CloudPOS Scanning Service Usage Document

Version=3.9.5

Catalog

1. Purpose and Audience

This document provides comprehensive instructions for using the Scan Service, encompassing interface and parameter descriptions, along with methods for invoking the service. It is primarily intended for developers utilizing the Scan Service in their projects.

2. Project Background

2.1. Advantages of Using the Custom Scan Service

The smart POS devices operate on a customized Android system, enhanced to meet specific requirements. Unlike standard Android systems, they don't inherently include barcode or 2D barcode scanning capabilities. These functionalities are often integrated using open-source services like Zxing/Zbar. Many applications on smart POS devices have successfully implemented rapid scanning features. However, numerous applications developed specifically for smart POS are not off-the-shelf commercial solutions. A significant portion of smart POS developers comes from a POS industry background rather than professional Android development. Consequently, when embarking on application development, they prefer access to a user-friendly scan API over the need to delve into the intricacies of Zxing/Zbar.

From a hardware perspective, the scanning components in smart POS devices often diverge from standard cameras, necessitating modifications. In certain scenarios, these devices require specialized scanning hardware. Direct application of Zxing/Zbar is generally unsuitable for smart POS systems without specific adaptations and customizations. Therefore, the development of the Scan Services aims to simplify the integration of scanning functionalities for third-party developers, enabling them to incorporate these features into their applications more efficiently and effectively.

2.2. Scan Service Usage Overview

The Scan Service operates as an application and is initiated using Android Interface Definition Language (AIDL). This setup allows third-party applications to customize their user interface (UI) by transmitting specific parameters to the service. This design choice ensures flexibility and adaptability for developers, enabling them to seamlessly integrate the scanning functionality into their unique application environments while tailoring the UI to their specific needs and preferences. The use of AIDL facilitates efficient communication between the Scan Service and third-party apps, ensuring a smooth and responsive user experience.

3. Guide to Scanning Interfaces and Parameters

3.1. Interface Description

3.1.1. Sync Scanning (scanBarcode):

This is a synchronous call interface. When invoked by an application, the Scan Service activates the camera as specified by the scan parameters and initiates the scan. Post-scan, the camera is deactivated, and the results are returned immediately.

Method `ScanResult scanBarcode(ScanParameter parameter);`

Parameters: `ScanParameter`

Return Type: `ScanResult`

3.1.2. Continuous Scanning (startScan):

An asynchronous call interface for initiating continuous scanning. When this interface is called, the Scan Service opens the camera as defined by the scan parameters and starts scanning. After each scan, results are returned via callback. Upon completion of each callback, the next scanning process begins.

Method : `void startScan(ScanParameter parameter, IScanCallBack callBack);`

Parameters: `ScanParameter, IScanCallBack`

Return Type: `void`

3.1.3. Scan Result Callback Interface (FoundBarcode in IScanCallBack):

Essential for the `startScan()` method. This interface must be implemented to receive the `ScanResult`. When this interface is called, the Scan Service enters a paused state. After the call returns, the next scanning process resumes. The paused scanning service can be terminated with `stopScan`.

Method : `void foundBarcode(ScanResult result);`

Parameters: `ScanResult`

Return Type: `void`

3.1.4. Stop Continuous Scanning (stopScan)

Stops the continuous scan and turns off the Scan Service's UI. Post-stop, other interfaces like `startScan` or `scanBarcode` can be invoked.
Method:`boolean stopScan();`

Return Type: `Boolean, indicating success (true) or failure (false).`

3.1.5. getScanType(int index)

Retrieves the scanner type.
Method: `String getScanType(int index);`

Parameters: `int index(0 or 1)`

Return Type: `String representing the scanner type: "Scanner", "Camera", or "Error".`

3.2. Parameter Introduction (ScanParameter)

`ScanParameter` is configured through the method `set(String key, String value)`, where keys are case-insensitive and parameters are stored as key-value pairs.

