扫码SDK说明文档

1. ScanManagerService

1.1 获取ScanManagerService对象

1.1.1 创建 .aidl文件

文件名：IScanManager.aidl

路径：kozen\os\scan\

```
package kozen.os.scan;

interface IScanManager {
    void openScanner();
    void closeScanner();
    void startScan();
    void stopScan();
    boolean isScannerOpen();
}
```

1.1.2 获取 Stub接口的实例

```
import kozen.os.scan.IScanManager;

IScanManager.Stub.asInterface(ServiceManager.getService("scan_service"));
```

1.2 aidl方法说明

方法名	参数	返回值	描述
openScanner	void	void	开启扫码头
closeScanner	void	void	关闭扫码头
startScan	void	void	获取图像
stopScan	void	void	停止获取图像
isScannerOpen	void	boolean	是否开启了扫码头

1.3 代码示例

```
try {
    if(!IScanManager.Stub.asInterface(ServiceManager.getService(
        "scan_service")).isScannerOpen()){
        return;
    }
} catch (RemoteException e) {
}

try{
    IScanManager.Stub.asInterface(ServiceManager.getService(
        "scan_service")).startScan();
} catch (RemoteException e) {
}
```

2. ScanService

2.1 获取ScanService

```
import vendor.kozen.hardware.scan.V1_0.IScan;

private static volatile IScan mScanService = null;

mScanService = IScan.getService();
```

2.2 ScanService方法说明

方法名	参数	返回值	描述
initContext	String	boolean	初始化解码库，返回是否成功
releaseContext	void	void	释放解码库
setCallback	IScanCallback	void	设置回调，IScanCallback实现见2.3
setDefaultCallback	IScanCallback	void	设置默认回调，setCallback(null)时生效
setFeedbackCallback	IScanCallback	void	反馈回调，结果反馈与结果回调分离
setScanResult	String, String	void	设置扫码结果
onTimeOut	void	void	执行IScanCallback中的onTimeOut
getSdkVersion	void	String	获得SDK版本号
startDecode	void	boolean	开始解码，返回是否成功
stopDecode	void	boolean	停止解码，返回是否成功
getDecodeState	void	int	获取解码状态，未获取到返回-1，空闲状态返回0，正在解码返回1，解码成功返回2
setTimeout	int	boolean	设置超时时间，单位：毫秒，返回是否成功
enableAllSymbol	void	boolean	开启解码器支持的所有码制，返回是否成功
disableAllSymbol	void	boolean	关闭解码器支持的所有码制，返回是否成功
setSymbolEnable	int, boolean	boolean	根据码制ID开关解码器码制，boolean参数为true开启，为false关闭；int参数见2.4
setDecoderTag	int, int	boolean	设置解码器码制属性，见2.7
getDecoderTag	int	int	获取解码器码制属性
setWorkMode	int	boolean	设置工作模式，返回是否成功
getWorkMode	void	int	获取工作模式，工作模式取值见2.5
setFlashMode	int	boolean	设置闪光灯模式，0为闪烁，1为静止
getFlashMode	void	int	获取闪光灯模式
setViewSize	int	boolean	设置视图大小，返回是否成功
getViewSize	void	int	获取视图大小，视图大小取值见2.6
enableFlash	boolean	boolean	开关闪光灯，返回是否成功
enableAim	boolean	boolean	开关瞄准灯（红灯），返回是否成功

2.3 IScanCallback实现

2.3.1 新建一个 ScanCallback类

2.3.2 导入 IScanCallback

```
import vendor.kozen.hardware.scan.V1_0.IScanCallback;
```

2.3.3 继承 IScanCallback.Stub

```
public class ScanCallback extends IScanCallback.Stub {
    private final String TAG = "JavaScanCallback";

    @Override
    public void onScanResult(String barcode, String result) {
        Log.d(TAG, "barcode = " + barcode);
        Log.d(TAG, "result = " + result);
        // ...
    }

    @Override
    public void onTimeOut(){
        // ...
    }
}
```

