User Manual

ZKFinger SDK for Android

Record of Release

Date	Version	Description	Author
2016-04-13	1.0.0	Basic version	Zhu Longhai
2016-04-27	1.0.1	Add some functions(See 3.1.7~3.1.12)	Scarchen

Important Claims

Thank you for using ZKTeco ZKFinger SDK. Please carefully read this document before using so that you can quickly grasp and use the ZKFinger SDK.

Copyright Statement

Document Use Instructions

Due to constant expansion of ZKFinger SDK software functions, the version of the ZKFinger SDK document is updated continuously. Therefore, please carefully read the ZKFinger SDK document while using the ZKFinger SDK software. Please forgive us for the inconvenience brought to you. You can contact our document writer and the contact information is as follows.

Company: ZKTeco (Xiamen) Software Base

Address: Room 403-02, No.32 Guanri Road, Phase II Software Park, Xiamen

Tel: 0592-5961369-8023

Website: www.zkteco.com

Email: scar.chen@zkteco.com

Table of Contents

1. Overview 1
2. Development Environment Setup 1
   2.1 Importing zkandroidfpreader.jar/zkandroidcore.jar 1
   2.2 Deploying the Algorithm Library 5
3. ZKFinger SDK 5
   3.1 FingerprintSensor.class 7
      3.1.1 open 7
      3.1.2 close 7
      3.1.3 setFingerprintfCaptureListener 9
      3.1.4 startCapture 11
      3.1.5 stopCapture 11
      3.1.6 destroy 13
   3.2 FingerprintCaptureListener.class 15
      3.2.1 captureOK 15
      3.2.2 captureError 17
      3.2.3 extractOK 17
      3.2.4 extractError 19
   3.3 ZKFingerService.class 19
      3.3.1 verify 19
      3.3.2 verifyId 20
      3.3.3 identify 21
      3.3.4 merge 24
      3.3.5 save 25
      3.3.6 get 26
      3.3.7 del 27
      3.3.8 clear 27
      3.3.9 count 28

3.4 Instance Codes (for details, see Demo-Android Studio) 28
3.4.1 MainActivity.java 28
Appendix 42

1. Overview

ZKFinger SDK is a set of application programming interfaces (APIs) provided by ZKTeco for development engineers. It is capable of managing ZKTeco fingerprint sensors in a unified manner. It can be used by developers in different classes to develop applications for controlling ZKTeco Android-based devices.

ZKFinger SDK supports the following functions:

Fingerprint sensors: ZKFinger SDK allows performing operations on fingerprint sensors, including initialization, startup, and shutdown.

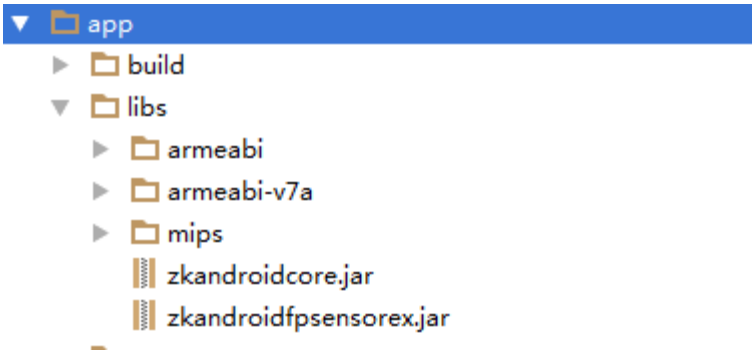
Fingerprint algorithm: The ZKFinger SDK supports 1:1 and 1:N fingerprint comparison.

2. Development Environment Setup

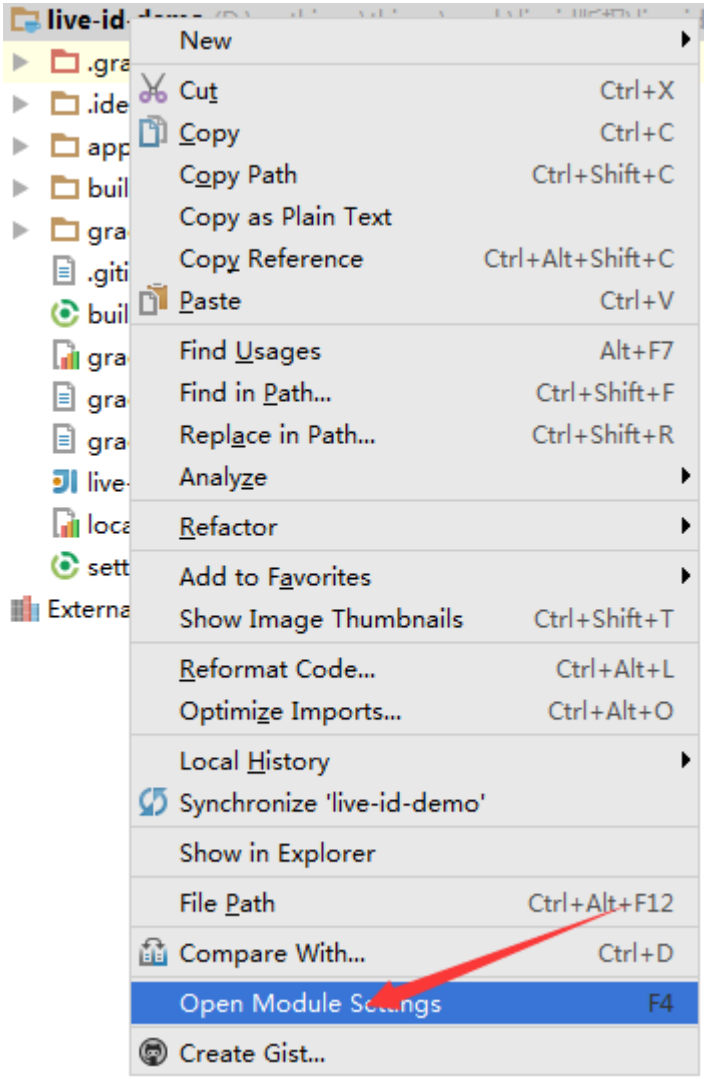
2.1 Importing zkandroidfpreader.jar/zkandroidcore.jar