Key	Value Type	Value Range/Defaulte	Description
window_top	int	Default: 0 Range: >0	Distance from the top of the screen to the upper-left corner of the window, effective only in floating window mode. Unit: dp
window_left	int	Default: 0 Range: >0	Distance from the left side of the screen, effective only in floating window mode. Unit: dp
window_width	int	Default: Screen width	Width of the scanning window, effective only

		Range: >0	in floating window mode. Unit: dp
window_height	int	Default: Screen height Range: >0	Height of the scanning window, effective only in floating window mode. Unit: dp
enable_scan_section	boolean	Default: true Range: True/false	If false, the entire display window is the scanning area, removing the scanning frame. If true, a custom scanning area is set, not requiring the entire image scan. The scanning frame will be centered and can be adjusted in width and height.
scan_section_width	int	Default: 300dp Range: >0	Width of the scanning frame.
scan_section_height	int	Default: 300dp Range: >0	Height of the scanning frame.
display_scan_line	String	Default: moving Range: No/fixed/moving	Display style of the red line indicator within the scanning area. Options are no display, fixed in the middle, or moving up and down.
enable_flash_icon	boolean	Default: true on W1, false on Q1 Range: True/false	Whether to show the floating button to toggle the flashlight.
enable_switch_icon	boolean	Default: true Range: True/false	Whether to show the button to switch cameras.
enable_indicator_light	boolean	Default: false Range: True/false	Whether to show the scanning indicator light, supported only on Q1 devices.
decodeformat	String	Default:	Decoding range.

		BARCODE_ALL Range: See appendix for barcode formats	Default is BARCODE_ALL. Multiple decoding ranges can be separated with a comma.
decoder_mode	int	Default: 2 Range: 0/1/2	Scanning mode. 0 for mode1, 1 for mode2, 2 for mode3.
enable_return_image	boolean	Default: false Range: True/false	Whether to return the recognized Bitmap image.
camera_index	int	Default: 0 Range: 0/1/2	0 for the main scanning head , 1 for the secondary camera , 2 for the customer display camera.
scan_time_out	long	Default: -1 (unit milliseconds) Range: >0	If ≤0, scanning is continuous. If >0, scanning occurs only within this timeframe, returning a timeout error if exceeded. Only effective for synchronous interfaces.
scan_section_border_color	int	Default: Color.WHITE	Border color of the scanning frame. Color is passed as an int value, can use Color.argb method.
scan_section_corner_color	int	Default: Color.argb(0xFF, 0x21, 0xDB, 0xD5)	Corner color of the scanning frame.
scan_section_line_color	int	Default: Color.RED	Color of the scanning line.
scan_tip_text	String	Default: “Align the image with the scanning frame for automatic scanning”	Tip text displayed below the scanning frame.
scan_tip_textSize	int	Default: 15	Unit: sp Description: Font size

			of the tip text below the scanning frame.
scan_tip_textColor	int	Default: Color.WHITE	Color of the tip text below the scanning frame.
scan_tip_textMargin	int	Default: 30	Unit: dp Description: Margin from the bottom of the scanning frame to the tip text.
flash_light_state	boolean	Default: false	Initializes the state of the flashlight switch. True for on, false for off.
indicator_light_state	boolean	Default: false	Initializes the state of the indicator light switch. True for on, false for off.
scan_mode	String	Default: dialog	Scanning frame mode. 'dialog' for a dialog-themed activity, 'overlay' for a floating window.
scan_camera_exposure	int	Default: 0	Camera exposure compensation value. Useful when scanning QR codes from mobile screens that are too bright. The specific value depends on the camera used and should be obtained through the Camera API. Effective only for zoom cameras, not for fixed-focus cameras.
scan_time_limit	int	Default: 50	Maximum decoding time. Increasing this value can increase the success rate but decrease scanning speed. 50 is optimal

			and modifications are not recommended.
enable_mirror_scan	boolean	Default: false	Whether to support mirror scanning. Default is off.
enable_hands_free	boolean	Default: true	HandsFree mode enables motion detection and motion lighting. This mode is used in continuous scanning scenarios. True to enable HandsFree mode, false to disable it. Note: This feature is available only for Zebra scanners.
enable_ui_by_zebra	boolean	Default: true	True to enable default scanning UI. False to hide it, which can speed up the scanning initiation. Note: This feature is available only for Zebra scanners.
enable_mobile_phone_screen_mode	boolean	Default: false	Mobile phone screen scanning mode, which can improve the scanning recognition rate. Enabling this mode increases the decoding time per scan. Note: This feature is available only for Zebra scanners.
enable_upca_country	booleann	Default: true	True to pass the country code after UPC_A decoding. False to hide the leading country code. Note: This feature is available only for Zebra scanners.

enable_decoding_illumination	boolean	Default: true	True to turn on the illumination light. False to turn it off. Note: This feature is available only for Zebra scanners.
enable_motion_illumination	boolean	Default: false	True for constant illumination light, false to automatically turn off the light when no object movement is detected. Note: This feature is available only for Zebra scanners in handsfree mode.

