Girgit

Programmers Guide

verifone

www.verifone.com

Girgit Programmers Guide
© 2025 Verifone, Inc.

The information contained in this document is subject to change without notice. Although Verifone has attempted to ensure the accuracy of the contents of this document, this document may include errors or omissions. The examples and sample programs are for illustration only and may not be suited for your purpose. You should verify the applicability of any example or sample program before placing the software into productive use. This document, including without limitation the examples and software programs, is supplied “As-Is.”

Comments? Please e-mail all comments on this document to your local Verifone Support Team.

Content

Preface

Introduction

This document provides a detailed and comprehensive information on packages and APIs used for integration of Girgit Service Application on Neo devices.

The document will be revised and updated whenever new functionality is developed in a new version of the application.

Audience

This guide is aimed at Integrators and POS Developers team who writes third party application on Neo devices.

Definitions and Abbreviations

The following terms are used in this document:

Abbreviation	Definition
AES	Advanced Encryption Standard
AIDL	Android Interface Definition Language
AID	Application Identifier
APDU	Application Protocol Data Unit
APN	Access Point Name
API	Application Programming Interface
APID	Application Priority Indicator
APNs	Access Point Names
ARQC	Authorization Request Cryptogram

ATR	Answer to Reset
CBC	Cipher Block Chaining
CTLS	Contactless
CT	Country Code Index
CVM	Cardholder Verification Method
DES	Data Encryption Standard
ECB	Electronic Codebook
EMV	Europay, MasterCard, and Visa
EvDo	Evolution-Data Optimized
ICCID	Integrated Circuit Card Identifier
IMSI	International Mobile Subscriber Identity
initVec	initialization Vector
IPC	Inter-Process Communication
MMS	Multimedia Messaging Service
OpenSDI	Open Smartcard Development Interface
PAN	Primary Account Number
PCI	Payment Card Industry
POS	Point of Sale
TC	Transaction Certificate
TDK	Track Data Key
TEK	Transaction Encryption Key
TLV	Tag-Length-Value

TTF	True Type Font
QR code	Quick Response code
SDI	Secure Device Interface
SDK	Software Development Kit
SIM	Subscriber Identity Module
UP	Universal Payment

Related Documentation

To learn more about Girgit and the migration to Neo devices, refer to the following set of documents:

- Girgit Migration Guide
- Android 13 Migration for the older applications (https://developer.android.com/about/versions/13/behavior-changes-13)

Revision History

Date	Document Version Number	Girgit Compatibility Version	Description
17-02-2025	1.0.0	1.0.0.4	First Release

1. Solution Overview

Girgit is a Middleware Software Development Kit (SDK) developed to enable the migration of applications from Verifone X990 devices to Verifone Neo devices. By abstracting Secure Device Interface (SDI) API calls and substituting them with VFI service API calls (X990-SDK), Girgit ensures a smooth transition with minimal disruption.

The word "*Girgit*" is derived from Hindi, which means "*chameleon*" in English. Chameleons are well-known for their ability to change their skin color to blend into different environments. This ability to adapt and change subtly, while fundamentally remaining the same organism, mirrors this solution’s goal of adapting applications to new environments with only slight modifications. In a broader cultural and linguistic context, using the name Girgit symbolizes the idea of transformation and seamless adaptation.

Girgit uses a Replatform and Device Update migration strategy, with focus on seamless application compatibility and enhanced performance on Neo devices. The transition process is structured into four distinct phases:

1. Updating application interface and configurations
2. Updating Android Interface Definition Language (AIDL) components
3. Upgrading Android OS version
4. Implementing package and API changes for structural compatibility

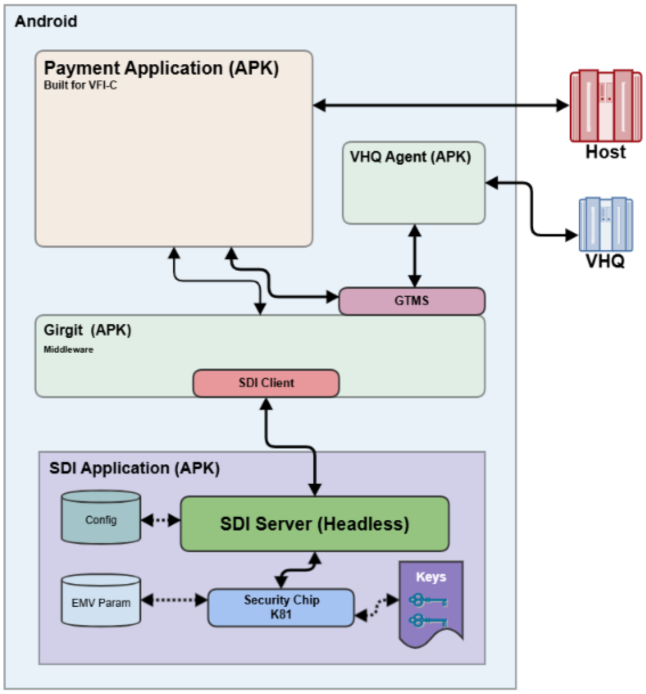


Figure 1: High-level architecture diagram

1.1 Installation of Girgit Service Application

1.1.1 Prerequisites

Ensure that the below applications are installed for successful migration:

1) Girgit
2) GirgitSystemService

These applications contain the necessary frameworks and additional AIDL interfaces (included under AIDL.zip file) to support the migration.

Both applications are integral components of the default Girgit package that contains two key services for maintaining functionality and performance:

- Girgit Device Service: This service is responsible for managing all Europay, MasterCard, and Visa (EMV) transactions, ensuring secure and efficient payment processing.
  - **Package name: com.vfi.smartpos.deviceservice.**

- Girgit System Service: This service handles all system configurations, maintaining optimal device operations and settings during and after migration.
    - **Package name: com.vfi.smartpos.system_service**

NOTE

Girgit is compatible with Android OS version 13 and above but does not support any earlier versions.

1.1.2 Installation Process- Side Loading

The installation of Girgit packages is done via side loading. Android Side Loader enables the direct loading and updating of Verifone Android devices from a USB stick that is connected to a host PC.

1.1.2.1 Supported Side Loadable Packages

APK File

- The android applications can be installed in the APK format. Make sure that the APK file is Verifone signed, or sponsor signed.

ZIP File

- A single ZIP, flash.zip is provided to perform a bulk installation of multiple components. Note that the security checking is done on the individual files within the flash.zip, and not on the zip file itself. There is no manifest used in the flash.zip. All files are expected to be at the top-level directory of the flash.zip. The type of installation method is determined by the file extension.

Order	Filename/Extension	File Description	Comment
1	sponsor.tgz	Initial sponsor package	Install this file first if the device has not been signed by a sponsor, so that any other packages can be installed after the device has been signed by a sponsor.

			NOTE • Verifone will issue a customer-specific signing certificate to the sponsor. • Only one sponsor will be allowed per device. • Once a device is bound to a sponsor, only those apps that are signed by that specific sponsor can be installed. • The sponsor.tgz is provided by Verifone.
2	...tgz, ...tar, ...tar.gz	Secure components package	This package is used to update the secure components for products that do not run on Engage devices. There can be maximum of one package of this type per flash.zip file. The package holds the following items: • SDI configuration files: (whitelist.json, sensitive_tags.json, card_ranges.json) • SDI security configuration (sccfg.json) • SDI static EMV configuration • VCL settings file NOTE This package is for non-engage secure processors only. This package can also be used to sponsor-sign a new terminal, as the package includes a customer-specific signing certificate to the sponsor Customers can create the specific signing certificate from the Verifone Development portal. Nevertheless, this feature is being developed, and you should contact the Verifone Technical support team to obtain the signing certificate.
3	...apk	Android package	There can be multiple .apk files in a flash.zip file.

			The Verifone Android platform will only support APKv2 signing scheme. Customers can use the Android default signing for Application Development units (APP-DEV). For Production units (PROD), customer APK signing is done on the VeriShield File Signing (VFS) Service at https://sign.verifone.com.
4	...zip	CP Package	This package requires a manifest file (.mft) that indicates whether applications are to be installed or uninstalled. Installed applications are expected to be within this zip file and in the apk format. There can be maximum of one package of this type per flash.zip file. Android Commerce Platform packages are APK files. Engage-CP.zip • cpapp1.apk • cpapp2.apk • cp-pkg.mft Engage Commerce Platform packages are ZIP files. Engage-CP.zip • cpdemo1-inst.zip • cpdemo2-inst.zip • cp-pkg.mft Manifest file must have application->type set to “CP_ANDROID” or “CP_ENGAGE” to identify the platform. For Android application->purpose is indicating an action to be taken: “CP_INSTALL” or “CP_UNINSTALL”.
5	...json	VRKv2	There can be multiple files in a flash.zip file. VRK key loading packages are encrypted with a VRKv2 warrantied key that is pre-installed on the system. Verifone creates these packages upon customer request.

			The VRKv2 key can currently only be loaded on Trinity and Neo terminals.
6	tgz, ...tar, ...tar.gz	Engage Only secure installer package	This package is used to update the secure components on products running on Engage terminals. These packages can also be used to sponsor sign a new terminal. There can be maximum of one package of this type per flash.zip file.
7	...zip	Android OTA	This file is used to update the Android operating system software. There can be maximum of one package of this type per flash.zip file. This file is identified by the contained meta-data created by Android tools. The Android OTA files are system signed and provided by Verifone.
8	Parameters.zip	Android settings	There can be only one parameter file called "parameters.zip" per flash.zip file and it needs to be in top-level of flash.zip.

1.1.2.2 Required Setup and Tools

USB Stick

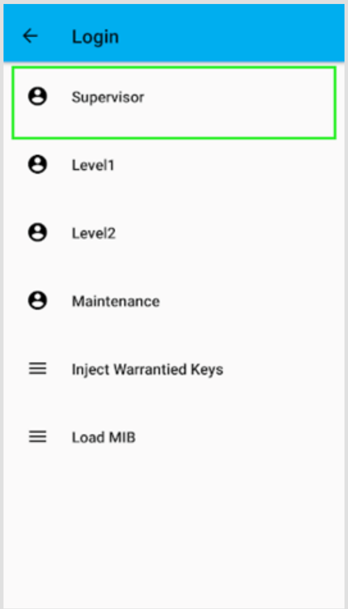
- FAT32 formatted stick with 4GB.

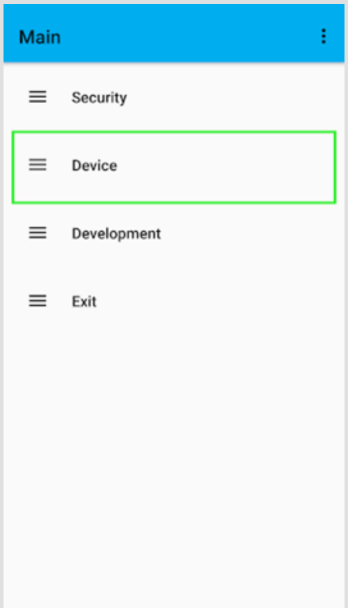
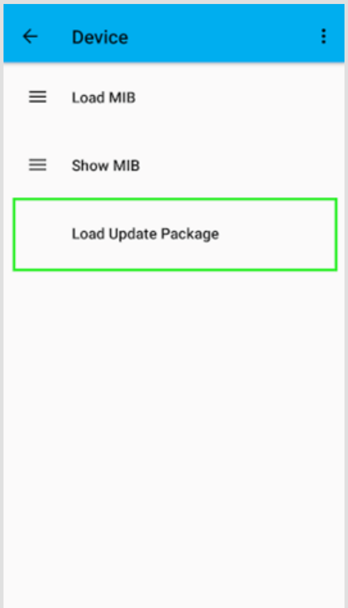
Packaging Tools - Creating loadable ZIP files

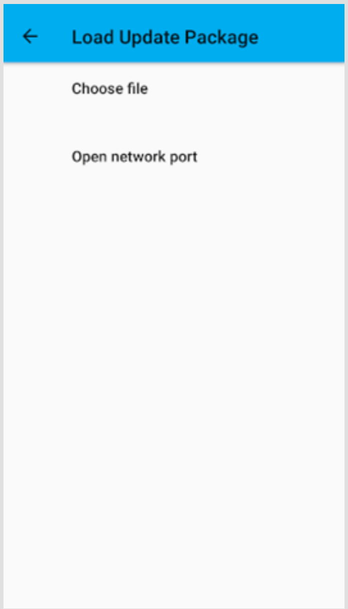
- Any standard tool can be used to create the zip file which will be used as the flash.zip.

1.1.2.3 Installation via USB using Sysmode

At any time through Sysmode, the Android Side Loader app can be launched by going to “Device” -> “Load update package”. This method can also be used at the time of development purpose, troubleshooting the device, or any maintenance purpose.

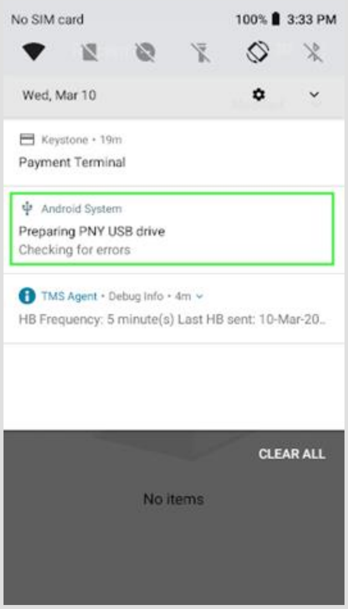
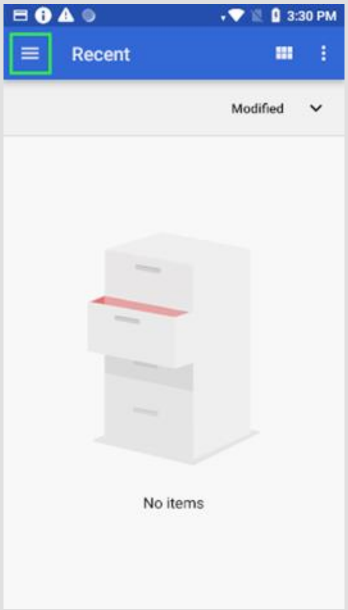
Step	Detail	Screen
1	From Sysmode, select **Supervisor**.	 Login Supervisor Level1 Level2 Maintenance Inject Warrantied Keys Load MIB

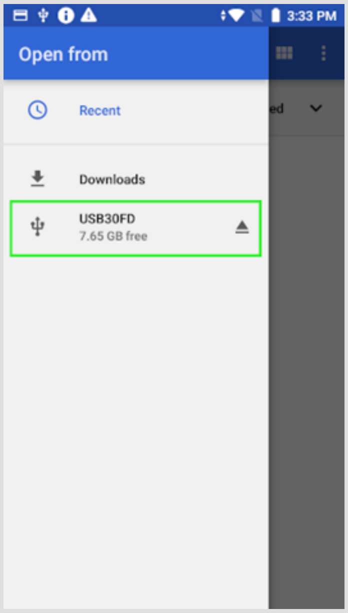
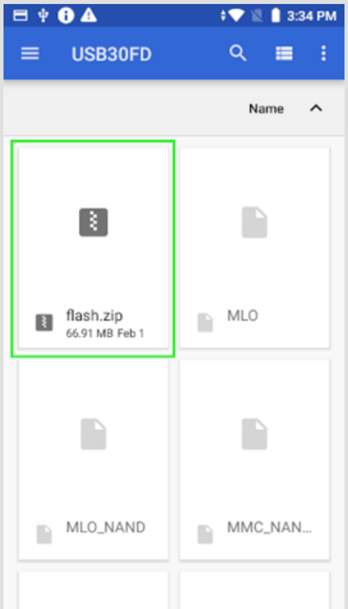
2	From the main screen, select **Device**.	
3	Select **Load Update Package**.	

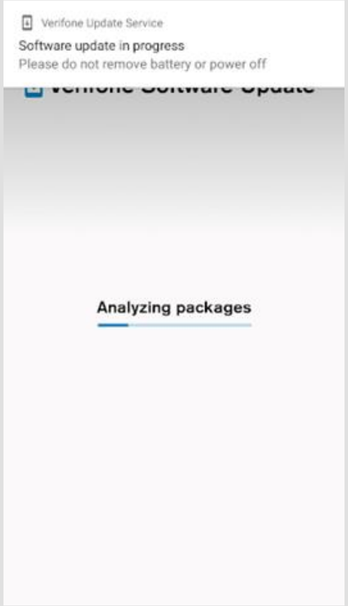
4	The application will present the following options: 1. Choose file 2. Open network port	 Load Update Package Choose file Open network port

Choose File

Upon selecting this option, a file browser will be launched, enabling users to select a file from a USB drive. The system restricts file selection to those with APK and ZIP extensions for installation purposes. Users may insert the USB drive at any time; however, it is essential to allow sufficient time for the Android operating system to recognize and process the device. The USB drive's connection status can be checked via the status bar.

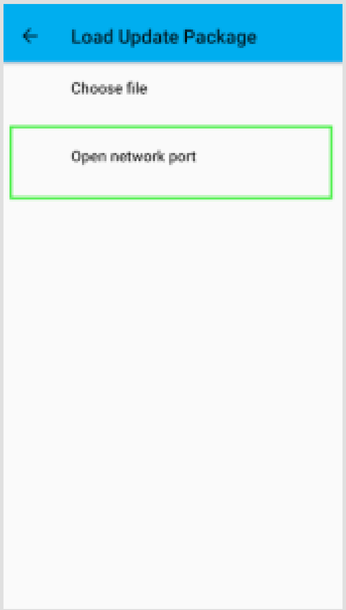
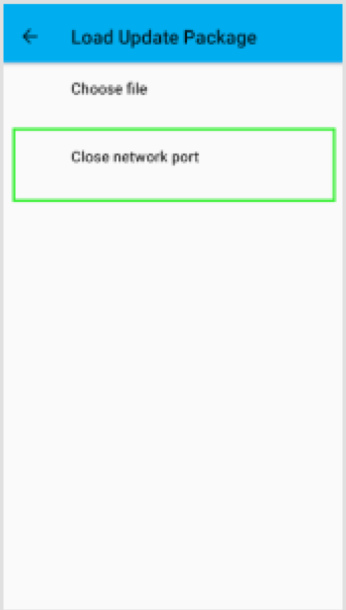
Step	Detail	Screen
1	Check the status of USB connection from the status bar.	 No SIM card 100% 3:33 PM Wed, Mar 10 Keystone • 19m Payment Terminal Android System Preparing PNY USB drive Checking for errors TMS Agent • Debug Info • 4m HB Frequency: 5 minute(s) Last HB sent: 10-Mar-20.. CLEAR ALL No items
2	Once the device is ready, select the file. Note that the file can have any name if it ends in .zip, and the file can be located in any folder as the user can use the UI to navigate and select the file to be used. Select the menu icon to change the browsing location.	 3:30 PM Recent Modified No items

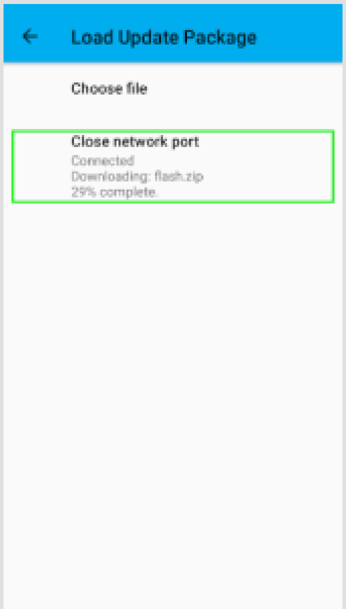
3	Select the USB drive.	 3:33 PM Open from Recent Downloads USB30FD 7.65 GB free
4	Select the ZIP file to be used.	 3:34 PM USB30FD Name flash.zip 66.91 MB Feb 1 MLO MLO_NAND MMC_NAN...

5	Upon selection, the file installation begins.	 Verifone Update Service Software update in progress Please do not remove battery or power off Verifone Software Update Analyzing packages

Open Network Port

Selecting this option will open the netloader port to allow an IP based file download. The port will remain open until closed by the user or the device is power cycled.

Step	Detail	Screen
1	Select **Open network port** before connecting with an external tool.	 Load Update Package Choose file Open network port
2	This will indicate the connection is open. There is no need to stay on this screen to keep the port open. Selecting this again will close the port.	 Load Update Package Choose file Close network port

3	While receiving a file and on this screen, the status of the download will be displayed here.	 Close network port Connected Downloading: flash.zip 29% complete.

1.1.2.4 Connecting to System Service and Device Service

After the package installation, it is necessary to bind both the system service and device service, as shown in the code snippets below.

Binding the System Service:

```
public boolean bindSystemService(Context context) {
        final boolean[] isConnected = {false};
        Executors.newSingleThreadExecutor().execute(() -> {
            /**
             * Create a new connection and keep the handler in global scope.
             * */
            ServiceConnection serviceConnection = new ServiceConnection() {
                @Override
                public void onServiceConnected(ComponentName componentName, IBinder iBinder)
{
                    setHandler(ISystemManager.Stub.asInterface(iBinder));
                    DeviceServiceManager.getInstance().bindPosService(context);
                    Toast.backgroundToast(context, "System onServiceConnected");
                }

                @Override
                public void onServiceDisconnected(ComponentName componentName) {
                    Toast.backgroundToast(context, "onServiceDisconnected");
                    Log.d(TAG, "onServiceDisconnected: Disconnected");
                    MainActivity.disconnected();
                    setHandler(null);
                    context.bindService(new
Intent().setAction(POS_SERVICE_ACTION).setPackage(POS_SERVICE_PACKAGE),
                            this, Context.BIND_AUTO_CREATE);
                }
            };

            boolean res = context.bindService(
                    new Intent()
                            .setAction(POS_SERVICE_ACTION)
                            .setPackage(POS_SERVICE_PACKAGE),
                    serviceConnection,
                    Context.BIND_AUTO_CREATE);
            if (!res) {
                Log.i(TAG, "System Service Failed");
                Toast.backgroundToast(context, "System Service Failed");
                getInstance().isConnected = false;
            } else {
                Log.i(TAG, "System Service Connected");
                Toast.backgroundToast(context, "System Service Connected");
                getInstance().isConnected = true;
            }
        });
        return getInstance().isConnected;
    }
```

Binding the Device Service:

```
public boolean bindPosService(Context context) {
        Executors.newSingleThreadExecutor().execute(() -> {
            /**
             * Create a new connection and keep the handler in global scope.
             * */
            ServiceConnection serviceConnection = new ServiceConnection() {
                @Override
                public void onServiceConnected(ComponentName componentName, IBinder iBinder)
{
                    setHandler(IDeviceService.Stub.asInterface(iBinder));
                    Toast.backgroundToast(context, "device onServiceConnected");

                }

                @Override
                public void onServiceDisconnected(ComponentName componentName) {
                    Toast.backgroundToast(context, "onServiceDisconnected");
                    Log.d(TAG, "onServiceDisconnected: Disconnected");
                    MainActivity.disconnected();
                    setHandler(null);
                    context.bindService(new
Intent().setAction(POS_SERVICE_ACTION).setPackage(POS_SERVICE_PACKAGE),
                            this, Context.BIND_AUTO_CREATE);
                }
            };

            boolean res = context.bindService(
                    new Intent()
                            .setAction(POS_SERVICE_ACTION)
                            .setPackage(POS_SERVICE_PACKAGE),
                    serviceConnection,
                    Context.BIND_AUTO_CREATE);
            if (!res) {
                Log.i(TAG, "Device Service Failed");
                Toast.backgroundToast(context, "Device Service FAILED");
                getInstance().isConnected = false;
            } else {
                Log.i(TAG, "Device Service Connected");
                Toast.backgroundToast(context, "Device Service Connected");
                getInstance().isConnected = true;
                Intent intent = new Intent(context, SubMainActivity.class);

                // Start the SubMenu activity
                context.startActivity(intent);
            }

        });
        return getInstance().isConnected;
    }
```

1.1.3 Supporting Files to be Installed

Apart from the default package, install the following files to ensure secure and proper handling of sensitive data:

1. **sccfg.json**: The sccfg.json file is essential for securely handling sensitive data operations. It contains the configuration for the ADK-Sec security service, which the SDI Server uses to perform secure operations. Proper configuration of this file is crucial for the successful management of all tasks involving sensitive data.
2. **GirgitJ_log.conf** and **GirgitN_log.conf**: These are the configuration files used for logging purposes to retrieve Girgit application logs.
3. **whitelist.json**: The `whitelist.json` file is not included in the default configuration provided with the SDI base and SDI config packages. However, if the user application requires it, it can be installed through a user config package. This file whitelists the set of cards that will block sensitive

payment-related data, such as the Primary Account Number (PAN). By default, the file contains the values [0,1,2,3,4,5,6,7,8,9].

NOTE	The above files should be as .zip files (flash.zip). Refer to Section 1.2.2.3 on how to install these files.

2. AIDL

AIDL is a powerful tool to facilitate inter-process communication (IPC). It allows different Android components, potentially running in separate processes, to interact and share data in a seamless and efficient manner. This is particularly useful for applications that need to perform background tasks or require services from other applications.

Key features of AIDL:

- **Inter-process Communication:** AIDL enables communication between different processes in Android, ensuring that data can be shared, and methods can be invoked across process boundaries.
- **Type Support:** AIDL supports a variety of data types, including primitive types, Strings, CharSequences, List, Map, and Parcelable objects, allowing for flexible and complex data structures to be communicated.
- **Automatic Code Generation:** When defining an AIDL interface, the Android build tools to automatically generate the necessary code to handle the IPC, minimizing the boilerplate code developers need to write.

This chapter describes the methods offered by Girgit service application.

2.1 AIDL: System Service

The system service AIDL interfaces provide a structured and efficient way to manage various system-level operations on Neo devices. These interfaces define a set of methods to perform critical system functions such as installing and uninstalling applications, managing device settings, and accessing network management functionalities.

2.1.1 INetworkManager

Package: com.vfi.smartpos.system_service.aidl.networks.INetworkManager

Overview:

This interface provides a set of methods to manage network-related operations on Neo payment devices. It includes functionalities to configure network types, enable/disable Wi-Fi, and Airplane mode, manage mobile data settings, and handle Access Point Names (APNs).

NOTE

The following methods are not supported at the device level in Android 13. These functionalities should instead be implemented at the application level.

- `isMultiNetwork()`
- `setMultiNetwork(boolean enable)`
- `getMultiNetworkPrefer()`
- `setMultiNetworkPrefer(String prefer)`

Public Member Functions:

Modifier and Type	Method
void	**setNetworkType** (int mode)
int	**getNetworkType** ()
void	**enableWifi** (boolean state)
void	**enableAirplayMode** (boolean state)
int	**setAPN** (in Bundle infos)
void	**enableMobileData** (boolean state)
Bundle	**getSelectedApnInfo** ()
int	**selectMobileDataOnSlot** (int slotIdx)
boolean	**isMultiNetwork** ()

void	**setMultiNetwork** (boolean enable)
String	**getMultiNetworkPrefer** ()
boolean	**setMultiNetworkPrefer** (String prefer)
void	**setEthernetStaticIp** (in Bundle bundle)
void	**setWifiStaticIp** (in Bundle bundle)
void	**setMobilePreferredNetworkType** (String type)
String	**getMobilePreferredNetworkType** ()

Member Function Documentation:

setNetworkType()

This method sets the network type to the specified mode. It is used to configure the network settings of the device, enabling the user to switch between different network types as required.

Prototype

```
void
com.vfi.smartpos.system_service.aidl.networks.INetworkManager.setNetworkType(int
mode)
```

Parameter

`mode` An integer value that indicates the required network mode to be set.

7 = Global	This mode represents a global (or auto) mode where the device can switch between multiple network types depending on availability. This supports both GSM and CDMA networks globally.

6 = EvDo only	This mode selects the EvDo (Evolution-Data Optimized) network type, which is a high-speed data technology used with CDMA networks, specifically in the 3G context.
5 = CDMA without EvDo	This mode forces the device to connect only to a CDMA network without using EvDo for data services. Typically, this refers to basic voice and limited data on older CDMA networks.
4 = CDMA / EvDo auto	This mode allows the device to automatically switch between CDMA for voice services and EvDo for high-speed data. It's a common mode for devices that need to support both voice and 3G data on CDMA networks.
3 = GSM / WCDMA auto	This mode allows the device to automatically switch between GSM (2G) for voice services and WCDMA (3G) for data, depending on availability.
2 = WCDMA only	This mode forces the device to connect to WCDMA networks (3G only), disabling GSM or any other network type.
1 = GSM only	This mode forces the device to use GSM networks only, typically for 2G voice and data services.
0 = GSM / WCDMA preferred	This is a common mode that prefers GSM for voice and WCDMA for data. If neither is available, the device might fall back to another supported network type (e.g., CDMA, if applicable).

Return Values

```
void
```

getNetworkType()

This method retrieves the current network type. It provides the configuration of the current network settings on the device.

Prototype

```
int
com.vfi.smartpos.system_service.aidl.networks.INetworkManager.getNetworkType()
```

Parameter

None.

Return Values

An integer representing the current network type below:

7 = Global	This mode represents a global (or auto) mode where the device can switch between multiple network types depending on availability. This likely supports both GSM and CDMA networks globally.
6 = EvDo only	This mode selects the EvDo (Evolution-Data Optimized) network type, which is a high-speed data technology used with CDMA networks, specifically in the 3G context.
5 = CDMA without EvDo	This mode forces the device to connect only to a CDMA network without using EvDo for data services. Typically, this could refer to basic voice and limited data on older CDMA networks.

4 = CDMA / EvDo auto	This mode allows the device to automatically switch between CDMA for voice services and EvDo for high-speed data. It's a common mode for devices that need to support both voice and 3G data on CDMA networks.
3 = GSM / WCDMA auto	This mode allows the device to automatically switch between GSM (2G) for voice services and WCDMA (3G) for data, depending on availability.
2 = WCDMA only	This mode forces the device to connect to WCDMA networks (3G only), disabling GSM or any other network type.
1 = GSM only	This mode forces the device to use GSM networks only, typically for 2G voice and data services.
0 = GSM / WCDMA preferred	This is a common mode that prefers GSM for voice and WCDMA for data. If neither is available, the device might fall back to another supported network type (e.g., CDMA, if applicable).

enableWifi()

This method enables or disables Wi-Fi based on the provided state.

Prototype

```
void com.vfi.smartpos.system_service.aidl.networks.INetworkManager.enableWifi
(boolean state)
```

Parameter

`state`	A Boolean value that indicates and controls the current Wi-Fi state of the device.

true: Enables Wi-Fi functionality.

false: Disable Wi-Fi functionality.

Return Values

```
void
```

enableAirplayMode()

This method enables or disables Airplane mode based on the provided state.

NOTE	Enabling Airplane mode will deactivate all wireless communications, while disabling it will restore connectivity.

Prototype

```
void com.vfi.smartpos.system_service.aidl.networks.INetworkManager.
enableAirplayMode(boolean state)
```

Parameter

`state`	A Boolean value that indicates and controls the Airplane mode state of the device.

true: Enables Airplane mode.

false: Disables Airplane mode.

Return Values

```
void
```

setAPN()

This method sets the APN settings based on the provided configuration details.

Prototype

```
int com.vfi.smartpos.system_service.aidl.networks.INetworkManager.setAPN(in
Bundle infos)
```

Code Snippet

```
Bundle infos = new Bundle();
     infos.putString("name", "test01");
     infos.putString("apn", "test01");
     infos.putString("authtype", "-1");
     infos.putString("numeric", "46002");
```

Parameter

`infos` A Bundle containing configuration details necessary for APN setup.

Key	Value	Description
name	test01	The name of the APN.
apn	test01	The actual APN string that the device will use to connect to the mobile network.
authtype	-1	Authentication type, usually -1 for none, 0 for PAP, 1 for CHAP, etc.
numeric	46002	The mobile network's numeric identifier (MCC + MNC). In this case, 460 is the Mobile Country Code (MCC) and 02 is the Mobile Network Code (MNC)
mcc	460	MCC - part of the identifier for the network.

mnc	02	MNC - part of the identifier for the network.
proxy	-	Proxy server address, if used (often left empty if no proxy is required).
port	-	The port number for the proxy (if used).
mmsproxy	-	The proxy server for Multimedia Messaging Service (MMS).
mmsport	-	The port number for the MMS proxy.
user	-	Username for authentication, if needed.
server	-	Server address (typically left empty).
password	-	Password for authentication (if needed).
mmsc	-	Multimedia Messaging Service Center (MMSC) URL.
current	1	Indicator if this is the current APN to be used (1 means current).
carrier_enabled	1	Carrier's status (1 for enabled, 0 for disabled).
protocol	IP	The protocol used for IP (e.g., IP, IPv6).
roaming_protocol	IP	The protocol used when roaming (e.g., IP, IPv6).
bearer	0	Bearer type (e.g., 0 for unspecified, 1 for GPRS, etc.).
max_conns	0	Maximum number of connections (0 for unlimited).
max_conns_time	0	Time in seconds before the number of connections is restricted (0 for no limit).

modem_cognitive	-	Modem cognitive settings (may not be used in all cases).
localized_name	-	Localized name for the APN (useful for specific regions).
mvno_match_data	-	Mobile Virtual Network Operator (MVNO) matching data.
mvno_type	-	Type of MVNO (e.g., SPN, IMSI, etc.).
profile_id	0	Profile ID for this APN configuration.
read_only	0	Whether the APN configuration is read-only (0 for no, 1 for yes).
sub_id	1	Subscription ID for the APN.
type		Type of the APN (e.g., default, mms, supl).
SLOT	1	SIM card slot number (1 for the first slot, 2 for the second, etc.).
fixed_numeric	46002	The fixed numeric identifier for the carrier.

Return Values

An integer indicating the result (success or failure) of the APN configuration.

0: The operation was successful.

Any negative integer: The operation failed.

enableMobileData()

This method enables or disables mobile data connectivity based on the provided state. When enabled, mobile data is activated, allowing for data transmission over the mobile network.

Prototype

```
void com.vfi.smartpos.system_service.aidl.networks.INetworkManager.
enableMobileData(boolean state)
```

Parameter

`state`	A Boolean value that indicates and controls the mobile data state on the device.
	true: Enables mobile data.
	false: Disables mobile data.

Return Values

```
void
```

getSelectedApnInfo()

This method retrieves information about the selected APN configuration on the device, such as APN name, type, username, password etc.

Prototype

```
bundle com.vfi.smartpos.system_service.aidl.networks.INetworkManager.
getSelectedApnInfo()
```

Parameter

None.