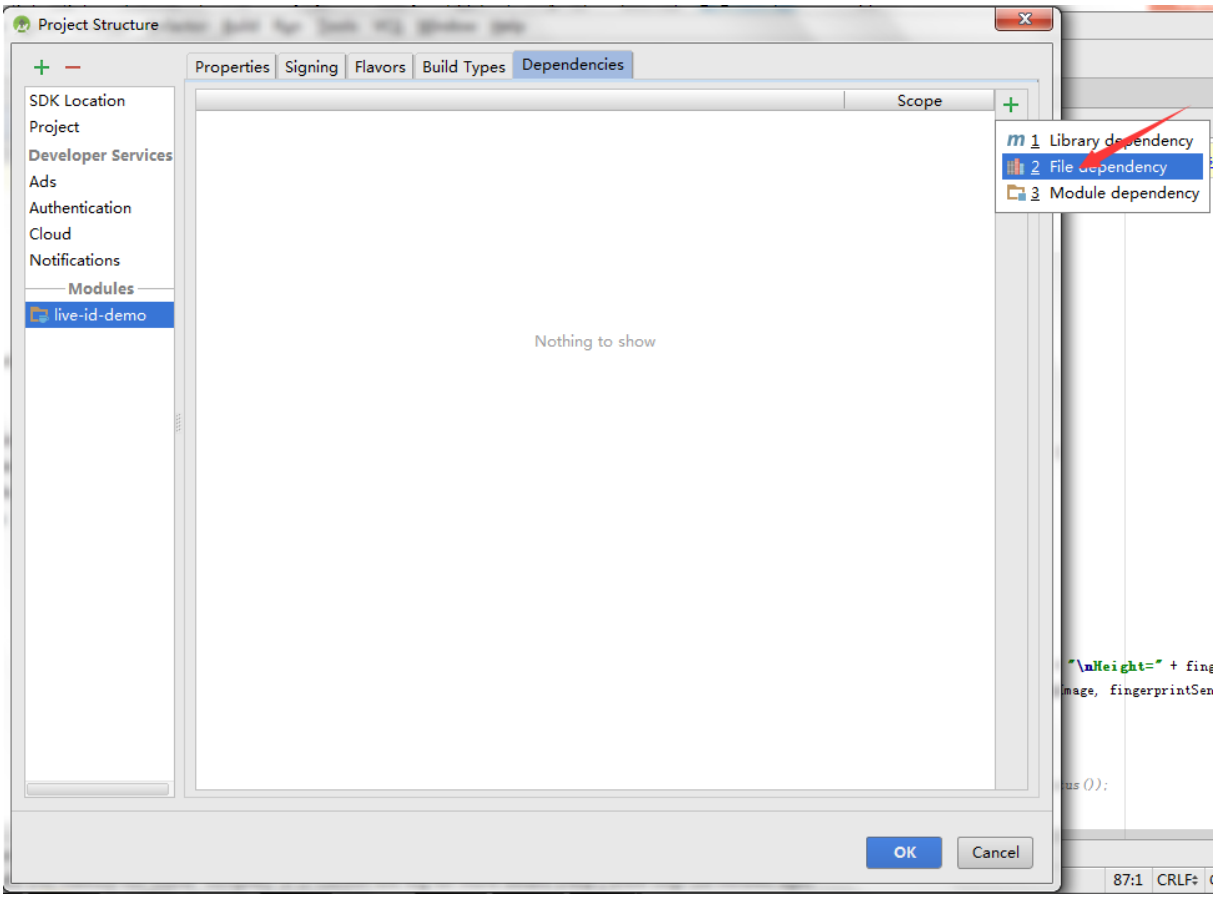
Open the SDK folder and import zkandroidfpreader.jar/zkandroidcore.jar in the Device directory to the application development tool (the following uses Android Studio as an example).