2.3.4 实现 onScanResult方法（必须）

```
@Override
public void onScanResult(String barcode, String result) {
    Log.d(TAG, "barcode = " + barcode);
    Log.d(TAG, "result = " + result);
    // ...
}
```

参数	说明
barcode	码制
result	扫码结果

2.3.5 实现 onTimeOut方法（必须）

```
@Override
public void onTimeOut(){
    // ...
}
```

2.3.6 ScanCallback类说明

方法	说明
onScanResult(String barcode, String result)	扫码成功后执行
onTimeOut()	扫码超时后执行

2.3.7 建议

使用时可以再建一个类来继承 ScanCallback，将自己的代码写到新建的类里，从而不用每次都实现onScanResult和 onTimeOut方法。

2.4 setSymbolEnable 参数

int参数为码制ID，对应如下：

```
// Native code type definition
public static final int E3_SYM_DOTCODE       = 0; // DotCode
public static final int E3_SYM_EAN13         = 1; // EAN-13
public static final int E3_SYM_EAN8          = 2; // EAN-8
public static final int E3_SYM_UPCA          = 3; // UPC-A
public static final int E3_SYM_UPCE          = 4; // UPC-E
public static final int E3_SYM_CODE11        = 5; // C11
public static final int E3_SYM_CODE32        = 6; // C32
public static final int E3_SYM_CODE39        = 7; // C39
public static final int E3_SYM_CODE93        = 8; // C93
public static final int E3_SYM_CODE128       = 9; // C128
public static final int E3_SYM_PDF417        = 10; // PDF417
public static final int E3_SYM_MICROPDF417   = 11; // MICROPDF417
public static final int E3_SYM_QRCODE        = 12; // QRCODE
public static final int E3_SYM_DATAMATRIX    = 13; // DATA MATRIX
public static final int E3_SYM_CODABAR       = 14; // Codabar
public static final int E3_SYM_AZTEC         = 15; // AZTEC
public static final int E3_SYM_GS1_128       = 16; // GSI-128
public static final int E3_SYM_GS1_DATABAR   = 17; // GSI DATABAR
public static final int E3_SYM_GS1_LIMITED   = 18; // GS1 DataBar Limited
public static final int E3_SYM_GS1_EXPANDED  = 19; // GS1 DataBar Expanded
public static final int E3_SYM_TRIOPTIC      = 20; // TRIOPTIC
public static final int E3_SYM_ITF25         = 21; // ITF25
public static final int E3_SYM_MATRIX25      = 22; // MATRIX 25
public static final int E3_SYM_IATA25        = 23; // IATA-25
public static final int E3_SYM_INDUSTRIAL25  = 24; // INDUSTRIAL 25
public static final int E3_SYM_CODABLOCK_F   = 25; // CODABLOCK F
public static final int E3_SYM_CODABLOCK_A   = 26; // CODABLOCK A
public static final int E3_SYM_MSI           = 27; // MSI
public static final int E3_SYM_COMPOSITE     = 28; // GS1 Composite
public static final int E3_SYM_TELEPEN       = 29; // TELEPEN
public static final int E3_SYM_MAXICODE      = 30; // MAXICODE
public static final int E3_SYM_HANXIN        = 31; // HANXIN
public static final int E3_SYM_USPS_4_STATE  = 32; // USPS 4-State
public static final int E3_SYM_HK25          = 33; // HK25
public static final int E3_SYM_GRID_MATRIX   = 34; // Grid Matrix
public static final int E3_SYM_GS1_DATAMATRIX = 35; // GS1 DATA MATRIX
```