Return Values

A Bundle containing details of the selected APN. Possible values:

Key	Value	Description
name	test01	The name of the APN.
apn	test01	The actual APN string that the device will use to connect to the mobile network.
authtype	-1	Authentication type, usually -1 for none, 0 for PAP, 1 for CHAP, etc.
numeric	46002	The mobile network's numeric identifier (MCC + MNC). In this case, 460 is the Mobile Country Code (MCC) and 02 is the Mobile Network Code (MNC)
mcc	460	MCC - part of the identifier for the network.
mnc	02	MNC - part of the identifier for the network.
proxy	-	Proxy server address, if used (often left empty if no proxy is required).
port	-	The port number for the proxy (if used).
mmsproxy	-	The proxy server for Multimedia Messaging Service (MMS).
mmsport	-	The port number for the MMS proxy.
user	-	Username for authentication, if needed.
server	-	Server address (typically left empty).
password	-	Password for authentication (if needed).

mmsc	-	Multimedia Messaging Service Center (MMSC) URL.
current	1	Indicator if this is the current APN to be used (1 means current).
carrier_enabled	1	Carrier's status (1 for enabled, 0 for disabled).
protocol	IP	The protocol used for IP (e.g., IP, IPv6).
roaming_protocol	IP	The protocol used when roaming (e.g., IP, IPv6).
bearer	0	Bearer type (e.g., 0 for unspecified, 1 for GPRS, etc.).
max_conns	0	Maximum number of connections (0 for unlimited).
max_conns_time	0	Time in seconds before the number of connections is restricted (0 for no limit).
modem_cognitive	-	Modem cognitive settings (may not be used in all cases).
localized_name	-	Localized name for the APN (useful for specific regions).
mvno_match_data	-	Mobile Virtual Network Operator (MVNO) matching data.
mvno_type	-	Type of MVNO (e.g., SPN, IMSI, etc.).
profile_id	0	Profile ID for this APN configuration.
read_only	0	Whether the APN configuration is read-only (0 for no, 1 for yes).
sub_id	1	Subscription ID for the APN.

type		Type of the APN (e.g., default, mms, supl).
SLOT	1	SIM card slot number (1 for the first slot, 2 for the second, etc.).
fixed_numeric	46002	The fixed numeric identifier for the carrier.

selectMobileDataOnSlot()

This method allows the selection of SIM slot to be used for mobile data connectivity by providing the corresponding slot index (either 1 or 2).

Prototype

```
int com.vfi.smartpos.system_service.aidl.networks.INetworkManager.
selectMobileDataOnSlot (int slotIdx)
```

Parameter

`slotldx` An integer value that indicates the SIM slot for mobile data usage. Acceptable values are 1 or 2.

1: Slot 1.

2: Slot 2.

Return Values

An integer indicating the result (success or failure) of the operation.

0: The operation was successful.

Any negative integer: The operation failed.

isMultiNetwork()

This method is used to check whether multi-network support is enabled on the device.

Prototype

```
boolean
com.vfi.smartpos.system_service.aidl.networks.INetworkManager.isMultiNetwork ()
```

Parameter

None.

Return Values

A Boolean value:

true: Multi-network support is enabled.

false: Multi-network support is disabled.

setEthernetStaticIp()

This method configures a static IP for Ethernet or switches to DHCP.

Prototype

```
void
com.vfi.smartpos.system_service.aidl.networks.INetworkManager.setEthernetStaticI
p(in Bundle bundle)
```

Parameter

`bundle` A Bundle object that contains the configuration details as shown below:

STATIC_IP:	The static IP address for the Ethernet interface. (e.g.192.168.1.1).
STATIC_GATEWAY:	The default gateway for the static IP (e.g. 192.168.1.1).
STATIC_NETMASK:	The subnet mask for the IP configuration (e.g. 255.255.255.0).
STATIC_DNS1:	The primary DNS server (e.g. 192.168.1.1).
STATIC_DNS2:	The secondary DNS server (optional) (e.g. 192.168.1.1).

Return Values

```
void
```

setWifiStaticIp()

This method allows to configure a static IP address or switch the Wi-Fi connection to DHCP mode.

Prototype

```
void
com.vfi.smartpos.system_service.aidl.networks.INetworkManager.setWifiStaticIp(in
Bundle bundle)
```

Parameter

`bundle`	A Bundle object that contains the configuration data including IP address, gateway, netmask, DNS servers, and a flag to switch to DHCP:

STATIC_IP:	The static IP address for the Wi-Fi interface. Setting this to "0.0.0.0" or "0" will switch to DHCP mode (e.g.192.168.1.1).
STATIC_GATEWAY:	The default gateway for the static IP (e.g. 192.168.1.1).
STATIC_NETMASK:	The subnet mask for the IP configuration (e.g. 255.255.255.0).
STATIC_DNS1:	The primary DNS server (e.g. 192.168.1.1).
STATIC_DNS2:	The secondary DNS server (optional) (e.g. 192.168.1.1).

Return Values

```
void
```

setMobilePreferredNetworkType()

This method enables the configuration of the preferred mobile network type on the current SIM card.

Prototype

```
void
com.vfi.smartpos.system_service.aidl.networks.INetworkManager.setMobilePreferred
NetworkType(String type)
```

Parameter

`type`	A string that specifies the preferred mobile network type. It can take the following values:

2G:	The device will only use 2G networks.
3G:	The device can use both 2G and 3G networks.
4G:	The device can use 2G, 3G, and 4G networks.

Return Values

```
void
```

getMobilePreferredNetworkType()

This method retrieves the preferred mobile network type for the current SIM card.

Prototype

```
String
com.vfi.smartpos.system_service.aidl.networks.INetworkManager.getMobilePreferred
NetworkType()
```

Parameter

None.

Return Values

The method returns the following values:

2G: If the device is set to use only 2G networks, this will be returned.

3G: If the device is set to use both 2G and 3G networks, this will be returned.

4G: If the device is set to use 2G, 3G, and 4G networks, this will be returned.

Null: If there is a failure to retrieve the current network type, it will return null.

2.1.2 IAppDeleteObserver

Package: com.vfi.smartpos.system_service.aidl.IAppDeleteObserver

Overview:

This interface provides a callback mechanism that enables monitoring of the application deletion process on the device. It includes a single method, `onDeleteFinished()`, which is invoked upon the completion of an application deletion operation.

Public Member Functions:

Modifier and Type	Method
void	**onDeleteFinished** (String packageName, int returnCode)

Member Function Documentation:

onDeleteFinished()

This method is a callback function that is triggered upon the completion of an application deletion. The method provides the package name of the deleted application and a return code to indicate the success or failure of the operation.

Prototype

```
void
com.vfi.smartpos.system_service.aidl.IAppDeleteObserver.onDeleteFinished(String
packageName, int returnCode)
```

Parameters

`packageName`	The package name of the application that was deleted.
`returnCode`	An integer indicating the result of the deletion operation. This code indicates whether the deletion was successful or if an error occurred.

Error Code	Value	Description
STATUS_SUCCESS	0	Package installation (or operation) was a success.
STATUS_FAILURE	1	Package installation (or operation) was a failure.

STATUS_FAILURE_LOW_BATTERY	2	The battery level is too low for an Android or Engage update.
STATUS_FAILURE_PACKAGE_NOT_A_ZIP_FILE	3	Package file was not a zip file as expected.
STATUS_FAILURE_PACKAGE_ZIP_FILE_EMPTY	4	Package file was an empty zip file.
STATUS_FAILURE_UNRECOGNIZED_UPDATE_ FILE_NAME	5	A file inside the package file has an unrecognized name.
STATUS_FAILURE_TOO_MANY_UPDATE_FILES	6	The package file contained more than one Android update file.
STATUS_FAILURE_UPDATE_FILE_VERIFY	7	The Android update file failed verification.
STATUS_FAILURE_UPDATE_FILE_APPLY	8	The update failed to apply.
STATUS_PENDING	9	The update is pending, expect a callback.

STATUS_SUCCESS_REBOOTING	10	The update is pending, expect a callback.
STATUS_FAILURE_SERVICE_NOT_BOUND	11	The service is not bound, no action taken.
STATUS_FAILURE_IO	12	The service is not bound, no action taken.
STATUS_FAILURE_NO_INFO_FILE	13	There is no info file present.
STATUS_FAILURE_INVALID_FILE_TYPE	14	The service is not bound, no action taken.
STATUS_PKG_TRANSFERING	15	The package is being transferred.
STATUS_PKG_VERIFING	16	The package is being verified.
STATUS_PKG_INSTALLING	17	The package is being installed.
STATUS_EMPTY_INSTALL	18	Install actions have been ignored or none have occurred.
STATUS_BUSY	19	Install is already in progress.

STATUS_FAILURE_LOW_FLASH_MEMORY	20	Terminal doesn't have enough storage space to process the request.
STATUS_UNKNOWN	21	The package status is unknown.

Return Values

`void`

2.1.3 IAppInstallObserver

Package: com.vfi.smartpos.system_service.aidl.IAppInstallObserver

Overview:

This interface provides a callback mechanism that enables monitoring of the application installation process on the device. It defines a single method, `onInstallFinished()`, which is invoked when the installation of an application has been completed.

Modifier and Type	Method
void	**onInstallFinished** (String packageName, int returnCode)

Member Function Documentation:

onInstallFinished()

This method is a callback function that is triggered upon the completion of an application installation. The method provides the package name of the installed application and a return code to indicate the success or failure of the operation.

Prototype

```
void
com.vfi.smartpos.system_service.aidl.IAppDeleteObserver.onInstallFinished(String
packageName, int returnCode)
```

Parameters

`packageName`	The package name of the application that was installed.
`returnCode`	An integer indicating the result of the installation operation. This code indicates whether the installation was successful or if an error occurred.

Error Code	Value	Description
STATUS_SUCCESS	0	Package installation (or operation) was a success.
STATUS_FAILURE	1	Package installation (or operation) was a failure.
STATUS_FAILURE_LOW_BATTERY	2	The battery level is too low for an Android or Engage update.
STATUS_FAILURE_PACKAGE_NOT_A_ZIP_FILE	3	Package file was not a zip file as expected.
STATUS_FAILURE_PACKAGE_ZIP_FILE_EMPTY	4	Package file was an empty zip file.

STATUS_FAILURE_UNRECOGNIZED_UPDATE_FILE_NAME	5	A file inside the package file has an unrecognized name.
STATUS_FAILURE_TOO_MANY_UPDATE_FILES	6	The package file contained more than one Android update file.
STATUS_FAILURE_UPDATE_FILE_VERIFY	7	The Android update file failed verification.
STATUS_FAILURE_UPDATE_FILE_APPLY	8	The update failed to apply.
STATUS_PENDING	9	The update is pending, expect a callback.
STATUS_SUCCESS_REBOOTING	10	The update is pending, expect a callback.
STATUS_FAILURE_SERVICE_NOT_BOUND	11	The service is not bound, no action taken.
STATUS_FAILURE_IO	12	The service is not bound, no action taken.
STATUS_FAILURE_NO_INFO_FILE	13	There is no info file present.

STATUS_FAILURE_INVALID_FILE_TYPE	14	The service is not bound, no action taken.
STATUS_PKG_TRANSFERING	15	The package is being transferred.
STATUS_PKG_VERIFING	16	The package is being verified.
STATUS_PKG_INSTALLING	17	The package is being installed.
STATUS_EMPTY_INSTALL	18	Install actions have been ignored or none have occurred.
STATUS_BUSY	19	Install is already in progress.
STATUS_FAILURE_LOW_FLASH_MEMORY	20	Terminal doesn't have enough storage space to process the request.
STATUS_UNKNOWN	21	The package status is unknown.

Return Values

```
void
```

2.1.4 ISystemManager

Package: com.vfi.smartpos.system_service.aidl.ISystemManager

Overview:

This interface provides a set of methods to manage system-level operations on the device, including installing and uninstalling applications, controlling device settings, and accessing network management functionalities.

Public Member Functions:

Modifier and Type	Method
void	**installApp** (String apkPath, IAppInstallObserver observer, String installerPackageName)
void	**uninstallApp** (String packageName, IAppDeleteObserver observer)
void	**reboot** ()
void	**isMaskHomeKey** (boolean state)
void	**isMaskStatusBard** (boolean state)
void	**updateROM** (String zipPath)
INetworkManager	**getNetworkManager** ()
void	**setLocationMode** (int status)
boolean	**isAdbMode** ()
boolean	**killApplication** (String packageName)
boolean	**restartApplication** (String packageName)
void	**initLogcat** (int logcatBufferSize, int logcatBufferSizeSuffix, in Bundle bundle)

String	**getLogcat** (String logcatFileName, int compressType)
Bundle	**getLaunchAppsInfo** (long beginTime, long endTime)
ISettingsManager	**getSettingsManager** ()
Bitmap	**takeCapture** ()
void	**shutdownDevice** ()

Member Function Documentation:

installApp()

This method installs an APK file onto the device by providing the necessary parameters, including the APK's location and an observer for installation status callbacks.

Ensure that the `apkPath` is valid and accessible and that the `installerPackageName` corresponds to a legitimate application package authorized to perform such actions on the device.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISystemManager.installApp(String
apkPath, IAppInstallObserver observer, String installerPackageName)
```

Parameters

`apkPath`	The absolute file path (URI) of the APK to be installed. This path must point to a valid APK file.
`observer`	A callback handler that allows the caller to receive updates on the installation process, such as success or failure notifications. Refer to `IAppInstallObserver`.
`installerPackageName`	The package name of the installer APK.

Return Values

`void`

uninstallApp()

This method uninstalls an application from the device by accepting the package name of the targeted application and an observer for tracking the uninstallation process.

Ensure that the provided `packageName` corresponds to an application currently installed on the device.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISystemManager.uninstallApp(String
packageName, IAppDeleteObserver observer)
```

Parameters

`packageName`	The unique package name of the application that is to be uninstalled. This must correspond to an installed application on the device.
`observer`	A callback handler that receives notifications regarding the status of the uninstallation process, including success or failure events. Refer to `IAppDeleteObserver`.

Return Values

`void`

reboot()

This method initiates a reboot of the device.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISystemManager.reboot()
```

Parameter

None.

Return Values

```
void
```

isMaskHomeKey()

This method enables or disables the masking of the Home key on the device. It controls the operational state of the Home key based on the application's requirements.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISystemManager.isMaskHomeKey(boolean
state)
```

Parameter

`state` A Boolean value that determines the action to be taken:

true:	Masks the Home key, preventing it from being used.
false:	Unmasks the Home key, allowing normal functionality.

Return Values

```
void
```

isMaskStatusBar()

This method enables or disables the masking of the Status Bar on the device. It allows the control of the visibility of the Status Bar by masking it based on the input parameter.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISystemManager.isMaskStatusBar(boolean
state)
```

Parameter

`state`	A Boolean value that indicates the status of the Status Bar masking: true: Enable masking of the Status Bar. false: Disable masking of the Status Bar.

Return Values

```
void
```

updateROM()

This method is called to update the ROM of the device.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISystemManager.updateROM(String
zipPath)
```

Parameter

`zipPath`	The file path to the ZIP containing the ROM update files. The specified path must be valid and accessible to ensure successful execution of the update process.

Return Values

```
void
```

getNetworkManager()

This method retrieves an instance of the network manager interface, allowing for management of network-related operations.

Prototype

```
INetworkManager getNetworkManager()
```

Parameter

None.

Return Values

An instance of `INetworkManager` such as network type, APN information, Mobile Data, Wi-Fi, and Ethernet Configuration.

setLocationMode()

This method sets the location mode of the device.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISystemManager.setLocationMode(int
status)
```

Parameter

`status`	An integer parameter that specifies the desired location mode for the device. The possible values for this parameter are predefined constants, which represent different location modes. 0: LOCATION_MODE_OFF (location services disabled). 1: LOCATION_MODE_SENSORS_ONLY (using only sensors for location, such as GPS). 2: LOCATION_MODE_BATTERY_SAVING (using Wi-Fi and mobile networks for location but with minimal power consumption). 3: LOCATION_MODE_HIGH_ACCURACY (using GPS, Wi-Fi, and mobile networks for high-precision location).

Return Values

```
void
```

isAdbMode()

This method returns the current status of ADB mode on the device.

Prototype

```
boolean com.vfi.smartpos.system_service.aidl.ISystemManager.isAdbMode ()
```

Parameter

None.

Return Values

A Boolean value:

true: ADB is enabled on the device.

false: ADB is disabled on the device.

killApplication()

This method is used to terminate the specified application by providing its package name.

Prototype

```
boolean
com.vfi.smartpos.system_service.aidl.ISystemManager.killApplication(String
packageName)
```

Parameter

`packageName`	The package name of the application user wants to terminate.

Return Values

A Boolean value:

true: If the application was successfully killed or terminated.

false: If there was a failure in killing the application.

restartApplication()

The method is used to restart a specific application by providing its package name.

Prototype

```
boolean
com.vfi.smartpos.system_service.aidl.ISystemManager.restartApplication(String
packageName)
```

Parameter

`packageName`	The package name of the application that user wants to restart.

Return Values

A Boolean value:

true: If the application was successfully restarted.

false: If there was a failure in restarting the application.

initLogcat()

This method is used to initialize or configure logcat settings on the device, such as the buffer size for logs. It provides flexibility in setting the logcat buffer size with different units (e.g., MB or KB).

Prototype

```
boolean com.vfi.smartpos.system_service.aidl.ISystemManager.initLogcat(int
logcatBufferSize, int logcatBufferSizeSuffix, in Bundle bundle)
```

Parameter

`logcatBufferSize`	The desired size of the logcat buffer. This determines how much memory will be allocated to store log messages.

	logcatBufferSize == 0:	The default buffer size will be applied, which is typically set by the system.
`logcatBufferSizeSuffix`	Specifies the unit for the buffer size.	
	0:	The suffix is "M" (megabytes).
	1:	The suffix is "K" (kilobytes).
`bundle`	An optional Bundle object that can be used to pass additional configuration options related to logcat.	

Return Values

A Boolean value:

true: If the operation was successful.

false: If the operation failed.

getLogcat()

This method retrieves the logcat file based on the parameters provided.

Prototype

```
String com.vfi.smartpos.system_service.aidl.ISystemManager.getLogcat(String
logcatFileName, int compressType)
```

Parameter

`logcatFileName`	This is the name of the logcat file user wish to retrieve.	
	logcatFileName == null:	If the value is null, the system will use a default logcat file name.
`compressType`	Defines the compression type for the log file.	

0:	No compression (the file will be in plain text format).
1:	Gzipped (the file will be compressed using `.gz` format).

Return Values

Returns the path or location of the log file as a String. The file path indicates where the logcat file is stored on the system. If no file is found, it may return null.

getLaunchAppsInfo()

This method retrieves the usage count of applications within a specified time range.

Prototype

```
Bundle
com.vfi.smartpos.system_service.aidl.ISystemManager.getLaunchAppsInfo(long
beginTime, long endTime)
```

Parameter

`beginTime`	The start time for retrieving app usage data. This is the timestamp (in milliseconds) from which to begin the data collection. For example, `beginTime = Calendar.getInstance().setDate(date).getTimeInMillis()`
`endTime`	The end time for retrieving app usage data. This is the timestamp (in milliseconds) up to which to collect app usage information. For example, `endTime = Calendar.getInstance().getTimeInMillis()`

Return Values

Returns a Bundle containing the usage data, specifically a JSON string that represents a list of `UsageStats` objects.

getSettingsManager()

This method retrieves an `ISettingsManager` object to perform settings-related actions.

Prototype

```
ISettingsManager
com.vfi.smartpos.system_service.aidl.ISystemManager.getSettingsManager ()
```

Parameter

None.

Return Values

Returns an instance of the `ISettingsManager` interface, which provides methods to interact with the system's settings.

takeCapture()

The method captures the current screen content and returns it as a `Bitmap` object, which can then be processed within the application.

Prototype

```
Bitmap com.vfi.smartpos.system_service.aidl.ISystemManager.takeCapture ()
```

Parameter

None.

Return Values

Returns a Bitmap object that represents the screenshot of the screen.

shutdownDevice()

This method is used to shut down the device.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISystemManager.shutdownDevice ()
```

Parameter

None.

Return Values

```
void
```

2.1.5 ISettingsManager

Package: com.vfi.smartpos.system_service.aidl.ISettingsManager

Overview:

This interface is used for managing different configuration settings on a device. It allows an application to adjust key device parameters, which can include system settings such as time configuration, screen brightness, and permissions.

NOTE

The `enableAlertWindow(String packageName)` method requires system-level permissions, which are not supported on Verifone Android 13 devices. This functionality can be enabled by modifying the application ID to start with com.priv.xxx (privilege access). For more details, refer to the Girgit Migration Guide.

Public Member Functions:

Modifier and Type	Method
int	**settingsSetActions** (int settingsType, in Bundle bundle)
Bundle	**settingsReadActions** (int settingsType, in Bundle bundle)
boolean	**settingPCIRebootTime** (int hour, int min, int sec)
long	**getPCIRebootTime** ()
void	**setScreenLock** (boolean isLock)
boolean	**setDeviceBrightnessLevel** (int level)
boolean	**isShowBatteryPercent** (boolean isShow)
void	**enableAlertWindow** (String packageName)
void	**clearCachesByPackageName** (String packageName)

Member Function Documentation:

settingsSetActions()

This method is used to apply configuration settings on a device. The method allows the system to set specific parameters based on the `settingsType` and a `bundle` containing the relevant data.

Prototype

```
int com.vfi.smartpos.system_service.aidl.ISettingsManager.settingsSetActions
(int settingsType, in Bundle bundle)
```

Code Snippet

```
//set system time whether to sync auto network time
Bundle bundle = new Bundle();
bundle.putString("SYSTEM_TIME_ACTIONS",
SettingsActions.SystemTimeActions.SET_AUTO_SYSTEM_TIME_STATE);
More informations of "SYSTEM_TIME_ACTIONS" pleae refer to
SettingsActions.SystemTimeActions class;
bundle.putInt("AUTO_SYSTEM_TIME", 0);
0: disable sync
1: sync

//set system time zone whether to sync auto network time zone
Bundle bundle = new Bundle();
bundle.putString("SYSTEM_TIME_ACTIONS",
SettingsActions.SystemTimeActions.SET_AUTO_SYSTEM_TIME_ZONE_STATE);
More informations of "SYSTEM_TIME_ACTIONS" pleae refer to
SettingsActions.SystemTimeActions class;
bundle.putInt("AUTO_SYSTEM_TIME_ZONE", 0);
0: disable sync
1: sync


//set system time
Bundle bundle = new Bundle();
bundle.putString("SYSTEM_TIME_ACTIONS",
SettingsActions.SystemTimeActions.SET_AUTO_SYSTEM_TIME_ZONE_STATE);
More informations of "SYSTEM_TIME_ACTIONS" pleae refer to
SettingsActions.SystemTimeActions class;
bundle.putString("SYSTEM_DATE", "20200707"); date format is yyyyMMdd;
bundle.putString("SYSTEM_TIME", "150629"); time format is HHmmss;


//set launcher
Bundle bundle = new Bundle();
bundle.putString("LAUNCHER_ACTIONS",
SettingsActions.LauncherActions.SET_LAUNCHER);
More informations of "LAUNCHER_ACTIONS" pleae refer to
SettingsActions.LauncherActions class;
bundle.putString("LAUNCHER_PACKAGE_NAME", "com.vfi.android.payment"); // passing
launcher application package name.
bundle.putBoolean("RUN_PACKAGE", true); //true: run application, false: not run
application.
```

Parameters

`settingsType` An integer representing the type of settings to be configured. Possible values:

LAUNCHER Settings related to the launcher.

DATE_TIME	Settings related to date and time.

bundle A Bundle containing the settings that need to be applied. Possible values:

Key	Value	Description
GET_AUTO_SYSTEM_TIME_STATE	1	Sync auto system time.
	0	Disable sync auto system time.
GET_AUTO_SYSTEM_TIME_ZONE_STATE	1	Sync auto system time zone.
	0	Disable sync auto system time zone.

Return Values

An integer indicating the result of the operation (e.g., success or failure).

0: The operation was successful, and the requested settings were retrieved.

-1: The operation failed, possibly due to an error while retrieving the settings.

settingsReadActions()

This method retrieves device settings based on the settingsType and returns them in a bundle.

Prototype

```
Bundle com.vfi.smartpos.system_service.aidl.ISettingsManager.settingsReadActions
(int settingsType, in Bundle bundle)
```

Parameters

`settingsType`	An integer indicating the type of settings to retrieve. The two possible values:

LAUNCHER	Settings related to the launcher.
DATE_TIME	Settings related to date and time.

`bundle`	A Bundle to receive the settings data. Possible values:

Key	Value	Description
GET_AUTO_SYSTEM_TIME_STATE	1	Sync auto system time.
	0	Disable sync auto system time.
GET_AUTO_SYSTEM_TIME_ZONE _STATE	1	Sync auto system time zone.
	0	Disable sync auto system time zone.

Return Values

A Bundle containing the retrieved settings.

If SettingActions is SystemTimeActions.GET_AUTO_SYSTEM_TIME_STATE SYSTEM_TIME_ACTIONS.

1: Sync auto system time.

0: Disable sync auto system time.

If SettingActions is SystemTimeActions.GET_AUTO_SYSTEM_TIME_ZONE_STATE AUTO_SYSTEM_TIME_ZONE.

1: Sync auto system time zone.

0: Disable sync auto system time zone.

settingPCIRebootTime()

This method is used to set the PCI reboot time on the device.

Prototype

```
boolean
com.vfi.smartpos.system_service.aidl.ISettingsManager.settingPCIRebootTime (int
hour, int min, int sec)
```

Parameters

`hour`	Hour of the reboot time (24-hour format; range: 0-23).
`min`	Minute of the reboot time (range: 0-59).
`sec`	Second of the reboot time (range: 0-59).

Return Values

A Boolean indicating whether the PCI reboot time was successfully set.

true: The PCI reboot time was successfully set.

false: The attempt to set the PCI reboot time failed.

getPCIRebootTime()

This method retrieves the current PCI reboot time set on the device.

Prototype

```
long com.vfi.smartpos.system_service.aidl.ISettingsManager.getPCIRebootTime ()
```

Parameters

None.

Return Values

A long representing the reboot time in seconds.

setScreenLock()

This method is used to lock or unlock the device's screen.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISettingsManager.setScreenLock
(boolean isLock)
```

Parameters

`isLock`	A Boolean representing the device screen lock status depending on its value: true: Locks the device screen. false: Unlocks the device screen.

Return Values

```
void
```

setDeviceBrightnessLevel()

This method sets the device's brightness level.

Prototype

```
boolean
com.vfi.smartpos.system_service.aidl.ISettingsManager.setDeviceBrightnessLevel
(int level)
```

Parameters

`level`	An integer value representing the brightness level range from 0 to 255.

Return Values

A Boolean indicating whether the brightness level was successfully set.

true: The brightness level was successfully set.

false: The attempt to set the brightness level failed.

isShowBatteryPercent()

This method enables or disables the display of battery percentage on the device.

Prototype

```
boolean
com.vfi.smartpos.system_service.aidl.ISettingsManager.isShowBatteryPercent
(boolean isShow)
```

Parameters

`isShow`	A Boolean indicating the display of the current battery percentage on the device's status bar. true: Shows the battery percentage on the device's status bar. false: Hides the battery percentage from the status bar.

Return Values

A Boolean value indicating whether the setting was successfully applied.

true: The setting was successfully applied.

false: The setting was not successfully applied.

enableAlertWindow()

This method enables an alert window for a specified application identified by its package name.

Prototype

```
void com.vfi.smartpos.system_service.aidl.ISettingsManager.enableAlertWindow
(String packageName)
```

Parameters

packageName	The package name of the app for which the alert window should be enabled.

Return Values

```
void
```

clearCachesByPackageName()

This method clears the cache of the specified application.

Prototype

```
void
com.vfi.smartpos.system_service.aidl.ISettingsManager.clearCachesByPackageName
(String packageName)
```

Parameters

packageName	The package name of the app whose cache should be cleared.

Return Values

```
void
```

2.2 AIDL: Device Service

The device service AIDL interfaces provide a structured and efficient way to manage various device-level operations on Neo devices. These interfaces define a set of methods to perform critical device functions such as card reading and activation, transaction operations, beeper, and scanner operations, and accessing device information and status functionalities etc.

2.2.1 CheckCardListener

Package: com.vfi.smartpos.deviceservice.aidl.CheckCardListener

Overview:

This interface provides callback methods for managing the card checking events such as card swiping, powering up, activation, timeout, and error conditions.

Public Member Functions:

Modifier and Type	Method
void	**onCardPowerUp** ()
void	**onCardActivate** ()
void	**onCardSwiped** (in Bundle track)
void	**onTimeout** ()
void	**onError** (int error, String message)

Member Function Documentation:

onCardPowerUp()

This method is called when the card reader has been successfully powered on. It indicates that the card reader is fully operational and ready for input, allowing users to proceed with card insertion, swiping, or tapping.

Run `IPBOC.startEMV` to start EMV workflow.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CheckCardListener.onCardPowerUp()
```

Parameters

None.

Return Values

```
void
```

See Also

Refer to `IPBOC.startEMV()` method under Section 2.2.15.

onCardActivate()

This method is called when a card is activated. It signals the application that the card is ready for further transaction processing operations (such as initiating a secure transaction or reading card data).

Run `IPBOC.startEMV()` to start EMV workflow.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CheckCardListener.onCardActivate()
```

Parameters

None.

Return Values

`void`

See Also

Refer to `IPBOC.startEMV()` method under Section 2.2.15.

onCardSwiped()

This method is triggered immediately when a card is successfully swiped through the card reader. It provides information about the card, which includes its PAN, track data, service code, and expiration date.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CheckCardListener.onCardSwiped (in
Bundle track)
```

Parameters

`track` A Bundle object containing card information. This includes:

PAN (String)	The Primary Account Number (PAN) of the card.
TRACK1 (String)	Track 1 data.
TRACK2 (String)	Track 2 data.
TRACK3 (String)	Track 3 data.

SERVICE_CODE (String)	The service code used to identify the card.
EXPIRED_DATE (String)	The expiration date of the card, typically formatted as MM/YY.

Return Values

```
void
```

See Also

- `onCardPowerUp()`
- `onCardActivate()`

onTimeout()

This method is called when a card operation exceeds the defined time limit. After timeout, the ongoing transaction is cancelled.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CheckCardListener.onTimeout()
```

Parameters

None.

Return Values

```
void
```

onError()

This method is called when an error occurs during the card checking process.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CheckCardListener.onError(int error,
String message)
```

Parameters

`error` An integer representing the error code associated with the event. The following error codes may be encountered:

Error Type	Error Code	Description
SERVICE_CRASH	(99)	Service crash.
REQUEST_EXCEPTION	(100)	Exception occurred while processing the request.
MAG_SWIPE_ERROR	(1)	Magnetic card read error.
IC_INSERT_ERROR	(2)	Smart card read error.
IC_POWERUP_ERROR	(3)	Smart card failed to power up.
RF_PASS_ERROR	(4)	Contactless card read error.
RF_ACTIVATE_ERROR	(5)	Contactless card activation error.
MULTI_CARD_CONFLICT_ERROR	(6)	Multiple cards are detected.
M1_CARD_UNSUPPORT_EMV_ERROR	(7)	[M1Sn]M1 card is not supported in EMV process.

	FELICA_CARD_UNSUPPORT_EMV_ERROR	(8)	Felica card is not supported in EMV process.
	DESFIRE_CARD_UNSUPPORT_EMV_ERROR	(9)	DesFireSN]DesFire card is not supported in EMV process.
`message`	A descriptive message providing additional context about the error encountered.		

Return Values

```
void
```

2.2.2 IBeeper

Package: com.vfi.smartpos.deviceservice.aidl.IBeeper

Overview:

This interface includes few methods to programmatically manage the beeping functionality of the device. It allows you to start and stop the beep sound, providing control over audio feedback during transaction processing.

Public Member Functions:

Modifier and Type	Method
void	**startBeep** (int msec)
void	**stopBeep** ()
void	**startBeepWithConfig** (int msec, in Bundle bundle)

Member Function Documentation:

startBeep()

This method enables the device beeper to emit sound for a duration specified in milliseconds. It is invoked when a card removal prompt appears on the device screen.

NOTE	This method is non-blocking, allowing the program to continue executing without waiting for the beeping to finish.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IBeeper.startBeep(int msec)
```

Parameters

`msec`	The duration for which the beeping sound will occur, specified in milliseconds. If this value is set to `0`, the beeper will be inactive, and no sound will be produced.

Return Values

```
void
```

stopBeep()

This method is called to stop the beeping sound instantly.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IBeeper.stopBeep()
```

Parameters

None.

Return Values

```
void
```

startBeepWithConfig()

This method is used to initiate a beeping sound based on a given configuration. It is a non-blocking method.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IBeeper.startBeepWithConfig (int msec,
in Bundle bundle)
```

Parameters

`msec`	Specifies the duration (in milliseconds) for how long the beep sound will play. If it is set as 0, the device will not beep.
`bundle`	A Bundle object that contains additional configuration options for the beep. It includes: The frequency of the beeping sound in Hertz (Hz). The default value is 850 Hz (the frequency can be set within a range of :20Hz to 20000Hz). Example: `bundle.putInt("HZ", 600).`

Return Values

```
void
```

2.2.3 ScannerListener

Package: com.vfi.smartpos.deviceservice.aidl.ScannerListener

Overview:

This interface manages several events associated with barcode scanning functionality on the device. It provides several callback methods to manage different outcomes of a scanning operation, such as success, error, timeout, and cancellation.

Public Member Functions:

Modifier and Type	Method
void	**onSuccess** (String barcode)
void	**onError** (int error, String message)
void	**onTimeout** ()
void	**onCancel** ()

Member Function Documentation:

onSuccess()

This method is called upon the successful scanning of a barcode.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.ScannerListener.onSuccess(String
barcode)
```

Parameters

`barcode`	A string that encapsulates the scanned barcode data. This value represents the unique identifier retrieved from the scanned barcode.

Return Values

```
void
```

onError()

This method is called when an error occurs during the barcode scanning process.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.ScannerListener.onError(int error,
String message)
```

Parameters

`error`	An integer representing the error code associated with the scanning failure. -1: Indicates that the scan has failed.
`message`	A string providing additional information about the error. It helps in understanding the nature of the failure.

Return Values

```
void
```

onTimeout()

This method is called when the barcode scanning operation exceeds the allowed time limit.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.ScannerListener.onTimeout()
```

Parameter

None.

Return Values

```
void
```

onCancel()

This method is called when the user cancels the barcode scanning operation.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.ScannerListener.onCancel()
```

Parameter

None.

Return Values

```
void
```

2.2.4 PrinterListener

Package: com.vfi.smartpos.deviceservice.aidl.PrinterListener

Overview:

This interface provides callback methods to notify users about the status of a print operation on a device.

Public Member Functions:

Modifier and Type	Method
void	**onFinish** ()

void	**onError** (int error)

Member Function Documentation:

onFinish()

This method is called when the print operation completes successfully, signaling the end of the printing.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PrinterListener.onFinish()
```

Parameters

None.