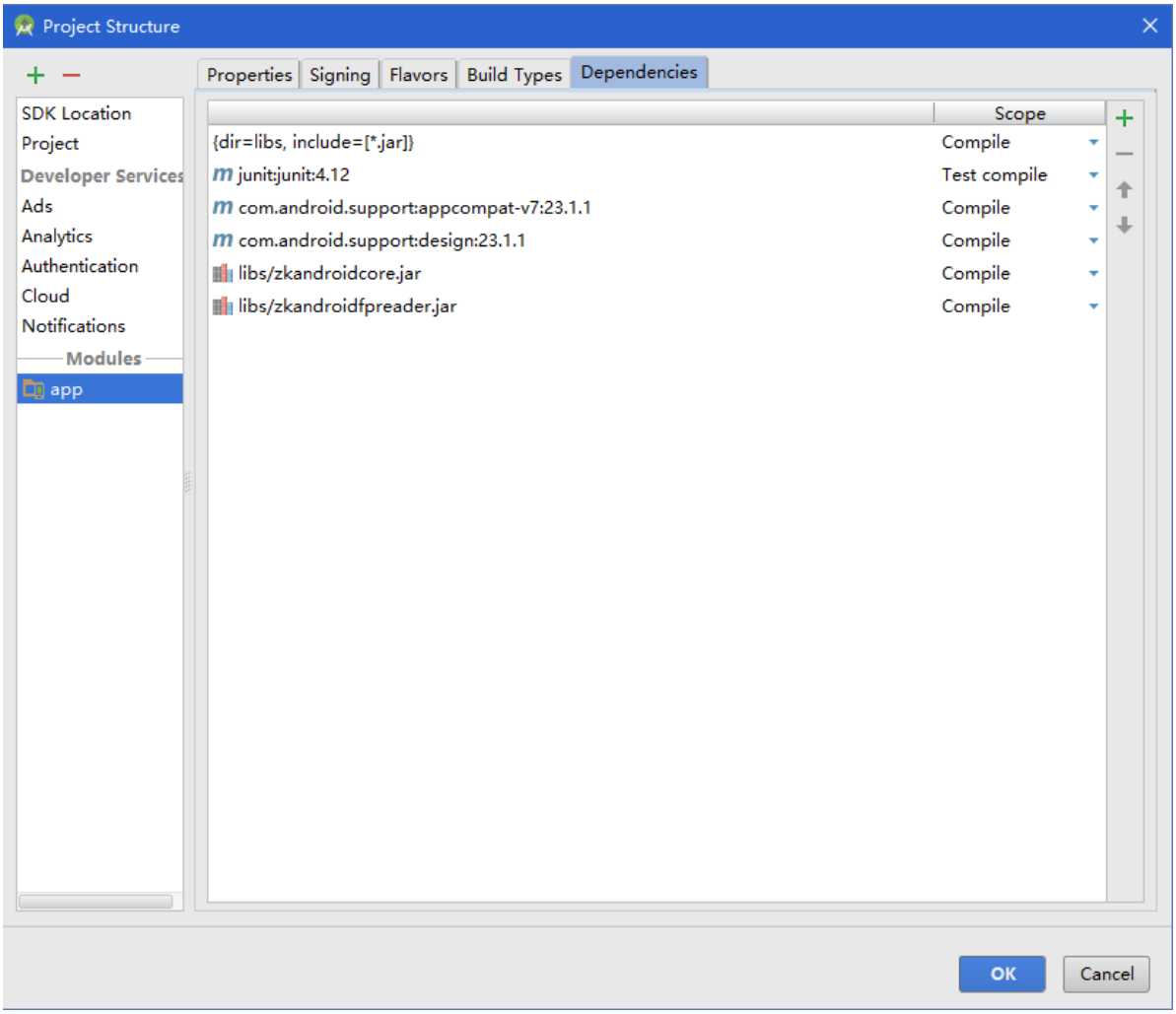
Step 1: Copy zkandroidfpreader.jar and zkandroidcore.jar into the app/libs directory.



Step 2: Right-click a project and choose Open Module Settings from the shortcut menu. In the Project Structure dialog box, click the Dependencies tab, click + and choose File dependency. Select zkandroidfpreader.jar/ zkandroidcore.jar in the libs directory and click OK to import zkandroidfpreader.jar/ zkandroidcore.jar.


New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails Ctrl+Shift+T
Reformat Code... Ctrl+Alt+L
Optimize Imports... Ctrl+Alt+O
Local History
Synchronize 'live-id-demo'
Show in Explorer
File Path Ctrl+Alt+F12
Compare With... Ctrl+D
Open Module Settings F4
Create Gist...


Project Structure
Properties
Signing
Flavors
Build Types
Dependencies
SDK Location
Project
Developer Services
Ads
Authentication
Cloud
Notifications
Modules
live-id-demo
Scope
Nothing to show
1 Library dependency
2 File dependency
3 Module dependency
OK
Cancel


Project Structure
Properties
Signing
Flavors
Build Types
Dependencies
SDK Location
Project
Developer Services
Ads
Analytics
Authentication
Cloud
Notifications
Modules
app
Scope
{dir=libs, include=[*.jar]}
Compile
junit:junit:4.12
Test compile
com.android.support:appcompat-v7:23.1.1
Compile
com.android.support:design:23.1.1
Compile
libs/zkandroidcore.jar
Compile
libs/zkandroidfpreader.jar
Compile
OK
Cancel

2.2 Deploying the Algorithm Library

Copy the zkfingerreader\libs directory to the corresponding libs directory in the project on the Android Studio.