2.5 WorkMode 取值

取值	描述
0	单次扫码
1	连续扫码不过滤
2	连续扫码过滤相同结果

2.6 ViewSize 取值

取值	视图大小
0	100%
1	75%
2	50%
3	25%
4	12.5%
5	6.25%
6	3.125%
7	1.5625%

2.7 DecoderTag

2.7.1 setDecoderTag 使用

其中tag相当于码制属性的ID，是固定的，且每个属性的ID都不相同；而value则是码制属性的取值，除了 "最小长度" 和 "最大长度" 取值是一个在码制所支持的长度内的数字，其它码制属性大多只有两个取值，对应功能开与关

```
public static int setDecoderTag(int tag, int value) {
    int ret = 0;
    try {
        if (null == mScanService) {
            ret = -1;
        } else {
            Log.d(TAG, "setDecoderTag, tag = " + tag + ",value = " + value);
            mScanService.setDecoderTag(tag, value);
        }
    } catch (RemoteException e) {
        Log.e(TAG, "setDecoderTag, Exception = " + e);
    }
    return ret;
}
```

2.7.2 以GS1 DATABAR为例

GS1 DATABAR码制有四个属性，如下：

属性	说明
TAG_RSS_LIMITED_ENABLED	是否支持限定型（取值1或0，限定型长度固定）
TAG_RSS_EXPANDED_ENABLED	是否支持扩展型（取值1或0）
TAG_RSS_EXPANDED_MIN_LENGTH	扩展型最小长度限制（最小取值为1）
TAG_RSS_EXPANDED_MAX_LENGTH	扩展型最大长度限制（最大取值为128）

```
public static final int TAG_RSS_LIMITED_ENABLED          = 0x1A022002;
public static final int TAG_RSS_EXPANDED_ENABLED         = 0x1A022003;
public static final int TAG_RSS_EXPANDED_MIN_LENGTH      = 0x1A022004;
public static final int TAG_RSS_EXPANDED_MAX_LENGTH      = 0x1A022005;
//GS1 DATABAR(RSS)
setDecoderTag(TAG_RSS_LIMITED_ENABLED, true ? 1 : 0);
setDecoderTag(TAG_RSS_EXPANDED_ENABLED, true ? 1 : 0);
setDecoderTag(TAG_RSS_EXPANDED_MIN_LENGTH, 1);
setDecoderTag(TAG_RSS_EXPANDED_MAX_LENGTH, 128);
```

这个例子中TAG_RSS_LIMITED_ENABLED和TAG_RSS_EXPANDED_ENABLED的取值都是1或0，但是并不是所有功能开关的取值都是1或0

3. 扫码与结果处理

3.1 初始化

3.1.1 扫码头

开启扫码头

```
try {
    IScanManager.Stub.asInterface(ServiceManager.getService(
            "scan_service")).openScanner();
} catch (RemoteException e) {
    Log.e(TAG, "initDecoderLibrary openScanner, Exception = " + e);
}
```

关闭扫码头

```
try {
    IScanManager.Stub.asInterface(ServiceManager.getService(
            "scan_service")).closeScanner();
} catch (RemoteException e) {
}
```

3.1.2 解码器

调用 initContext(String name) 初始化解码库

```
if(null == mScanService) {
    try {
        mScanService = IScan.getService();
        initResult = mScanService.initContext("e3scan");
        Log.d(TAG, "initScanContext, initResult = " + initResult);
    } catch (RemoteException e) {
        Log.e(TAG, "initDecoderLibrary, Exception = " + e);
    }
}
```

初始化设置

1. 码制开关
2. 工作模式
3. 闪光灯开关

4. 瞄准灯开关
5. 码制属性设置

3.1.3 设置回调

1. setCallback
2. setDefaultCallback
3. setFeedbackCallback

示例

```
if(null != mScanService) {
    try {
        //setDefaultCallback
        mScanService.setDefaultCallback(callback);
    } catch (RemoteException e) {
        Log.e(TAG, "setupDefaultScanResultCallback, Exception = " + e);
    }
} else {
    Log.e(TAG, "setupDefaultScanResultCallback mScanService is null");
}
```

三个设置回调函数实现的效果相同，其中 setCallback 会覆盖掉 setDefaultCallback，setFeedbackCallback 分离出来专用于扫码结果提示

3.2 开始解码，停止解码

3.2.1 开始解码

1. 判断扫码头是否开启

```
try {
    if(!IScanManager.Stub.asInterface(ServiceManager.getService(
            "scan_service")).isScannerOpen()){
        return;
    }
} catch (RemoteException e) {
}
```