Return Values

```
void
```

onError()

This method is called when an error occurs during the print process. It provides an error code that indicates the nature of the issue encountered.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PrinterListener.onError (int error)
```

Parameters

`error`	Contains the specific error code associated with the print operation failure. Possible error code values:

Error Type	Error Code	Description
ERROR_NONE	(0x00)	Normal operation.
ERROR_PAPERENDED	(0xF0)	Out of paper.
ERROR_NOCONTENT	(0xF1)	No content to print.
ERROR_HARDERR	(0xF2)	Printer hardware error.
ERROR_OVERHEAT	(0xF3)	Printer overheating.
ERROR_NOBM	(0xF6)	No black mark detected.
ERROR_BUSY	(0xF7)	Printer is busy.
ERROR_MOTORERR	(0xFB)	Motor malfunction.
ERROR_LOWVOL	(0xE1)	Low battery voltage.
ERROR_NOTTF	(0xE2)	No True Type Font (TTF) available.
ERROR_BITMAP_TOOWIDE	(0xE3)	Bitmap width exceeds limit.

Return Values

```
void
```

2.2.5 PinInputListener

Package: com.vfi.smartpos.deviceservice.aidl.PinInputListener

Overview:

This interface provides callback methods to manage PIN input operations on the device. It handles the events related to processing PIN input, confirming entries, managing cancel events, and responding to error conditions.

Public Member Functions:

Modifier and Type	Method
void	**onInput** (int len, int key)
void	**onConfirm** (byte[] data, boolean isNonePin)
void	**onCancel** ()
void	**onError** (int errorCode)

Member Function Documentation:

onInput()

This method is called upon PIN input process, when a user enters a PIN on the device.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PinInputListener.onInput(int len, int
key)
```

Parameters

`len`	The length of the entered PIN (length ranges from 4 to 10 digits).
`key`	The masked key value associated with the input. It is used for PIN character masking.

Return Values

```
void
```

onConfirm()

This method is called when the PIN entry has been completed and confirmed.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PinInputListener.onConfirm (byte[]
data,boolean isNonePin)
```

Parameters

`data`	The byte array representation of the entered PIN number. If no PIN is entered, this will be 'null'.
`isNonePin`	A Boolean value that indicates whether the user entered a PIN or not. true: No PIN is entered. false: PIN is entered.

Return Values

```
void
```

onCancel()

This method is called when the user cancels the PIN input process before completion.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PinInputListener.onCancel ()
```

Parameters

None.

Return Values

```
void
```

onError()

This method is called when an error occurs during the PIN entry process, such as input validation failures, encryption issues, or other unforeseen exceptions.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PinInputListener.onError (int
errorCode)
```

Parameters

`errorCode` Contains the type of error encountered during the PIN input process. Possible values:

Error Code	Error Type	Description
-1	Input exception	An unexpected input scenario occurred.
-2	Input time out	The user took too long to enter the PIN.
-3	Plain text is null	The input data had no value.
-4	Encrypt error	There was a failure in encrypting the entered PIN.
-5	Cipher text is null	The resulting cipher text from the process is empty.
0xff	Other error	Indicates an unspecified error occurred.

Return Values

```
void
```

2.2.6 PBOCHandler

Package: com.vfi.smartpos.deviceservice.aidl.PBOCHandler

Overview:

This interface manages payment transaction operations within the application, adhering to the PBOC standards. It includes callback methods that facilitate the handling of various stages of payment transactions, including transaction amount requests, application selection, card information confirmation, PIN entry, and transaction outcomes.

Public Member Functions:

Modifier and Type	Method
void	**onRequestAmount** ()
void	**onSelectApplication** (in List< String > appList)
void	**onConfirmCardInfo** (in Bundle info)
void	**onRequestInputPIN** (boolean isOnlinePin, int retryTimes)
void	**onConfirmCertInfo** (String certType, String certInfo)
void	**onRequestOnlineProcess** (in Bundle aaResult)
void	**onTransactionResult** (int result, in Bundle data)

Member Function Documentation:

onRequestAmount()

This method is called to request the transaction amount.

NOTE	This method is deprecated and will not be called. The amount should be set while calling the `IPBOC.startEMV`.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PBOCHandler.onRequestAmount ()
```

Code Snippet

```
// show the amount screen and import the amount
ipboc.importAmount((int) amount);
// abort
ipboc.abortPBOC();
```

Parameters

None.

Return Values

```
void
```

See Also

Refer to `IPBOC.startEMV()` and `IPBOC.abortPBOC()` methods under Section 2.2.15.

onSelectApplication()

This method is called to select a specific application from a list of applications available on the smart card.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PBOCHandler.onSelectApplication (in
List<String> appList)
```

Code Snippet

```
IPBOC ipboc;
@Override
public void onSelectApplication(List<String> appList) throws RemoteException {
    int i = 1;
    //check the cancel flag
    Boolean cancelSelectApplication = false;
    if (cancelSelectApplication) {
        ipboc.abortPBOC();
        // ("user cancel");
        return;
    }
    for (String str : appList) {
        msg = i++ + ".AppName=" + str + "\n";
    }
    //show the application list, get the index
    ipboc.importAppSelect(index);
}
```

Parameters

`appList`	The list of application names available on the smart card.

Return Values

`void`

See Also

Refer to `IPBOC.abortPBOC()` and `IPBOC.importAppSelect()` methods under Section 2.2.15.

onConfirmCardInfo()

This method is called for confirming card information before proceeding with the transaction. It contains the necessary card details, including the card number, service codes, and expiration dates.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PBOCHandler.onConfirmCardInfo (in
Bundle info)
```

Code Snippet

```
@Override
    public void onConfirmCardInfo(Bundle info) throws RemoteException {
        // display the card infor
        String result = "onConfirmCardInfo callback, \nPAN:" +
info.getString("PAN") +
                "\nTRACK2:" + info.getString("TRACK2") +
                "\nCARD_SN:" + info.getString("CARD_SN")+
                "\nSERVICE_CODE:" + info.getString("SERVICE_CODE") +
                "\nEXPIRED_DATE:" + info.getString("EXPIRED_DATE");
        if (true) {
                byte[] strs = ipboc.getCardData("9F51");
            } else {
        }
        // get the result
        if ( cancel ) {
            ipboc.abortPBOC();
            ipboc.importCardConfirmResult(false);
        } else {
            ipboc.importCardConfirmResult(true);
        }
    }
}
```

Parameters

`info` A Bundle containing card information:

PAN(String)	The Primary Account Number (PAN).
TRACK2(String)	Track 2 data.

CARD_SN(String)	Card serial number.
SERVICE_CODE(String)	Service code used to identify the card.
EXPIRED_DATE(String)	Expiration date of the card, typically formatted as MM/YY.

Return Values

```
void
```

See Also

Refer to `IPBOC.abortPBOC()` and `IPBOC.importCardConfirmResult()` methods under Section 2.2.15.

onRequestInputPIN()

This method is called to request input for a PIN from the user.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PBOCHandler.onRequestInputPIN (boolean
isOnlinePin, int retryTimes)
```

Parameters

`isOnlinePin`	A Boolean value indicating whether the PIN should be processed online or offline. true: The PIN input is required for an online transaction. false: The PIN input is required for an offline transaction.
`retryTimes`	The maximum number of retry attempts allowed for entering the PIN in case of an offline transaction.

Return Values

```
void
```

onConfirmCertInfo()

This method is used to confirm certificate information of the card holder.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PBOCHandler.onConfirmCertInfo (String
certType, String certInfo)
```

Parameters

`certType`	The type of certificate being confirmed.
`certInfo`	The certificate information associated with the certificate type.

Return Values

```
void
```

onRequestOnlineProcess()

This method is called to request online transaction process.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PBOCHandler.onRequestOnlineProcess (in
Bundle aaResult)
```

Code Snippet

```
@Override public void onRequestOnlineProcess(Bundle aaResult) throws
RemoteException { String result = "onRequestOnlineProcess callback RESULT:" +
aaResult.getInt(BUNDLE_TRANS_RESULT) + "\nARQC_DATA:" +
aaResult.getString(BUNDLE_ARQC_DATA) + "\nREVERSAL_DATA:" +
aaResult.getString(BUNDLE_REVERSAL_DATA) + "\nCARD_ORG:" +
aaResult.getString(BUNDLE_EMV_CARDORG) + "\nPAN:" +
aaResult.getString(BUNDLE_PBOC_PAN) + "\nTRACK2:" +
aaResult.getString(BUNDLE_PBOC_TRACK2) + "\nEXPIRD_DATE:" +
aaResult.getString(BUNDLE_PBOC_EXPIRED_DATE) + "\nCARD_SN:" +
aaResult.getString(BUNDLE_PBOC_CARD_SN) + "\nDATE:" +
aaResult.getString(BUNDLE_EMV_DATE) + "\nTIME:" +
aaResult.getString(BUNDLE_EMV_TIME) + "\nCARD_HOLDER_NAME:" +
aaResult.getString(BUNDLE_EMV_HOLDERNAME);

Boolean getAppTLVListOption = true; if (getAppTLVListOption) { String[] strlist
= {"9F33", "9F40", "9F10", "9F26", "95", "9F37", "9F1E", "9F36", "82", "9F1A",
"9A", "9B", "50", "84", "5F2A", "8F"}; String strs =
ipboc.getAppTLVList(strlist); }

Boolean getData = importData.getBoolean(BUNDLE_IMPORT_IS_GET_PBOC_DATA); if
(getData) { String strs = "PAN:" + ipboc.getPBOCData(BUNDLE_PBOC_PAN) + "\n" +
"TRACK2:" + My15getPBOCData(BUNDLE_PBOC_TRACK2) + "\n" + "CARD_SN:" +
My15getPBOCData(BUNDLE_PBOC_CARD_SN) + "\n" + "EXPIRED_DATE:" +
My15getPBOCData("EXPIRED_DATE") + "\n" + "DATE:" + My15getPBOCData("DATE") +
"\n" + "TIME:" + My15getPBOCData("TIME") + "\n" + "BALANCE:" +
My15getPBOCData("BALANCE") + "\n" + "CURRENCY:" + My15getPBOCData("CURRENCY");

Log.d(TAG, "getPBOCData : " + strs); }

String[] tvr = {"95"}; ipboc.getAppTLVList(tvr);

//online String online_result = "";

//do online finish process ipboc.inputOnlineResult(importData, new
MyOnlineResultHandler()); }
```

Parameters

`aaResult` A Bundle containing the result data for the online process. It includes:

RESULT(int)	An integer indicating the outcome of the online process. QPBOC_ARQC (201) - QPBOC_ARQC, online request, part of PBOC standard. AARESULT_ARQC (2) - AARESULT ARQC, the action analysis result.

	PAYPASS_MAG_ARQC (302)-The mode of magnetic card on PayPass request. PAYPASS_EMV_ARQC (303)- The mode of EMV on PayPass request.
ARQC_DATA(String)	The ARQC (Authorization Request Cryptogram) data. It requests some of Field55 data. You can also use getAppTLVList() method to retrieve this data. CTLS data include: "9F26, 9F27, 9F10, 9F37, 9F1A, 9F36, 95, 9A, 9C, 9F02, 5F2A, 82, 9F03, 9F33, 9F34, 9F35, 84, 9F1E, 9F09, 9F41"). IC data include: "9F26, 9F27, 9F10, 9F37, 9F36, 95, 9A, 9C, 9F02, 5F2A, 82, 9F1A, 9F03, 9F33, 9F34, 9F35, 9F1E, 84, 9F09, 9F41").
REVERSAL_DATA(String)	Data for reversing the transaction if necessary. This is a subset of the Field55 data for the IC card. (IC data include "9F26, 9F27, 9F10, 9F37, 9F36, 95, 9A, 9C, 9F02,

5F2A, 82, 9F1A, 9F03,
9F33, 9F34, 9F35,
9F1E, 84, 9F09, 9F41").

You can get the following from ARQC data:

CARD_ORG(String)	The card organization.
PAN(String)	The PAN.
TRACK2(String)	The track 2.
EXPIRED_DATE(String)	Expired date.
CARD_SN(String)	The card SN.
DATE(String)	The date of transaction.
TIME(String)	The time of transaction.
CARD_HOLDER_NAME(String)	The card holder name.

Return Values

```
void
```

See Also

Refer to `IPBOC.getAppTLVList()`, `IPBOC.getPBOCData()` and `IPBOC.inputOnlineResult()` methods under Section 2.2.15.

onTransactionResult()

This method is called to process the outcome of a transaction request, logging the result (success or failure) and handling any associated data or errors.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.PBOCHandler.onTransactionResult
(int result, in Bundle data)
```

Code Snippet

```
@Override
public void onTransactionResult(int result, Bundle data) throws RemoteException
{
    Log.i(TAG, "onTransactionResult callback, result:" + result +
            "\nTC_DATA:" + data.getString(BUNDLE_TC_DATA) +
            "\nREVERSAL_DATA:" + data.getString(BUNDLE_REVERSAL_DATA) +
            "\nERROR:" + data.getString(BUNDLE_TRANS_ERROR));
    if ((type == 3) && (transType == EC_BALANCE || transType == Q_QUERY)) {
        String ecBalance = ipboc.getPBOCData("BALANCE");
        Log.i(TAG, "BALANCE:" + ecBalance);

        if (ecBalance != null)
            msg = ":" + ecBalance;
        logUtils.addCaseLog(msg);

    }
    msg = "result:" + result + "\n" + data.getString(BUNDLE_TRANS_ERROR);
    Message message1 = new Message();
    message1.getData().putString("message", msg);
    handler.sendMessage(message1);
}
```

Parameters

`result` An integer representing the specific result of the EMV transaction. The values can signify various states, including success, errors, and other critical transaction conditions. Below are the defined result codes:

Error Type	Error Code	Description
EMV_NO_APP	(8)	Indicates that no EMV application is found (aid parameter).
EMV_COMPLETE	(9)	The EMV transaction completed successfully.
EMV_OTHER_ERROR	(11)	General error indicating that the transaction has been aborted.

EMV_FALLBACK	(12)	Transaction fallback initiated.
EMV_DATA_AUTH_FAIL	(13)	Offline data authentication failure during EMV processing.
EMV_APP_BLOCKED	(14)	The application has been locked and is not usable.
EMV_NOT_ECCARD	(15)	Indicates the card presented is not an electronic cash card.
EMV_UNSUPPORT_ECCARD	(16)	The transaction does not support electronic cash cards.
EMV_AMOUNT_EXCEED_ON_PURELYEC	(17)	The consumption amount of pure electronic cash card exceeds the limit.
EMV_SET_PARAM_ERROR	(18)	Set parameter fail on 9F7A.
EMV_PAN_NOT_MATCH_TRACK2	(19)	The PAN does not match the track 2 data.
EMV_CARD_HOLDER_VALIDATE_ERROR	(20)	Cardholder authentication failed.
EMV_PURELYEC_REJECT	(21)	Transaction declined for purely electronic cash card.
EMV_BALANCE_INSUFFICIENT	(22)	Insufficient balance.
EMV_AMOUNT_EXCEED_ON_RFLIMIT_CHECK	(23)	Check if the transaction amount exceeds the contactless limit.

EMV_CARD_BIN_CHECK_FAIL	(24)	Card reading failed.
EMV_CARD_BLOCKED	(25)	Card is locked.
EMV_MULTI_CARD_ERROR	(26)	Doka conflict.
EMV_BALANCE_EXCEED	(27)	Balance exceeds the limit.
EMV_RFCARD_PASS_FAIL	(60)	Tap card failure.
EMV_IN_QPBOC_PROCESS	(99)	qPBOC is processing.
AARESULT_TC	(0)	TC on action analysis.
AARESULT_AAC	(1)	Refuse on action analysis.
QPBOC_AAC	(202)	Refuse on qPBOC.
QPBOC_ERROR	(203)	Error while processing qPBOC.
QPBOC_TC	(204)	TC on qPBOC.
QPBOC_CONT	(205)	Need contact for further processing.
QPBOC_NO_APP	(206)	Result of qPBOC, indicating no application (UP Card may be available).
QPBOC_NOT_CPU_CARD	(207)	The card present is not a CPU card.
QPBOC_ABORT	(208)	Transaction abort.
EMV_SEE_PHONE	(150)	PayPass result, please check the result on phone.

`data`	A Bundle object containing associated data related to the transaction result.

TC_DATA(String)	The string of TC; use getAppTLVList() to get the data. TC data include "9F26, 9F27, 9F10, 9F37, 9F36, 95, 9A, 9C, 9F02, 5F2A, 82, 9F1A, 9F03, 9F33, 9F34, 9F35, 9F1E, 84, 9F09, 9F41, 9F63, 91".
REVERSAL_DATA(String)	The string of reversal data.
ERROR(String)	The error description (from the result of PBOC).

Return Values

`void`

See Also

Refer to `IPBOC.getPBOCData()` method under Section 2.2.15.

2.2.7 OnlineResultHandler

Package: com.vfi.smartpos.deviceservice.aidl.OnlineResultHandler

Overview:

This interface handles the result of the online processing for PBOC transactions on devices. It provides one callback method, `onProccessResult()`, which is invoked upon the completion of an online transaction.

Public Member Functions:

Modifier and Type	Method

void	**onProccessResult** (int result, in Bundle data)

Member Function Documentation:

onProccessResult()

This method is called to process the result of an online transaction, with its outcome indicated by the `result` parameter.

Prototype

```
void
com.vfi.smartpos.deviceservice.aidl.OnlineResultHandler.onProccessResult(int
result, in Bundle data)
```

Parameters

`result` An integer indicating the status of the online transaction. The potential outcomes are defined by several error codes, each representing a distinct result of the transaction process.

Error Type	Error Code	Description
ONLINE_RESULT_TC	(0)	Online transaction successful.
ONLINE_RESULT_AAC	(1)	Online transaction refused.
ONLINE_RESULT_OFFLINE_TC	(101)	Online transaction failed; offline transaction successful.

ONLINE_RESULT_SCRIPT_NOT_EXECUTE	(102)	Script not executed.
ONLINE_RESULT_SCRIPT_EXECUTE_FAIL	(103)	Script execution failed.
ONLINE_RESULT_NO_SCRIPT	(104)	Online transaction failed; no script sent.
ONLINE_RESULT_TOO_MANY_SCRIPT	(105)	Online transaction failed; more than one script sent.
ONLINE_RESULT_TERMINATE	(106)	Online transaction failed, transaction terminated (GAC return value is not 9000, transaction termination indicated by 0x8F).
ONLINE_RESULT_ERROR	(107)	Online transaction failed, EMV kernel error.
ONLINE_RESULT_OTHER_ERROR	(110)	Other errors.

`data` A Bundle containing additional data related to the transaction result.

TC_DATA(String)	The TC (Transaction Certificate) data in TLV

	(Tag-Length-Value) format.
SCRIPT_DATA(String)	The script result data in TLV format.
REVERSAL_DATA(String)	The reversal data in TLV format.

Return Values

```
void
```

2.2.8 EMVHandler

Package: com.vfi.smartpos.deviceservice.aidl.EMVHandler

Overview:

This interface provides a set of callback methods for handling different events during an EMV transaction process. These methods cover critical phases, including amount requests, application selections, card information confirmations, PIN inputs, certificate verifications, online processing, and transaction results.

Public Member Functions:

Modifier and Type	Method
void	**onRequestAmount** ()
void	**onSelectApplication** (in List< Bundle > appList)
void	**onConfirmCardInfo** (in Bundle info)
void	**onRequestInputPIN** (boolean isOnlinePin, int retryTimes)
void	**onConfirmCertInfo** (String certType, String certInfo)
void	**onRequestOnlineProcess** (in Bundle aaResult)
void	**onTransactionResult** (int result, in Bundle data)

int[]	**onGetCTLSAppPriority** (in List< Bundle > appList)

Member Function Documentation:

onRequestAmount()

This method is called to request the transaction amount.

NOTE	This method is deprecated and will not be called. The amount should be set while calling `IEMV.startEMV()` method.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.EMVHandler.onRequestAmount()
```

Parameters

None.

Return Values

```
void
```

See Also

Refer to `IEMV.startEMV()` and `IEMV.abortEMV()` methods under Section 2.2.10.

onRequestInputPIN()

This method is called to request PIN input from the user.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.EMVHandler.onRequestInputPIN(boolean
isOnlinePin, int retryTimes)
```

Parameters

`isOnlinePin`	A Boolean value indicating whether the PIN should be processed online or offline. true: The PIN input is required for an online transaction. false: The PIN input is required for an offline transaction.
`retryTimes`	The maximum number of retry attempts allowed for entering the PIN in case of an offline transaction.

Return Values

```
void
```

onRequestOnlineProcess()

This method is called to request online process of an EMV transaction.

During this process, the payment application performs various checks and validations, such as verifying the cardholder's identity, checking for sufficient funds, and obtaining authorization for the transaction.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.EMVHandler.onRequestOnlineProcess(in
Bundle aaResult)
```

Parameters

`aaResult`	A Bundle containing the result data from the application authentication process. It includes:

SIGNATURE(Boolean)	Indicates whether a signature is required.
CTLS_CVMR(int)	Contains the Cardholder Verification Method (CVM) result for contactless (CTLS) transactions. 0: No CVM required. 1: CVM with PIN. 2: CVM with signature. 3: CDCVM (Consumer Device Cardholder Verification Method).
RESULT(int)	The type of result from the online process. CTLS_ARQC (201): Online request part of the EMV standard. AARESULT_ARQC (2): Action analysis result. PAYPASS_MAG_ARQC (302): Magnetic card mode on PayPass request. PAYPASS_EMV_ARQC (303): EMV mode on PayPass request.
ARQC_DATA(String)	The ARQC (Authorization Request Cryptogram) data. It requests some of Field55 data. You can also use getAppTLVList() method to retrieve this data.

	CTLS data include: 9F26, 9F27, 9F10, 9F37, 9F36, 95, 9A, 9C, 9F02, 5F2A, 82, 9F03, 9F33, 9F34, 9F35, 84, 9F1E, 9F09, 9F41.
REVERSAL_DATA(String)	Data for reversing the transaction if necessary. This is a subset of the Field55 data for the IC card. Example IC data include: 9F26, 9F27, 9F10, 9F37, 9F36, 95, 9A, 9C, 9F02, 5F2A, 82, 9F03, 9F33, 9F34, 9F35, 9F1E, 84, 9F09, 9F41.

Return Values

```
void
```

See Also

Refer to `IEMV.getAppTLVList()`, `IEMV.getEMVData()` and `IEMV.inputOnlineResult()` methods under Section 2.2.10.

onSelectApplication()

This method is called to select a specific application from a list of applications available on the smart card.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.EMVHandler.onSelectApplication(in List<
Bundle >appList)
```

Code Snippet

```
IEMV iemv;
```

```
@Override
public void onSelectApplication(List<Bundle> appList) throws RemoteException {
    int i = 1;
    //check the cancel flag
    Boolean cancelSelectApplication = false;
    if (cancelSelectApplication) {
        iemv.abortEMV();
        // ("user cancel");
        return;
    }
    for (Bundle app : appList) {
        .....
        .....
    }
    //show the application list, get the index
    iemv.importAppSelection(index);
}
```

Parameters

`appList` A collection of application bundles, each representing an application available on the smart card. Each application bundle contains following details of the application.

aidName(String)	TAG9F12 Application Preferred Name.
aidLabel(String)	TAG50 Application Label.
aid(String)	Application Identifier.
aidPriority(int)	TAG87 Application Priority Indicator.
aidIssuerIdx(int)	TAG9F11 Issuer Code Table Index.

Return Values

`void`

See Also

Refer to `IEMV.abortEMV()` and `IEMV.importAppSelection()` methods under Section 2.2.10.

onConfirmCardInfo()

This method is called for confirming card information before proceeding with the transaction. It contains the necessary card details, including the card number, service codes, and expiration dates.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.EMVHandler.onConfirmCardInfo(in Bundle
info)
```

Code Snippet

```
@Override
    public void onConfirmCardInfo(Bundle info) throws RemoteException {
        // display the card infor
        String result = "onConfirmCardInfo callback, \nPAN:" +
info.getString("PAN") +
                "\nTRACK2:" + info.getString("TRACK2") +
                "\nCARD_SN:" + info.getString("CARD_SN")+
                "\nSERVICE_CODE:" + info.getString("SERVICE_CODE") +
                "\nEXPIRED_DATE:" + info.getString("EXPIRED_DATE");

        if (true) {
                byte[] strs = iemv.getCardData("9F51");
            } else {
        }

        // get the result
        if ( cancel ) {
            iemv.abortEMV();
            iemv.importCardConfirmResult(false);
        } else {
            iemv.importCardConfirmResult(true);
        }
    }
}
```

Parameters

`info`	Contains card information.	
	PAN(String)	The PAN.

TRACK1(String)	Track 1 data.
TRACK2 (String)	Track 2 data.
CARD_SN(String)	Card serial number.
SERVICE_CODE(String)	Service code.
EXPIRED_DATE(String)	Expiration date of the card.
CARD_TYPE(int)	Card type by CTLS. 0: EMV card. 1: MST stripe card. 2: Other.

Return Values

```
void
```

See Also

Refer to `EMV.abortEMV()` and `IEMV.importCardConfirmResult()` methods under Section 2.2.10.

onConfirmCertInfo()

This method is called to confirm cardholder certification information during EMV transaction process.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.EMVHandler.onConfirmCertInfo(String
certType, String certInfo)
```

Parameters

`certType`	The type of certificate being confirmed.
`certInfo`	The certificate information associated with the certificate type.

Return Values

`void`

onTransactionResult()

This method is called to provide the result of an EMV transaction, including successful completion and multiple error scenarios.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.EMVHandler.onTransactionResult(int result, in
Bundle data)
```

Parameters

`result` An integer representing the specific result of the EMV transaction. The values can signify various states, including success, errors, and other critical transaction conditions. Below are the defined result codes:

Error Type	Error Code	Description
EMV_NO_APP	(8)	EMV no application found (aid param).
EMV_COMPLETE	(9)	EMV transaction completed.
EMV_OTHER_ERROR	(11)	Other error encountered, transaction abort.

EMV_FALLBACK	(12)	Fallback procedure initiated.
EMV_DATA_AUTH_FAIL	(13)	Data authentication failure.
EMV_APP_BLOCKED	(14)	Application has been blocked.
EMV_NOT_ECCARD	(15)	Card is not recognized as an EMV card.
EMV_UNSUPPORT_ECCARD	(16)	Unsupported EMV card.
EMV_AMOUNT_EXCEED_ON_PURELYEC	(17)	Amount exceeds the EC limit.
EMV_SET_PARAM_ERROR	(18)	Set parameter fail on 9F7A.
EMV_PAN_NOT_MATCH_TRACK2	(19)	PAN does not match track 2 data.
EMV_CARD_HOLDER_VALIDATE_ERROR	(20)	Cardholder validation error.
EMV_PURELYEC_REJECT	(21)	Purely EC transaction rejected.
EMV_BALANCE_INSUFFICIENT	(22)	Balance insufficient.
EMV_AMOUNT_EXCEED_ON_RFLIMIT_CHECK	(23)	Amount exceeded the CTLS limit.
EMV_CARD_BIN_CHECK_FAIL	(24)	Check card failed.
EMV_CARD_BLOCKED	(25)	Card has been blocked.
EMV_MULTI_CARD_ERROR	(26)	Multiple card conflict.

EMV_INITERR_GPOCMD	(27)	GPO Processing Options response error.
EMV_GACERR_GACCMD	(28)	GAC response error.
EMV_TRY_AGAIN	(29)	Try again.
EMV_ODA_FAILED	(30)	ODA failed.
EMV_CVM_FAILED	(31)	CVM response error.
EMV_RFCARD_PASS_FAIL	(60)	Tap card failure.
AARESULT_TC	(0)	TC on action analysis.
AARESULT_AAC	(1)	Refuse on action analysis.
CTLS_AAC	(202)	Refuse on CTLS.
CTLS_ERROR	(203)	Error on CTLS.
CTLS_TC	(204)	Approval on CTLS.
CTLS_CONT	(205)	Need contact.
CTLS_NO_APP	(206)	Result of CTLS, no application (UP Card maybe available).
CTLS_NOT_CPU_CARD	(207)	The card is not a CPU card.
CTLS_ABORT	(208)	Transaction aborted.
CTLS_ISSUERUPDATE_APPROVE	(209)	Second tap for issuer update approved.
CTLS_CARD_BLOCK	(210)	6A81 error card block.
CTLS_SEL_FILE_INVALID	(211)	6283 error Selected file invalidated.

EMV_SEE_PHONE	(150)	PayPass result, please check the result on phone.
QPBOC_KERNAL_INIT_FAILED	(301)	CTLS kernel initialization failed.

`data` A data Bundle containing additional result-associated information.

TC_DATA(String)	The string of TC; use getAppTLVList() to get it. TC data include "9F26, 9F27, 9F10, 9F37, 9F36, 95, 9A, 9C, 9F02, 5F2A, 82, 9F1A, 9F03, 9F33, 9F34, 9F35, 9F1E, 84, 9F09, 9F41, 9F63, 91")
REVERSAL_DATA(String)	The string of reversal data.
ERROR(String)	The error description (from the result of EMV).
SIGNATURE(boolean)	Specifies whether a signature is required when result is "CTLS_TC (204)".
CTLS_CVMR(int)	Provides the Cardholder Verification Method (CVM) result for CTLS when result is "CTLS_TC (204)". 0: NO_CVM. 1: CVM_PIN. 2: CVM_SIGN. 3: CVM_CDCVM.
CARD_TYPE(int)	Indicates the type of card processed via CTLS. 0: EMV card. 1: Magnetic stripe card. 2: Other.

SUB_ERROR(int)

A field queried when receiving the CTLS_CONT (205) result.

Error Code	Error Type	Description
0:	NONE.	None.
7:	M1_CARD_UNSUPPORT_EMV_ERROR	[M1Sn] M1 card not supported in EMV process.
8:	FELICA_CARD_UNSUPPORT_EMV_ERROR	EMV not supported Felica card.
9:	DESFIRE_CARD_UNSUPPORT_EMV_ERROR	[DesFireSN] DesFire card not supported in EMV process.

CARD_SN(String)

This is only applicable for NFC cards; could be empty depending on card type.

Return Values

`void`

See Also

Refer to `IEMV.getEMVData()` method under Section 2.2.10.

onGetCTLSAppPriority()

This method retrieves the current priority order of applications available on a smart card during a CTLS transaction.

When this method is invoked, it analyzes the list of applications provided through the Bundle `appList` parameter. The method then returns an array of integers, where each integer represents the explicit priority for the corresponding applications. The priority can be modified via the `appList` parameter.

> NOTE
>
> The application can set the card priority only if the `ctlsPriority(byte)` parameter under `IEMV.startEMV()` method is set to 0xFF.

Prototype

```
int[] com.vfi.smartpos.deviceservice.aidl.EMVHandler.onGetCTLSAppPriority (in
List< Bundle > appList)
```

Parameters

`appList`

A collection of contactless application list, each representing applications available in smartcard. The list contains the following details of the CTLS application:

aidName(String)	TAG9F12 Application Preferred Name.
aid(String)	Application Identifier. It is a unique identifier that distinguishes each application.
aidPriority(int)	TAG87 Application Priority Indicator.

Return Values

Returns an array of integers containing the card application priorities.

For example, if CTLS priority is set as 0xFF in `IEMV.startEmv()`, then the method will return values similar to the following:

‘{0x04, 0x01, 0x02, 0x03}’: This indicates that application 4 has the highest priority, followed by application 1, then 2, and finally 3.

0x01- Visa

0x02- Mastercard

0x03- Amex

0x04- MCCS debit

See Also

Refer to `IEMV.startEMV()` method under Section 2.2.10.

2.2.9 IDeviceInfo

Package: com.vfi.smartpos.deviceservice.aidl.IDeviceInfo

Overview:

This interface provides several methods for effectively managing and retrieving information about a device. It provides a collection of public member functions that allow retrieval of various device attributes, management of system settings, and monitoring of device power status.

Public Member Functions:

Return Type	Method
String	**getSerialNo** ()
String	**getIMSI** ()
String	**getIMEI** ()

String	**getICCID** ()
String	**getManufacture** ()
String	**getModel** ()
String	**getAndroidOSVersion** ()
String	**getAndroidKernelVersion** ()
String	**getROMVersion** ()
String	**getFirmwareVersion** ()
String	**getHardwareVersion** ()
boolean	**updateSystemTime** (String date, String time)
boolean	**setSystemFunction** (Bundle bundle)
TusnData	**getTUSN** (int mode, in byte[] input)
String	**getPN** ()
void	**setPowerStatus** (boolean status)
String	**getRamTotal** ()
String	**getRamAvailable** ()
String	**getRomTotal** ()
String	**getRomAvailable** ()
String	**getMobileDataUsageTotal** ()
String	**getBootCounts** ()
String	**getPrintPaperLen** ()
String	**getMagCardUsedTimes** ()
String	**getSmartCardUsedTimes** ()

String	**getCTLSCardUsedTimes** ()
String	**getBatteryTemperature** ()
String	**getBatteryLevel** ()
String	**getK21Version** ()
String	**getMEID** ()
String	**getTamperCode** ()
String	**getServiceVersion** ()
Bundle	**getKernelVersion** ()
String	**getCertificate** (int mode)
String	**getBatteryChargingTimes** ()
int	**getDeviceStatus** (in Bundle bundle)
String	**getButtonBatteryVol** ()
Bundle	**getDeviceInfo** ()

Member Function Documentation:

getSerialNo()

This method is called to retrieve the unique Serial Number (SN) of the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getSerialNo ()
```

Parameters

None.

Return Values

A string value representing the unique SN of the device.

getIMSI()

This method is called to retrieve the International Mobile Subscriber Identity (IMSI) for a device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getIMSI ()
```

Parameters

None.

Return Values

A string value representing the IMSI of the device.

getIMEI()

This method is called to retrieve the International Mobile Equipment Identity (IMEI) of a device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getIMEI ()
```

Parameters

None.

Return Values

A string value representing the IMEI of the device.

getICCID()

This method is called to retrieve the Integrated Circuit Card Identifier (ICCID) of a SIM card in the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getICCID ()
```

Parameters

None.

Return Values

A string value representing the ICCID. The ICCID includes up to 19 digits and is unique to each SIM card.

getManufacture()

This method is called to retrieve the manufacturer of the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getManufacture ()
```

Parameters

None.

Return Values

A string value representing the manufacturer of the device.

getModel()

This method is called to retrieve the model's name of the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getModel ()
```

Parameters

None.

Return Values

A string value representing the device's model name.

getAndroidOSVersion()

This method is called to retrieve the android operating system (OS) version currently running on the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getAndroidOSVersion ()
```

Parameters

None.