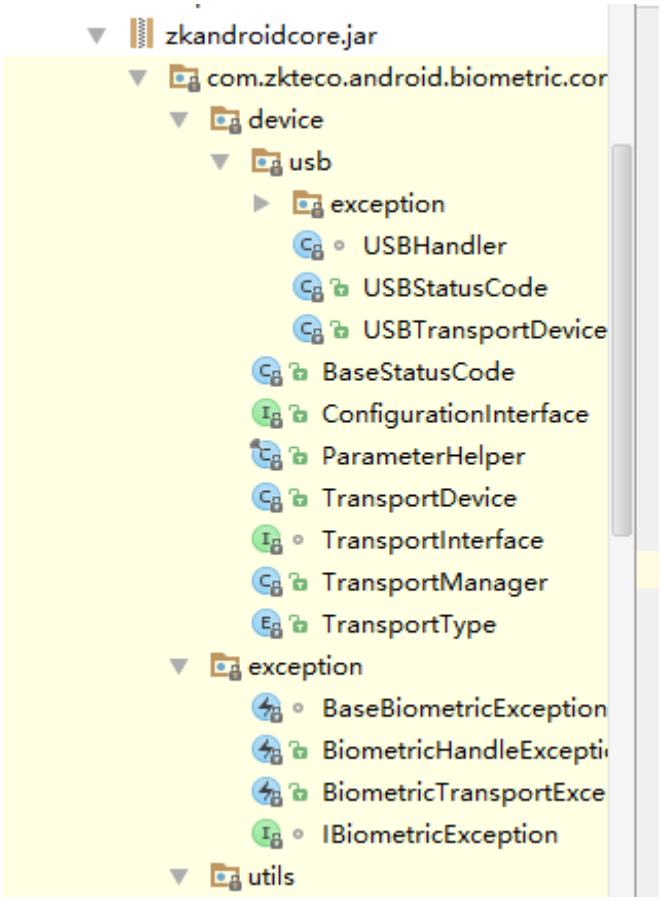
3. ZKFinger SDK

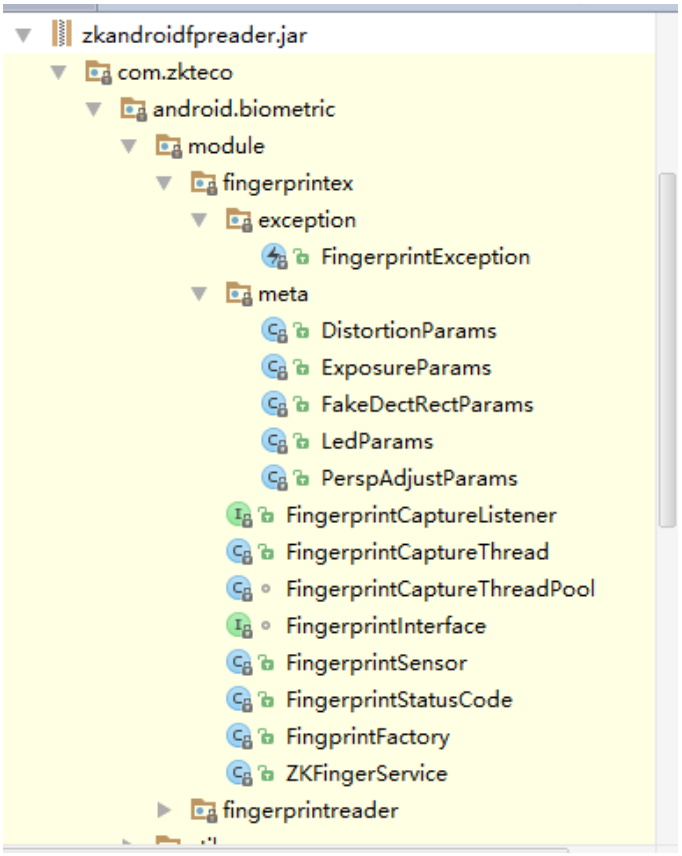
ZKFinger SDK abstracts function modules as classes. Users can call methods in classes to complete operations on the underlying hardware as well as handling of the fingerprint algorithm.

ZKFinger SDK includes the fingerprint sensor class, control device class, constant class, exception handling class, and algorithm handling class. The following table lists the types of the classes.

Class Name	Type
com.zkteco.android.biometric.module.fingerprintreader.FingerprintSensor	Fingerprint sensor class
com.zkteco.android.biometric.module.fingerprintreader.FingerpringStatusCode	Error code
com.zkteco.android.biometric.module.fingerprintreader.exception.FingerprintException	Exception handling class
com.zkteco.android.biometric.core.utils.LogHelper	Log tool class
com.zkteco.android.biometric.core.utils.ToolUtils	Auxiliary tool class
com.zkteco.android.biometric.module.fingerprintreader.FingerprintCaptureListener	Capture listening event class
com.zkteco.android.biometric.module.fingerprintreader.ZKFingerService	Algorithm handling class

The figures below show the structure of the SDK package.


zkandroidcore.jar
com.zkteco.android.biometric.cor
device
usb
exception
USBHandler
USBStatusCode
USBTransportDevice
BaseStatusCode
ConfigurationInterface
ParameterHelper
TransportDevice
TransportInterface
TransportManager
TransportType
exception
BaseBiometricException
BiometricHandleExcepti
BiometricTransportExce
IBiometricException
utils


zkandroidfpreader.jar
com.zkteco
android.biometric
module
fingerprintex
exception
FingerprintException
meta
DistortionParams
ExposureParams
FakeDectRectParams
LedParams
PerspAdjustParams
FingerprintCaptureListener
FingerprintCaptureThread
FingerprintCaptureThreadPool
FingerprintInterface
FingerprintSensor
FingerprintStatusCode
FingprintFactory
ZKFingerService
fingerprintreader

3.1 FingerprintSensor.class

FingerprintSensor.class is a class for controlling fingerprint sensors, which can be used to start and shut down a finger sensor, start fingerprint capture, and stop fingerprint capture.