3.3. Scan Result (ScanResult) Description

Key	Type	Description
resultCode	Int	Indicates the success or failure of the scan. A value greater than or equal to 0 signifies success; a value less than 0 indicates failure. Refer to the error code definitions for more details.
text	String	The string result of the scan. This is returned only when the scan is successful. The text is in UTF-8 format; if another format is needed, convert the **rawBuffer** accordingly.
rawBuffer	Byte[]	The raw byte stream result of the scan.
bitmap	Bitmap	The image captured during a successful scan. This field contains a value only if the parameter **enable_return_image** is

		set to true.
barcodeFormat	String	The format of the barcode scanned. See the appendix table for barcode format details.

3.4. Scanner Mode: Functionality and Application

This document outlines two distinct modes for integrating the scanner UI in Android applications for smart POS systems: Dialog Mode and Floating Window (Overlay) Mode.

- Dialog Mode

In Dialog Mode, the camera scanner service fully manages the scanner UI.
Developers integrating this mode don't need to design or manage the UI; it's automatically handled.

- Floating Window(Overlay) Mode

In Floating Window Mode, users can customize the UI outside the scanning frame. This allows simultaneous operation of their own UI and the scanning frame. Therefore, customers can define their own buttons for switching the camera, toggling the flashlight, and turning the indicator light on or off, as well as designing elements like the title bar.

Control of switching the camera, toggling the flashlight, and the indicator light can be achieved either through buttons on the scanning frame itself or by sending broadcasts.

Switching the Camera:

Broadcast Action: `com.cloudpos.scanner.setcamera`
Broadcast Value Key: overlay_config
Values: 0 for the main scanning head, 1 for the secondary camera , 2 for the customer display camera.

Toggling the Flashlight:

Broadcast Action: `com.cloudpos.scanner.setflashlight`
Broadcast Value Key: overlay_config
Values: true to turn on, false to turn off.

Toggling the Indicator Light:

Broadcast Action: `com.cloudpos.scanner.setindicator`
Broadcast Value Key: overlay_config
Values: true to turn on, false to turn off.

The above values can be directly copied, but be mindful of case sensitivity or use constants from `ScanParameter`.

Example for turning on the flashlight:

```
Intent intent = new Intent();
intent.setAction(ScanParameter.BROADCAST_SET_FLASHLIGHT);
intent.putExtra(ScanParameter.BROADCAST_VALUE, true);
sendBroadcast(intent);
```

It's important to note that in this mode, the scanning window cannot capture the back and home keys. Developers need to override the `onKeyDown` method to capture these key operations and call the `stopScan` method. Failing to do so may result in scenarios where the user's page disappears, but the scanning window remains active.

3.5. Zebra Camera Scanning

To use the Zebra camera for scanning, the following conditions must be met:

Existence of Zebra Camera: The terminal device must be equipped with a Zebra camera.
ScanParameter Setting: The scanning head needs to be set as the main scanning head (0) in the scan parameters.
Screen Off Limitation: When the screen is off, the Zebra camera will not be functional.

Manually hide UI: When using the Zebra camera for scanning, it's necessary to manually set parameters to hide the UI. This can be achieved by setting the `enable_ui_by_zebra` parameter to false. This setting will effectively hide the UI during Zebra camera scanning operations.

3.6. Error codes

Value	Description
1	Success:Indicates that the scanning operation was successful.

0	User Cancelled: The user has cancelled the scanning operation.
2	Scan Window Fully Visible Notification: Notification that the scan window is fully visible on the screen.
-1	Scan Service Occupied: The scanning service is currently in use or occupied by another process.
-2	Scanner/Camera Cannot Be Opened: The scanning head or camera could not be opened, possibly due to hardware issues or conflicts.
-3	Scanning Timeout: The scanning process exceeded the allotted time limit without successfully reading a barcode.
-4	Parameter Error: There is an error in the parameters provided for the scanning operation, which could include incorrect values or formats.