Return Values

A string value representing the android OS version of the device.

getAndroidKernelVersion()

This method is called to retrieve the android kernel version of the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getAndroidKernelVersion
()
```

Parameters

None.

Return Values

A string value representing the android kernel version of the device.

getROMVersion()

This method is called to retrieve the android ROM version of the device

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getROMVersion ()
```

Parameters

None.

Return Values

A string value representing the ROM version of the device.

getFirmwareVersion()

This method is called to retrieve the Firmware version currently running on the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getFirmwareVersion ()
```

Parameters

None.

Return Values

A string value representing the firmware version of the device.

getHardwareVersion()

This method is called to retrieve the Hardware version of the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getHardwareVersion ()
```

Parameters

None.

Return Values

A string value representing the hardware version of the device.

updateSystemTime()

This method is called to update the system's date and time on the device.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.updateSystemTime (String
date, String time)
```

Parameters

`date`	A string representing the date format YYYYMMDD user wants to set on the device.
`time`	A string representing the time format HHMMSS user wants to set on the device.

Return Values

A Boolean value:

true: Indicates that the date and time were successfully updated.

false: Indicates that the update operation failed.

setSystemFunction()

This method is called to set specific system functions based on the data passed in the bundle.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.setSystemFunction (in
Bundle bundle)
```

Parameters

`bundle`	The Bundle contains specific system functions to perform.	
	HOMEKEY(Boolean)	Indicates whether to enable or disable the Home key function.
	STATUSBARKEY(Boolean)	Indicates whether to enable or disable the Status bar key function in the drop-down menu.

Return Values

A Boolean value:

true: Indicates that the operation was successful.

false: Indicates that the operation failed.

getTUSN()

This method is called to retrieve the Terminal Unique Serial Number (TUSN) of a UnionPay terminal.

Prototype

```
TusnData com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getTUSN (int mode,
in byte[] input)
```

Parameters

`mode`	An integer that specifies mode for retrieving the TUSN. The mode parameter is reserved and must be set to 0.
`input`	Refers to the random number involved in the calculation when calculating MAC for TUSN. The allowed range is 4 to 10 bytes.

Return Values

Returns TUSN data which includes Terminal Type, MAC and TUSN successfully; returns null if failed.

getPN()

This method is called to retrieve the PN associated with a device (such as a UnionPay terminal).

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getPN ()
```

Parameters

None.

Return Values

A string value representing the PN.

setPowerStatus()

This method is called to set the operational status of the power key on the device. It allows you to enable or disable the power key's functionality.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.setPowerStatus (boolean
status)
```

Parameters:

`status` A Boolean indicating the power key status.

true: Disable the power key.

false: Enable the power key.

Return Values

```
void
```

getRamTotal()

This method is called to retrieve the total amount of Random Access Memory (RAM) on the device. The returned value is expressed in bytes and represents the overall memory capacity available for the application.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getRamTotal ()
```

Parameters

None.

Return Values

A string value representing the overall RAM capacity of the device in bytes.

getRamAvailable()

This method is called to retrieve the amount of available RAM (in bytes) on the device that is currently free for use.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getRamAvailable ()
```

Parameters

None.

Return Values

A string value representing the available RAM capacity of the device.

getRomTotal()

This method is called to retrieve the total amount of Flash Read-Only Memory (ROM) available on the device. The returned value is expressed in bytes and represents the overall storage capacity.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getRomTotal ()
```

Parameters

None.

Return Values

A string value representing the overall ROM capacity of the device in bytes.

getRomAvailable()

This method is called to retrieve the amount of available Flash ROM memory (in bytes) currently free for use.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getRomAvailable ()
```

Parameters

None.

Return Values

A string value representing the available ROM capacity of the device in bytes.

getMobileDataUsageTotal()

This method is called to retrieve the total mobile data usage of the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getMobileDataUsageTotal
()
```

Parameters

None.

Return Values

A string value representing the total mobile data usage of the device in bytes.

getPrintPaperLen()

This method is called to retrieve the length of print paper.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getPrintPaperLen ()
```

Parameters

None.

Return Values

A string value representing the length of the print paper unit in millimeter (mm).

getMagCardUsedTimes()

This method is called to track and retrieve the number of times the magnetic card reader has been used on the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getMagCardUsedTimes ()
```

Parameters

None.

Return Values

A string value representing the number of times a magnetic card reader has been used.

getSmartCardUsedTimes()

This method is called to track and retrieve the number of times a smart card reader has been used on the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getSmartCardUsedTimes ()
```

Parameters

None.

Return Values

A string value representing the number of times a smart card reader has been used.

getCTLSCardUsedTimes()

This method is called to track and retrieve the number of times a contactless (CTLS) card reader has been used on the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getCTLSCardUsedTimes ()
```

Parameters

None.

Return Values

A string value representing the number of times a CTLS card reader has been used.

getBatteryTemperature()

This method is called to retrieve the temperature of the device’s battery.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getBatteryTemperature ()
```

Parameters

None.

Return Values

A string value representing the temperature of the device’s battery.

getBatteryLevel()

This method is called to retrieve the current charging level of the device's battery. The return value provides a clear indication of the battery's remaining capacity.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getBatteryLevel ()
```

Parameters

None.

Return Values

A string value representing the current charging level of the device’s battery.

getK21Version()

This method is called to retrieve the K21 version specific to the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getK21Version ()
```

Parameters

None.

Return Values

A string value representing the K21 version of the device.

getMEID()

This method is called to retrieve the Mobile Equipment Identifier (MEID) of the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getMEID ()
```

Parameters

None.

Return Values

A string value representing the MEID of the device.

getTamperCode()

This method is called to retrieve the tamper detection code for the device. This code indicates if the device has been tampered.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getTamperCode ()
```

Parameters

None.

Return Values

A string value representing the tamper detection code.

getServiceVersion()

This method is called to retrieve the service version of the device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getServiceVersion ()
```

Parameters

None.

Return Values

A string that represents the service version of the device.

getKernelVersion()

This method is called to retrieve the version of the kernel currently running on the device.

Prototype

```
Bundle com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getKernelVersion ()
```

Parameters

None.

Return Values

A Bundle object containing various keys with string values related to payment types below:

SmartEMV(String)	Represents Smart EMV payment method.
Visa(String)	Represents Visa payment method.
MasterCard(String)	Represents MasterCard payment method.
JCB(String)	Represents JCB payment method.
AMEX(String)	Represents American Express payment method.
Discover(String)	Represents Discover payment method.
QuickPass(String)	Represents QuickPass payment method.
GemaltoPure(String)	Represents Gemalto Pure payment method.

getCertificate()

This method is called to retrieve and display the digital certificate of the device based on the specified mode.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getCertificate (int mode)
```

Parameters

`mode`	An integer that specifies the mode for retrieving the certificate:

O:	Retrieve the certificate for sponsor digest.
Any other value:	Not supported; return an empty string.

Return Values

A string value representing the value of the digital certificate.

getBatteryChargingTimes ()

This method is called to retrieve the total duration for which the device's battery has been in a charging state.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo. getBatteryChargingTimes
()
```

Parameters

None.

Return Values

A string value representing the duration of the battery’s charging time.

getDeviceStatus()

This method checks the status of various device components such as printers, card readers, PIN pad, cameras, and SD cards.

Prototype

```
bundle com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getDeviceStatus (in
Bundle bundle)
```

Parameters

`bundle` A Bundle object that contains a key-value pair, where the key is "DeviceType" and the value is a string specifying the type of device. The device types are predefined, and you need to provide one of the following types:

PRINTER:	For checking the status of a printer.
ICCARDREADER_SLOT1:	For the first ICC card reader slot.
ICCARDREADER_SLOT2:	For the second ICC card reader slot.
RFCARDREADER:	For an RFID card reader.
SAMCARDREADER_SLOT1:	For the first SAM card reader slot.
SAMCARDREADER_SLOT2:	For the second SAM card reader slot.
PINPAD:	For checking the status of a PIN pad.

CAMERA_FRONT: For checking the status of the front camera.

CAMERA_REAR: For checking the status of the rear camera.

SDCARD: For checking the status of the SD card.

Return Values

An integer value to indicate the device working status:

0: The device is normal.

-1: The device is abnormal.

getButtonBatteryVol()

The method is used to retrieve the current voltage level of the button battery, a coin cell battery, which is often used to power certain hardware components when the device is turned off.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getButtonBatteryVol ()
```

Parameters

None.

Return Values

A string value representing the voltage of the button battery. It indicates the measured voltage unit in volts (V).

getDeviceInfo()

This method is used to retrieve a Bundle containing information about the device. It collects important identifiers and details related to the device, such as serial number, model, firmware version, and more. The collected data can be used for device management, diagnostics, and configuration purposes.

Prototype

```
Bundle com.vfi.smartpos.deviceservice.aidl.IDeviceInfo.getDeviceInfo ()
```

Parameters

None.

Return Values

A Bundle containing the following key-value pairs:

SN: Serial Number of the device.

PN: Product Number.

IMSI: International Mobile Subscriber Identity (for mobile network identification).

IMEI: International Mobile Equipment Identity (for mobile device identification).

MEID: Mobile Equipment Identifier.

manufacture: The manufacturer of the device.

deviceModel: The model of the device.

androidOsVer: The version of Android OS the device is running.

androidKernalVer: The version of the Android kernel.

romVer: The version of the device’s ROM.

firmwareVer: The version of the device’s firmware.

hardwareVer: The version of the device’s hardware.

k21Ver: Version of K21 (specific to the device, possibly related to a proprietary software or firmware).

GirgitSerivceVer: Version of Girgit Service (likely related to some system service version).

VRKSn: VRK Serial Number (a specific identifier, perhaps for a part or accessory).

SponsorID: Sponsor ID (used in cases where a sponsor or specific partner is associated with the device).

2.2.10 IEMV

Package: com.vfi.smartpos.deviceservice.aidl.IEMV

Overview:

The `IEMV` interface provides essential methods for interacting with an EMV-compliant card reader, facilitating a secure and efficient payment process. This interface includes functions for processing transactions, reading card data, and handling user interactions.

Public Member Functions:

Modifier and Type	Method
void	**checkCard** (in Bundle cardOption, int timeout, **CheckCardListener** listener)
void	**stopCheckCard** ()
void	**startEMV** (int processType, in Bundle intent, **EMVHandler** handler)
void	**abortEMV** ()

boolean	**updateAID** (int operation, int aidType, String aid)
boolean	**updateRID** (int operation, String rid)
void	**importAmount** (long amount)
void	**importAppSelection** (int index)
void	**importPin** (int option, in byte[] pin)
void	**importCertConfirmResult** (int option)
void	**importCardConfirmResult** (boolean pass)
void	**importOnlineResult** (in Bundle onlineResult, **OnlineResultHandler** handler)
void	**setEMVData** (in List< String > tlvList)
String	**getAppTLVList** (in String[] taglist)
byte[]	**getCardData** (String tagName)
String	**getEMVData** (String tagName)
String[]	**getAID** (int type)
String[]	**getRID** ()
int	**getProcessCardType** ()

void	**registerKernelAID** (in Map customAidList)
void	**inputOnlineResult** (in Bundle onlineResult, **OnlineResultHandler** handler)
boolean	**updateVisaAPID** (int operation, in **DRLData** drlData)
boolean	**updateCardBlk** (int operation, in **BLKData** blkData, int type)
int	**emvProcessingRequestOnline** ()
String[]	**getCAPK** (int type)
void	**enableTrack** (int trkNum)
boolean	**setCtlsPreProcess** (in Bundle param)
void	**checkCardMs** (in Bundle cardOption, long timeout, **CheckCardListener** listener)

Member Function Documentation:

checkCard()

The `checkCard()` method is used to detect and read the card before the actual EMV transaction can proceed. This step is performed when the user inserts, swipes, or taps the card on a reader.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.checkCard (in Bundle cardOption,
int timeout, CheckCardListener listener)
```

Parameters

`cardOption`	Indicates the types of cards that the reader supports. It includes: supportMagCard(boolean) — Indicates whether the card reader supports magnetic cards. supportSmartCard(boolean) — Indicates whether the card reader supports smart cards. supportCTLSCard(boolean) — Indicates whether the card reader supports CTLS cards.
`timeout`	The timeout duration in seconds.
`listener`	A callback listener that gets triggered when a card is detected, refer to `CheckCardListener`.

Return Values

`void`

See Also

- `stopCheckCard()`
- `startEMV()`
- Refer to `CheckCardListener` interface under Section 2.2.1.

stopCheckCard()

This method is used to stop any ongoing card checking processes, such as reading card data or verifying card details.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.stopCheckCard ()
```

Parameters

None.

Return Values

```
void
```

See Also

- `checkCard()`
- `startEMV()`
- Refer to `CheckCardListener` interface under Section 2.2.1.

startEMV()

The `startEMV()` method is used to initiate and manage an EMV transaction after the card has been detected (inserted, tapped, or swiped). It handles the transaction phases such as authorization, PIN entry, and other critical transaction-related activities.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.startEMV (int processType,
in Bundle intent, EMVHandler handler)
```

Parameters

`processType`	Specifies the type of EMV process to be initiated.		
	1:		EMV processing.
	2:		EMV simplified processing.
`intent`	A bundle that holds transaction settings.		
	cardType(int)		Specifies the card type.
		CARD_INSERT(0)	For smart IC card.
		CARD_RF(1)	For CTLS card.

crdDetectTimeOut(long)	Indicates the maximum time allowed for a CTLS transaction to complete.
transProcessCode(byte)	A 1Byte code translation type (9C first two digits of the ISO 8583:1987 Processing Code).
authAmount(long)	The transaction amount that requires authorization.
isSupportSM(boolean)	Indicates support for the Secure Messaging (SM).
isForceOnline(boolean)	If true, the transaction will be processed online.
merchantName(String)	The name of the merchant (variable bytes).
merchantId(String)	The merchant's ID (15 bytes).
terminalId(String)	The terminal's ID (8 bytes).
transCurrCode(String)	Currency code (5F2A). If not set, the kernel will attempt to find it in the AID string.
otherAmount(String)	Sets the value for the Other Amount (9F03).
panConfirmTimeOut(int)	Sets the timeout for PAN confirmation (default is 60 seconds, applicable only for smart cards).
appSelectTimeOut(int)	Sets the timeout for application selection (default is 60 seconds, applicable only for smart cards).
traceNo(String)	The trace number (variable bytes).

ctlsPriority(byte)	CTLS application priority (optional). For example, b0 for MyDebit; b1 to b7 are to be defined. Set this value as 0xFF for setting from application.
isForceOffline(boolean)	Indicates if the transaction should be forced offline. Default is false (only for AMEX kernel).
transSeqCounter(int)	Transaction sequence counter (9F41). Valid range: 0 <= transSeqCounter <= 99999999.

`handler`	The callback handler manages events during the transaction process, refer to `EMVHandler`.

Return Values

`void`

See Also

- `startEMV()`
- `abortEMV()`

abortEMV()

This method is used to terminate any ongoing EMV transaction process.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.abortEMV ()
```

Parameters

None.

Return Values

```
void
```

updateAID()

This method is used to update the AID for payment applications in a payment processing system. This method allows user to perform three main operations (like add, remove, and clear all AIDs) related to the AIDs.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IEMV.updateAID (int operation,
int aidType, String aid)
```

Parameters

`operation` Specifies the type of operation to perform on the AID.

1: (append) Adds a new AID to the existing list of applications.

2: (remove) Removes a specific AID from the list.

3: (clear all) Clears all AIDs, resetting the application settings.

`aidType` Indicates the type of AID being managed.

1: (contact) Refers to a traditional smart card application that requires physical contact with the reader.

2: (contactless) Refers to applications that utilize contactless technology, allowing transactions through NFC or Radio Frequency Identification (RFID).

`aid` The actual AID to be added, removed, or cleared. This identifier is crucial for selecting the appropriate payment application during transactions.

Return Values

A Boolean value:

true: Indicates that the operation was successful.

false: Indicates that the operation failed.

See Also

```
getAID()
```

updateRID()

This method is used to update a Record Identifier (RID) associated with a Certificate Authority (CA) public key in a payment processing system. This method allows user to perform three main operations (like add, remove, and clear all RIDs) related to the RIDs.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IEMV.updateRID (int operation,
String rid)
```

Parameters

`operation` Specifies the type of operation to perform on the RID.

1: (append) Add a new RID associated with a new CA public key.

2: (remove) Remove an existing RID for a CA public key.

3: (clear all) Clears all RIDs, resetting the application settings.

`rid` The actual RID that corresponds to the CA public key being updated.

Return Values

A Boolean value:

true: Indicates that the operation was successful.

false: Indicates that the operation failed.

See Also

`getRID()`

importAmount()

This method is used to import a specified amount for processing within an electronic mobile payment system.

NOTE	This method has been deprecated.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.importAmount (long amount)
```

Parameters

`amount`	Indicates a monetary value to be imported, specified in the smallest currency unit.

Return Values

`void`

See Also

`startEMV()`

importAppSelect()

This method is used to select a specific application from a multi-application card. It allows the system to determine which payment application to use for processing a transaction based on the provided index.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.importAppSelection (int index)
```

Parameters

`index`	The index of the application to select.	
	Start from 1:	This indicates the first application in the list.
	0:	Indicates a cancel action.

Return Values

```
void
```

See Also

Refer to `EMVHandler.onSelectApplication()` method under Section 2.2.8.

importPin()

This method is used to import a PIN for transaction authentication in electronic mobile payments.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.importPin (int option,
in byte[] pin)
```

Parameters

`option` Specifies the action to be taken regarding the PIN.

CANCEL (0):	Indicates that the PIN entry should be cancelled.
CONFIRM (1):	Indicates that the PIN entry should be confirmed and processed.

`pin` An array of bytes that indicates the PIN data.

Return Values

`void`

See Also

Refer to `EMVHandler.onRequestInputPIN()` method under Section 2.2.8.

importCertConfirmResult()

This method is part of the EMV transaction flow, specifically related to the process of certificate validation. Certificates are used to ensure the authenticity of the card, issuer, or transaction during the payment process. This method confirms the result of certificate validation, signalling whether the certificate verification was successful or not.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.importCertConfirmResult (int
option)
```

Parameters

`option` Indicates the result of the cardholder verification process.

CANCEL (0):	Indicates that the verification process was cancelled (bypassed).
CONFIRM (1):	Indicates that the verification was successful and confirmed.

NOTMATCH (2): Indicates that the entered information does not match the cardholder's data.

Return Values

```
void
```

See Also

Refer to `EMVHandler.onConfirmCertInfo()` method under Section 2.2.8.

importCardConfirmResult()

This method is part of the EMV transaction flow and is used to confirm the result of the cardholder verification process. This step is critical in ensuring that the person using the card is authorized to complete the transaction, through methods such as PIN entry, signature, or biometric authentication.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.importCardConfirmResult (boolean
pass)
```

Parameters

`pass`	Indicates the outcome of the cardholder verification.

true:	The cardholder verification was successful.
false:	The cardholder verification failed.

Return Values

```
void
```

See Also

Refer to `EMVHandler.onConfirmCardInfo()` method under Section 2.2.8.

importOnlineResult()

This method is used to import and process the response received from an online transaction in electronic mobile payment systems, allowing the system to take appropriate actions based on the result.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.importOnlineResult (in Bundle
onlineResult, OnlineResultHandler handler)
```

Parameters

`onlineResult` A Bundle that contains the results of the online operation. It includes:

isOnline(boolean) Indicates whether the transaction was processed online.

respCode(String) The response code returned from the online operation, indicating the status of the transaction.

authCode(String) The authorization code provided for the transaction, if applicable.

field55(String) The response data for field 55, which may contain additional transaction details or messages.

`handler` This is an instance of an interface intended to manage the results of the online transaction. Refer to `OnlineResultHandler` under Section 2.2.7.

Return Values

`void`

See Also

Refer to `EMVHandler.onRequestOnlineProcess()` method under Section 2.2.8.

setEMVData()

This method is essential for setting or modifying EMV kernel data during the transaction process for Dynamic Currency Conversion (DCC). It is especially applicable in the EMV flow during callbacks such as `onConfirmCardInfo()` or `onRequestInputPIN()`, and it specifically supports smart card transactions. For example:

- Initially, user might set a currency code (e.g., 5F2A=0156). If user need to change this during the `onConfirmCardInfo()` callback (e.g., to 5F2A=0116), user can use this method to update the tag.

- User may start with an initial authorization amount (e.g., `authAmount`=100 using tag 9F02). During the `onConfirmCardInfo()` callback, user can modify this amount as needed.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.setEMVData (in List< String >
tlvList)
```

Parameters

`tlvList`	A list of strings formatted in Tag-Length-Value (TLV) structure that represents the EMV data user wants to set or modify. Each entry must comply with the TLV format for proper processing.

Return Values

```
void
```

getAppTLVList()

This method is used to retrieve kernel data in the TLV format from the PBOC kernel, specifically for the EMV applications.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IEMV.getAppTLVList (in String[]
taglist)
```

Code Snippet

```
{
    String[] strlist = {"9F33", "9F40", "9F10", "9F26", "95", "9F37", "9F1E",
"9F36",
            "82", "9F1A", "9A", "9B", "50", "84", "5F2A", "8F"};

    String strs = iemv.getAppTLVList(strlist);
 }
```

Parameters

`taglist`	An array of strings representing the specific tags user wants to query.

Return Values

Returns either of the two values:

TLV Format: The method returns data in the TLV format.

Null: Indicating that there is no response available.

getCardData()

This method is called to retrieve the EMV card data based on a specified tag identifier.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IEMV.getCardData (String tagName)
```

Parameters

`tagName`	The name of the tag for which the data is requested.

Return Values

The EMV data associated with the specified tag name.

getEMVData()

This method is used to retrieve the specific EMV data, such as card number, validity date, card serial number, and other relevant details.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IEMV.getEMVData (String tagName)
```

Parameters

`tagName` The name of the tag for which the EMV data is requested. It includes:

PAN card No.	PAN card number.
TRACK2	Track data from Track 2 of the magnetic stripe.
CARD_SN	Card Serial Number.
EXPIRED_DATE	Expiration date of the card.
DATE	Current date.
TIME	Current time.
BALANCE	Current balance.
CURRENCY	Currency code associated with the card.

Return Values

A string value representing the requested EMV data. If the specified tag is not available at the current EMV processing stage or if there is an error retrieving the data, the method will return null.

See Also

`getCardData()`

getAID()

This method is used to retrieve the AID based on a specified application type during the EMV transactions.

Prototype

```
String[] com.vfi.smartpos.deviceservice.aidl.IEMV.getAID (int type)
```

Code Snippet

```
   @brief get the AID parameter

   @param type - 1-contact aid 2-contactless aid
   @return null if the AID is unavailable
   \code
  demo returns from getAID(1)
{"9F0607A0000000031010DF0101009F09020140DF1105C000000000DF1205000000000DF13050
0000000009F1B0400000000DF150400000009F7B06000000000000DF1906000000000000DF2006
0099999999995F2A0201569F1A0201569F3303E0F9C89F4005FF00F0A0019F66042600000809F350
122DF150400000000DF160101DF170101DF14039F3704DF1801009F1D00",
"9F0607A0000000032010DF0101009F09020140DF1105D84004A800DF1205D84000F800DF130500
100000009F1B0400000000DF150400000009F7B06000000000000DF1906000000000000DF20060
099999999995F2A0201569F1A0201569F3303E0F9C89F4005FF00F0A0019F66042600000809F3501
22DF150400000000DF160101DF170101DF14039F3704DF1801009F1D00"
```

Parameters

`type`	The type of application for which the AID is requested.

1: Contact AID for EMV cards that require physical contact.

2: Contactless AID for EMV cards that support contactless transactions.

Return Values

A string representing the AID of the application.

See Also

```
updateAID()
```

getRID()

This method is used to retrieve the RID associated with the EMV transaction.

> NOTE
>
> This interface method has been deprecated.

Prototype

```
String[] com.vfi.smartpos.deviceservice.aidl.IEMV.getRID ()
```

Parameters

None.

Return Values

Returns either of the two values:

RID: The method returns the available RID.

Null: Indicating that the RID is unavailable.

See Also

```
getCAPK()
```

getProcessCardType()

This method is used to obtain the CTLS card type (In `onRequestOnlineProcess` callback you can use this interface to obtain the CTLS card type) during an EMV transaction.

NOTE

This interface method has been deprecated.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IEMV.getProcessCardType ()
```

Parameters

None.

Return Values

The type of card currently being processed during an EMV transaction.

0:	No Type	Indicates an unknown or unsupported card type.
1:	JCB_CHIP	Represents a JCB chip card.
2:	JCB_MSD	Represents a JCB magnetic stripe card.
3:	JCB_Legcy	Represents a legacy JCB card.
4:	VISA_w1	Represents a Visa card using the w1 processing.
5:	VISA_w3	Represents a Visa card using the w3 processing.
6:	Master_EMV	Represents a MasterCard EMV chip card.
7:	Master_MSD	Represents a MasterCard magnetic stripe card.
8:	qPBOC_qUICS.	Represents a card using qPBOC/qUICS standards.

See Also

Refer to `EMVHandler.onRequestOnlineProcess()` method under Section 2.2.8.

registerKernelAID()

This method is used to set custom AID for the EMV kernel before initiating an EMV transaction with `startEMV()`.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.registerKernelAID (in Map
customAidList)
```

Parameters

`customAidList`	An Integer representing the kernel ID associated with that AID, refer to `CTLSKernelID` for valid ID's.

Return Values

`void`

See Also

Refer to `CTLSKernelID` class under Section 2.3.9.

inputOnlineResult()

This method allows a payment application to import and process the online response related to a PIN input operation.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.inputOnlineResult (in Bundle
onlineResult, OnlineResultHandler handler)
```

Parameters

`onlineResult`	A Bundle that contains the results of the online operation. It includes:

isOnline(boolean)	Indicates whether the transaction was processed online.
respCode(String)	The response code returned from the online operation, indicating the status of the transaction.
authCode(String)	The authorization code provided for the transaction, if applicable.
field55(String)	The response data for field 55, which may contain additional transaction details or messages.

`handler`	An instance of `OnlineResultHandler` that processes the results of the online operation. Refer to `OnlineResultHandler`.

Return Values

```
void
```

See Also

Refer to `EMVHandler.onRequestOnlineProcess()` method under Section 2.2.8.

updateVisaAPID()

This method is used to perform operations related to Visa transactions. It updates or manages data associated with a Visa transaction based on the provided parameters.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IEMV.updateVisaAPID (int operation,
in DRLData drlData)
```

Parameters

`operation`	Specifies the type of operation to perform on the Visa transaction data: 1: (append) This operation adds new data to the existing transaction information. 2: (clear) This operation clears or resets the existing data associated with the transaction.
`DRLData`	Represents an instance of the `DRLData` class, which contains all necessary information related to the Visa transaction.

Return Values

A Boolean value:

true: if the operation was successful.

false: if the operation failed.

See Also

Refer to `DRLData` class under Appendix A Supporting Classes 3.

updateCardBlk()

This method is used to manage and update data related to card blocking operations in a payment processing system.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IEMV.updateCardBlk (int operation,
in BLKData blkData, int type)
```

Parameters

`operation`	Specifies the action to perform:

1: (append) Adds new blocking data to the existing card information.

2: (clear) Removes any existing blocking data associated with the card.

`blkData` An instance of the `BLKData` class that contains the data related to the card blocking operation.

`type` Indicates the type of card being blocked:

1: contact (smart card) for traditional smart cards that require physical contact for transactions.

2: contactless for cards that support contactless transactions, allowing for quicker payments without physical contact.

Return Values

A Boolean value:

true: if the operation was successful.

false: if the operation failed.

See Also

Refer to `BLKData` class under Appendix A Supporting Classes 1.

emvProcessingRequestOnline()

This method is used to initiate an online transaction request for a smart card immediately after the cardholder has selected a payment application on the terminal.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IEMV.emvProcessingRequestOnline ()
```

Parameters

None.

Return Values

Returns either of the two values:

0: Indicates success, meaning the online request was processed successfully.

other values (non-zero): Indicates failure, which may represent various error conditions (e.g., communication errors, authorization issues).

getCAPK()

This method is used to retrieve the Certificate Authority Public Key (CAPK) corresponding to a specified type, which is essential for secure EMV transaction processing.

Prototype

```
String[] com.vfi.smartpos.deviceservice.aidl.IEMV.getCAPK (int type)
```

Parameters

`type` Specifies the type of CAPK to retrieve:

1: Contact AID for EMV cards that require physical contact.

2: Contactless AID for EMV cards that support contactless transactions.

Return Values

Returns either of the two values:

CAPK: The method returns the available CAPK.

Null: Indicating that the CAPK is unavailable.

enableTrack()

This method is used to enable a specific track of card data during transaction processing, although this may not be strictly necessary for all operations.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.enableTrack (int trkNum)
```

Parameters

`trkNum` This parameter uses bitwise flags to determine which tracks to enable:

bit0:	If set to 1, enables Track 1.
bit1:	If set to 1, enables Track 2.
bit2:	If set to 1, enables Track 3.

Return Values

```
void
```

setCtlsPreProcess()

This method is used to configure parameters for the CTLS transaction pre-processing, preparing the terminal for contactless interactions.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IEMV.setCtlsPreProcess (in Bundle
param)
```

Parameters

`param` A Bundle including various settings to configure contactless transaction processing.

traceNo(String)	The trace number for the transaction (variable-length bytes).
transProcessCode(byte)	Specifies the transaction type as a single byte, specifically the first two digits of the ISO 8583:1987 Processing Code (e.g., 9C).
transCurrCode(String)	Currency code (e.g., 5F2A). If not set, the kernel finds this tag in the AID string.
otherAmount(String)	Sets the Other Amount value (tag 9F03), which may represent additional fees.
authAmount(long)	The authorization amount for the transaction.
isForceOnline(boolean)	Indicates if the transaction should be forced online for authorization.
ctlsPriority(byte)	CTLS application priority (optional); b0 for MyDebit, b1 to b7 for other definitions.
sForceOffline(boolean)	Indicates if the transaction should be forced offline. Default is false.

Return Values

A Boolean value:

true: Indicates that the parameters for contactless pre-processing were successfully set.

false: Indicates a failure in setting the parameters.

checkCardMs()

This method is used to initiate a non-blocking check for a magnetic stripe card, allowing the terminal to detect card data while keeping the application responsive.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IEMV.checkCardMs (in Bundle cardOption,
long timeout, CheckCardListener listener)
```

Parameters

`cardOption` A Bundle that specifies the card types that the terminal will check for during the magnetic stripe card detection process.

supportMagCard(boolean)	Indicates supports for magnetic stripe cards.
supportSmartCard(boolean)	Indicates support for smart cards.
supportCTLSCard(boolean)	Indicates support for CTLS cards.

`timeout` The timeout duration in seconds.

`listener` A callback interface that handles events related to card detection. Refer to `CheckCardListener`.

Return Values

`void`

See Also

- `stopCheckCard()`
- `startEMV()`
- Refer to `CheckCardListener` interface under Section 2.2.1.

2.2.11 ISmartCardReader

Package: com.vfi.smartpos.deviceservice.aidl.ISmartCardReader

Overview:

This interface is used for interacting with smart cards, including contact cards or IC cards. This interface provides methods to facilitate communication between applications and smart cards, enabling various operations such as reading and writing data.

Public Member Functions:

Modifier and Type	Method
boolean	**powerup** ()
boolean	**powerDown** ()
boolean	**isCardIn** ()
byte[]	**exchangeApdu** (in byte[] apdu)
boolean	**isPSAMCardExists** ()
byte	**checkCardStatus** ()
byte[]	**getPowerUpATR** ()

Member Function Documentation:

powerUp()

This method is used to power up a smart card reader that supports various smart card technologies, including contact and contactless cards. This method is part of an interface that handles a broader range of smart card operations, ensuring that the reader is initialized and ready to interact with the card once it's powered up.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.ISmartCardReader.powerUp ()
```

Parameters

None.

Return Values

A Boolean value:

true: Indicates that the card reader has been successfully powered up and is ready for use.

false: Indicates a failure to power up the card reader.

See Also

`powerDown()`

powerDown()

This method is called to power off the smart card reader, ensuring it is no longer active and conserving battery life or reducing wear on the device.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.ISmartCardReader.powerDown ()
```

Parameters

None.

Return Values

A Boolean value:

true: Indicates that the card reader has been successfully powered down.

false: Indicates a failure to power down the card reader.

See Also

```
powerup()
```

isCardIn()

This method is used to check whether a smart card (contact card or IC card) is currently inserted into the card reader.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.ISmartCardReader.isCardIn ()
```

Parameters

None.

Return Values

A Boolean value:

true: Indicates that a card is currently available.

false: Indicates that a card is unavailable.

exchangeApdu()

This method is fundamental for APDU's data communication between an application and a smart card. This method enables the sending of commands to the card and receiving its responses, allowing for various interactions

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.ISmartCardReader.exchangeApdu (in
byte[] apdu)
```

Parameters

`apdu`	The APDU command to be sent to the smart card, structured according to the command requirements.

Return Values

A byte array containing the response from the smart card.

Valid Response: A non-null byte array indicating the data returned by the card.

Null: Indicates that no response was received from the smart card.

isPSAMCardExists()

This method is used to check whether a Payment System Access Module (PSAM) card is currently present in the card reader.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.ISmartCardReader.isPSAMCardExists ()
```

Parameters

None.

Return Values

A Boolean value:

true: Indicates that the PSAM card is currently in place and detected by the card reader.

false: Indicates that the PSAM card is not in place or not detected.

checkCardStatus()

This method is called to check whether the smart card is ready for operations, powered down, or in an error state, allowing the application to take appropriate actions based on the card's status.

Prototype

```
byte com.vfi.smartpos.deviceservice.aidl.ISmartCardReader.checkCardStatus ()
```

Parameters

None.