3.1.1 open

[Function]

public void open(int index)

[Purpose]

The function is used to connect to a device.

[Parameter Description]

Index

Device index number. The value is determined based on the total number of connected fingerprint sensors.

Example:

When a total of one fingerprint sensor is connected, the index is 0.

When a total of two fingerprint sensors are connected, the index is 0 or 1.

. . .

[Return Value]

[Note]

An error code is thrown out through an exception.

3.1.2 close

[Function]

public void close(int index)

[Purpose]

This function is used to shut down a device.

[Parameter Description]

index

Device index number. The value is determined based on the total number of connected fingerprint sensors.

Example:

When a total of one fingerprint sensor is connected, the index is 0.

When a total of two fingerprint sensors are connected, the index is 0 or 1.

. . .

[Return Value]

[Note]

An error code is thrown out through an exception.

3.1.3 setFingerprintfCaptureListener

[Function]

public void setFingerprintCaptureListener(int index, FingerprintCaptureListener listener)

[Purpose]

This function is used to set a fingerprint capture listening event.

[Parameter Description]

index

Device index number. The value is determined based on the total number of connected fingerprint sensors.

Example:

When a total of one fingerprint sensor is connected, the index is 0.

When a total of two fingerprint sensors are connected, the index is 0 or 1.

. . .

Listener

Listening object

[Return Value]

[Note]

3.1.4 startCapture

[Function]

public void startCapture(int index)

[Purpose]

This function is used to capture a fingerprint image.

[Parameter Description]

index

Device index number. The value is determined based on the total number of connected fingerprint sensors.

Example:

When a total of one fingerprint sensor is connected, the index is 0.

When a total of two fingerprint sensors are connected, the index is 0 or 1.

. . .

[Return Value]

[Note]

Images are captured asynchronously and the callback interface set by setFingerprintCaptureListener returns fingerprint image templates. For details, see the demo.

3.1.5 stopCapture

[Function]

public void stopCapture (int index)

[Purpose]

This function is used to stop capturing fingerprint images asynchronously.

[Parameter Description]

Index

Device index number. The value is determined based on the total number of connected fingerprint sensors

Example:

When a total of one fingerprint sensor is connected, the index is 0.

When a total of two fingerprint sensors are connected, the index is 0 or 1.

. . .

[Return Value]

[Note]

Asynchronous capture of images is stopped.

3.1.6 destroy

[Function]

public static void destroy(FingerprintSensorfingerprintSensor)

[Purpose]

This function is used to destroy resources.

[Parameter Description]

fingerprintSensor

Device operation instance object

[Return Value]

[Note]

3.1.7 getImageWidth

[Function]

public static intgetImageWidth()

[Purpose]

This function is used to acquire the width of a fingerprint image.

[Parameter Description]

[Return Value]

Width of the fingerprint image

[Note]

3.1.8 getImageHeight

[Function]

public static intgetImageHeight()

[Purpose]

This function is used to acquire the height of a fingerprint image.

[Parameter Description]

[Return Value]

Height of the fingerprint image

[Note]

3.1.9 getLastTempLen

[Function]

public static intgetLastTempLen()

[Purpose]

This function is used to acquire the data length of a fingerprint template.

[Parameter Description]

[Return Value]

Data length of the fingerprint template

[Note]

See FingerprintCaptureListener.extractOK.

3.1.10 setFakeFunOn

[Function]

public static void setFakeFunOn(intfakeFunOn)

[Purpose]

This function is used to set the anti-fake function.

[Parameter Description]

fakeFunOn

0: Disables the anti-fake function.

1: Enables the anti-fake function.

[Return Value]

[Note]

The function supports only SILK20R.

3.1.11 getFakeFunOn

[Function]

public static intsetFakeFunOn()

[Purpose]

This function is used to acquire information about the anti-fake function.

[Parameter Description]

[Return Value]

0: Indicates that the anti-fake function is disabled.

1: Indicates that the anti-fake function is enabled.

[Note]

The function supports only SILK20R.

3.1.12 getFakeStatus

[Function]

public static intgetFakeStatus()

[Purpose]

This function is used to acquire the current fingerprint status.

[Parameter Description]

[Return Value]

If the lowest five bits are all 1's, the fingerprint image is true: (value&0x1F) == 31.

If other values are returned, the fingerprint image is suspicious.

[Note]

The function supports only SILK20R and setFakeFunOn(1) must be set.

3.2 FingerprintCaptureListener.class

FingerprintCaptureListener.class is a fingerprint capture listening event class.

3.2.1 captureOK

[Function]

voidcaptureOK(byte[] fpImage);

[Purpose]

This function indicates that a fingerprint image is captured successfully.

[Parameter Description]

fpImage

Fingerprint image

[Return Value]

[Note]

3.2.2 captureError

[Function]

voidcaptureError(FingerprintSensorException e)

[Purpose]

This function indicates that a fingerprint image fails to be captured.

[Parameter Description]

e: abnormal object (see 3.3)

[Return Value]

None.

[Note]

3.2.3 extractOK

[Function]

voidextractOK(byte[] fpTemplate);

[Purpose]

This function indicates that a fingerprint template is extracted successfully.