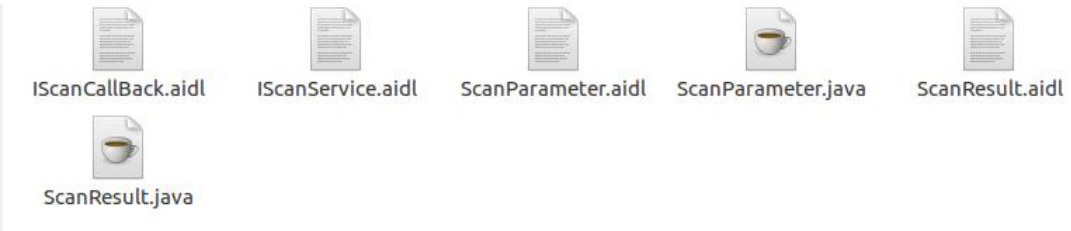
4. Usage Instructions

4.1. Integration of Scanning Service: Step-by-Step Process

The scanning service described in this document is invoked through the Android Interface Definition Language (AIDL). This requires the inclusion of necessary AIDL files in the project.

To successfully integrate the scanning service, you will need to include several specific files. These files are essential for the proper functioning of the scanning service and ensure seamless communication between the application and the service. The detailed steps for including these files will be outlined in the subsequent sections, tailored for Android Studio environments.

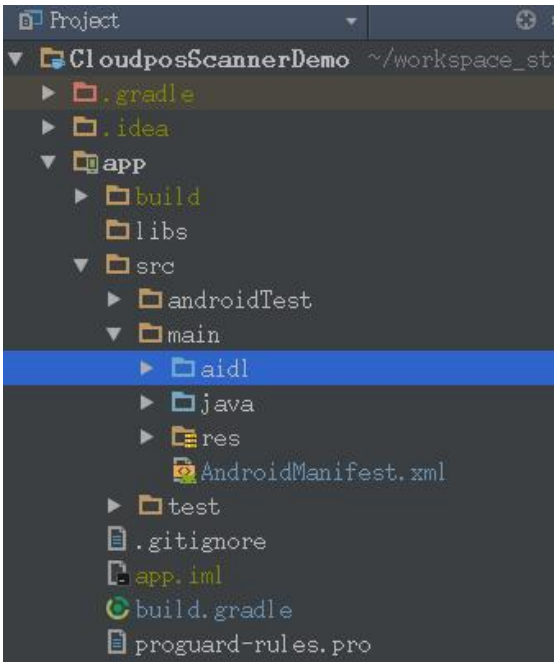
To integrate the scanning service, the following files are required:



Separate AIDL and Java Files: The AIDL files and Java files need to be placed in separate folders within your project structure. This separation is crucial for maintaining a clear project organization and ensuring that the build system correctly processes these files.

- Creating an AIDL Folder in Your Project:

1. Navigate to the "src--main" directory in your project.
2. Right-click and create a new folder within this directory.
   Name this new folder "aidl".



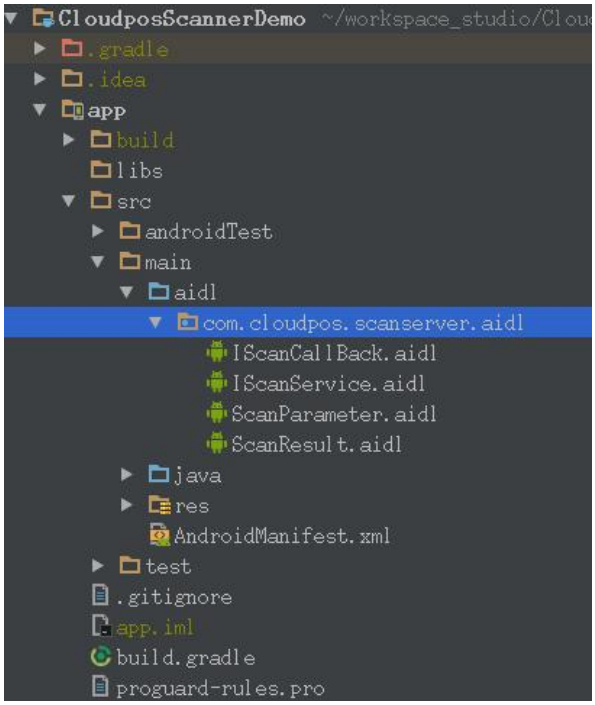
- To create a new package named `com.cloudpos.scanserver.aidl` in the aidl folder and place four AIDL files from the scanning SDK in it, follow these steps:

1. Locate the aidl Folder: In the 'Project' view, find the aidl folder. It's usually located in the src/main/ directory of your app module.