Return Values

An integer value representing the card's status:

0x00:	Card does not exist (not detected in the reader).
0x01:	Card exists (detected and ready for operations).
0x02:	Card is powered on (active and ready for communication).

getPowerUpATR()

This method is used to obtain the ATR (Answer to Reset) after powering on the smart card, which is essential for understanding how to communicate with the card.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.ISmartCardReader.getPowerUpATR ()
```

Parameters

None.

Return Values

A byte array representing the ATR data of the smart card.

2.2.12 IScanner

Package: com.vfi.smartpos.deviceservice.aidl.IScanner

Overview:

The `IScanner` interface is intended for managing the operations of barcode and QR code scanners, allowing applications to initiate scans, receive results, and handle scanning events seamlessly.

Public Member Functions:

Modifier and Type	Method
void	**startScan** (in Bundle param, long timeout, **ScannerListener** listener)
void	**stopScan** ()

Member Function Documentation:

startScan()

This method is called to start a scanning operation with specified parameters and process results via a listener.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IScanner.startScan (in Bundle param,
long timeout, ScannerListener listener)
```

Parameters

`param` A collection of parameters that define how the scanning should be performed. This can include:

topTitleString(String)	This parameter sets the title displayed at the top of the scanning interface. The text alignment is centered.
upPromptString(String)	This string is used as a prompt displayed above the scan box, providing guidance to the user. The text alignment is centered.

downPromptString(String)		This prompt appears below the scan box, further assisting the user during the scanning process. The text alignment is centered.
showScannerBorder(boolean)		Controls the visibility of the scanner border and the prompt strings. Default Value: true (shows the border and prompts). When false: Hides the scanner border and both the `upPromptString` and `downPromptString`.
scannerSelect(int)		Selects the scanner position.
	0:	Use the front scanner.
	1:	Use the rear scanner.
	Default:	If this parameter is not provided, the method will default to using the scanner position specified by the `IDeviceService.getScanner()` method

`timeout`	Sets the maximum duration in milliseconds that the scanner will attempt to scan before timing out.
`listener`	A callback listener that receives updates and results from the scanning operation. This typically includes methods for successful scans, errors, and timeouts. Refer to `ScannerListener` under Section 2.2.3.

Return Values

```
void
```

stopScan()

This method is used to stop the current scanning session, regardless of whether a barcode or QR code has been successfully detected.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IScanner.stopScan ()
```

Parameters

None.

Return Values

```
void
```

2.2.13 IPrinter

Package: com.vfi.smartpos.deviceservice.aidl.IPrinter

Overview:

This interface enables the payment applications to interact effectively with printer services on a device. It facilitates printing tasks, allowing users to print different formats such as strings, barcodes, QR codes, and images. It offers a clear set of methods for managing print jobs, checking printer status, and handling printing processes.

Public Member Functions:

Modifier and Type	Method
int	**getStatus** ()

void	**setGray** (int gray)
void	**addText** (in Bundle format, String text)
void	**addTextInLine** (in Bundle format, String lString, String mString, String rString, int mode)
void	**addBarCode** (in Bundle format, in String barcode)
void	**addQrCode** (in Bundle format, String qrCode)
void	**addQrCodesInLine** (in List< **QrCodeContent** > qrCodes)
void	**addImage** (in Bundle format, in byte[] imageData)
void	**feedline** (int lines)
void	**startPrint** (**PrinterListener** listener)
void	**startSaveCachePrint** (**PrinterListener** listener)
void	**setLineSpace** (int space)
void	**startPrintInEmv** (**PrinterListener** listener)
int	**cleanCache** ()
void	**addBmpImage** (in Bundle format, in Bitmap image)

Member Function Documentation:

getStatus()

This method is called to retrieve the current status of the printer.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IPrinter.getStatus()
```

Parameters

None.

Return Values

Returns the operational status of the printer.

Error Type	Error Code	Description
ERROR_NONE	(0x00)	Normal operation.
ERROR_PAPERENDED	(0xF0)	Out of paper.
ERROR_NOCONTENT	(0xF1)	No content to print.
ERROR_HARDERR	(0xF2)	Printer hardware error.
ERROR_OVERHEAT	(0xF3)	Printer overheating.
ERROR_NOBM	(0xF6)	No black mark detected.
ERROR_BUSY	(0xF7)	Printer is busy.
ERROR_MOTORERR	(0xFB)	Motor malfunction.
ERROR_LOWVOL	(0xE1)	Low battery voltage.
ERROR_NOTTF	(0xE2)	No TTF available.
ERROR_BITMAP_TOOWIDE	(0xE3)	Bitmap width exceeds limit.

setGray()

This method is called to set the gray level for printing.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.setGray(int gray)
```

Parameters

`gray`	An integer indicating the desired level of gray, from 0 to 7.

Return Values

```
void
```

addText()

This method adds a specified text string to the print queue for printing. It allows for flexible formatting options through a Bundle that defines the text's appearance (such as font size, style, alignment, etc.).

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.addText (in Bundle format,
String text)
```

Parameters

`format` A Bundle containing various settings for customizing the text:

font(int)	Specifies the font size:
0:	Small (size 16).
1:	Normal (size 24).

	2:	Normal_bigger (size 24, double height and bold).
	3:	Large (size 32).
	4:	Large_bigger (size 32, double height and bold).
	5:	Huge (size 48).
	6:	Normal_wide (size 24, double width and bold).
	7:	Large_wide (size 32, double width and bold).
fontStyle(String)		Specifies the font style:
Custom font		Specify an absolute path to a custom font by user (e.g., /xxxx/xx.ttf).
align(int)		Text alignment:
	0:	Left.
	1:	Center.
	2:	Right.
bold(boolean)		Text to print:
	true:	Bold.
	false:	Normal.
newline(boolean)		A new line is added after printing the text:
	true:	New line after print.
	false:	Normal.
scale_w(float)		Multiple Width scaling.
scale_h(float)		Multiple Height scaling.

`text` Indicates the content is going to be printed on the receipt.

Return Values

```
void
```

See Also

```
addTextInLine()
```

addTextInLine()

This method enables the user to print three separate text strings in a single line, allowing for better layout control in receipts.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.addTextInLine (in Bundle
format, String lString, String mString, String rString, int mode)
```

Parameters

`format` A Bundle that contains key-value pairs for text formatting settings:

fontSize(int)		Specifies the size of the font:
	0:	Small (size 16).
	1:	Normal (size 24).
	2:	Normal_bigger (size 24, double height and bold).
	3:	Large (size 32).
bold(boolean)		Text to print:
	true:	Bold.
	false:	Normal.

fontStyle(String)	Specifies the font style:
Chinese (Android font)	Use the default system font for Chinese characters.
English (Android font)	Use the default system font for English text.
Arabic (Android font)	Use the default system font for Arabic text.
Custom font	Specify an absolute path to a custom font by user (e.g., /xxxx/xx.ttf).

`lString` The string to be printed on the left side, justified to the left.

`mString` The string to be printed in the middle or center.

`rString` The string to be printed on the right side, justified to the right.

`mode` An integer specifies the formatting behaviour for the text:

0:	The left and right justified text divide the width equally.
1:	The left and right justified text divide the width flexible.

Return Values

`void`

See Also

`addText()`

addBarCode()

This method is called to generate and print a barcode (CodeType Code128) based on a given string.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.addBarCode (in Bundle format,
in String barcode)
```

Code Snippet

```
public enum BarcodeFormat {
    AZTEC,
    CODABAR,
    CODE_39,
    CODE_93,
    CODE_128,
    DATA_MATRIX,
    EAN_8,
    EAN_13,
    ITF,
    MAXICODE,
    PDF_417,
    QR_CODE,
    RSS_14,
    RSS_EXPANDED,
    UPC_A,
    UPC_E,
    UPC_EAN_EXTENSION;

    private BarcodeFormat() {
    }
}
```

Parameters

`format` A Bundle object that contains key-value pairs specifying the settings for generating the barcode:

align(int)	Specifies the alignment of the barcode:
0:	Left.

1:	Center.
2:	Right.
height(int)	The height of the barcode in units.
barCodeType(int)	Specifies the type of barcode to be printed: The default barcode type is "BarcodeFormat.CODE_128.ordinal()", which corresponds to the Code 128 barcode format.

`barcode` A string that represents the data encoded in the barcode.

Return Values

```
void
```

addQrCode()

This method is called to enable printing of QR codes with customizable options.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.addQrCode (in Bundle format,
String qrCode)
```

Parameters

`format` A Bundle object that specifies the print format, which includes settings for the positioning and appearance of the QR code:

offset(int)	Specifies the horizontal offset from the left edge of the print area. This allows for positioning the QR code at a specific location.
expectedHeight(int)	The expected height and width of the QR code. It should be a multiple of the minimum

	pixel size required for QR codes to ensure proper readability.
`qrCode`	The specific data to be encoded in the QR code.

Return Values

```
void
```

addQrCodesInLine()

This method is used to print multiple QR codes in a single line, allowing for efficient layout and space management.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.addQrCodesInLine (in List<
QrCodeContent > qrCodes)
```

Parameters

`qrCodes`	A list of QrCodeContent objects, where each object represents the data to be encoded in an individual QR code.

Return Values

```
void
```

addImage()

This method is called to add an image to the print queue, allowing for customization of the image's printing parameters.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.addImage (in Bundle format,
in byte[] imageData)
```

Code Snippet

```
@brief Add an image to print
  @param format - the format setting
  <ul>
  <li>offset(int) - the offset from left</li>
  <li>width(int) - the width of the image want to print.(MAX = 384)</li>
  <li>height(int) - the height want to print</li>
  <li>gray(int) - set pixcel gray to pint (0~255 default = 128) </li>
  </ul>
  @param imageData - the image buffer
  \en_e
  <p>
  \code{.java}
     // get image buffer from id
    private byte[] getBitmapByte(int id) {
         BitmapFactory.Options bfoOptions = new BitmapFactory.Options();
         bfoOptions.inScaled = false;
         ByteArrayOutputStream out = new ByteArrayOutputStream();
         BitmapFactory.decodeResource(context.getResources(), id,
bfoOptions).compress(Bitmap.CompressFormat.JPEG, 100, out);
         try {
             out.flush();
             out.close();
         } catch (IOException e) {
             e.printStackTrace();
         }
         return out.toByteArray();
     }
  \endcode
  \code{.java}

// get image buffer from file public byte[] image2byte(String path) { byte[]
data = null; FileInputStream input = null; try { input = new FileInputStream(new
File(path)); ByteArrayOutputStream output = new ByteArrayOutputStream(); byte[]
buf = new byte[1024]; int numBytesRead = 0; while ((numBytesRead =
input.read(buf)) != -1) { output.write(buf, 0, numBytesRead); } data =
output.toByteArray(); output.close(); input.close(); } catch
(FileNotFoundException ex1) { ex1.printStackTrace(); } catch (IOException ex1) {
ex1.printStackTrace(); } return data; }
```

Parameters

`format`	A Bundle object that contains formatting options for the image (Bitmap.CompressFormat.JPEG).

`imageData`	A byte array containing the actual image data to be printed.

Return Values

```
void
```

feedLine()

This method is used to command a printer to advance the paper by a specified number of lines, allowing for appropriate spacing during printing.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.feedLine (int lines)
```

Parameters

`lines` The number of lines to feed the paper. The input value must satisfy the following outcomes:

Must be greater than 1 (`lines` > 1).

Must be less than or equal to 50 (`lines` <= 50).

The actual number of lines advanced will be `lines` + 1, as the current line being printed is also counted.

Return Values

```
void
```

startPrint()

This method is called to initiate the printing operation. It allows for monitoring the printing status through a listener.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.startPrint (PrinterListener
listener)
```

Parameters

`listener`	A callback interface to notify the result of the print operation. Refer to `PrinterListener`.

Return Values

```
void
```

startSaveCachePrint()

This method is called to start the print process while retaining the cached data for future use.

NOTE

This method has been deprecated. Refer to `IPrinter.startPrint()` instead.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.startSaveCachePrint
(PrinterListener listener)
```

Parameters

`listener`	A callback interface to notify the result of the print operation. Refer to `PrinterListener`.

Return Values

```
void
```

See Also

```
IPrinter.startPrint()
```

setLineSpace()

This method is called to set the line spacing for printed text.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.setLineSpace (int space)
```

Parameters

`space`	An integer that specifies the line spacing value, which must be within the range of 0 to 50.

Return Values

```
void
```

startPrintInEmv()

This method is called to start the printing operations during an EMV transaction process. It facilitates printing tasks that are part of the EMV flow without terminating the ongoing transaction.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.startPrintInEmv
(PrinterListener listener)
```

Parameters

`listener`	A callback interface to notify the outcome of the EMV print operation. The listener will provide feedback on whether the printing was successful or if an error occurred during the process. Refer to `PrinterListener`.

Return Values

```
void
```

cleanCache()

This method is called to clear the printer's cache.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IPrinter.cleanCache ()
```

Parameters

None.

Return Values

The status of clearing the printer’s cached data:

1: Succeeded - Indicates that the cache was successfully cleared.

0: Failed - Indicates that the operation to clear the cache was unsuccessful.

addBmpImage()

This method is called to add a bitmap image in BMP format to the printer's cache for subsequent printing.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPrinter.addBmpImage (in Bundle format,
in Bitmap image)
```

Parameters

`format` A Bundle object that contains key-value pairs indicating the format and settings of the image.

offset(int)	The horizontal offset from the left edge where the image will be printed.
width(int)	The width of the image to be printed, with a maximum value of 384 pixels.
height(int)	The height of the image to be printed.
gray(int)	Sets the pixel gray level for the image, ranging from 0 to 255 with a default value of 128.

`imageData` The BMP image data that will be added to the printer's cache for printing.

Return Values

```
void
```

2.2.14 IPinpad

Package: com.vfi.smartpos.deviceservice.aidl.IPinpad

Overview:

This interface is used for communicating with Pinpad devices in payment processing applications. It provides essential methods for key management, PIN input processing, and data encryption/decryption for Pinpad.

Public Member Functions:

Modifier and Type	Method
boolean	**isKeyExist** (int keyType, int keyId)
boolean	**loadTEK** (int keyId, in byte[] key, in byte[] checkValue)
boolean	**loadTEKWithAlgorithmType** (int keyId, in byte[] key, in byte algorithmType, in byte[] checkValue)
boolean	**loadEncryptMainKey** (int keyId, in byte[] key, in byte[] checkValue)
boolean	**loadEncryptMainKeyWithAlgorithmType** (int keyId, in byte[] key, int algorithmType, in byte[] checkValue)
boolean	**loadMainKey** (int keyId, in byte[] key, in byte[] checkValue)
boolean	**loadMainKeyWithAlgorithmType** (int keyId, in byte[] key, int algorithmType, in byte[] checkValue)
boolean	**loadDukptKey** (int keyId, in byte[] ksn, in byte[] key, in byte[] checkValue)
boolean	**loadWorkKey** (int keyType, int mkId, int wkId, in byte[] key, in byte[] checkValue)
boolean	**loadWorkKeyWithDecryptType** (int keyType, int mkId, int wkId, int decKeyType, in byte[] key, in byte[] checkValue)
byte[]	**calcMAC** (int keyId, in byte[] data)
byte[]	**calcMACWithCalType** (int keyId, int type, in byte[] CBCInitVec, in byte[] data, int desType, boolean dukptRequest)
byte[]	**encryptTrackData** (int mode, int keyId, in byte[] trkData)

byte[]	**encryptTrackDataWithAlgorithmType** (int mode, int keyId, int algorithmType, in byte[] trkData, boolean dukptRequest)
void	**startPinInput** (int keyId, in Bundle param, in Bundle globleParam, **PinInputListener** listener)
void	**submitPinInput** ()
void	**stopPinInput** ()
String	**getLastError** ()
byte[]	**colculateData** (int mode, int desType, in byte[] key, in byte[] data)
byte[]	**dukptEncryptData** (int destype, int algorithm, int keyid, in byte[] data, in byte[] CBCInitVec)
boolean	**savePlainKey** (int keyType, int keyId, in byte[] key)
byte[]	**getDukptKsn** ()
Bundle	**generateSM2KeyPair** ()
byte[]	**getSM3Summary** (in byte[] data)
byte[]	**getSM2Sign** (in Bundle bundle)
byte[]	**getKeyKCV** (int keyIndex, int keyType)
Map	**initPinInputCustomView** (int keyId, in Bundle param, in List< **PinKeyCoorInfo** > pinKeyInfos, **PinInputListener** listener)
void	**startPinInputCustomView** ()
void	**endPinInputCustomView** ()
byte[]	**calculateData** (int mode, int desType, in byte[] key, in byte[] data)
byte[]	**calculateDataEx** (int mode, int desType, in byte[] key, in byte[] data, in byte[] initVec)
byte[]	**encryptPinFormat0** (int pinKeyId, int desType, in byte[] cardNumber, String passwd)
byte[]	**calculateByDataKey** (int keyId, int encAlg, int encMode, int encFlag, in byte[] data, in byte[] initVec)

boolean	**loadEncryptMainKeyEX** (int keyId, in byte[] key, int algorithmType, in byte[] checkValue, in Bundle extend)
boolean	**loadWorkKeyEX** (int keyType, int mkId, int wkId, int decKeyType, in byte[] key, in byte[] checkValue, in Bundle extend)
boolean	**clearKey** (int keyId, int keyType)
boolean	**loadDukptKeyEX** (int keyId, in byte[] ksn, in byte[] key, in byte[] checkValue, in Bundle extend)
boolean	**loadTEKEX** (int keyId, in byte[] key, in byte algorithmType, in byte[] checkValue, in Bundle extend)
byte[]	**calculateByWorkKey** (int keyId, int keyType, int encAlg, int encMode, int encFlag, in byte[] data, in Bundle extend)
byte[]	**calculateByMSKey** (int keyId, int keyType, int algorithmMode, in byte[] data, in Bundle extend)

Member Function Documentation:

isKeyExist()

This method is used to check if a specific key exists in the Pinpad's key storage. It currently supports only the ECB (Electronic Codebook) key type.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.isKeyExist (int keyType,
int keyId)
```

Parameters

`keyType` Indicates the type of cryptographic key.

0: MASTER (main) key.

1: MAC key.

2: PIN (work) key.

3: TD key.

4: (SM) MASTER key.

5: (SM) MAC key.

6: (SM) PIN key.

7: (SM) TD key.

8: (AES) MASTER key.

9: (AES) MAC key.

10: (AES) PIN key.

11: (AES) TD key.

12: DUKPT key.

13: TEK.

14: (SM)TEK.

15: (AES)TEK.

`keyId` The index of the key has a range of 0 to 4 for DUKPT (Derived Unique Key Per Transaction) keys and 0 to 99 for other key types.

Return Values

A Boolean value:

true: Indicates that the specified cryptographic key exists on the device.

false: Indicates that the specified key does not exist.

loadTEK()

This method is called to load a plain Transaction Encryption Key (TEK) onto the Pinpad device. The default algorithm is set to 2, which corresponds to a 3DES plain key. The TEK is essential for encrypting the master key during transaction processing.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadTEK (int keyId,
in byte[] key, in byte[] checkValue)
```

Parameters

`keyId`	An index ranging from 0 to 99.
`key`	The cryptographic key that is encrypted using TEK.
`checkValue`	To validate the integrity and suitability of the cryptographic key.

Return Values

A Boolean value:

true: Indicates that the TEK was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadTEKWithAlgorithmType()`
- `loadEncryptMainKey()`
- `loadEncryptMainKeyWithAlgorithmType()`

loadTEKWithAlgorithmType()

This method is called to load a TEK key, which is used as the transfer key to encrypt the master key. The default algorithm type for this method is ECB mode**.**

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadTEKWithAlgorithmType
(int keyId, in byte[] key, in byte algorithmType, in byte[] checkValue)
```

Parameters

`keyId` An index ranging from 0 to 99.

`key` The cryptographic key that is encrypted using the TEK.

`algorithmType` A byte that specifies the encryption algorithm used for the TEK.

1: 3DES encrypted key.

2: 3DES plain key.

3: SM4 encrypted key.

4: SM4 plain key.

5: AES encrypted key.

6: AES plain key.

`checkValue` To validate the integrity and suitability of the cryptographic key.

Return Values

A Boolean value:

true: Indicates that the TEK was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadTEK()`
- `loadEncryptMainKey()`
- `loadEncryptMainKeyWithAlgorithmType()`

loadEncryptMainKey()

This method is called to load an encrypted master key onto the Pinpad device. The master key is encrypted using the 3DES ECB algorithm, and the decryption uses the TEK with an index of 0.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadEncryptMainKey (int
keyId, in byte[] key, in byte[] checkValue)
```

Parameters

`keyId`	An index ranging from 0 to 99.
`key`	The cryptographic key that is encrypted using TEK.
`checkValue`	To validate the integrity and suitability of the cryptographic key. The default is set to null.

Return Values

A Boolean value:

true: Indicates that the master key was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadTEK()`
- `loadTEKWithAlgorithmType()`
- `loadEncryptMainKeyWithAlgorithmType()`

- `loadMainKey loadWorkKey loadWorkKeyWithDecryptType`

loadEncryptMainKeyWithAlgorithmType()

This method is called to load a master encryption key onto the Pinpad device, specifying the encryption algorithm type to be used. The default is ECB mode, and it decrypts using the TEK with an index of 0.

Prototype

```
boolean
com.vfi.smartpos.deviceservice.aidl.IPinpad.loadEncryptMainKeyWithAlgorithmType
(int keyId, in byte[] key, int algorithmType, in byte[] checkValue)
```

Parameters

`keyId` An index ranging from 0 to 99.

`key` The cryptographic key that is encrypted using the TEK.

`algorithmType` An integer that specifies the encryption algorithm used for the master key.

1: 3DES algorithm.

3: SM4 algorithm.

5: AES algorithm.

`checkValue` To validate the integrity and suitability of the cryptographic key. The default is set to null.

Return Values

A Boolean value:

true: Indicates that the master key was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadTEK()`
- `loadTEKWithAlgorithmType()`
- `loadEncryptMainKey()`
- `loadMainKey()`
- `loadWorkKey()`
- `loadWorkKeyWithDecryptType()`

loadMainKey()

This method is called to load a plain master key onto the Pinpad device. The default encryption algorithm used for this key is 3DES ECB.

NOTE	As per PCI Compliance, direct plain key injection is not allowed. Hence to use this API, requires few dependency files which loads the plain master key (3DES ECB algorithm default) as per Girgit Service. For further assistance, please contact Verifone Support Team.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadMainKey (int keyId,
in byte[] key, in byte[] checkValue)
```

Parameters

`keyId`	An index ranging from 0 to 99.
`key`	The cryptographic key that is encrypted using the TEK.
`checkValue`	To validate the integrity and suitability of the cryptographic key. The default is set to null.

Return Values

A Boolean value:

true: Indicates that the master key was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadEncryptMainKey()`
- `loadEncryptMainKeyWithAlgorithmType()`
- `loadMainKeyWithAlgorithmType()`
- `loadWorkKey()`
- `loadWorkKeyWithDecryptType()`

loadMainKeyWithAlgorithmType()

This method is called to load a plain master key onto the Pinpad device using a specified encryption algorithm type. This method supports both ECB and CBC modes of operation.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadMainKeyWithAlgorithmType
(int keyId, in byte[] key, int algorithmType, in byte[] checkValue)
```

Parameters

`keyId`	An index ranging from 0 to 99.
`key`	The cryptographic key that is encrypted using the TEK.
`algorithmType`	An integer values to indicate specific cryptographic algorithms used for the plain master key.

0x02:	3DES algorithm.
0x04:	SM4 algorithm.
0x06:	AES algorithm.

`checkValue`	To validate the integrity and suitability of a cryptographic key. The default is set to null.

Return Values

A Boolean value:

true: Indicates that the master key was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadEncryptMainKey()`
- `loadEncryptMainKeyWithAlgorithmType()`
- `loadMainKey()`
- `loadWorkKey()`
- `loadWorkKeyWithDecryptType()`

loadDukptKey()

This method is called to load an encrypted DUKPT key onto the Pinpad device. It decrypts the DUKPT key using TEK with an index of 0.

NOTE

- This interface method has been deprecated. Refer to `IDUKPT.aidl`.
- As per PCI Compliance, direct plain key injection is not allowed. Hence to use this API, requires few dependency files which loads the plain DUKPT key (3DES ECB algorithm default) as per Girgit Service. For further assistance, please contact Verifone Support Team.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadDukptKey (int keyId,
in byte[] ksn, in byte[] key, in byte[] checkValue)
```

Parameters

`keyId`	An index ranging from 0 to 4.
`ksn`	The Key Serial Number (KSN) associated with the DUKPT key.
`key`	The cryptographic key that is encrypted using the TEK.
`checkValue`	To validate the integrity and suitability of the cryptographic key. The default is set to null.

Return Values

A Boolean value:

true: Indicates that the DUKPT key was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

loadWorkKey()

This method is called to load a work key onto the Pinpad device. The work key is encrypted using the 3DES ECB algorithm and is decrypted by the corresponding master key during the loading process.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadWorkKey (int keyType,
int mkId, int wkId, in byte[] key, in byte[] checkValue)
```

Parameters

`keyType` Select the type of work key:

1: MAC key.

2: PIN key.

3: TD key.

`mkId`	The ID of the master key used for decrypting the work key.
`wkId`	Set the index for the work key ID ranging from 0 to 99.
`key`	The cryptographic key that will be loaded onto the Pinpad.
`checkValue`	To validate the integrity and suitability of the cryptographic key. The default is set to null for none.

Return Values

A Boolean value:

true: Indicates that the work key was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadWorkKeyWithDecryptType()`
- `loadEncryptMainKey()`
- `loadEncryptMainKeyWithAlgorithmType()`
- `loadMainKey()`
- `loadMainKeyWithAlgorithmType()`

loadWorkKeyWithDecryptType()

This method is called to load the work key onto the Pinpad device with the specified decryption key type.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadWorkKeyWithDecryptType
(int keyType, int mkId, int wkId, int decKeyType, in byte[] key, in byte[]
checkValue)
```

Parameters

`keyType` Select the type of work key:

1: MAC key.

2: PIN key.

3: TD key.

5: (SM4) MAC key.

6: (SM4) PIN key.

7: (SM4) TD key.

9: (AES) MAC key.

10: (AES) PIN key.

11: (AES) TD key.

`mkId` The ID of the master key used for decrypting the work key.

`wkId` Set the index for the work key ID ranging from 0 to 99.

`decKeyType` Specifies the type of decryption key used:

0: 3DES master key.

1: Transport key.

2: SM4 master key.

3: AES master key.

`key` The cryptographic key that will be loaded onto the Pinpad.

`checkValue` To validate the integrity and suitability of the cryptographic key. The default is set to null for none.

Return Values

A Boolean value:

true: Indicates that the work key was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadWorkKey()`
- `loadEncryptMainKey()`
- `loadEncryptMainKeyWithAlgorithmType()`
- `loadMainKey loadMainKeyWithAlgorithmType()`

calcMAC()

This method is called to calculate the Message Authentication Code (MAC) for a given data input using the default 3DES encryption algorithm with X919 format.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.calcMAC (int keyId, in byte[]
data)
```

Parameters

`keyId`	The index of the MAC key used for the calculation.
`data`	The source data input for which the MAC is to be calculated.

Return Values

Returns the calculated MAC value as a byte. It returns Null if there is failure in the MAC calculation.

See Also

- `loadWorkKey()`
- `loadWorkKeyWithDecryptType()`

calcMACWithCalType()

This method calculates the MAC using a specified calculation type. It supports additional options like CBC initialization vector (initVec) and DUKPT requests.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.calcMACWithCalType (int
keyId, int type, in byte[] CBCInitVec, in byte[] data, int desType, boolean
dukptRequest)
```

Parameters

`keyId` The index of the key has a range of 0 to 99 for MAC Keys, and 0 to 4 for DUKPT keys.

`type` The MAC's calculation mode.

0x00:	MAC X99.
0x01:	MAC X919.
0x02:	ECB (CUP standard ECB algorithm).
0x03:	MAC 9606.
0x04:	CBC MAC calculation.

`CBCInitVec` The CBC initialization vector used for encryption. This should have a fixed length of 8 bytes. It can be set to null, in which case it defaults to an 8-byte array of 0x00.

`data` The source data for which the MAC is to be calculated.

`desType` Type of encryption algorithm for MAC calculation process.

0x00:	DES.
0x01:	3DES.

	0x02: SM4.
	0x03: AES.
`dukptRequest`	A Boolean value indicating if the operation uses the DUKPT key.
	true: Indicates that the keyId refers to the DUKPT key ID.

Return Values

Returns the calculated MAC value as a byte. It returns Null if there is failure in the MAC calculation.

See Also

- `loadWorkKey()`
- `loadWorkKeyWithDecryptType()`
- `calcMAC()`

encryptTrackData()

This method encrypts the track data using the default 3DES algorithm.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.encryptTrackData (int mode,
int keyId, in byte[] trkData)
```

Parameters

`mode`	An integer specifying the encryption mode to be used:
	0: ECB.
	1: CBC.
`keyId`	The ID of the Track Data Key (TDK) that will be used in the encryption process.

`trkData` A byte array that represents the track data to be encrypted.

Return Values

Returns the encrypted track data. If the encryption fails, the method will return Null.

encryptTrackDataWithAlgorithmType()

This method encrypts track data using a specified encryption algorithm and mode, allowing for more flexibility in the encryption process.

Prototype

```
byte[]
com.vfi.smartpos.deviceservice.aidl.IPinpad.encryptTrackDataWithAlgorithmType
(int mode, int keyId, int algorithmType, in byte[] trkData, boolean
dukptRequest)
```

Parameters

`mode` An integer specifying the encryption mode to be used:

0: ECB.

1: CBC.

`keyId` The ID of the Track Data Key (TDK) that will be used in the encryption process. It ranges from 0 to 4 for DUKPT keys and 0 to 99 for other key types.

`AlgorithmType` An integer that indicates which cryptographic algorithm should be used for encrypting the track data.

0x01: 3DES.

0x02: SM4.

0x03: AES.

`trkData` A byte array that represents the track data to be encrypted.

`dukptRequest`	A Boolean value indicating if the operation uses the DUKPT key.
	true: Indicates that the keyId refers to the DUKPT key ID.

Return Values

Returns the encrypted track data. If the encryption fails, the method will return Null.

startPinInput()

This method is called to start the process of collecting user’s Personal Identification Number (PIN).

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPinpad.startPinInput (int keyId,
in Bundle param, in Bundle globleParam, PinInputListener listener)
```

Parameters

`keyId`	The index of the key has a range of 0 to 99 for PIN Keys, and 0 to 4 for DUKPT keys.
`param`	A Bundle object containing specific parameters for the PIN input operation.

pinLimit(byte[])	An array defining valid lengths for the PIN. For example, {0, 4, 5, 6} indicates valid lengths of 0, 4, 5, or 6 digits.
timeout(int)	The timeout duration in seconds.
isOnline(boolean)	Indicates if the PIN is processed online.
promptString(String)	A string to display as a prompt for the user.

pan(String)	The Primary Account Number (PAN) for encrypting an online PIN.
desType(int)	Specifies the calculation type for encryption: 0x01 MK/SK(master key/session key) + 3DES (default). 0x02 MK/SK + AES. 0x03 MK/SK + SM4. 0x04 DUKPT + 3DES.
numbersFont(String)	URL of numbers TTF font (value "" is android system fonts).
promptsFont(String)	URL of prompt TTF font (value "" is android system fonts).
otherFont(String)	URL of other TTF font (confirm button and backspace button) (value "" is android system fonts).
displayKeyValue(byte[])	Custom the sequence key number of Pinpad.
random(byte[])	Random number participation in pinblock calculation. The default is not set.
notificatePinLenError(boolean)	Notification password is not long enough. The default is set to false.
randomize_PED(boolean)	Whether or not the pin input digits shall be randomized. The default is set to true.

`globalParam` Set global display options. If null, the service defaults to Arabic numerals (0 to 9) and English labels (confirm/backspace buttons).

Display_One(String)	Text for the first display item.
Display_Two(String)	Text for the second display item.
Display_Three(String)	Text for the third display item.
Display_Four(String)	Text for the fourth display item.
Display_Five(String)	Text for the fifth display item.
Display_Six(String)	Text for the sixth display item.
Display_Seven(String)	Text for the seventh display item.
Display_Eight(String)	Text for the eighth display item.
Display_Nine(String)	Text for the ninth display item.
Display_Zero(String)	Text for the zero display item.
Display_Confirm(String)	Text for the confirm button.
Display_BackSpace(String)	Text for the backspace button.

`listener` A callback interface that receives notifications regarding the PIN input process. Refer to `PinInputListener`.

Return Values

```
void
```

submitPinInput()

This method is used to submit PIN for processing.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPinpad.submitPinInput ()
```

Parameters

None.

Return Values

```
void
```

stopPinInput()

This method is used to stop the PIN input process on the Pinpad device.

NOTE

This method has been deprecated.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPinpad.stopPinInput ()
```

Parameters

None.

Return Values

```
void
```

getLastError()

This method is called to retrieve the last error that occurred during the operation of the Pinpad device.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IPinpad.getLastError ()
```

Parameters

None.

Return Values

A string representing the description of the last error that occurred.

colculateData()

This method is used for performing cryptographic operations, including encryption or decryption of data on the Pinpad device.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.colculateData (int mode, int
desType, in byte[] key, in byte[] data)
```

Parameters

`mode` Specifies the mode of operation for the encryption or decryption operation.

0x00	MK/SK Encrypt.
0x01	MK/SK Decrypt.

`desType` Indicates the type of encryption or decryption standard.

TYPE_DES	0x00 DES Type.
TYPE_3DES	0x01 3DES Type (EBC).
TYPE_SM4	0x02 SM4 Type.
TYPE_AES	0x03 AES Type.
TYPE_SM2_PUBKEY	0x04 SM2 Type (use public key).
TYPE_SM2_PRIVKEY	0x05 SM2 Type (use private key).
TYPE_3DES	0x06 3DES Type (CBC, with an initVec of 00000000).

`key` A byte array that represents the cryptographic key used for the operation.

`data` The source data to be encrypted or decrypted.

Return Values

The encrypted or decrypted data as a byte array. Returns Null if the operation fails.

See Also

`calculateData()`

dukptEncryptData()

This method is used for encrypting data using the DUKPT key management scheme.

NOTE This method has been deprecated. Refer to `IDUKPT.aidl.`

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.dukptEncryptData (int
destype, int algorithm, int keyid, in byte[] data, in byte[] CBCInitVec)
```

Parameters

`desType` The type of encryption standard to be used:

TYPE_DES	0x00 DES Type.
TYPE_3DES	0x01 3DES Type.
TYPE_SM4	0x02 SM4 Type.
TYPE_AES	0x03 AES Type.

`algorithm` The type of algorithm used for encryption:

0x01:	CBC.
0x02:	ECB.

`keyId` The index of the key used for encryption. The value can range from 0 to 4 for DUKPT keys.

`data` The source data to be encrypted.

`CBCInitVec` The CBC initialization vector used for encryption. This should have a fixed length of 8 bytes. It can be set to null, in which case it defaults to an 8-byte array of 0x00.

Return Values

Returns a byte array containing the encrypted data using the specified parameters.

savePlainKey()

This method is used to save the plain key (unencrypted key) into the Pinpad device, particularly supporting the 3DES encryption standard.

NOTE

This method has been deprecated.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.savePlainKey (int keyType,
int keyId, in byte[] key)
```

Parameters

`keyType` Indicates the type of key being saved.

1: MAC.

2: PIN.

3: TD.

`keyId` The index of the key being saved.

`key` The source data key to be saved.

Return Values

A Boolean value:

true: The plain key was successfully saved into the Pinpad device.

false: There was a failure in saving the plain key.

getDukptKsn()

This method is used to retrieve the current DUKPT KSN from the Pinpad device.

NOTE

This method has been deprecated. Refer to `IDUKPT.aidl`.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.getDukptKsn ()
```

Parameters

None.