[Parameter Description]

fpTemplate

Fingerprint template

[Return Value]

None.

[Note]

3.2.4 extractError

[Function]

voidextractError(interrno)

[Purpose]

This function indicates that a fingerprint template fails to be extracted.

[Parameter Description]

For the description of error codes, see Appendix

[Return Value]

[Note]

3.3 ZKFingerService.class

ZKFingerService.class is a fingerprint algorithm handling class.

3.3.1 verify

[Function]

static public int verify(byte[] temp1, byte[] temp2)

[Purpose]

This function is used to compare two fingerprint templates.

[Parameter Description]

temp1

The first fingerprint template data array

temp2

The second fingerprint template data array

[Return Value]

Fingerprint template matching score, with the value ranging from 0 to 100.

Suggestion: When the return value is larger than the preset threshold or default value (23), it is considered that two fingerprint templates are matched successfully.

[Note]

For the description of error codes, see Appendix

.

3.3.2 verifyId

[Function]

static public intverifyId(byte[] temp, String id)

[Purpose]

This function is used to match a fingerprint template with the fingerprint template of a specified ID in the buffer (1:1 comparison). A matching score will be returned.

[Parameter Description]

temp

Fingerprint template data array

id

ID of the target fingerprint template for comparison. The ID is a string, with the length not longer than 20 bytes.

Example: String id = "ZKTeco_20_01"

[Return Value]

Fingerprint template matching score, with the value ranging from 0 to 100.

When the return value is larger than the preset threshold or default value (23), two fingerprint templates are matched successfully.

3.3.3 identify

[Function]

static public int identify(byte[] temp, byte []idstr, int threshold, int count)

[Purpose]

This function is used to match fingerprint template data with fingerprint template data in the buffer (that is, 1:N comparison). It returns the count of matching scores that are higher than the threshold. If the actual count is smaller than the preset count, the actual count is returned. For example, if count is set to 2, the return value may be 0, 1, or 2.

[Parameter Description]

temp

Fingerprint template data array

idstr

Array of the IDs and scores of fingerprint templates that are matched successfully.

It is recommended that the array size be set to 50*count.

Multiple results are sorted by matching score in descending order. Each result contains the ID and matching score. IDs and matching scores are separated by/t and results are separated by /n.

Example: String resultStrs = new String(idstr);

If the return value of resultStrs is 1001'\t'95'\n'1002'\t'85'\n'1003'\t75, three comparison results are returned:

Result 1: ID: 1001, score: 95

Result 2: ID: 1002, score: 85

Result 3: ID: 1003, score: 75

For details about codes, see [Example Code]

.

threshold

Matching threshold, with the value ranging from 0 to 100. Note: This value will replace the preset threshold.

Matching threshold determines the level of similarity between two fingerprint templates. A higher matching threshold indicates a stricter requirement for the similarity. When two fingerprint templates are compared and the return value is greater than the preset threshold, the comparison is successful. Otherwise, the comparison fails. Flexible adjustment can be made based on conditions in specific applications.

Matching threshold description

False Rejection Rate	False Accept Rate	Matching Threshold	
		1:N	1:1
High	Low	65	45
Intermediate	Intermediate	55	35
Low	High	45	30

count

Expected number of returned results. The recommended value is 1.

Example:

When threshold is set to 50 and count is set to 3, if only two fingerprint templates meet the matching threshold during comparison, the return value is 2.

[Return Value]

Succeeded The actual comparison result count is returned. The return value may be smaller than value of count. The value 1 indicates that one fingerprint template is matched successfully. The value 0 indicates that no fingerprint template is matched successfully.

Otherwise, an error code is returned.

[Note]

For the description of error codes, see Appendix

[Example Code]

```
int ret = 0;

byte[] idstr = new byte[50];

ret = ZKFingerServer.identify(tempIdentify, idstr, 50, 1);

if( ret == 1 ){

String resultString = new String(idstr);

textView.setText("id: " + resultString.split("\t")[0] + ", score: " +
resultString.split("\t")[1] );

}else{

textView.setText("Please try again!");

}
```

3.3.4 merge

[Function]

static public int merge(byte[] temp1, byte[] temp2, byte[] temp3, byte[] temp)

[Purpose]

This function is used to merge three fingerprint templates as an enrolled template.

[Parameter Description]

temp1

Fingerprint template 1 to be enrolled

temp2

Fingerprint template 2 to be enrolled

temp3

Fingerprint template 3 to be enrolled

temp

Output enrolled fingerprint template

[Return Value]

>0 the length of enrolled fingerprint template

Otherwise, an error code is returned.

[Note]

For the description of error codes, see Appendix

.

3.3.5 save

[Function]

static public save (byte[] temp, String id)

[Purpose]

This function is used to store fingerprint template data to the buffer. Users can save required fingerprint data in the database to the buffer by invoking the save method.

[Note]

Ensure that the fingerprint algorithm library is initialized successfully when the save method is called.

[Parameter Description]

temp

Transferred fingerprint template data array

id

Fingerprint template ID

The ID is a string, with the length not longer than 20 bytes.

Example: String id = "ZKTeco_20_01"

[Return Value]

0 Succeeded

Otherwise, an error code is returned.

[Note]

For the description of error codes, see Appendix

.