2. 调用 startScan() 获取图像

```
try{
    IScanManager.Stub.asInterface(ServiceManager.getService(
            "scan_service")).startScan();
} catch (RemoteException e) {
}
```

3. 调用 srartDocode() 解码

```
if(null != mScanService) {
    try {
        mScanService.startDecode();
    } catch (RemoteException e) {
        Log.e(TAG, "doDecode, Exception = " + e);
    }
}
```

3.2.2 停止解码

1. 判断是否正在解码

```
private static final int SCAN_STATE_UNKNOWN = -1;
private static final int SCAN_STATE_IDLE = 0;
private static final int SCAN_STATE_DECODE = 1;
private static final int SCAN_STATE_DONE = 2;

public static boolean IsDecoding() {
    Log.d(TAG, "isDecoding+");
    int state = SCAN_STATE_UNKNOWN;
    boolean isDecoding = false;

    try {
        if (null != mScanService) {
            state = mScanService.getDecodeState();
        }
    } catch (RemoteException e) {
        Log.e(TAG, "isDecoding, Exception = " + e);
    }

    switch (state) {
        case SCAN_STATE_UNKNOWN:
            isDecoding = false;
            break;
        case SCAN_STATE_IDLE:
            isDecoding = false;
            break;
        case SCAN_STATE_DECODE:
            isDecoding = true;
            break;
        case SCAN_STATE_DONE:
            isDecoding = false;
            break;
    }

    Log.d(TAG, "isDecoding-, state = " + state);
    return isDecoding;
}
```

2. 停止解码

```
if (IsDecoding()) {
    try{
        IScanManager.Stub.asInterface(ServiceManager.getService(
                "scan_service")).stopScan();
    } catch (RemoteException e) {
```

```
    }

    try {
        if (null != mScanService) {
            mScanService.stopDecode();
        }
    } catch (RemoteException e) {
        Log.e(TAG, "stopDecode, Exception = " + e);
    }
}
```

3.3 扫码结果输出

3.3.1 广播输出

```
String actionName = "com.xcheng.scanner.action.BARCODE_DECODING_BROADCAST";
String barcodeData = "EXTRA_BARCODE_DECODING_DATA";
String symbologyType = "EXTRA_BARCODE_DECODING_SYMBOLE";

Intent intent = new Intent(actionName);
intent.putExtra(symbologyType, symbology);
intent.putExtra(barcodeData, barcode);
sendBroadcast(intent);
```

3.3.2 模拟键盘输出

```
char[] chars = setEncodingFormat(barcode).toCharArray();
Instrumentation inst = new Instrumentation();
for(int i = 0; i < chars.length; i++) {
    Log.i(TAG, "sendKeyboardData: result = " + String.valueOf(chars[i]));
    switch (chars[i]) {
        case '\t':
            inst.sendKeyDownUpSync(KeyEvent.KEYCODE_TAB);
            break;
        case '\n':
            inst.sendKeyDownUpSync(KeyEvent.KEYCODE_ENTER);
            break;
        case ' ':
            inst.sendKeyDownUpSync(KeyEvent.KEYCODE_SPACE);
            break;
        default:
            inst.sendStringSync(String.valueOf(chars[i]));
            break;
    }
    try {
        Thread.currentThread().sleep(0);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

3.3.3 剪切板输出

```
ClipboardManager copy =
(ClipboardManager)getSystemService(Context.CLIPBOARD_SERVICE);
copy.setText(setEncodingFormat(barcode));
// paste
inst.sendCharacterSync(KeyEvent.KEYCODE_PASTE);
```

3.4 其它功能

1. 前后缀，在barcode前后增加字符串
2. 字符串替换，替换barcode中的特定字符串
3. 回车功能

```
import android.view.KeyEvent;
import android.app.Instrumentation;

Instrumentation inst = new Instrumentation();
inst.sendKeyDownUpSync(KeyEvent.KEYCODE_ENTER);
```