Return Values

The current DUKPT KSN.

generateSM2KeyPair()

This method is called to generate a key pair (public key and private key) for the SM2 encryption algorithm.

Prototype

```
Bundle com.vfi.smartpos.deviceservice.aidl.IPinpad.generateSM2KeyPair ()
```

Parameters

None.

Return Values

A Bundle containing the following:

publicKey(string): A string representing the generated public key.

privateKey(string): A string representing the generated private key.

getSM3Summary()

This method is called to retrieve the SM3 data summary by computing the SM3 hash (a cryptographic hash function) of the provided input data.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.getSM3Summary (in byte[]
data)
```

Parameters

`data`	A byte array indicating the input data that the user wants to hash using the SM3 algorithm.

Return Values

The computed SM3 data summary as a byte array.

getSM2Sign()

This method is called to retrieve the SM2 digital signature for the provided data. It uses the SM2 algorithm to sign the data, ensuring its authenticity and integrity.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.getSM2Sign (in Bundle bundle)
```

Parameters

`bundle` A Bundle object that contains the necessary data required for signing including:

prikey(byte[]) The private key used for signing; passed as a byte array.

data(byte[]) The byte array of the data that the user wants to sign.

Return Values

Returns the SM2 digital signature as a byte array.

getKeyKCV()

This method retrieves the Key Check Value (KCV) for a specified key. The KCV is a value used to verify the integrity of cryptographic keys, ensuring they have not been altered or corrupted.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.getKeyKCV (int keyIndex,
int keyType)
```

Parameters

`keyIndex` The index of the key for which the KCV is requested.

`keyType` Indicates the type of the key for which the KCV is being retrieved.

0x01: Data encryption key.

0x02: PIN working key.

0x03: MAC key.

0x04: Transfer key

0x05: Main key.

0x11:	Data encryption key (SM4).
0x12:	PIN working key (SM4).
0x13:	MAC key (SM4).
0x14:	Transport key (SM4).
0x15:	Master key (SM4).
0x21:	DATA key (AES).
0x22:	PIN key (AES).
0x23:	MAC key (AES).
0x24:	AES transmission key.
0x25:	AES master key.

Return Values

The check value (KCV) of the specified key as a byte array.

initPinInputCustomView()

This method is used to initialize a custom user interface for PIN input process on the Pinpad device.

Prototype

```
Map com.vfi.smartpos.deviceservice.aidl.IPinpad.initPinInputCustomView (int
keyId, in Bundle param, in List< PinKeyCoorInfo > pinKeyInfos, PinInputListener
listener)
```

Parameters

`keyId`	The pinKey ID is the ID of the loadWorkKey (PIN) ID.

`param` A Bundle that holds additional parameters for the PIN input setup.

pinLimit(byte[])	An array defining valid lengths for the PIN. For example, {0, 4, 5, 6} indicates valid lengths of 0, 4, 5, or 6 digits).
timeout(int)	The timeout duration in seconds.
isOnline(boolean)	Indicates if the PIN is processed online.
pan(String)	The Primary Account Number (PAN) for encrypting an online PIN.
desType(int)	Specifies the calculation type for encryption.
displayKeyValue(byte[])	Customizes the sequence of key numbers displayed on the Pinpad.
random(byte[])	A random number to participate in the pinblock calculation. The default is not set.
randomize_PED(boolean)	Indicates whether the PIN input digits should be randomized. The default is set to true.

`pinKeyInfos` A list of PinKeyCoorInfo objects that define the coordinates and layout of the PIN keys on the custom view.

`listener` A listener interface that receives callbacks related to the PIN input process, such as success or error notifications. Refer to `PinInputListener`.

Return Values

map<String String>: A map indicating the display values for the keys from 0 to 9.

startPinInputCustomView()

This method is used to start the custom user interface for PIN input process on the Pinpad device.

NOTE

If you retrieve a Map<String, String> from a previous initialization method, you should traverse the map to get the values associated with the keys to display them on the PIN input interface.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPinpad.startPinInputCustomView ()
```

Parameters

None.

Return Values

```
void
```

endPinInputCustomView()

This method is called to end the custom PIN input flow on the Pinpad device. It stops any ongoing PIN input session that was initiated through the custom user interface.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPinpad.endPinInputCustomView ()
```

Parameters

None.

Return Values

```
void
```

calculateData()

This method is used for performing cryptographic operations, including encryption or decryption of data.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.calculateData (int mode,
int desType, in byte[] key, in byte[] data)
```

Parameters

`mode` Specifies the mode of operation for the encryption or decryption operation.

0x00	MK/SK Encrypt.
0x01	MK/SK Decrypt.

`desType` Indicates the type of encryption or decryption standard.

TYPE_DES	0x00 DES Type.
TYPE_3DES	0x01 3DES Type (EBC).
TYPE_SM4	0x02 SM4 Type.
TYPE_AES	0x03 AES Type.
TYPE_SM2_PUBKEY	0x04 SM2 Type (use public key).
TYPE_SM2_PRIVKEY	0x05 SM2 Type (use private key).
TYPE_3DES	0x06 3DES Type (CBC, with an initVec of 00000000).

`key`	A byte array that represents the cryptographic key used for the operation.
`data`	The source data to be encrypted or decrypted.

Return Values

The encrypted or decrypted data as a byte array. Returns Null if the operation fails.

calculateDataEx()

This method is used for performing more advanced and complex cryptographic operations, including encryption or decryption of sensitive data, with support for an initialization vector for certain modes.

NOTE This method has been deprecated.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.calculateDataEx (int mode,
int desType, in byte[] key, in byte[] data, in byte[] initVec)
```

Parameters

`mode` Specifies the mode of operation for the encryption or decryption.

0x00	MK/SK Encrypt.
0x01	MK/SK Decrypt.

`desType` Indicates the type of encryption or decryption standard.

TYPE_DES	0x00 DES Type.
TYPE_3DES	0x01 3DES Type (EBC).

TYPE_SM4	0x02 SM4 Type.
TYPE_AES	0x03 AES Type.
TYPE_SM2_PUBKEY	0x04 SM2 Type (use public key).
TYPE_SM2_PRIVKEY	0x05 SM2 Type (use private key).
TYPE_3DES	0x06 3DES Type (CBC).

NOTE

WorkKey (TD) ID = 60 indicates that this specific identifier is reserved and should not be used by user applications.

`key`	A byte array that represents the cryptographic key used for the operation.
`data`	The source data to be encrypted or decrypted.
`initVec`	An initVec required for the 3DES CBC mode.

Return Values

The encrypted or decrypted data as a byte array. Returns Null if the operation fails.

encryptPinFormat0()

This method is called to securely encrypt the PIN using the provided card number and password.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.encryptPinFormat0 (int
pinKeyId, int desType, in byte[] cardNumber, String passwd)
```

Parameters

`pinKeyId`	Indicates the index of the pin key used for encryption. It ranges from 0 to 99.
`desType`	Specifies the type of encryption standard to be used during the encryption process. 0x01 MK/SK + 3DES (default). 0x02 MK/SK + AES. 0x03 MK/SK + SM4. `0x04 DUKPT + 3DES.
`cardNumber`	Indicates the card number represented in ASCII. For example, the input "1234", should be converted to a byte array as follows: byte[4] = {31, 32, 33, 34}.
`passwd`	Indicates the plain password as a String, such as `"1234"`. This will be used in conjunction with the card number for PIN block encryption.

Return Values

A byte array containing the encrypted PIN block.

calculateByDataKey()

This method is called to perform cryptographic operations such as encryption or decryption using a specified data key.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.calculateByDataKey (int
keyId, int encAlg, int encMode, int encFlag, in byte[] data, in byte[] initVec)
```

Parameters

`KeyId` Indicates the index of the data key used for the cryptographic operation. It ranges from 0 to 99.

`encAlg` Indicates the encryption algorithm to be used.

0x01: 3DES.

0x02: SM4.

0x03: AES.

`encMode` Indicates the mode of operation for the encryption algorithm.

0x01: ECB.

0x02: CBC.

`encFlag` Indicates an encryption flag that provides additional control over the encryption or decryption process.

0x00: Encrypt.

0x01: Decrypt.

`data` The source data to be encrypted or decrypted.

`initVec` An initialization vector is used in specific modes of encryption. The default value is set to null.

Return Values

Returns the encrypted or decrypted data.

loadEncryptMainKeyEX()

This method is called to load an encrypted master key onto a secure Pinpad, based on the specified encryption algorithm type.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadEncryptMainKeyEX (int
keyId, in byte[] key, int algorithmType, in byte[] checkValue, in Bundle extend)
```

Parameters

`keyId`	Indicates the index of the key slot. It ranges from 0 to 99.
`key`	Indicates the encrypted master key to be loaded.
`algorithmType`	The type of algorithm used for encryption.

0x01:	3DES algorithm.
0x03:	SM4 algorithm.
0x05:	AES algorithm.
0x81:	3DES (CBC).
0x83:	SM4 (CBC).
0x85:	AES (CBC).

`checkValue`	A byte array used for integrity verification of the key being loaded. The default value is set to null.
`extend`	A Bundle object that allows you to pass additional optional parameters related to the key loading process.

isCBCType(boolean)	Indicates whether the master key encryption mode is CBC mode. The default is
initVec(byte[])	Initialization vector for CBC mode. The default is set to 16 byte 0.
isMasterEncMasterMode(boolean)	Indicates whether the MK (master key) can encrypt another MK; applicable only if a master key has already been loaded.

decryptKeyIndex(int)	The index (0 to 99) of the decryption key. If not set, the method will use the `keyId`. The last key will be overwritten if specified.

Return Values

A Boolean value:

true: Indicates that the operation was successful.

false: Indicates that the operation failed.

See Also

- `loadTEK loadTEKWithAlgorithmType()`
- `loadEncryptMainKey()`
- `loadMainKey()`
- `loadWorkKey()`
- `loadWorkKeyWithDecryptType()`

loadWorkKeyEX()

This method is called to load a work key onto the Pinpad device with a specified decryption type.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadWorkKeyEX (int keyType,
int mkId, int wkId, int decKeyType, in byte[] key, in byte[] checkValue, in
Bundle extend)
```

Parameters

`keyType` Specifies the type of work key being loaded, including:

1:	MAC key.
2:	PIN key.
3:	TD key.
5:	(SM4) MAC key.
6:	(SM4) PIN key.
7:	(SM4) TD key.
9:	(AES) MAC key.
10:	(AES) PIN key.
11:	(AES) TD key.

`mkId` The master key's ID for decrypting a work key.

`wkId` The work key ID ranging from 0 to 99.

`decKeyType` Select the type of decryption key.

0x00:	3DES master key.
0x01:	Transport key.
0x02:	SM4 master key.
0x03:	AES master key.
0x04:	SM4 transport key.
0x05:	AES transport key.
0x80:	CBC 3DES master key.
0x81:	CBC transport key.

	0x82:	CBC SM4 master key.
	0x83:	CBC AES master key.
	0x84:	CBC SM4 transport key.
	0x85:	CBC AES transport key.
`key`	The cryptographic key for decrypting the work key.	
`checkValue`	A byte array used for integrity verification of the key being loaded (null for none).	
`extend`	A Bundle object that allows you to pass additional optional parameters related to the key loading process.	
	isCBCType(boolean)	Indicates whether the mk encryption mode is in CBC mode.
	initVec(byte[])	An initialization vector used in CBC mode for encryption and decryption processes. The default is set to 16 byte 0.

Return Values

A Boolean Value:

true: Indicates that the operation was successful.

false: Indicates that the operation failed.

See Also

- `loadWorkKey()`
- `loadEncryptMainKey()`
- `loadEncryptMainKeyWithAlgorithmType()`
- `loadMainKey()`
- `loadMainKeyWithAlgorithmType()`

clearKey()

This method is called to remove both master key and work key from the Pinpad. It is applicable for devices running on K21 version greater than 169.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.clearKey (int keyId,
int keyType)
```

Parameters

`keyId` Specifies the ID of the key that needs to be cleared.

`keyType` Specifies the type of key that the user wants to clear from the Pinpad's memory.

0x00:	DES MK.
0x01:	SM4 MK.
0x02:	AES MK.
0x10:	DES PIN.
0x11:	SM4 PIN.
0x12:	AES PIN.
0x20:	DES MAC.
0x21:	SM4 MAC.
0x22:	AES MAC.
0x30:	DES DATA.
0x31:	SM4 DATA.
0x32:	AES DATA.

0x40: DUKPT.

Return Values

A Boolean value:

true: Indicates that the key was successfully cleared.

false: Indicates that the operation failed.

loadDukptKeyEX()

This method is used to load the DUKPT key with additional configuration options.

> NOTE
>
> This method has been deprecated. Refer to `IDUKPT.aidl`.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadDukptKeyEX (int keyId,
in byte[] ksn, in byte[] key, in byte[] checkValue, in Bundle extend)
```

Parameters

`keyId`	An index ranging from 0 to 4.
`ksn`	The KSN associated with the DUKPT key.
`key`	The cryptographic key that is encrypted using the TEK.
`checkValue`	To validate the integrity and suitability of the cryptographic key. The default is set to null.

extend	The extend parameter, which is of type Bundle, is used to pass additional configuration options when loading the DUKPT key.	
	loadPlainKey(boolean)	Indicates whether to loadPlainKey in unencrypted format or as an encrypted key.
	TEKIndex(int)	If loadPlainKey is set to false, you need to specify the TEKIndex, which indicates the index of the TEK to use.

Return Values

A Boolean value:

true: Indicates that the DUKPT key was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

loadTEKEX()

This method is used to load a plain TEK with specific parameters onto the Pinpad device.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPinpad.loadTEKEX (int keyId,
in byte[] key, in byte algorithmType, in byte[] checkValue, in Bundle extend)
```

Parameters

keyId	An index ranging from 0 to 99.	
key	The cryptographic key that is encrypted using the TEK.	
algorithmType	The type of algorithm used for encryption.	
	0x01:	3DES encrypted key.

0x02:	3DES plain key.
0x03	SM4 encrypted key.
0x04:	SM4 plain key.
0x05:	AES encrypted key.
0x06:	AES plain key.
0x81:	CBC 3DES encrypted key.
0x82:	CBC 3DES plain key.
0x83:	CBC SM4 encrypted key.
0x84:	CBC SM4 plain key.
0x85:	CBC AES encrypted key.
0x86:	CBC AES plain key.

`checkValue` To validate the integrity and suitability of the cryptographic key.

`extend` The extend parameter, which is of type Bundle, is used to pass additional configuration options when loading the DUKPT key.

isCBCType(boolean)	Indicates whether the mk encryption mode is in CBC mode. The default is set to false.
initVec(byte[])	An initialization vector used in CBC mode for encryption and decryption processes. The default is set to 16 byte 0.

Return Values

A Boolean value:

true: Indicates that the TEK was successfully loaded onto the Pinpad.

false: Indicates that the operation failed.

See Also

- `loadTEK()`
- `loadEncryptMainKey()`
- `loadEncryptMainKeyWithAlgorithmType()`

calculateByWorkKey()

This method is used to perform encryption or decryption of data using a specified work key.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.calculateByWorkKey (int
keyId, int keyType, int encAlg, int encMode, int encFlag, in byte[] data, in
Bundle extend)
```

Parameters

`keyId` Indicates the index of the data key used for the cryptographic operation. It ranges from 0 to 99.

`keyType` Specifies the type of key being used.

`encAlg` This indicates the encryption algorithm to be used.

0x01:	3DES.
0x02:	SM4.
0x03:	AES.

`encMode` This indicates the mode of operation for the encryption algorithm.

0x01:	ECB.
0x02:	CBC.

`encFlag` Indicates an encryption flag that provides additional control over the encryption or decryption process.

0x00:	Encrypt.
0x01:	Decrypt.

`data` The source data to be encrypted or decrypted.

`extend` A Bundle parameter called extend, which allows for additional configuration options, including specifying an initialization vector for encryption modes that require it, such as CBC.

Return Values

A byte array containing the result of the encryption or decryption operation.

calculateByMSKey()

This method is used for performing encryption or decryption operations using the specified Master Key (MK) on the Pinpad device.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPinpad.calculateByMSKey (int keyId,
int keyType, int algorithmMode, in byte[] data, in Bundle extend)
```

Parameters

`keyId` The index for the data key, ranging from 0 to 99.

`keyType` Specifies the type of key being used.

0x01:	Master key.
0x02:	SM4 master key.
0x03:	AES master key.

`algorithmMode`	The mode of the encryption algorithm:	
	0x00:	Encrypt ECB
	0x01:	Decrypt ECB.
	0x02:	Encrypt CBC
	0x03:	Decrypt CBC.
`data`	A byte array representing the input data that the user wants to encrypt or decrypt.	
`extend`	A Bundle containing additional configuration parameters:	
	initVec(byte[])	An optional initialization vector for the CBC mode. If not set, it defaults to null.

Return Values

A byte array containing the result of the encryption or decryption operation.

2.2.15 IPBOC

Package: com.vfi.smartpos.deviceservice.aidl.IPBOC

Overview:

The IPBOC interface is part of the AIDL framework, intended to facilitate communication between Android applications and POS devices. It provides several methods for card processing, transaction management, device control, callbacks and listeners and error handling.

Public Member Function:

Modifier and Type	Method
void	**checkCard** (in Bundle cardOption, int timeout, **CheckCardListener** listener)

void	**stopCheckCard** ()
void	**readUPCard** (**UPCardListener** listener)
void	**startPBOC** (int transType, in Bundle intent, **PBOCHandler** handler)
void	**startEMV** (int processType, in Bundle intent, **PBOCHandler** handler)
void	**abortPBOC** ()
boolean	**updateAID** (int operation, int aidType, String aid)
boolean	**updateRID** (int operation, String rid)
Void	**importAmount** (long amount)
void	**importAppSelect** (int index)
void	**importPin** (int option, in byte[] pin)
void	**importCertConfirmResult** (int option)
void	**importCardConfirmResult** (boolean pass)
void	**inputOnlineResult** (in Bundle onlineResult, **OnlineResultHandler** handler)
void	**setEMVData** (in List< String > tlvList)
String	**getAppTLVList** (in String[] taglist)
byte[]	**getCardData** (String tagName)
String	**getPBOCData** (String tagName)
CandidateAppInfo	**getCandidateAppInfo** ()

String[]	**getAID** (int type)
String[]	**getRID** ()
int	**getProcessCardType** ()

Member Function Documentation:

checkCard()

This method is used to initiate a card reading operation in a non-blocking manner, allowing Android applications to interact with POS devices effectively.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.checkCard (in Bundle cardOption,
int timeout, CheckCardListener listener)
```

Code Snippet

```
public void My17startPBOC() {
    Bundle cardOption = new Bundle();
    cardOption.putBoolean("supportMagCard", true);
    cardOption.putBoolean("supportICCard", true);
    cardOption.putBoolean("supportRFCard", true);

     ipboc.checkCard( cardOption,  30, new CheckCardListener.Stub() {
      String msg;

      @Override
      public void onCardSwiped(Bundle track) throws RemoteException {
          ipboc.stopCheckCard();
      }

      @Override
      public void onCardPowerUp() throws RemoteException {
          // Smart card
          ipboc.stopCheckCard();
          try {
              Bundle intent1 = intent;
              // don't change transaction type for test case K17, card type will
be changed according the check card result for other test case
              if (intent.getBoolean(BUNDLE_NOT_CHANGE_CARDTYPE) == false)
```

```
                  intent1.putInt(BUNDLE_STARTPBOCPARAM_CARDTYPE, 0);
              ipboc.startPBOC(transType, intent1, pbochandler);
          } catch (RemoteException e) {
              e.printStackTrace();
          }
      }

      @Override
      public void onCardActivate() throws RemoteException {
          // CTLS
          ipboc.stopCheckCard();
          try {
              Bundle intent1 = intent;

              // don't change transaction type for test case K17, card type will
be changed according the check card result for other test case
              if (intent.getBoolean(BUNDLE_NOT_CHANGE_CARDTYPE) == false)
                  intent1.putInt(BUNDLE_STARTPBOCPARAM_CARDTYPE, 1);
              ipboc.startPBOC(transType, intent1, pbochandler);
          } catch (RemoteException e) {
              e.printStackTrace();
          }
      }

      @Override
      public void onTimeout() throws RemoteException {
          ipboc.stopCheckCard();
      }

      @Override
      public void onError(int error, String message) throws RemoteException {
          // msg = "error" + error + message;
          ipboc.stopCheckCard();

          if (error == 3)
              message1.getData().putString("message", msg + "Fallback");
          else
              message1.getData().putString("message", msg);

          if (error == 3) {

          }
      }
  });
 }
```

Parameters

`cardOption`	Indicates the types of cards that the POS device should support.

	supportMagCard(boolean)	Indicates support for the magnetic card.
	supportICCard(boolean)	Indicates support for the IC card.
	supportRFCard(boolean)	Indicates support for the CTLS card.
`timeout`	The timeout duration in seconds.	
`listener`	A callback interface to notify the result of the card reading operation. Refer to `CheckCardListener`.	

Return Values

`void`

See Also

- `stopCheckCard()`
- `startPBOC()`
- `startEMV()`
- Refer to `CheckCardListener` interface under Section 2.2.1.

stopCheckCard()

This method is used to stop an ongoing card reading operation that was initiated by the `checkCard()` method.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.stopCheckCard ()
```

Parameters

None.

Return Values

`void`

See Also

- `checkCard()`
- `startPBOC()`
- `startEMV()`
- Refer to `CheckCardListener` interface under Section 2.2.1.

readUPCard()

This method is used to read a UP card, including cards that have a chip embedded in the SIM card.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.readUPCard (UPCardListener
listener)
```

Parameters

`listener`	A callback interface that is used to handle the results of the UP card reading operation. Refer to `UPCardListener`.

Return Values

`void`

See Also

`UPCardListener`

startPBOC()

This method is used to initiate a PBOC transaction, allowing users to make payments by simply tapping or waving their card or mobile device near the POS terminal.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.startPBOC (int transType,
in Bundle intent, PBOCHandler handler)
```

Parameters

`transType` Specifies the type of transaction to be performed.

EC_BALANCE(1)	Query the balance.
TRANSFER(2)	Transfer funds.
EC_LOAD(3)	Load electronic cash (EC LOAD).
EC_LOAD_U(4)	EC LOAD without assigning an account.
EC_LOAD_CASH(5)	EC LOAD with cash.
EC_LOAD_CASH_VOID(6)	Void an EC LOAD with cash.
PURCHASE(7)	Make a purchase.
Q_PURCHASE(8)	Quick purchase.
CHECK_BALANCE(9)	Get the balance.
PRE_AUTH(10)	Pre-authorization.
SALE_VOID(11)	Void a sale.
SIMPLE_PROCESS(12)	Simple processing transaction.
REFUND(13)	Process a refund (full process).

`intent`	A Bundle that contains additional information or parameters needed for the transaction.	
	cardType(int)	Specifies the card type.
	CARD_INSERT(0)	For smart IC card.
	CARD_RF(1)	For CTLS card.
	authAmount(long)	The transaction amount that requires authorization.
	isSupportQ(boolean)	Indicates support for the QPBOC.
	isSupportSM(boolean)	Indicates support for the SM.
	isQPBOCForceOnline(boolean)	Indicates if the QPBOC is forced to go online.
	merchantName(String)	The name of the merchant.
	merchantId(String)	The merchant's ID.
	terminalId(String)	The terminal's ID.
`handler`	A callback interface that receives the result of the EMV transaction. Refer to `PBOCHandler` under Section 2.2.6.	

Return Values

`void`

See Also

`startEMV()`

startEMV()

This method is used to initiate an EMV transaction, allowing for secure processing of chip card payments.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.startEMV (int processType,
in Bundle intent, PBOCHandler handler)
```

Parameters

`processType`	An integer specifies the type of processing for the transaction.		
	1:		Full process.
	2:		Simple process.
`intent`	A Bundle that contains additional information or parameters needed for the transaction.		
	cardType(int)		Specifies the card type.
		CARD_INSERT(0)	For smart IC card.
		CARD_RF(1)	For CTLS card.
	transProcessCode(byte)		A 1 Byte code representing translation type (9C first two digits of the ISO 8583:1987 Processing Code).
	authAmount(long)		The transaction amount that requires authorization.
	isSupportQ(boolean)		Indicates support for the QPBOC.
	isSupportSM(boolean)		Indicates support for the SM.

isQPBOCForceOnline(boolean)	Indicates if the QPBOC is forced to go online.
merchantName(String)	The name of the merchant.
merchantId(String)	The merchant's ID.
terminalId(String)	The terminal's ID.

`handler`	A callback interface to handle the results of the PBOC transaction. Refer to `PBOCHandler` under Section 2.2.6.

Return Values

```
void
```

See Also

- `startPBOC()`
- `abortPBOC()`

abortPBOC()

This method is used to terminate an active PBOC transaction, ensuring that the system can recover from any errors without leaving the transaction in an uncertain state.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.abortPBOC ()
```

Parameters

None.

Return Values

```
void
```

updateAID()

This method is used to update the AID for payment applications in a payment processing system. This method allows user to perform three main operations (like add, remove, and clear all AIDs) related to the AIDs.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPBOC.updateAID (int operation,
int aidType, String aid)
```

Parameters

`operation` Specifies the type of operation to perform on the AID.

1: (append) Adds a new AID to the existing list of applications.

2: (remove) Removes a specific AID from the list.

3: (clear all) Clears all AIDs, resetting the application settings.

`aidType` Indicates the type of AID being managed.

1: (contact) Refers to a traditional smart card application that requires physical contact with the reader.

2: (contactless) Refers to applications that utilize contactless technology, allowing transactions through NFC or RFID.

`aid` The actual AID to be added, removed, or cleared. This identifier is crucial for selecting the appropriate payment application during transactions.

Return Values

A Boolean value:

true: Indicates that the operation was successful.

false: Indicates that the operation failed.

See Also

`getAID()`

updateRID()

This method is used to update the RID associated with the CA public key in a payment processing system. This method allows user to perform three main operations (like add, remove, and clear all RIDs) related to the RIDs.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IPBOC.updateRID (int operation,
String rid)
```

Parameters

`operation` Specifies the type of operation to perform on the RID.

1: (append) Add a new RID associated with a new CA public key.

2: (remove) Remove an existing RID for a CA public key.

3: (clear all) Clears all RIDs, resetting the application settings.

`rid` The actual RID that corresponds to the CA public key being updated.

Return Values

A Boolean value:

true: Indicates that the operation was successful.

false: Indicates that the operation failed.

See Also

`getRID()`

importAmount()

This method is essential for importing and handling monetary values within financial applications. It serves to register the amount, which can then be used for further processing, like initiating EMV transactions.

NOTE

This method has been deprecated.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.importAmount (long amount)
```

Parameters

`amount`	Indicates a monetary value to be imported, specified in the smallest currency unit.

Return Values

`void`

See Also

`startEMV()`

importAppSelect()

This method is used to select a specific application from a multi-application card. It allows the system to determine which payment application to use for processing a transaction based on the provided index.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.importAppSelect (int index)
```

Parameters

`index`	The index of the application to select.		
		Start from 1:	This indicates the first application in the list.
		0:	Indicates a cancel action.

Return Values

```
void
```

See Also

Refer to `PBOCHandler.onSelectApplication()` method under Section 2.2.6.

importPin()

This method is used to import or process the PIN for authentication during financial transactions.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.importPin (int option,
in byte[] pin)
```

Parameters

`option` Specifies the action to be taken regarding the PIN.

CANCEL (0): Indicates that the PIN entry should be cancelled.

CONFIRM (1): Indicates that the PIN entry should be confirmed and processed.

`pin` An array of bytes that indicates the PIN data.

Return Values

`void`

See Also

Refer to `PBOCHandler.onRequestInputPIN()` method under Section 2.2.6.

importCertConfirmResult()

This method is part of the Unified Payment (UP) card processing flow, specifically related to certificate confirmation. This method is called to confirm the result of a certificate validation process during a transaction, to secure payment or authentication processes. Certificates are used to verify the authenticity of the card, issuer, or transaction, ensuring that the transaction is legitimate and secure.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.importCertConfirmResult (int
option)
```

Parameters

`option` Indicates the result of the cardholder verification process.

CANCEL (0): Indicates that the verification process was cancelled (bypassed).

CONFIRM (1):	Indicates that the verification was successful and confirmed.
NOTMATCH (2):	Indicates that the entered information does not match the cardholder's data.

Return Values

`void`

See Also

Refer to `PBOCHandler` interface under Section 2.2.6.

importCardConfirmResult()

This method is part of the UP card processing flow, specifically related to cardholder verification during a payment transaction. This method is used to confirm the outcome of a cardholder authentication process, such as when the system verifies that the person using the card is authorized to complete the transaction.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.importCardConfirmResult (boolean
pass)
```

Parameters

`pass` Indicates the outcome of the card verification.

true:	The card verification was successful.
false:	The card verification failed.

Return Values

`void`

See Also

Refer to `PBOCHandler.onConfirmCardInfo()` method under Section 2.2.6.

inputOnlineResult()

This method is used to import and process the response received from an online transaction, allowing the system to take appropriate actions based on the result.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.inputOnlineResult (in Bundle
onlineResult, OnlineResultHandler handler)
```

Parameters

`onlineResult` A Bundle that contains the results of the online operation. It includes:

isOnline(boolean) Indicates whether the transaction was processed online.

respCode(String) The response code returned from the online operation, indicating the status of the transaction.

authCode(String) The authorization code provided for the transaction, if applicable.

field55(String) The response data for field 55, which may contain additional transaction details or messages.

`handler` This is an instance of an interface or class intended to manage the results of the online transaction. Refer to `OnlineResultHandler` under Section 2.2.7.

Return Values

```
void
```

See Also

Refer to `PBOCHandler.onRequestOnlineProcess()` method under Section 2.2.6.

setEMVData()

This method is used to set the EMV kernel data before initiating the EMV transaction flow. Properly setting this data is essential for the transaction's success, as it includes important information required for processing.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IPBOC.setEMVData (in List< String >
tlvList)
```

Code Snippet

```
IPBOC ipboc;
EMVL2 emvl2;
public void K12001() {
    byte[] acquirerID = {(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00};
    byte[] termCap = {(byte) 0xe0, (byte) 0xf1, (byte) 0xc8};
    byte[] addTermCap = {(byte) 0xe0, (byte) 0x00, (byte) 0xF0, (byte) 0xA0,
(byte) 0x01};
    byte[] countryCode = {(byte) 0x01, (byte) 0x56};
    byte[] currencyCode = {(byte) 0x01, (byte) 0x56};
    byte[] termType = {(byte) 0x22};

    Collection<String> tlvList = new ArrayList<String>();

    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F15020000")));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F160F")) +
Utils.byte2HexStr("123456789012345".getBytes()));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F4E0D")) +
Utils.byte2HexStr("Verifone Test".getBytes()));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F1C08")) +
Utils.byte2HexStr("12345678".getBytes()));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F0106")) +
Utils.byte2HexStr(acquirerID));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F1E08")) +
Utils.byte2HexStr("50342027".getBytes()));
```

```
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F3501")) +
Utils.byte2HexStr(termType));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F3303")) +
Utils.byte2HexStr(termCap));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F4005")) +
Utils.byte2HexStr(addTermCap));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F1A02")) +
Utils.byte2HexStr(countryCode));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("5F2A02")) +
Utils.byte2HexStr(currencyCode));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("5F3601" + "02")));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F3C02")) +
Utils.byte2HexStr(currencyCode));
    tlvList.add(Utils.byte2HexStr(Utils.asc2Bcd("9F3D0102")));
    ipboc.setEMVData(tlvList);
}
```

Parameters

`tlvList` This list contains Tag-Length-Value (TLV) formatted strings. The method supports setting the following tags, which are essential for EMV transaction processing. If there is a conflict with an AID list that has the same tag, the AID list takes priority over this method.

9F33:	Terminal Capability.
9F15:	Merchant Category Code (MCC).
9F16:	Merchant Identifier.
9F4E:	Merchant Name.
9F1C:	Terminal Identifier.
9F35:	Terminal Type.
9F3C:	Transaction Reference Currency Code.
9F3D:	Transaction Reference Currency Exponent.
5F2A:	Transaction Currency Code.
5F36:	Transaction Currency Exponent.
9F1A:	Terminal Country Code.

9F40: Additional Terminal Capability.

Return Values

```
void
```

getAppTLVList()

This method is called to retrieve a list of TLV data from the PBOC kernel for specific tags related to the EMV application.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IPBOC.getAppTLVList (in String[]
taglist)
```

Code Snippet

```
{
    String[] strlist = {"9F33", "9F40", "9F10", "9F26", "95", "9F37", "9F1E",
"9F36",
            "82", "9F1A", "9A", "9B", "50", "84", "5F2A", "8F"};

    String strs = ipboc.getAppTLVList(strlist);
 }
```

Parameters

`taglist`	An array of strings representing the tags you want to query from the PBOC kernel.

Return Values

Returns either of the two values:

TLV Format: The method returns data in the TLV format.

Null: Indicating that there is no response available.

getCardData()

This method is called to retrieve the EMV card data based on a specified tag.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IPBOC.getCardData (String tagName)
```

Parameters

`tagName`	The name of the tag for which the data is requested.

Return Values

The EMV data associated with the specified tag name.

See Also

`getPBOCData()`

getPBOCData()

This method is used to retrieve the specific PBOC or EMV data, such as card number, validity date, card serial number, and other relevant details.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.IPBOC.getPBOCData (String tagName)
```

Parameters

`tagName` The name of the tag for which the PBOC data is requested. It includes:

PAN card No.	Primary Account Number (card number).
TRACK2	Track data from Track 2 of the magnetic stripe.
CARD_SN	Card Serial Number.

EXPIRED_DATE	Expiration date of the card.
DATE	Current date.
TIME	Current time.
BALANCE	Current balance.
CURRENCY	Currency code associated with the card.

Return Values

The requested PBOC data associated with the specified tag name.

See Also

`getCardData()`

getCandidateAppInfo()

This method is used to retrieve candidate application information, specifically for uploading electronic signatures (e-signatures) during EMV transactions.

Prototype

```
CandidateAppInfo com.vfi.smartpos.deviceservice.aidl.IPBOC.getCandidateAppInfo
()
```

Parameters

None.

Return Values

A list of strings containing details about the candidate applications.

getAID()

This method is used to retrieve the AID based on a specified application type during EMV transactions.