3.3.6 get

[Function]

static publicint get (byte[] temp, String id)

[Purpose]

This function is used to acquire the fingerprint template data of a specified ID in the buffer.

[Parameter Description]

temp

Fingerprint template data array of a specified ID. It is recommended that the array size be set to 1024*3, to ensure that there is sufficient space for storing fingerprint template data.

id

ID of the fingerprint template to be acquired

The ID is a string, with the length not longer than 20 bytes.

Example: String id = "ZKTeco_20_01"

[Return Value]

>0 Succeeded. The value is the fingerprint template length.

Otherwise, an error code is returned.

[Note]

For the description of error codes, see Appendix

.

3.3.7 del

[Function]

static publicint del(String id)

[Purpose]

This function is used to delete an enrolled fingerprint template from the memory.

[Parameter Description]

id

ID of the fingerprint template to be deleted

The ID is a string, with the length not longer than 20 bytes.

Example: String id = "ZKTeco_20_01"

[Return Value]

0 Succeeded

Otherwise, an error code is returned.

[Note]

For the description of error codes, see Appendix

.

3.3.8 clear

[Function]

static public int clear ()

[Purpose]

This function is used to clear all fingerprint templates from the buffer.

[Parameter Description]

None.

[Return Value]

0 Succeeded

Otherwise, an error code is returned.

[Note]

For the description of error codes, see Appendix

.

3.3.9 count

[Function]

static public int count()

[Purpose]

This function is used to acquire the number of fingerprint templates stored in the buffer.

[Parameter Description]

None.

[Return Value]

The number of fingerprint templates stored in the buffer is returned.