2. Create a New Package: Right-click on the aidl folder. Choose 'New' > 'Package'. Enter `com.cloudpos.scanserver.aidl` as the package name and confirm by clicking OK.

3. Add AIDL Files:

   1) Copy these files.
   2) Navigate to the newly created `com.cloudpos.scanserver.aidl` package.
   3) Paste the AIDL files into this package.

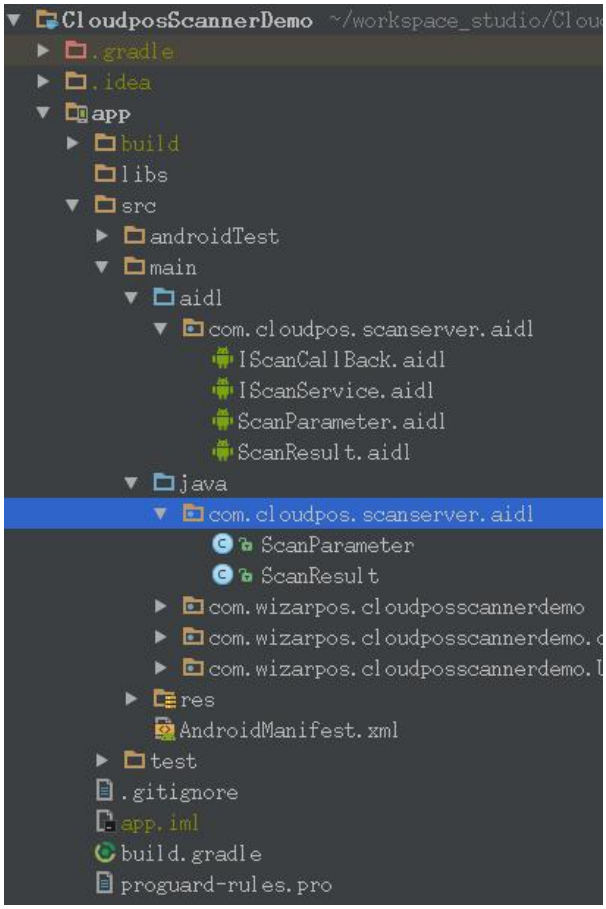


- To create a new package named `com.cloudpos.scanserver.aidl` under the src/main/java directory and add two Java files from the scanning SDK to this package in your Android project, follow these steps:

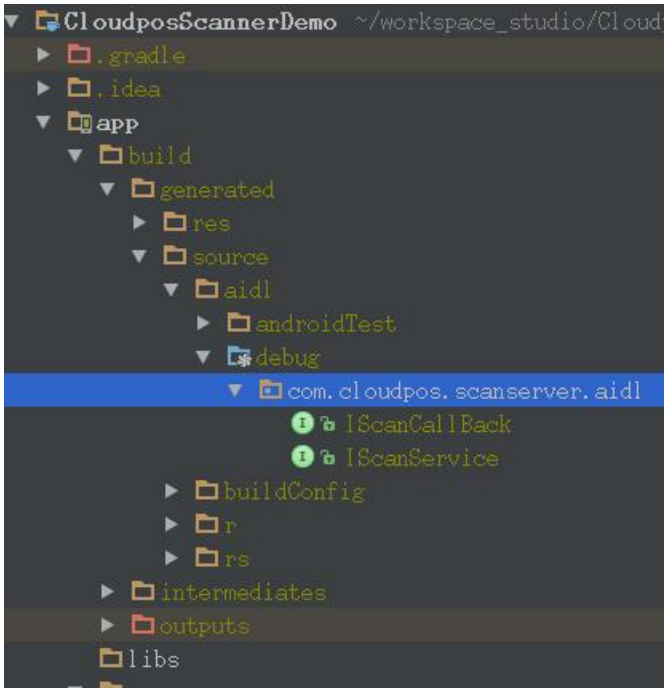
1. Locate the src/main/java Directory: In the 'Project' view of Android Studio, navigate to the src/main/java directory. This is the standard location for Java source files in an Android project.

