# Scan SDK documentation

## 1. ScanManagerService

### 1.1 Get instance of ScanManagerService

#### 1.1.1 Create .aidl file

FileName: IScanManager.aidl

Path: kozen\os\scan\

```
package kozen.os.scan;

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

#### 1.1.2 Get instance of Stub interface

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

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

### 1.2 aidl method explain

| Method name | Parameter | Return | Explain |
| --- | --- | --- | --- |
| openScanner | void | void | Turn on the scan code camera |
| closeScanner | void | void | Turn off the scan code camera |
| startScan | void | void | Get images |
| stopScan | void | void | Stop getting images |
| isScannerOpen | void | boolean | Is the scan code camera on |

### 1.3 Code example

```
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 Get ScanService

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

private static volatile IScan mScanService = null;

mScanService = IScan.getService();
```

## 2.2 ScanService method explain

| Method name | Parameter | Return | Explain |
| --- | --- | --- | --- |
| initContext | String | boolean | Initialize the decoding library and return whether it is successful |
| releaseContext | void | void | Release the decoding library |
| setCallback | IScanCallback | void | Set callback, How to implement IScanCallback see 2.3 |
| setDefaultCallback | IScanCallback | void | Set default callback, Effective when setCallback(null) |
| setFeedbackCallback | IScanCallback | void | Feedback callback, Separation of feedback and results |
| setScanResult | String, String | void | Set scan result |
| onTimeOut | void | void | Execute onTimeOut in IScanCallback |
| getSdkVersion | void | String | Get SDK version |
| startDecode | void | boolean | Start decoding and return whether it is successful |
| stopDecode | void | boolean | Stop decoding and return whether it is successful |
| getDecodeState | void | int | Get the decoding state, return -1 if it is not obtained, return 0 in idle state, return 1 in decoding, return 2 when decoding is successful |
| setTimeout | int | boolean | Set the timeout period, unit: milliseconds, return whether it is successful |
| enableAllSymbol | void | boolean | Turn on all symbologies supported by the decoder and return whether it is successful |
| disableAllSymbol | void | boolean | Turn off all symbologies supported by the decoder and return whether it is successful |
| setSymbolEnable | int, boolean | boolean | According to the code system ID switch decoder code system, boolean parameter is true to open, false to close; int parameter see 2.4 |
| setDecoderTag | int, int | boolean | Set decoder symbology attribute, see 2.7 |
| getDecoderTag | int | int | Get decoder symbology attribute |
| setWorkMode | int | boolean | Set the working mode and return whether it is successful |

| Method name | Parameter | Return | Explain |
| --- | --- | --- | --- |
| getWorkMode | void | int | Get the working mode, working mode value see 2.5 |
| setFlashMode | int | boolean | Set the flash mode, 0 means flashing, 1 means static |
| getFlashMode | void | int | Get the flash mode |
| setViewSize | int | boolean | Set the view size and return whether it is successful |
| getViewSize | void | int | Set the view size, view size value see 2.6 |
| enableFlash | boolean | boolean | Turn on and off the flash, and return whether it is successful |
| enableAim | boolean | boolean | Switch the aiming light (red light) and return whether it is successful |

## 2.3 Implement IScanCallback

### 2.3.1 Create ScanCallback class

### 2.3.2 Import IScanCallback

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

### 2.3.3 Inherit 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 Implement onScanResult method (required)

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

| Parameter | Explain |
| --- | --- |
| barcode | Symbology |
| result | Scan code result |

#### 2.3.5 Implement onTimeOut method (required)

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

#### 2.3.6 ScanCallback class explain

| Method | Explain |
| --- | --- |
| onScanResult(String barcode, String result) | Execute after successful scanning |
| onTimeOut() | Execute after scan code timeout |

#### 2.3.7 Suggestion

When using, you can create another class to inherit ScanCallback and write your own code into the newly created class, so you don't need to implement the onScanResult and onTimeOut methods every time.

## 2.4 setSymbolEnable parameter

int parameter is symbology ID, corresponding to the following:

```
// 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 Working mode value

| value | Explain |
| --- | --- |
| 0 | Single scan |
| 1 | Scan code continuously without filtering duplicate results |
| 2 | Scan code continuously with filtering duplicate results |

## 2.6 View size value

| value | View size |
| --- | --- |
| 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 use

The tag is equivalent to the ID of the symbology attribute, which is fixed, and the ID of each attribute is different; and the value is the value of the symbology attribute, except for "minimum length" and "maximum length", which are both in For numbers within the length supported by the symbology, most of the other symbology attributes only have two values, corresponding to the function on and off

```
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 Take GS1 DATABAR as an example

GS1 DATABAR code system has four attributes, as follows:

| Parameter | Explain |
| --- | --- |
| TAG_RSS_LIMITED_ENABLED | Whether to support the limited type (the value is 1 or 0, the length of the limited type is fixed) |
| TAG_RSS_EXPANDED_ENABLED | Whether to support the expanded type (the value is 1 or 0) |
| TAG_RSS_EXPANDED_MIN_LENGTH | Extended minimum length limit (minimum value is 1) |
| TAG_RSS_EXPANDED_MAX_LENGTH | Extended maximum length limit (maximum value is 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);
```

In this example, the values of TAG_RSS_LIMITED_ENABLED and TAG_RSS_EXPANDED_ENABLED are both 1 or 0, but not all function switches are 1 or 0

# 3. Scan code and result processing

## 3.1 Initialization

#### 3.1.1 Scan code camera

Open scan code camera

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

Close scan code camera

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

#### 3.1.2 Decoder

Call initContext(String name) to initialize the decoding library

```
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);
    }
}
```

Initialize settings

1. Symbology switch
2. Working mode
3. Flash switch
4. Aiming light switch
5. Symbology attribute setting

#### 3.1.3 Set callback

1. setCallback
2. setDefaultCallback
3. setFeedbackCallback

Example:

```
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");
}
```

The three setting callback functions achieve the same effect, among which setCallback will override setDefaultCallback, and setFeedbackCallback will be separated and dedicated to the scanning result prompt

## 3.2 Start decoding, stop decoding

### 3.2.1 Start decoding

1. Determine whether the scan code camera is turned on

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

2. Call startScan() to get images

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

3. Call srartDocode() to decode

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

### 3.2.2 Stop decoding

1. Determine if it is decoding

```
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. Stop decoding

```
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 Scan result output

### 3.3.1 Broadcast output

```
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 Keyboard output

```
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 Clipboard output

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

## 3.4 Other functions

1. Prefix and postfix, add string before and after barcode
2. String replacement, replace a specific string in the barcode
3. Carriage return function, code show as below:

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

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