3.4 Instance Codes (for details, see Demo-Android Studio)

3.4.1 MainActivity.java

```
packagecom.zkteco.live_id_demo;


importandroid.graphics.Bitmap;
importandroid.os.Bundle;
importandroid.support.design.widget.FloatingActionButton;
importandroid.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
```

```
importandroid.util.Log;

importandroid.view.View;

importandroid.view.Menu;

importandroid.view.MenuItem;

importandroid.widget.ImageView;

importandroid.widget.TextView;


importcom.zkteco.android.biometric.core.device.ParameterHelper;

importcom.zkteco.android.biometric.core.device.TransportType;

importcom.zkteco.android.biometric.core.utils.LogHelper;

importcom.zkteco.android.biometric.core.utils.ToolUtils;

import
com.zkteco.android.biometric.module.fingerprintreader.FingerprintCapture
Listener;

import
com.zkteco.android.biometric.module.fingerprintreader.FingerprintSensor;

import
com.zkteco.android.biometric.module.fingerprintreader.FingprintFactory;

import
com.zkteco.android.biometric.module.fingerprintreader.ZKFingerService;

import
com.zkteco.android.biometric.module.fingerprintreader.exception.Fingerpr
intException;


importjava.io.File;

importjava.io.FileNotFoundException;

importjava.io.FileOutputStream;
```

```
importjava.io.IOException;

importjava.util.HashMap;
```

```
importjava.util.Map;


public class MainActivity extends AppCompatActivity {

private static final int VID = 6997;

private static final int PID = 288;

privateTextViewtextView = null;

privateImageViewimageView = null;

privatebooleanbstart = false;

privatebooleanisRegister = false;

privateintuid = 1;

private byte[][] regtemparray = new byte[3][2048];  //register template
buffer array

privateintenrollidx = 0;

private byte[] lastRegTemp = new byte[2048];


privateFingerprintSensorfingerprintSensor = null;

    @Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity_main);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);

setSupportActionBar(toolbar);


FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);

fab.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
public void onClick(View view) {
Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
        }
    });

textView = (TextView)findViewById(R.id.textView);
imageView = (ImageView)findViewById(R.id.imageView);

startFingerprintSensor();
  }

private void startFingerprintSensor() {
    // Define output log level
LogHelper.setLevel(Log.WARN);
    // Start fingerprint sensor
    Map fingerprintParams = new HashMap();
    //set vid
fingerprintParams.put(ParameterHelper.PARAM_KEY_VID, VID);
    //set pid
fingerprintParams.put(ParameterHelper.PARAM_KEY_PID, PID);
```

```
fingerprintSensor = FingprintFactory.createFingerprintSensor(this,
TransportType.USB, fingerprintParams);
    }
    @Override
publicbooleanonCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.menu_main, menu);
return true;
    }

    @Override
publicbooleanonOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
int id = item.getItemId();

        //noinspectionSimplifiableIfStatement
if (id == R.id.action_settings) {
return true;
        }

returnsuper.onOptionsItemSelected(item);
    }
```

```
public void saveBitmap(Bitmap bm) {
        File f = new File("/sdcard/fingerprint", "test.bmp");
if (f.exists()) {
f.delete();
        }
FileOutputStream out = null;
try {
out = new FileOutputStream(f);
bm.compress(Bitmap.CompressFormat.PNG, 90, out);
out.flush();
out.close();
            } catch (FileNotFoundException e) {
e.printStackTrace();
}catch (IOException e) {
e.printStackTrace();
            }
    }
public void OnBnBegin(View view) throws FingerprintException
    {
try {
if (bstart) return;
fingerprintSensor.open(0);
finalFingerprintCaptureListener listener = new
FingerprintCaptureListener() {
                @Override
```

```
public void captureOK(final byte[] fpImage) {

runOnUiThread(new Runnable() {

                @Override

public void run() {

if(null != fpImage)

                    {

ToolUtils.outputHexString(fpImage);

LogHelper.i("width=" + fingerprintSensor.getImageWidth() + "\nHeight=" +
fingerprintSensor.getImageHeight());

                        Bitmap bitmapFp =
ToolUtils.renderCroppedGreyScaleBitmap(fpImage,
fingerprintSensor.getImageWidth(), fingerprintSensor.getImageHeight());

                        //saveBitmap(bitmapFp);

imageView.setImageBitmap(bitmapFp);

                    }

                    //textView.setText("FakeStatus:" +
fingerprintSensor.getFakeStatus());

                }

            });

        }

        @Override

public void captureError(FingerprintException e) {

finalFingerprintExceptionexp = e;

runOnUiThread(new Runnable() {

                @Override

public void run() {
```

```
LogHelper.d("captureErrorerrno=" + exp.getErrorCode() +
                        ",Internal error code: " + exp.getInternalErrorCode()
+ ",message=" + exp.getMessage());
                }
            });
        }
        @Override
public void extractError(final int err)
        {
runOnUiThread(new Runnable() {
                @Override
public void run() {
textView.setText("extract fail, errorcode:" + err);
                }
            });
        }

        @Override
public void extractOK(final byte[] fpTemplate)
        {
final byte[] tmpBuffer = fpTemplate;
runOnUiThread(new Runnable() {
                @Override
public void run() {
if (isRegister) {
```

```
byte[] bufids = new byte[256];

int ret = ZKFingerService.identify(tmpBuffer, bufids, 55, 1);

if (ret > 0)

                    {

                        String strRes[] = new String(bufids).split("\t");

textView.setText("the finger already enroll by " + strRes[0] + ",cancel
enroll");

isRegister = false;

enrollidx = 0;

return;

                    }

if (enrollidx> 0 &&ZKFingerService.verify(regtemparray[enrollidx-1],
tmpBuffer) <= 0)

                    {

textView.setText("please press the same finger 3 times for the
enrollment");

return;

                    }

System.arraycopy(tmpBuffer, 0, regtemparray[enrollidx], 0, 2048);

enrollidx++;

if (enrollidx == 3) {

byte[] regTemp = new byte[2048];

if (0 < (ret = ZKFingerService.merge(regtemparray[0], regtemparray[1],
regtemparray[2], regTemp))) {

ZKFingerService.save(regTemp, "test" + uid++);

System.arraycopy(regTemp, 0, lastRegTemp, 0, ret);
```

```
textView.setText("enroll succ");
                        } else {
textView.setText("enroll fail");
                        }
isRegister = false;
                    } else {
textView.setText("You need to press the " + (3 - enrollidx) + "time
fingerprint");
                    }
                } else {
byte[] bufids = new byte[256];
int ret = ZKFingerService.identify(tmpBuffer, bufids, 55, 1);
if (ret > 0) {
                        String strRes[] = new String(bufids).split("\t");
textView.setText("identify succ, userid:" + strRes[0] + ", score:" +
strRes[1]);
                    } else {
textView.setText("identify fail");
                    }
                }
            }
        });
    }
```

```
        };


fingerprintSensor.setFingerprintCaptureListener(0, listener);
fingerprintSensor.startCapture(0);
bstart = true;
textView.setText("start capture succ");
}catch (FingerprintException e)
      {
textView.setText("begin capture fail.errorcode:"+ e.getErrorCode() + "err
message:" + e.getMessage() + "inner code:" + e.getInternalErrorCode());
      }
   }

public void OnBnStop(View view) throws FingerprintException
   {
try {
if (bstart)
        {
            //stop capture
fingerprintSensor.stopCapture(0);
bstart = false;
fingerprintSensor.close(0);
textView.setText("stop capture succ");
        }
else
```

```
            {
textView.setText("already stop");
            }
        } catch (FingerprintException e) {
textView.setText("stop fail, errno=" + e.getErrorCode() + "\nmessage=" +
e.getMessage());
        }
    }

public void OnBnEnroll(View view) {
if (bstart) {
isRegister = true;
enrollidx = 0;
textView.setText("You need to press the 3 time fingerprint");
        }
else
        {
textView.setText("please begin capture first");
        }
    }

public void OnBnVerify(View view) {
if (bstart) {
isRegister = false;
enrollidx = 0;
```

```
}else {
textView.setText("please begin capture first");
        }
    }
}
```

Appendix

Error code	Description
-20001	Failed to start the device.
-20002	Failed to shut down the device.
-20003	Failed to acquire GPIO data.
-20004	Failed to set the GPIO.
-20005	Failed to read data in the EEPROM.
-20006	Failed to acquire images from the USB flash drive.
-20007	Failed to probe images in the USB flash drive.
-20008	Insufficient input buffer.
-20009	Data reading abnormality
-20010	Failed to capture a fingerprint image.
-20011	Failed to decrypt image data.
-20012	Failed to start the capture thread.
-20013	Failed to stop the capture thread.
-20014	Failed to initialize the fingerprint sensor.
-20015	Failed to set correction parameters.
ERR_NOT_FOUND -5000	Failed to find the fingerprint template of the specified ID.
ERR_PARAM -5002	Parameter error.
ERR_TEMPLATE -5003	Fingerprint template error.
ERR_METHOD -5004	Method error.