2. Create a New Package: Right-click on the java directory. Choose 'New' > 'Package'. Enter `com.cloudpos.scanserver.aidl` as the package name and click OK.

3. Add Java Files from the Scanning SDK:
   1) Copy these files from their current location.
   2) Navigate to the newly created `com.cloudpos.scanserver.aidl` package in your project.
   3) Right-click on the package and select 'Paste' to add the files.



- Sync and Build Your Project: After adding the AIDL files and Java files, use the 'Sync Project with Gradle Files' option in Android Studio to sync your project. Then, build your project to compile the new files and ensure there are no errors. If you successfully compile files in the build/generated/source/aidl/debug directory of your Android project, it indicates that the AIDL (Android Interface Definition Language) files have been properly processed, and the corresponding Java interfaces have been generated. This means you are now ready to use these interfaces to communicate with the service defined by the AIDL files.



4.2 Bind Service：Establishing Connection with Scan Service

1. Place the Interface and Implementation:

The interface `IAIDLListener` and its implementation should be placed in any package within your project. Make sure they are properly imported wherever used.

Obtain the interface and its implementation from the \source\aidlControl directory in the barcode SDK package.

2. Bind to the Scanner Service:

Use the `AidlController.getInstance().startScanService(this, this)`; method to bind to the scanner service.

The first this in the method call refers to the Context, typically your Activity or Application, and the second this refers to the IAIDLListener implementation.

3. Implement the `IAIDLListener` Interface:

Implement the `IAIDLListener` interface in your Activity or other component from where you wish to use the scanner service.

In the `serviceConnected` method, check if the objService is an instance of IScannService. If so, assign it to a local variable (scanService) and keep a reference to the `ServiceConnection` (scanConn).

```
private IScannService scanService;
    private ServiceConnection scanConn;

    @Override
    public void serviceConnected(Object objService,
ServiceConnection connection) {
        if(objService instanceof IScannService){
            scanService = (IScannService) objService;
            scanConn = connection;
        }
    }
}
```

4. Use the Scanner Service:

Once the service is connected and you have a reference to IScannService, you can use this reference to call functions provided by the scanner service.

5. Unbind from the Service:

To properly clean up, unbind from the service when it's no longer needed or when your component is being destroyed.

Ensure that scanService is not null before calling unbindService(scanConn) and then set scanService and scanConn to null to avoid memory leaks.

```
@Override
    protected void onDestroy() {
        if(scanService != null){
            this.unbindService(scanConn);
            scanService = null;
            scanConn = null;
        }
        super.onDestroy();
    }
```

6. Refer to the Demo Project:

For detailed implementation and usage, refer to the demo project provided with the barcode SDK package. This will likely have examples of how to use the scanner service effectively.

Remember to handle any exceptions appropriately and ensure that all resources

are released when no longer needed. Also, ensure that your application has the necessary permissions to use the scanner service, especially if it involves hardware access.

5. **Appendix**: Barcode Formats

Usage Method: To specify the barcode formats for scanning, use the ScanParameter class. Set the desired formats using `ScanParameter.KEY_DECODEFORMAT`. Example:

```
ScanParameter parameter = new ScanParameter();
parameter.set(ScanParameter.KEY_DECODEFORMAT, "AZTEC, QR");
```

Compound Barcode Formats(In addition to specifying individual barcode formats, you can use the following special strings to encompass multiple formats)	
BARCODE_ALL	Includes all listed barcode formats.
BARCODE_1D	Includes all 1D barcode formats listed below.
BARCODE_2D	Includes all 2D barcode formats listed below.
Barcode Formats	
AZTEC	2D barcode
DATAMATRIX	2D barcode
QR	2D barcode
MAXICODE	2D barcode
PDF417	2D barcode
CODABAR	1D barcode
CODE39	1D barcode
CODE93	1D barcode
CODE128	1D barcode
EAN8	1D barcode
EAN13	1D barcode
ITF	1Dbarcode(Interleaved Two of Five)
RSS_14	1D barcode
RSS_EXPANDED	1D barcode
UPCA	1D barcode
UPCE	1D barcode
CODE11	1D barcode