Prototype

```
String[] com.vfi.smartpos.deviceservice.aidl.IPBOC.getAID (int type)
```

Code Snippet

```
@brief get the AID parameter

   @param type - 1-contact aid 2-contactless aid
   @return null if the AID is unavailable
   \code
  demo returns from getAID(1)

{"9F0607A0000000031010DF0101009F09020140DF1105C000000000DF12050000000000DF130
50000000000009F1B0400000000DF1504000000009F7B06000000000000DF1906000000000000DF
20060099999999995F2A0201569F1A0201569F3303E0F9C89F4005FF00F0A0019F660426000080
9F350122DF150400000000DF160101DF170101DF14039F3704DF1801009F1D00",
"9F0607A0000000032010DF0101009F09020140DF1105D84004A800DF1205D84000F800DF13050
0100000009F1B0400000000DF1504000000009F7B06000000000000DF1906000000000000DF200
6009999999999995F2A0201569F1A0201569F3303E0F9C89F4005FF00F0A0019F6604260000809F3
50122DF150400000000DF160101DF170101DF14039F3704DF1801009F1D00"
```

Parameters

`type`	The type of application for which the AID is requested.

1: Contact AID for EMV cards that require physical contact.

2: Contactless AID for EMV cards that support contactless transactions.

Return Values

A string representing the AID of the application.

See Also

```
updateAID()
```

getRID()

This method is used to retrieve the RID associated with the EMV transaction.

Prototype

```
String[] com.vfi.smartpos.deviceservice.aidl.IPBOC.getRID ()
```

Code Snippet

```
demo returns from getRID()
{"9F0605A0000000039F220199DF050420291231DF03144ABFFD6B1C51212D05552E431C5B
17007D2F5E6DDF070101DF060101DF028180AB79FCC9520896967E776E64444E5DCDD6E136
11874F3985722520425295EEA4BD0C2781DE7F31CD3D041F565F747306EED62954B17EDABA3
A6C5B85A1DE1BEB9A34141AF38FCF8279C9DEA0D5A6710D08DB4124F041945587E20359BAB4
7B7575AD94262D4B25F264AF33DEDCF28E09615E937DE32EDC03C54445FE7E382777DF04030
00003"}
```

Parameters

None.

Return Values

Returns either of the two values:

RID: The method returns the available RID.

Null: Indicating that the RID is unavailable.

See Also

```
updateRID()
```

getProcessCardType()

This method is used to obtain the CTLS card type (In `onRequestOnlineProcess()` callback you can use this interface to obtain the CTLS card type) during an EMV transaction.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IPBOC.getProcessCardType ()
```

Parameters

None.

Return Values

The type of card currently being processed during an EMV transaction.

0:	No Type	Indicates an unknown or unsupported card type.
1:	JCB_CHIP	Represents a JCB chip card.
2:	JCB_MSD	Represents a JCB magnetic stripe card.
3:	JCB_Legcy	Represents a legacy JCB card.
4:	VISA_w1	Represents a Visa card using the w1 processing.
5:	VISA_w3	Represents a Visa card using the w3 processing.
6:	Master_EMV	Represents a MasterCard EMV chip card.
7:	Master_MSD	Represents a MasterCard magnetic stripe card.
8:	qPBOC_qUICS.	Represents a card using qPBOC/qUICS standards.

See Also

Refer to `PBOCHandler.onRequestOnlineProcess()` method under Section 2.2.6.

2.2.16 IInsertCardReader

Package: com.vfi.smartpos.deviceservice.aidl.IInsertCardReader

Overview:

This interface is a critical component for interacting with smart card readers. It includes methods for powering up and down the card reader, checking the presence of various types of cards, and executing the APDU commands. These functionalities are essential for applications that require secure and efficient card transactions.

Public Member Functions:

Modifier and Type	Method
boolean	**powerup** ()
boolean	**powerDown** ()
boolean	**isCardIn** ()
byte[]	**exchangeApdu** (in byte[] apdu)
boolean	**isPSAMCardExists** ()

Member Function Documentation:

powerUp()

This method is invoked for card readers that require the physical insertion of cards, often referred as contact readers. This method is essential for powering on the reader and preparing it to interact with cards inserted into a designated slot.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IInsertCardReader.powerUp ()
```

Parameters

None.

Return Values

A Boolean value:

true: Indicates that the card reader has been successfully powered up and is ready for use.

false: Indicates a failure to power up the card reader.

powerDown()

This method is called to power off the smart card reader, ensuring it is no longer active and conserving battery life or reducing wear on the device.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IInsertCardReader.powerDown ()
```

Parameters

None.

Return Values

A Boolean value:

true: Indicates that the card reader has been successfully powered down.

false: Indicates a failure to power down the card reader.

isCardIn()

This method is used to check whether a smart card (contact card or IC card) is currently inserted into the card reader.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IInsertCardReader.isCardIn ()
```

Parameters

None.

Return Values

A Boolean value:

true: Indicates that a card is currently available.

false: Indicates that a card is unavailable.

exchangeApdu()

This method is fundamental for the APDU data communication between an application and a smart card. This method enables the sending of commands to the card and receiving its responses, allowing for various interactions.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IInsertCardReader.exchangeApdu (in
byte[] apdu)
```

Parameters

apdu	The APDU command to be sent to the smart card, structured according to the command requirements.

Return Values

A byte array containing the response from the smart card. Possible value:

Valid Response: A non-null byte array indicating the data returned by the card.

Null: Indicates that no response was received from the smart card.

isPSAMCardExists()

This method is used to check whether a PSAM card is currently present in the card reader.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IInsertCardReader.isPSAMCardExists
()
```

Parameters

None.

Return Values

A Boolean value:

true: Indicates that the PSAM card is currently in place and detected by the card reader.

false: Indicates that the PSAM card is not in place or not detected.

2.2.17 IDeviceService

Package: com.vfi.smartpos.deviceservice.aidl.IDeviceService

Overview:

This interface acts as a central point in an Android application, facilitating access to various peripheral device services associated with a terminal. Each method in this interface connects to a specific peripheral, enabling operations related to that device.

Public Member Functions:

Modifier and Type	Method
IBeeper	**getBeeper** ()
ILed	**getLed** ()
ISerialPort	**getSerialPort** (String deviceType)
IScanner	**getScanner** (int cameraId)
IMagCardReader	**getMagCardReader** ()
IInsertCardReader	**getInsertCardReader** (int slotNo)
IRFCardReader	**getRFCardReader** ()
IPinpad	**getPinpad** (int kapId)
IPrinter	**getPrinter** ()
IPBOC	**getPBOC** ()

IDeviceInfo	**getDeviceInfo** ()
IExternalSerialPort	**getExternalSerialPort** ()
IUsbSerialPort	**getUsbSerialPort** ()
ISmartCardReader	**getSmartCardReader** (int slotNo)
IEMV	**getEMV** ()
IDukpt	**getDUKPT** ()
IFelica	**getFelica** ()
IUtils	**getUtils** ()
WirelessConnectListener	**getWirelessConnectionMgr** ()
IGirgitExt	**getGirgitExt** ()
IFFBase	**getFFBase** ()

Member Function Documentation:

getBeeper()

This method is called to provide access to a beeper service within an Android application. This allows applications to interact with the beeper for various purposes, such as alerting users or providing audio feedback.

Prototype

```
IBeeper com.vfi.smartpos.deviceservice.aidl.IDeviceService.getBeeper ()
```

Parameters

None.

Return Values

An object of type `IBeeper`. This object provides methods to control beeper functionalities within your Android application. Refer to `IBeeper.aidl`.

See Also

Refer to `IBeeper` interface under Section 2.2.2.

getLed()

This method is used to obtain the current status or configuration of the device's Light Emitting Diode (LED). This could include information such as whether the LED is on or off, its color, or its blinking pattern.

Prototype

```
ILed com.vfi.smartpos.deviceservice.aidl.IDeviceService.getLed ()
```

Parameters

None.

Return Values

An object of type `ILed`, which provides access to an interface that allows interaction with LED functionalities. Refer to `ILed.aidl`.

See Also

Refer to `ILed` interface Appendix A Supporting Classes 5.

getSerialPort()

This method is used to retrieve a serial port connection for a specific type of device, identified by the `deviceType` parameter. It enables communication between the system and various hardware components (such as printers, scanners, card readers, etc.) via serial communication protocols.

Get the `ISerialPort` for serial port connection for non X990 terminal. If the application wants to use more than one serial port during single execution e.g. SERIAL_0 and SERIAL_8, then the application must invoke this API with both the port first i.e. getSerialPort(SERIAL_0) followed by getSerialPort(SERIAL_8) before trying to open and initializing from either of the `ISerialPort` object.

Prototype

```
ISerialPort com.vfi.smartpos.deviceservice.aidl.IDeviceService.getSerialPort
(String deviceType)
```

Parameters

`deviceType`	A String representing the type of serial device user want to access.	
	Physical Serial TTY	These are Physical Serial TTY, ttyHSL0, ttyHSL1,"SERIAL_0" "SERIAL_1" "SERIAL_2" "SERIAL_3" "SERIAL_4" "SERIAL_5" "SERIAL_6" "SERIAL_7" e.g. "SERIAL_0" These serial ports (e.g., ttyHSL0) are hardware ports on devices such as the T650c and T650t, the cable should be connected to the RS232 port on the terminal side.
	Virtual Serial TTY	These are Virtual Serial TTY, ttyGS1, ttyGS2, "SERIAL_8" "SERIAL_9" "SERIAL_10" "SERIAL_11" "SERIAL_12" "SERIAL_13" "SERIAL_14" "SERIAL_15" e.g. "SERIAL_8". These

	virtual serial ports are established when devices like the T650c and T650t are connected via a USB Type-C to Type-A cable.
Dongle Serial TTY	These are Dongle Serial TTY, ttyACM0, ttyACM1, "SERIAL_16" "SERIAL_17" "SERIAL_18" "SERIAL_19" "SERIAL_20" "SERIAL_21" "SERIAL_22" "SERIAL_23" e.g. "SERIAL_16" These are virtual serial ports associated with dongles that create serial connections. For example, when you connect one terminal configured to use "SERIAL_8" (a virtual serial port), it can communicate with another terminal, which may then show up as a ttyACM port.
Dongle Serial TTY	These are Dongle Serial TTY, ttyUSB0, ttyUSB1,"SERIAL_32" "SERIAL_33" "SERIAL_34" "SERIAL_35" "SERIAL_36" "SERIAL_37" "SERIAL_38" "SERIAL_39" e.g. "SERIAL_32". These ports are used for USB to serial converters, allowing serial communication via USB connections. They are recognized on the terminal as /dev/ttyUSB devices. The USB side of the cable can be connected to the terminal either through the USB-A port on devices like the T650c or via the USB-C port with an On-The-Go (OTG) connector.

Return Values

An object `ISerialPort`, it indicates that a method will return an instance of the `ISerialPort` interface. Refer to `ISerialPort.aidl`.

See Also

`ISerialPort`

getScanner()

This method is used to retrieve an `IScanner` object associated with a specific camera, identified by the provided `cameraId`.

Prototype

```
IScanner com.vfi.smartpos.deviceservice.aidl.IDeviceService.getScanner (int
cameraId)
```

Parameters

`cameraId`	An integer parameter to specify which camera to use for scanning. 0: Represents the rear camera. 1: Represents the front camera.

Return Values

An object of type `IScanner` interface, which allows for operations related to scanning. Refer to `IScanner.aidl`.

See Also

Refer to `IScanner` interface under Section 2.2.12.

getMagCardReader()

This method is used to obtain a reference to a magnetic card reader, which can read data from magnetic stripe cards.

Prototype

```
IMagCardReader
com.vfi.smartpos.deviceservice.aidl.IDeviceService.getMagCardReader ()
```

Parameters

None.

Return Values

An `IMagCardReader` object, which provides the necessary functions to interact with magnetic stripe cards. Refer to `IMagCardReader.aidl`.

See Also

Refer to `IMagCardReader` Appendix A Supporting Classes 6.

getInsertCardReader()

This method is used to obtain an `IInsertCardReader` object for interacting with smart cards and PSAM cards.

Prototype

```
IInsertCardReader
com.vfi.smartpos.deviceservice.aidl.IDeviceService.getInsertCardReader (int
slotNo)
```

Parameters

`slotNo`	An integer parameter specifying the slot number for the card reader, allowing you to choose the appropriate reader for your needs. 0: Represents the IC card slot. 1: Represents the SAM1 card slot. 2: Represents the SAM2 card slot.

Return Values

An `IInsertCardReader` object, which provides the functionality to interact with the selected card slot. With this object, user can perform operations like inserting cards, reading data from the card, and handling actions specific to smart card and PSAM card. Refer to `IInsertCardReader.aidl.`

See Also

Refer to `IInsertCardReader` interface under Section 2.2.16.

getRFCardReader()

This method retrieves a reference to a contactless card reader interface (`IRFCardReader`), enabling the application to perform operations related to reading data from contactless cards (such as NFC-enabled cards).

Prototype

```
IRFCardReader com.vfi.smartpos.deviceservice.aidl.IDeviceService.getRFCardReader
()
```

Parameters

None.

Return Values

An `IRFCardReader` object. This object provides the necessary functions to interact with contactless cards. Refer to `IRFCardReader.aidl`.

See Also

`IRFCardReader`

getPinpad()

This method is used to obtain an `IPinpad` object, which allows interaction with a Pinpad for entering and processing PIN or other input.

Prototype

```
IPinpad com.vfi.smartpos.deviceservice.aidl.IDeviceService.getPinpad (int kapId)
```

Parameters

`kapId`	This parameter refers to the index of the key set for the Pinpad.

Return Values

An `IPinpad` object, which provides methods for managing Pinpad operations, such as accepting PIN entries and handling input. Refer to `IPinpad.aidl`.

See Also

Refer to `IPinpad` interface under Section 2.2.14.

getPrinter()

This method is used to obtain an `IPrinter` object, which allows interaction with a printer for printing tasks.

Prototype

```
IPrinter com.vfi.smartpos.deviceservice.aidl.IDeviceService.getPrinter ()
```

Parameters

None.

Return Values

An `IPrinter` object that provides functionalities for printing documents, images, or other data. Refer to `IPrinter.aidl`.

See Also

Refer to `IPrinter` interface under Section 2.2.13.

getPBOC()

This method is used to obtain an `IPBOC` object, which facilitates interactions with PBOC or EMV compliant card processing.

Prototype

```
IPBOC com.vfi.smartpos.deviceservice.aidl.IDeviceService.getPBOC ()
```

Parameters

None.

Return Values

An `IPBOC` object. This object allows you to manage transactions and interact with PBOC or EMV cards. Refer to `IPBOC.aidl`.

See Also

Refer to `IPBOC` interface under Section 2.2.15.

getDeviceInfo()

This method is used to obtain an `IDeviceInfo` object, which provides access to information about the device.

Prototype

```
IDeviceInfo com.vfi.smartpos.deviceservice.aidl.IDeviceService.getDeviceInfo ()
```

Parameters

None.

Return Values

An object of type `IDeviceInfo`, which provides access to various details about the device. Refer to `IDeviceInfo.aidl.`

See Also

Refer to `IDeviceInfo` interface under Section 2.2.17.

getExternalSerialPort()

This method is used to retrieve an instance of `IExternalSerialPort`, specifically for accessing the serial port in a docking station.

Prototype

```
IExternalSerialPort
com.vfi.smartpos.deviceservice.aidl.IDeviceService.getExternalSerialPort ()
```

Parameters

None.

Return Values

An instance of `IExternalSerialPort`. Refer to `IExternalSerialPort.aidl`.

See Also

Refer to `IExternalSerialPort` Appendix A Supporting Classes 4.

getUsbSerialPort()

This method is called to facilitate communication with USB serial devices, such as X9 or C520H, connected via an OTG cable.

Prototype

```
IUsbSerialPort
com.vfi.smartpos.deviceservice.aidl.IDeviceService.getUsbSerialPort ()
```

Parameters

None.

Return Values

An instance of `IUsbSerialPort`. Refer to `IUsbSerialPort.aidl`.

See Also

`IUsbSerialPort`

getSmartCardReader()

This method is used to access a smart card reader associated with a specific slot number on a device.

Prototype

```
ISmartCardReader
com.vfi.smartpos.deviceservice.aidl.IDeviceService.getSmartCardReader (int
slotNo)
```

Parameters

`slotNo`	An integer parameter specifying the slot number for the card reader.

Return Values

An instance of `ISmartCardReader`. Refer to `ISmartCardReader.aidl`.

See Also

- `getInsertCardReader()`
- Refer to `ISmartCardReader` interface under Section 2.2.11.

getEMV()

This method is used to obtain an `IEMV` object, which facilitates interactions with EMV card processing.

Prototype

```
IEMV com.vfi.smartpos.deviceservice.aidl.IDeviceService.getEMV ()
```

Parameters

None.

Return Values

An object of type `IEMV`, which provides the necessary functionalities for managing the EMV card processing. Refer to `IEMV.aidl`.

getDUKPT()

This method is used to obtain a DUKPT object, which facilitates secure key management for transactions.

Prototype

```
IDukpt com.vfi.smartpos.deviceservice.aidl.IDeviceService.getDUKPT ()
```

Parameters

None.

Return Values

An object of type `IDUKPT`, which provides functionalities for managing the DUKPT keys. Refer to `IDukpt.aidl`.

getFelica()

This method is used to obtain a Felica object, which allows interaction with Felica card technology.

Prototype

```
IFelica com.vfi.smartpos.deviceservice.aidl.IDeviceService.getFelica ()
```

Parameters

None.

Return Values

An instance of a Felica object which provides methods for interacting with Felica cards. Refer to `IFelica.aidl`.

getUtils()

This method is used to retrieve an object that contains a set of utility methods for performing common tasks or operations related to the device.

Prototype

```
IUtils com.vfi.smartpos.deviceservice.aidl.IDeviceService.getUtils ()
```

Parameters

None.

Return Values

An instance of a utilities object. Refer to `IUtils.aidl`.

getWirelessConnectionMgr()

This method is used to retrieve a `WirelessConnectListener` object, which allows to manage and interact with wireless network connections on the device.

Prototype

```
WirelessConnectListener
com.vfi.smartpos.deviceservice.aidl.IDeviceService.getWirelessConnectionMgr ()
```

Parameters

None.

Return Values

An instance of `WirelessConnectListener` object. Refer to `WirelessConnectListener.aidl`.

getGirgitExt()

This method is used to retrieve an instance of the `GirgitExt` object, which provides access to specific functionalities related to the Girgit system within the application.

Prototype

```
IGirgitExt com.vfi.smartpos.deviceservice.aidl.IDeviceService.getGirgitExt ()
```

Parameters

None.

Return Values

An instance of `GirgitExt` object.

getFFBase()

This method is used to retrieve an instance of the `IFFBase` object, which can then be used to interact with the base unit, involved in network communication.

Prototype

```
IFFBase com.vfi.smartpos.deviceservice.aidl.IDeviceService.getFFBase ()
```

Parameters

None.

Return Values

An instance of `IFFBase` object. Refer to `IFFBase.aidl`.

2.2.18 ISystemService

Package: com.vfi.smartpos.deviceservice.aidl.ISystemService

Overview:

The `ISystemService` interface is part of the AIDL in the POS system. It mainly provides a method for managing system-level operations.

Public Member Functions:

Modifier and Type	Method
void	**reboot** ()

Member Function Documentation:

reboot()

This method is called to initiate a system reboot. This action is necessary for applying system updates, recovering from errors for improved performance.

NOTE

When an application calls the `reboot()` method, it internally invokes a native reboot method that performs the actual reboot operation at the system level.

Prototype

```
void org.verifone.girgit.systemservice.ISystemService.reboot ()
```

Parameters

None.

Return Values

```
void
```

2.2.19 IFFBase

Package: com.vfi.smartpos.deviceservice.aidl.IFFBase

Overview:

This interface provides methods to retrieve the base IP address and check the connection status with the base unit.

Public Member Functions:

Modifier and Type	Method
byte[]	**getBaseIpAddress** ()
boolean	**isBaseConnected** ()

Member Function Documentation:

getBaseIpAddress()

This method is used to retrieve the IP address of the base unit in the POS system.

Prototype

```
byte com.vfi.smartpos.deviceservice.aidl.IFFBase.getBaseIpAddress ()
```

Parameters

None.

Return Values

Returns a byte, which is used to represent the IP address as a sequence of bytes.

isBaseConnected()

This method checks whether there is an active connection to the base unit, which ensures that the application can communicate with the base unit without issues.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IFFBase.isBaseConnected ()
```

Parameters

None.

Return Values

A Boolean value:

true: if the application is connected to the base unit (the connection is active).

false: if the connection to the base unit is not established or is no longer active.

2.3 Constant Definitions

2.3.1 BarcodeFormat

Package: com.vfi.smartpos.deviceservice.constdefine.BarcodeFormat

Overview:

The `BarcodeFormat` enum represents various types of barcode formats that can be processed by the system. It extends java.lang.Enum<BarcodeFormat> and implements the interfaces java.io.Serializable and java.lang.Comparable<`BarcodeFormat`>.

Enum Declaration:

```
public enum BarcodeFormat extends java.lang.Enum<BarcodeFormat>
```

Enum Constant Summary:

The following table lists all the constants in the `BarcodeFormat` enum, along with brief descriptions of each and constant detail:

Enum Constant	Description	Enum Constant Detail
AZTEC	Aztec 2D barcode format.	`public static final BarcodeFormat AZTEC`
CODABAR	Codabar 1D barcode format.	`public static final BarcodeFormat CODABAR`
CODE_39	Code 39 1D barcode format.	`public static final BarcodeFormat CODE_39`
CODE_93	Code 93 1D barcode format.	`public static final BarcodeFormat CODE_93`
CODE_128	Code 128 1D barcode format.	`public static final BarcodeFormat CODE_128`
DATA_MATRIX	Data Matrix 2D barcode format.	`public static final BarcodeFormat DATA_MATRIX`

EAN_8	EAN-8 1D barcode format.	`public static final BarcodeFormat EAN_8`
EAN_13	EAN-13 1D barcode format.	`public static final BarcodeFormat EAN_13`
ITF	ITF (Interleaved Two of Five) 1D barcode format.	`public static final BarcodeFormat ITF`
MAXICODE	MaxiCode 2D barcode format.	`public static final BarcodeFormat MAXICODE`
PDF_417	PDF417 2D barcode format.	`public static final BarcodeFormat PDF_417`
QR_CODE	QR Code 2D barcode format.	`public static final BarcodeFormat QR_CODE`
RSS_14	RSS-14 (Reduced Space Symbology) 1D barcode format.	`public static final BarcodeFormat RSS_14`
RSS_EXPANDED	RSS-Expanded 1D barcode format.	`public static final BarcodeFormat RSS_EXPANDED`
UPC_A	UPC-A (Universal Product Code Version) 1D barcode format.	`public static final BarcodeFormat UPC_A`
UPC_E	UPC-E 1D barcode format.	`public static final BarcodeFormat UPC_E`
UPC_EAN_EXTENSION	UPC/EAN extension 1D barcode format (e.g., UPC+2, UPC+5).	`public static final BarcodeFormat UPC_EAN_EXTENSION`

Method Summary:

The BarcodeFormat enum includes the following static methods:

Modifier and Type	Method
static BarcodeFormat	**valueOf** (java.lang.String name)
static BarcodeFormat[]	**values** ()

Method Detail:

valueOf()

This method is used to returns the enum constant of this type with the specified name. The name must exactly match one of the enum constant identifiers, and extraneous whitespace characters are not permitted.

Prototype

```
public static BarcodeFormat valueOf(java.lang.String name)
```

Parameters

`name`	A string that matches an enum constant name.

Return Values

The `BarcodeFormat` enum constant with the specified name.

Throws

java.lang.IllegalArgumentException: Thrown if there is no enum constant with the specified name.

java.lang.NullPointerException: Thrown if the argument is null.

values()

This method is used to return an array containing all the constants of the enum type, in the order they were declared. This allows you to easily iterate over all the enum constants.

Prototype

```
public static BarcodeFormat[] values()
```

Code Snippet

```
for (BarcodeFormat c: BarcodeFormat.values())
    System.out.println(c);
```

Parameters

None.

Return Values

An array of `BarcodeFormat` constants in the order they were declared.

2.3.2 ConstCheckCardListener

Package: com.vfi.smartpos.deviceservice.constdefine.ConstCheckCardListener

Overview:

The `ConstCheckCardListener` class provides a structure for extracting and managing various pieces of data from a card when it is swiped through a card reader. This can include both magnetic stripe data and chip data**.**

Class Declaration:

```
public class ConstCheckCardListener extends java.lang.Object
```

Main Class Summary:

This is the main class responsible for handling card swipe events.

Modifier and Type	Class and Description
class	**ConstCheckCardListener**

Constructor Summary:

The constructor is public and initializes an instance of the `ConstCheckCardListener` class. It sets up internal event handling or prepares the class for listening to card swipe events.

Constructor	Constructor Detail
ConstCheckCardListener()	public ConstCheckCardListener()

Nested Class Summary:

This nested class is intended to handle the actual swipe event. When a card is swiped through a POS terminal, this class processes the data captured from the card.

Modifier and Type	Class and Description
class	**onCardSwiped**

Constructor Summary:

The constructor initializes an instance of the `onCardSwiped` class, preparing it to capture and process the data from the swipe event.

Constructor	Constructor Detail
onCardSwiped()	public onCardSwiped()

Nested Class Summary:

This nested class handles data from the individual tracks of the magnetic stripe on the card (Track 1, Track 2, Track 3). Each track contains different pieces of information, such as account data, cardholder information, and more.

Modifier and Type	Class and Description
class	**track**

Constructor Summary:

This constructor initializes the `track` class, which captures and processes the data for each individual track of the magnetic stripe. These tracks contain different formats and types of data:

Constructor	Constructor Detail
track()	public track()

Static Public Attributes:

The class defines several static final fields that are used to represent specific pieces of information that can be extracted from the card data:

Modifier and Type	**Field Name**	Description	Field Detail
static final String	**KEY_PAN_String**	Represents the Primary Account Number (PAN).	`public static final java.lang.String KEY_PAN_String`

static final String	**KEY_TRACK1_String**	Contains data from Track 1 of the magnetic stripe.	`public static final java.lang.String KEY_TRACK1_String`
static final String	**KEY_TRACK2_String**	Contains data from Track 2 of the magnetic stripe.	`public static final java.lang.String KEY_TRACK2_String`
static final String	**KEY_TRACK3_String**	Contains data from Track 3 of the magnetic stripe.	`public static final java.lang.String KEY_TRACK3_String`
static final String	**KEY_SERVICE_CODE_String**	Represents the service code for the card.	`public static final java.lang.String KEY_SERVICE_CODE_String`
static final String	**KEY_EXPIRED_DATE_String**	Indicates the expiration date of the card.	`public static final java.lang.String KEY_EXPIRED_DATE_String`

2.3.3 ConstILed

Package: com.vfi.smartpos.deviceservice.constdefine.ConstILed

Overview:

The `ConstILed` class is part of a system created to manage LED indicators on a smart POS device. It defines constants or fields for various LED states, making it easier for developers to control or manage the behavior of LEDs programmatically.

Class Declaration:

```
public class ConstILed extends java.lang.Object
```

Main Class Summary:

The `ConstILed` class is part of a system designed to manage LED indicators on a smart POS device.

Modifier and Type	Class and Description
class	**ConstILed**

Constructor Summary:

The constructor `ConstILed()` is a default constructor used to initialize the `ConstILed` class.

Constructor	Constructor Detail
ConstILed()	public ConstILed()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Value, Description, and Field Detail:

Modifier and Type	Field Name	Value	Description	Field Detail
static final int	**BLUE**	1	Represents the blue LED.	`public static final int BLUE`
static final int	**YELLOW**	2	Represents the yellow LED.	`public static final int YELLOW`
static final int	**GREEN**	3	Represents the green LED.	`public static final int GREEN`
static final int	**RED**	4	Represents the red LED.	`public static final int RED`

2.3.4 ConstIPBOC

Package: com.vfi.smartpos.deviceservice.constdefine.ConstIPBOC

Overview:

The ConstIPBOC is part of a smart POS system's device service framework. It is created to handle integrated payment operations on smart cards. The ConstIPBOC class serves as a container for constants and operations related to IPBOC, which encompasses a variety of activities, including card detection, data import, EMV transaction processing, and other payment-related operations.

The class is used in scenarios where a smart POS terminal is involved in processing card transactions, particularly those that require secure communications, such as EMV transactions.

Class Declaration:

```
public class ConstIPBOC extends java.lang.Object
```

Main Class Summary:

The ConstIPBOC class provides constants and has a series of inner classes or methods that interact with card readers, manage transaction flows, and handle necessary card data processing.

Modifier and Type	Class and Description
class	**ConstIPBOC**

Constructor Summary:

This is a default constructor would initialize the object and prepare it to interact with other classes or methods of the payment processing system.

Constructor	Constructor Detail
ConstIPBOC()	public ConstIPBOC()

Nested Class Summary:

The ConstIPBOC class consists of several nested classes, each handling a distinct part of the EMV card transaction process.

Modifier and Type	Class and Description
class	**checkCard**

class	**importCardConfirmResult**
class	**importCertConfirmResult**
class	**inputOnlineResult**
class	**startEMV**
class	**updateAID**
class	**updateRID**

Nested Class Summary:

The `checkCard` class is used to perform operations related to checking the card’s state, type, or integrity.

Modifier and Type	Class and Description
class	**checkCard**

Constructor Summary:

The default constructor for `checkCard()` initializes an instance of this class.

Constructor	Constructor Detail
checkCard()	public checkCard()

Nested Class Summary:

The `cardOption` class is a nested class within `checkCard` and is created to encapsulate options related to a card, such as its type and its status.

Modifier and Type	Class and Description
class	**cardOption**

Constructor Summary:

The default constructor for `cardOption()` initializes an instance of this class.

Constructor	Constructor Detail
cardOption()	public cardOption()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Description, and Field Detail:

Modifier and Type	Field Name	Description	Field Detail
static final String	**KEY_MagneticCard_boolean**	The string value "`supportMagCard`" is used as a reference to indicate whether magnetic stripe cards are supported or not.	`public static final java.lang.String KEY_MagneticCard_boolean`
static final String	**KEY_SmartCard_boolean**	The string value `"supportICCard"` indicates whether smart card functionality is supported by the system.	`public static final java.lang.String KEY_SmartCard_boolean`
static final String	**KEY_Contactless_boolean**	The string value `"supportRFCard"` indicates whether RFID-based contactless cards (e.g., NFC cards) are supported.	`public static final java.lang.String KEY_Contactless_boolean`
static final boolean	**VALUE_supported**	A boolean value of true, which indicates that a feature support is available in the system.	`public static final boolean VALUE_supported`
static final boolean	**VALUE_unsupported**	A boolean value of false, which indicates that a feature support is not available in the system.	`public static final boolean VALUE_unsupported`

Nested Class Summary:

The `importCardConfirmResult` class is created to manage operations related to the confirmation of card imports.

Modifier and Type	Class and Description
class	**importCardConfirmResult**

Constructor Summary:

The default constructor for the `importCardConfirmResult` class, which initializes an instance of this class.

Constructor	Constructor Detail
importCardConfirmResult()	public importCardConfirmResult()

Nested Class Summary:

The `pass` nested class within `importCardConfirmResult` appears to be used for indicating successful card import operations.

Modifier and Type	Class and Description
class	**pass**

Constructor Summary:

The default constructor for the `pass` nested class. This constructor initializes an instance of the `pass` class, which could serve as an indicator that the card import was successfully completed.

Constructor	Constructor Detail
pass()	public pass()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Description, and Field Detail:

Modifier and Type	Field Name	Description	Field Detail
static final boolean	**allowed**	If the value is true, it indicates that the operation has been allowed.	`public static final boolean allowed`
static final boolean	**refused**	If the value is false, it indicates that the operation has been refused.	`public static final boolean refused`

Nested Class Summary:

This class handles the confirmation of card imports, or the result of a card import process in a system.

Modifier and Type	Class and Description
class	**importCertConfirmResult**

Constructor Summary:

This is the default constructor for the class. It initializes an instance of the `importCertConfirmResult` class, which holds or manages the result of confirming a card import.

Constructor	Constructor Detail
importCertConfirmResult()	public importCertConfirmResult()

Nested Class Summary:

The nested `option` class define a set of constant values that represent various possible outcomes for a card import confirmation operation.

Modifier and Type	Class and Description
class	**option**

Constructor Summary:

This is the default constructor for the nested `option` class, which might be used to initialize instances of the option class.

Constructor	Constructor Detail
option()	public option()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Value, Description, and Field Detail:

Modifier and Type	Field Name	Value	Description	Field Detail
static final int	**CANCEL**	0	Indicates that the card import was cancelled.	`public static final int CANCEL`
static final int	**CONFIRM**	1	Indicates that the card import was successfully confirmed.	`public static final int CONFIRM`
static final int	**NOTMATCH**	2	Indicates that the card import did not match the expected criteria (e.g., invalid card or data mismatch).	`public static final int NOTMATCH`

Nested Class Summary:

This class handles the results of an online transaction input or a response from an online transaction.

Modifier and Type	Class and Description
class	**inputOnlineResult**

Constructor Summary:

This is the default constructor for the `inputOnlineResult` class. It initializes an instance of this class.

Constructor	Constructor Detail
inputOnlineResult()	public inputOnlineResult()

Nested Class Summary:

The nested `onlineResult` class defines constants used to handle specific data or response fields in the context of an online transaction.

Modifier and Type	Class and Description
class	**onlineResult**

Constructor Summary:

This is the default constructor for the nested `onlineResult` class, which may initialize certain internal states if required.

Constructor	Constructor Detail
onlineResult()	public onlineResult()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Description, and Field Detail:

Modifier and Type	Field Name	Description	Field Detail
static final String	**KEY_isOnline_boolean**	Represents whether the transaction is processed online.	`public static finaljava.lang.String KEY_isOnline_boolean`
static final String	**KEY_respCode_String**	Represents the response code returned after processing the transaction.	`public static final java.lang.String KEY_respCode_String`
static final String	**KEY_authCode_String**	Represents the authorization code returned for a successful transaction.	`public static final java.lang.String KEY_authCode_String`

static final String	**KEY_field55_String**	Represents additional transaction data, such as PIN or other sensitive information (often used in EMV transactions).	`public static final java.lang.String KEY_field55_String`

Nested Class Summary:

This class handles the results of an online transaction input or a response from an online transaction.

Modifier and Type	Class and Description
class	**startEMV**

Constructor Summary:

This is the default constructor for the `inputOnlineResult` class. It initializes an instance of this class.

Constructor	Constructor Detail
startEMV()	public startEMV()

Nested Class Summary:

The `startEMV` class and its nested classes are likely part of a system that handles EMV card transactions.

Modifier and Type	Class and Description
class	**intent**
class	**processType**

Nested Class Summary:

The `intent` class is a data container or configuration class used to store various settings or values for a transaction or payment processing flow.

Modifier and Type	Class and Description

class	**intent**

Constructor Summary:

A constructor for the `intent` class, which might initialize default values or configurations for transaction parameters (e.g., card type, process code, support for specific technologies like QPBOC, etc.).

Constructor	Constructor Detail
intent()	public intent()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Description, and Field Detail:

Modifier and Type	Field Name	Description	Field Detail
static final String	**KEY_isPanConfirmOnSimpeProcess_boolean**	Key for confirming whether the PAN (Primary Account Number) should be verified during a simplified payment process. This could be useful for fraud prevention or as part of a security protocol.	`public static final java.lang.String KEY_isPanConfirmOnSimpeProcess_boolean`
static final String	**KEY_cardType_int**	Key used to specify the type of card being processed (e.g., smart card, contactless card).	`public static final java.lang.String KEY_cardType_int`
static final int	**VALUE_cardType_smart_card**	Value representing a smart card type (0). A smart card could refer to a chip card with embedded microprocessors.	`public static final int VALUE_cardType_smart_card`

static final int	**VALUE_cardType_contactless**	Value representing a contactless card type (1). This could be used to distinguish contactless payment methods (like NFC or RFID-based cards).	`public static final int VALUE_cardType_contactless`
static final String	**KEY_transProcessCode_byte**	Key for storing the transaction process code. This likely corresponds to the type of transaction (e.g., purchase, refund, authorization).	`public static final java.lang.String KEY_transProcessCode_byte`
static final String	**KEY_authAmount_long**	Key for specifying the authorization amount. This value is used during payment processing to define the amount to be authorized for the transaction.	`public static final java.lang.String KEY_authAmount_long`
static final String	**KEY_isSupportQ_boolean**	Key to indicate whether the system supports QPBOC (Quick Pass Based on Online Communication).	`public static final java.lang.String KEY_isSupportQ_boolean`
static final boolean	**VALUE_supported**	Value true to indicate that QPBOC is supported	`public static final boolean VALUE_supported`
static final boolean	**VALUE_unsupported**	Value false to indicate that QPBOC is not supported.	`public static final boolean VALUE_unsupported`
static final String	**KEY_isSupportSM_boolean**	Key to indicate whether Secure Messaging (SM) is supported.	`public static final java.lang.String KEY_isSupportSM_boolean`

		SM could be important for ensuring secure communication between devices.	
static final String	**KEY_isQPBOCForceOnline_boolean**	Key to determine whether QPBOC should always be processed online (forced to go online).	`public static final java.lang.String KEY_isQPBOCForceOnline_boolean`
static final boolean	**VALUE_forced**	Value true to indicate that QPBOC should always go online for processing.	`public static final boolean VALUE_forced`
static final boolean	**VALUE_unforced**	Value false to indicate that QPBOC does not need to be forced to go online and can be processed offline.	`public static final boolean VALUE_unforced`
static final String	**KEY_merchantName_String**	Key for the merchant's name. This is used to identify the merchant in the system or on receipts.	`public static final java.lang.String KEY_merchantName_String`
static final String	**KEY_merchantId_String**	Key for the merchant’s unique ID. This could be used to track transactions or identify the merchant for processing.	`public static final java.lang.String KEY_merchantId_String`
static final String	**KEY_terminalId_String**	Key for the terminal ID. Each POS terminal typically has a unique identifier used for transaction processing or reporting.	`public static final java.lang.String KEY_terminalId_String`

Nested Class Summary:

This class is used to group related constants together, specifically constants that define the types of transaction processes available in the system.

Modifier and Type	Class and Description
class	**processType**

Constructor Summary:

The constructor `processType()` is public, this is a default constructor provided for convenience.

Constructor	Constructor Detail
processType()	public processType()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Description, and Field Detail:

Modifier and Type	Field Name	Description	Field Detail
static final int	**full_process**	Represents a constant value of 1, which indicates a full transaction process.	`public static final int full_process`
static final int	**simple_process**	Represents a constant value of 2, which indicates a simplified or fast transaction process.	`public static final int simple_process`

Nested Class Summary:

The `updateAID` class is responsible for handling updates related to an Application Identifier (AID) during an online transaction process.

Modifier and Type	Class and Description

class	**updateAID**

Constructor Summary:

This is the default constructor of the `updateAID` class. It initializes an instance of this class, which is involved in updating the AID based on the current transaction data.

Constructor	Constructor Detail
updateAID()	public updateAID()

Nested Class Summary:

The `updateAID` class and its nested classes are part of a system that handles EMV card transactions.

Modifier and Type	Class and Description
class	**aidType**
class	**operation**

Nested Class Summary:

The `aidType` class represent the type of the Application Identifier (AID) an online transaction, especially in EMV systems.

Modifier and Type	Class and Description
class	**aidType**

Constructor Summary:

This constructor initializes default values or properties related to the AID, such as the application identifier and associated information.

Constructor	Constructor Detail
aidType()	public aidType()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Description, and Field Detail:

Modifier and Type	Field Name	Description	Field Detail
static final int	**smart_card**	Represents a constant value of 1, which indicates a smart card (or chip card) in the system.	`public static final int smart_card`
static final int	**contactless**	Represents a constant value of 2, which indicates a contactless card (NFC/RFID-enabled card).	`public static final int contactless`

Nested Class Summary:

The `operation` class could represent an operation or transaction type that is performed as part of an EMV card transaction.

Modifier and Type	Class and Description
class	**operation**

Constructor Summary:

The constructor public `operation()` initializes an object of the operation class.

Constructor	Constructor Detail
operation()	public operation()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Description, and Field Detail:

Modifier and Type	Field Name	Description	Field Detail

static final int	**append**	A value 1 represents the operation to append an item to a collection, list, or array.	`public static final int append`
static final int	**remove**	A value 2 represents the operation to remove an item from a collection or data structure.	`public static final int remove`
static final int	**clear**	A value 3 represents the operation to clear or empty the entire collection.	`public static final int clear`

Nested Class Summary:

The `updateRID` class seems to represent an operation or functionality related to updating or managing Routing ID (RID) data.

Modifier and Type	Class and Description
class	**updateRID**

Constructor Summary:

The constructor `updateRID()` is public, meaning instances of this class can be created from outside the class.

Constructor	Constructor Detail
updateRID()	public updateRID()

Nested Class Summary:

The `operation` class is a simple utility class that defines constants for various types of operations, which can be applied to a collection or set of data, such as adding, removing, or clearing data.

Modifier and Type	Class and Description
class	**operation**

Constructor Summary:

The `operation()` constructor is public, it allows you to instantiate an operation object if necessary.

Constructor	Constructor Detail
operation()	public operation()

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Description, and Field Detail:

Modifier and Type	Field Name	Description	Field Detail
static final int	**append**	A value 1 represents the operation to append an item to a collection, list, or array.	`public static final int append`
static final int	**remove**	A value 2 represents the operation to remove an item from a collection or data structure.	`public static final int remove`
static final int	**clear**	A value 3 represents the operation to clear or empty the entire collection.	`public static final int clear`

2.3.5 ConstIPinpad

Package: com.vfi.smartpos.deviceservice.constdefine.ConstIPinpad

Overview:

The `ConstIPinpad` class is part of an SDK or API that provides constants and settings specifically related to the Pinpad device integration within a POS system. A Pinpad device is used in payment systems to securely capture and process sensitive information such as PIN entries, card details, and cryptographic operations during financial transactions.

Class Declaration:

```
public class ConstIPinpad extends java.lang.Object
```

Main Class Summary:

This class is mainly focused on providing constant values that govern the Pinpad device's behavior in a POS environment.

Modifier and Type	Class and Description
class	**ConstIPinpad**

Constructor Summary:

The constructor is simple and initializes the `ConstIPinpad` class.

Constructor	Constructor Detail
ConstIPinpad()	public ConstIPinpad()

Nested Class Summary:

The `ConstIPinpad` class also defines some nested classes, which are used for specific tasks related to the Pinpad device.

Modifier and Type	Class and Description
class	**calculateByMSKey**
class	**startPinInput**

Nested Class Summary:

This nested class handles operations related to calculating or using the Master Session Key (MSK), which is used for cryptographic operations.

Modifier and Type	Class and Description
class	**calculateByMSKey**

Constructor Summary:

This is a default constructor for this nested class. This constructor would initialize the class for cryptographic operations that involve the Master Session Key (MSK).

Constructor	Constructor Detail
calculateByMSKey()	public calculateByMSKey()

Nested Class Summary:

The `calculateByMSKey` class also defines some nested classes, which are used for specific tasks related to the Pinpad device.

Modifier and Type	Class and Description
class	**algorithmMode**
class	**keyType**

Nested Class Summary:

The `algorithmMode` class defines the supported cryptographic algorithms that can be used in conjunction with the MSK.

Modifier and Type	Class and Description
class	**algorithmMode**

Constructor Summary:

The `algorithmMode()` constructor initializes the class but may not necessarily perform any action because the class primarily consists of constants related to cryptographic algorithms.

Constructor	Constructor Detail
algorithmMode()	public algorithmMode()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `algorithmMode` class:

Modifier and Type	Field Name	Value	Description	Field Detail
static final int	**Encrypt_ECB**	0x00	Encrypt data using Electronic Codebook (ECB) mode.	`public static final int Encrypt_ECB`
static final int	**Decrypt_ECB**	0x01	Decrypt data using ECB mode.	`public static final int Decrypt_ECB`
static final int	**Encrypt_CBC**	0x02	Encrypt data using Cipher Block Chaining (CBC) mode.	`public static final int Encrypt_CBC`
static final int	**Decrypt_CBC**	0x03	Decrypt data using CBC mode.	`public static final int Decrypt_CBC`

Nested Class Summary:

The `keyType` class is a nested class within the `calculateByMSKey` class of the `ConstIPinpad` class. This class defines key types used for encryption, decryption, and other cryptographic operations within a POS system.

Modifier and Type	Class and Description
class	**keyType**

Constructor Summary:

The `keyType` class has a default constructor. It does not take any parameters and does not perform any specific initialization beyond the default behavior, as it mainly serves to define the constants for key types.

Constructor	Constructor Detail
keyType()	public keyType()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `keyType` class:

Modifier and Type	Field Name	Value	Description	Field Detail
static final int	**Master_Key**	0x00	Represents a Master Key used for general encryption purposes.	`public static final int Master_Key`
static final int	**SM4_Master_Key**	0x01	Represents an SM4 Master Key, used for the SM4 encryption algorithm.	`public static final int SM4_Master_Key`
static final int	**AES_Master_Key**	0x02	Represents an AES Master Key, used for the AES encryption algorithm.	`public static final int AES_Master_Key`

Nested Class Summary:

The `startPinInput` class is a nested class within the `ConstIPinpad` class. It is responsible for handling the PIN input process on a Pinpad device.

Modifier and Type	Class and Description
class	**startPinInput**

Constructor Summary:

This is the default constructor for the `startPinInput` class. It is used to instantiate the class and could potentially perform any necessary initialization tasks.

Constructor	Constructor Detail
startPinInput()	public startPinInput()

Nested Class Summary:

The `startPinInput` class also defines some nested classes, which are used for specific tasks related to the Pinpad device.

Modifier and Type	Class and Description
class	**globleParam**
class	**param**

Nested Class Summary:

The `globleParam` class is a nested class within the `startPinInput` class. This nested class defines global parameters that are used for configuring or managing the PIN input process on a Pinpad device in a POS system.

Modifier and Type	Class and Description
class	**globleParam**

Constructor Summary:

This constructor is part of the `globleParam` nested class and is responsible for initializing any default settings or parameters related to the PIN input process.

Constructor	Constructor Detail
globleParam()	public globleParam()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `globleParam` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**KEY_Display_One_String**	"Display_One"	Represents the "1" key.	`public static final java.lang.String KEY_Display_One_String`

static final String	**KEY_Display_Two_String**	"Display_Two"	Represents the "2" key.	`public static final java.lang.String KEY_Display_Two_String`
static final String	**KEY_Display_Three_String**	"Display_Three"	Represents the "3" key.	`public static final java.lang.String KEY_Display_Three_String`
static final String	**KEY_Display_Four_String**	"Display_Four"	Represents the "4" key.	`public static final java.lang.String KEY_Display_Four_String`
static final String	**KEY_Display_Five_String**	"Display_Five"	Represents the "5" key.	`public static final java.lang.String KEY_Display_Five_String`
static final String	**KEY_Display_Six_String**	"Display_Six"	Represents the "6" key.	`public static final java.lang.String KEY_Display_Six_String`
static final String	**KEY_Display_Seven_String**	"Display_Seven"	Represents the "7" key.	`public static final java.lang.String KEY_Display_Seven_String`
static final String	**KEY_Display_Eight_String**	"Display_Eight"	Represents the "8" key.	`public static final java.lang.String KEY_Display_Eight_String`
static final String	**KEY_Display_Nine_String**	"Display_Nine"	Represents the "9" key.	`public static final java.lang.String KEY_Display_Nine_String`
static final String	**KEY_Display_Zero_String**	"Display_Zero"	Represents the "0" key.	`public static final java.lang.String KEY_Display_Zero_String`

static final String	**KEY_Display_Confirm_String**	"Display_Confirm"	Represents the Confirm button.	`public static final java.lang.String KEY_Display_Confirm_String`
static final String	**KEY_Display_BackSpace_String**	"Display_BackSpace"	Represents the Backspace button.	`public static final java.lang.String KEY_Display_BackSpace_String`

Nested Class Summary:

The `param` class within the `startPinInput` class of the `ConstIPinpad` class represents the parameters that are specific to the PIN input process.

Modifier and Type	Class and Description
class	**param**

Constructor Summary:

The `param` class might have a default constructor or constructors that initialize the session parameters with specific values.

Constructor	Constructor Detail
param()	public param()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `param` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**KEY_isOnline_boolean**	"isOnline"	Indicates whether the device is in online or offline mode.	`final String com.vfi.smartpos.deviceservice.constdefine.ConstIPi`

				npad.startPinInput.param.KEY_isOnline_boolean ="isOnline"
static final String	**KEY_pan_String**	"pan"	Represents the PAN for the card.	final String com.vfi.smartpos.deviceservice.constdefine.ConstIPinpad.startPinInput.param.KEY_pan_String ="pan"
static final String	**KEY_pinLimit_ByteArray**	"pinLimit"	Defines the PIN length or limit.	final String com.vfi.smartpos.deviceservice.constdefine.ConstIPinpad.startPinInput.param.KEY_pinLimit_ByteArray ="pinLimit"
static final String	**KEY_timeout_int**	"timeout"	Defines the timeout period for the PIN entry process.	public static final java.lang.String KEY_timeout_int
static final String	**KEY_desType_int**	"desType"	Defines the encryption algorithm type to be used.	final String com.vfi.smartpos.deviceservice.constdefine.ConstIPinpad.startPinInpu

				`t.param.KEY_desType_int ="desType"`
static final int	**Value_desType_3DES**	1	3DES encryptions mode. Default option.	`final int com.vfi.smartpos.deviceservice.constdefine.ConstIPinpad.startPinInput.param.Value_desType_3DES = 1`
static final int	**Value_desType_AES**	2	AES (Advanced Encryption Standard) encryption mode.	`public static final int Value_desType_AES`
static final int	**Value_desType_SM4**	3	SM4 encryption (Chinese Standard) mode.	`public static final int Value_desType_SM4`
static final int	**Value_desType_DUKPT_3DES**	4	DUKPT with 3DES encryption.	`public static final int Value_desType_DUKPT_3DES`
static final String	**KEY_promptString_String**	"promptString"	The prompt string displayed to the user during PIN input.	`public static final java.lang.String KEY_promptString_String`
static final String	**KEY_randomize_PED**	"randomize_PED"	Defines whether the PIN Entry Device (PED)	`public static final`

			layout should be randomized for security.	`java.lang.String KEY_ randomize_PED`

2.3.6 ConstIPrinter

Package: com.vfi.smartpos.deviceservice.constdefine.ConstIPrinter

Overview:

The `ConstIPrinter` class contains constants specifically for controlling or configuring the printer within that POS system. These constants could define various printer states, commands, or configuration parameters.

Class Declaration:

```
public class ConstIPrinter extends java.lang.Object
```

Main Class Summary:

The `ConstIPrinter` class is a utility class within the POS system's software package, created specifically to define constants for controlling and configuring the behavior of a printer in a POS system.

Modifier and Type	Class and Description
class	**ConstIPrinter**

Constructor Summary:

The `ConstIPrinter()` is simple and initializes the `ConstIPrinter` class, ensuring that the constants defined within the class are accessible.

Constructor	Constructor Detail
ConstIPrinter()	public ConstIPrinter()

Nested Class Summary:

The ConstIPrinter includes nested classes, each likely representing a different aspect or functionality of the printer:

Modifier and Type	Class and Description
class	**addBarCode**
class	**addQrCode**
class	**addText**
class	**addTextInLine**

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the ConstIPrinter class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**BUNDLE_PRINT_FONT**	"font"	Defines the font setting. Default is small font.	`final String com.vfi.smartpos.deviceservice.constdefine.ConstIPrinter.BUNDLE_PRINT_ALIGN = "align"`
static final String	**BUNDLE_PRINT_ALIGN**	"align"	Defines the alignment setting. Default is left-aligned.	`final String com.vfi.smartpos.deviceservice.constdefine.ConstIPr`

				inter.BUNDLE_PRINT_FONT = "font"

Nested Class Summary:

The `addBarCode` class is a nested class inside the `ConstIPrinter` class, and its purpose is likely to handle tasks related to printing barcodes on the POS printer.

Modifier and Type	Class and Description
class	**addBarCode**

Constructor Summary:

The `addBarCode()` constructor initializes the class, but it might not be frequently used for instantiation, especially if the class is focused on handling static methods or constants for barcode printing.

Constructor	Constructor Detail
addBarCode()	public addBarCode()

Nested Class Summary:

The `format` class is a nested class inside the `ConstIPrinter` class. Its primary role appears to be defining constants related to various formatting options for printing in the POS system.

Modifier and Type	Class and Description
class	**format**

Constructor Summary:

The constructor `format()` is public, indicating that it can be instantiated, though, like other utility or constant classes.

Constructor	Constructor Detail
format()	public format()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `format` class:

Modifier and Type	Field Name	Value	Description	Field Detail
static final String	**KEY_Alignment_int**	"align"	This is the key for the alignment setting, likely used to refer to alignment in configuration or command contexts.	`public static final java.lang.String KEY_Alignment_int`
static final int	**VALUE_Alignment_LEFT**	0	An integer value representing alignment option (left) that can be used when printing text.	`public static final int VALUE_Alignment_LEFT`
static final int	**VALUE_Alignment_CENTER**	1	An integer value representing alignment option (center) that can be used when printing text.	`public static final int VALUE_Alignment_CENTER`
static final int	**VALUE_Alignment_RIGHT**	2	An integer value representing alignment option (right) that can be used when printing text.	`public static final int VALUE_Alignment_RIGHT`

static final String	**KEY_PixelWidthMode_int**	"pixelPoint"	This is a key used for identifying pixel width configurations.	`public static final java.lang.String KEY_PixelWidthMode_int`
static final int	**VALUE_PixelWidthMode_AUTO**	1	It sets the printer to automatically determine the pixel width, possibly adjusting the content's width to fit within the printable area.	`public static final int VALUE_PixelWidthMode_AUTO`
static final String	**KEY_Height_int**	"height"	This is the key used to reference height settings in the context of printing.	`public static final java.lang.String KEY_Height_int`

Nested Class Summary:

The `addQrCode` class is another nested class within `ConstIPrinter`, created specifically for handling QR code printing in the POS system.

Modifier and Type	Class and Description
class	**addQrCode**

Constructor Summary:

The constructor `addQrCode()` is public, and it likely initializes the `addQrCode` class for use.

Constructor	Constructor Detail
addQrCode()	public addQrCode()

Nested Class Summary:

The `format` class is a nested class within the `ConstIPrinter` class. This class is created to provide a set of constants for formatting settings that can be used when working with printed content in a POS system.

Modifier and Type	Class and Description
class	**format**

Constructor Summary:

The constructor `format()` is public, meaning that the class can be instantiated. However, the `format` class is mainly created to hold static constants.

Constructor	Constructor Detail
format()	public format()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `format` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**KEY_Offset_int**	"offset"	This constant is used to represent the key for configuring the offset of printed content.	`public static final java.lang.String KEY_Offset_int`
static final String	**KEY_Height_String**	"expectedHeight"	This constant is used to represent the key for configuring the height of printed content, such as	`public static final java.lang.String KEY_Height_String`

			the height of a QR code or text.	

Nested Class Summary:

The `addText` class is a nested class within `ConstIPrinter` that is created specifically to handle the printing of text in a POS system.

Modifier and Type	Class and Description
class	**addText**

Constructor Summary:

The constructor `addText()` is public, meaning that an instance of the `addText` class can be created and accessed by other parts of the program. It initializes the `addText` class, preparing it to handle text printing tasks.

Constructor	Constructor Detail
addText()	public addText()

Nested Class Summary:

The `format` class is a nested class within the `ConstIPrinter` class, which is a part of the POS system package.

Modifier and Type	Class and Description
class	**format**

Constructor Summary:

The constructor `format()` is public, meaning that instances of the `format` class can be created. However, as a utility class holding constants.

Constructor	Constructor Detail
format()	public format()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `format` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**KEY_FontSize_int**	"font"	Key for specifying the font size.	`public static final java.lang.String KEY_FontSize_int`
static final int	**VALUE_FontSize_SMALL_16_16**	0	Small font size (16x16).	`public static final int VALUE_FontSize_SMALL_16_16`
static final int	**VALUE_FontSize_NORMAL_24_24**	1	Normal font size (24x24).	`public static final int VALUE_FontSize_NORMAL_24_24`
static final int	**VALUE_FontSize_NORMAL_DH_24_48_IN_BOLD**	2	Normal font size with double-height (24x48), in bold.	`public static final int VALUE_FontSize_NORMAL_DH_24_48_IN_BOLD`
static final int	**VALUE_FontSize_LARGE_32_32**	3	Large font size (32x32).	`public static final int VALUE_FontSize_LARGE_32_32`
static final int	**VALUE_FontSize_LARGE_DH_32_64_IN_BOLD**	4	Large font size with double-height (32x64), in bold.	`public static final int VALUE_FontSize_LARGE_DH_32_64_IN_BOLD`

static final int	**VALUE_FontSize_HUGE_48**	5	Huge font size (48x48).	`public static final int VALUE_FontSize_HUGE_48`
static final String	**KEY_Alignment_int**	"align"	Key for specifying the alignment of content.	`public static final java.lang.String KEY_Alignment_int`
static final int	**VALUE_Alignment_LEFT**	0	Left alignment.	`public static final int VALUE_Alignment_LEFT`
static final int	**VALUE_Alignment_CENTER**	1	Center alignment.	`public static final int VALUE_Alignment_CENTER`
static final int	**VALUE_Alignment_RIGHT**	2	Right alignment.	`public static final int VALUE_Alignment_RIGHT`
static final String	**KEY_StyleBold_boolean**	"bold"	Key for specifying whether the text should be in bold.	`public static final java.lang.String KEY_StyleBold_boolean`
static final boolean	**VALUE_StyleBold_YES**	true	Apply bold style.	`public static final boolean VALUE_StyleBold_YES`
static final boolean	**VALUE_StyleBold_NO**	false	Do not apply bold style.	`public static final boolean VALUE_StyleBold_NO`
static final String	**KEY_newline_boolean**	"newline"	Key for specifying if a newline (CRLF) should be appended.	`public static final java.lang.String KEY_newline_boolean`
static final boolean	**VALUE_newline_AppendCRLF**	true	Append a CRLF (Carriage Return +	`public static final boolean`

			Line Feed) at the end of the text.	`VALUE_newline_Append CRLF`
static final boolean	**VALUE_newline_NoCRLF**	false	Do not append CRLF.	`public static final boolean VALUE_newline_NoCRLF`

Nested Class Summary:

The `addTextInLine` class is a nested class within the `ConstIPrinter` utility class, which is part of the POS system package.

Modifier and Type	Class and Description
class	**addTextInLine**

Constructor Summary:

The `addTextInLine` is simple and initialize the class for use, but the actual functionality comes from using the constants and nested classes.

Constructor	Constructor Detail
addTextInLine()	public addTextInLine()

Nested Class Summary:

The `addTextInLine` class contains nested classes that further break down the functionality of in-line text handling.

Modifier and Type	Class and Description
class	**format**
class	**mode**

Nested Class Summary:

The `format` class within `addTextInLine` deals with the formatting rules for inline text. It may include constants for things like font size, alignment, and other layout aspects, but specifically for the in-line placement.

Modifier and Type	Class and Description
class	**format**

Constructor Summary:

The constructor `format()` is public and initializes the format class. However, as the class mainly contains static constants, it is not likely to require any specific instantiation.

Constructor	Constructor Detail
format()	public format()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `format` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**KEY_FontSize_int**	"fontSize"	Key for specifying the font size.	`public static final java.lang.String KEY_FontSize_int`
static final int	**VALUE_FontSize_SMALL_16_16**	0	Small font size (16x16).	`public static final int VALUE_FontSize_SMALL_16_16`
static final int	**VALUE_FontSize_NORMAL_24_24**	1	Normal font size (24x24).	`public static final int VALUE_FontSize_NORMAL_24_24`

static final int	**VALUE_FontSize_NORMAL_DH_24_48_IN_BOLD**	2	Normal font size with double-height (24x48), in bold.	`public static final int VALUE_FontSize_NORMAL_DH_24_48_IN_BOLD`
static final int	**VALUE_FontSize_LARGE_32_32**	3	Large font size (32x32).	`public static final int VALUE_FontSize_LARGE_32_32`
static final String	**KEY_StyleBold_boolean**	"bold"	Key for specifying whether the text should be in bold.	`public static final java.lang.String KEY_StyleBold_boolean`
static final boolean	**VALUE_StyleBold_YES**	true	Apply bold style.	`public static final boolean VALUE_StyleBold_YES`
static final boolean	**VALUE_StyleBold_NO**	false	Do not apply bold style.	`public static final boolean VALUE_StyleBold_NO`
static final String	**KEY_GlobalFont_String**	"fontStyle"	Key for specifying the font style (language).	`public static final java.lang.String KEY_GlobalFont_String`
static final String	**VALUE_GlobalFont_CHINESE**	"Chinese"	Set the font style to Chinese.	`public static final java.lang.String VALUE_GlobalFont_CHINESE`
static final String	**VALUE_GlobalFont_English**	"English"	Set the font style to English.	`public static final java.lang.String VALUE_GlobalFont_English`
static final String	**VALUE_GlobalFont_Arabic**	"Arabic"	Set the font style to Arabic.	`public static final java.lang.String VALUE_GlobalFont_Arabic`

Nested Class Summary:

The `mode` class defines layout modes for in-line text. Specifically, it provides constants for how the text should be divided across the line.

Modifier and Type	Class and Description
class	**mode**

Constructor Summary:

The `mode()` is Public constructor that initializes the class.

Constructor	Constructor Detail
mode()	public mode()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `format` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final int	**Devide_Equally**	0	Divide the text equally across the available space.	`public static final int Devide_Equally`
static final int	**Devide_flexible**	1	Allow flexible division, which might adjust based on content size or other factors.	`public static final int Devide_flexible`

2.3.7 ConstOnlineResultHandler

Package: com.vfi.smartpos.deviceservice.constdefine.ConstOnlineResultHandler

Overview:

The `ConstOnlineResultHandler` class is part of the POS system that is created to handle the results of online transactions, such as payment processing, account verification, or other types of online communications between the POS system and external services.

Class Declaration:

```
public class ConstOnlineResultHandler extends java.lang.Object
```

Main Class Summary:

This class contains static constants that represent various outcomes of online transactions responses.

Modifier and Type	Class and Description
class	**ConstOnlineResultHandler**

Constructor Summary:

The constructor is public, which means instances of the class can be created. However, this class contains static constants, user would not need to instantiate it directly.

Constructor	Constructor Detail
ConstOnlineResultHandler()	public ConstOnlineResultHandler()

Nested Class Summary:

The `onProccessResult` class is responsible for managing the results of an online transaction or interaction. It includes constants for different states (e.g., error codes, transaction statuses) and categories (e.g., data and result).

Modifier and Type	Class and Description
class	**onProccessResult**

Constructor Summary:

The `onProccessResult()` is a public constructor, though it's a utility class with constants, it is not be instantiated directly in most cases.

Constructor	Constructor Detail
onProccessResult()	public onProccessResult()

Nested Class Summary:

The `onProccessResult` class contains two nested classes, that further defines important constants and values that represent different aspects of the transaction result.

Modifier and Type	Class and Description
class	**data**
class	**result**

Nested Class Summary:

The `data` class within `onProccessResult` handles specific data related to the transaction process, such as transaction codes, scripts, and reversal data. These constants are used to store and refer to specific types of data within the transaction process.

Modifier and Type	Class and Description
class	**data**

Constructor Summary:

The `data()` is a public constructor, it initialize static constants, meaning this class is mostly used to hold static data rather than to create instances.

Constructor	Constructor Detail
data()	public data()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `data` class:

Modifier and Type	Field Name	Value	Description	Field Detail
static final String	**KEY_TC_DATA_String**	"TC_DATA"	Transaction data.	`public static final java.lang.String KEY_TC_DATA_String`
static final String	**KEY_SCRIPT_DATA_String**	"SCRIPT_DATA"	Script data for transaction processing.	`public static final java.lang.String KEY_SCRIPT_DATA_String`
static final String	**KEY_REVERSAL_DATA_String**	"REVERSAL_DATA"	Reversal data for transaction reversals.	`public static final java.lang.String KEY_REVERSAL_DATA_String`

Nested Class Summary:

The `result` class defines constants related to the outcome of an online transaction. These result codes likely represent different transaction states that the system can encounter.

Modifier and Type	Class and Description
class	**result**

Constructor Summary:

The `result()` is a public constructor, it's primarily used to define static constants rather than to instantiate objects.

Constructor	Constructor Detail
result()	public result()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `result` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final int	**TC**	0	Transaction code (Success).	`public static final int TC`
static final int	**Online_AAC**	1	Online Account Authentication Code (AAC).	`public static final int Online_AAC`
static final int	**Offline_TC**	101	Offline Transaction Code.	`public static final int Offline_TC`
static final int	**SCRIPT_NOT_EXECUTE**	102	Script execution failed.	`public static final int SCRIPT_NOT_EXECUTE`
static final int	**SCRIPT_EXECUTE_FAIL**	103	Script execution failure.	`public static final int SCRIPT_EXECUTE_FAIL`
static final int	**NO_SCRIPT**	104	No script available.	`public static final int NO_SCRIPT`
static final int	**TOO_MANY_SCRIPTNO**	105	Too many scripts.	`public static final int TOO_MANY_SCRIPTNO`

static final int	**TERMINATE**	106	Terminate the process.	`public static final int TERMINATE`
static final int	**ERROR**	107	General error.	`public static final int ERROR`

2.3.8 ConstPBOCHandler

Package: com.vfi.smartpos.deviceservice.constdefine.ConstPBOCHandler

Overview:

The `ConstPBOCHandler` class is part of the POS system, specifically within the software package. It is a utility class used for handling PBOC transactions involving smart cards or chip cards.

Class Declaration:

```
public class ConstPBOCHandler extends java.lang.Object
```

Main Class Summary:

The `ConstPBOCHandler` class is created to manage various aspects of a PBOC transaction process, which is used in EMV smart card transactions.

Modifier and Type	Class and Description
class	**ConstPBOCHandler**

Constructor Summary:

The `ConstPBOCHandler` class has a simple constructor. This constructor initializes the handler, possibly setting up resources or configuration necessary for the PBOC transaction flow.

Constructor	Constructor Detail
ConstPBOCHandler()	public ConstPBOCHandler()

Nested Class Summary:

There are several nested classes within `ConstPBOCHandler` that handle various stages of the PBOC transaction process:

Modifier and Type	Class and Description
class	**onConfirmCardInfo**
class	**onRequestOnlineProcess**
class	**onTransactionResult**

Nested Class Summary:

The nested class `onConfirmCardInfo` within the `ConstPBOCHandler` class play a critical role in handling the card information during a PBOC transaction process.

Modifier and Type	Class and Description
class	**onConfirmCardInfo**

Constructor Summary:

This constructor initializes an instance of the `onConfirmCardInfo` class. this class is responsible for handling the process where the system confirms the card information, after reading data from the smart card in a PBOC transaction.

Constructor	Constructor Detail
onConfirmCardInfo()	public onConfirmCardInfo()

Nested Class Summary:

The `info` class has a simple data container for holding different pieces of card-related information. It is used within the `onConfirmCardInfo` class to store the card details that are confirmed during the transaction process.

Modifier and Type	Class and Description
class	**info**

Constructor Summary:

This constructor initializes an instance of the `info` class, which holds key card-related data, and prepares it for usage in the confirmation process.

Constructor	Constructor Detail
info()	public info()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `info` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**KEY_PAN_String**	"PAN"	The PAN card number.	`public static final java.lang.String KEY_PAN_String`
static final String	**KEY_TRACK2_String**	"TRACK2"	Track 2 data.	`public static final java.lang.String KEY_TRACK2_String`
static final String	**KEY_CARD_SN_String**	"CARD_SN"	Card serial number.	`public static final java.lang.String KEY_CARD_SN_String`
static final String	**KEY_SERVICE_CODE_String**	"SERVICE_CODE"	Service code.	`public static final java.lang.String KEY_SERVICE_CODE_String`

static final String	**KEY_EXPIRED_DATE_String**	"EXPIRED_DATE"	Expiration date.	`public static final java.lang.String KEY_EXPIRED_DATE_String`

Nested Class Summary:

The nested class `onRequestOnlineProcess` play important roles in managing the online transaction process, specifically in the PBOC transactions.

Modifier and Type	Class and Description
class	**onRequestOnlineProcess**

Constructor Summary:

The constructor for the `onRequestOnlineProcess` class is a simple public constructor that initializes an instance of the class.

Constructor	Constructor Detail
onRequestOnlineProcess()	public onRequestOnlineProcess()

Nested Class Summary:

The `aaResult` class, nested within onRequestOnlineProcess, to handle the result of the online transaction request process.

Modifier and Type	Class and Description
class	**aaResult**

Constructor Summary:

This constructor initializes an instance of the `aaResult` class. It is used to handle and store the result of an online transaction request.

Constructor	Constructor Detail
aaResult()	public aaResult()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `aaResult` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**KEY_RESULT_int**	"RESULT"	Identifies the result of the transaction.	`public static final java.lang.String KEY_RESULT_int`
static final int	**VALUE_RESULT_QPBOC_ARQC**	201	Authorization Request Cryptogram (ARQC) result code for PBOC.	`public static final int VALUE_RESULT_QPBOC_ARQC`
static final int	**VALUE_RESULT_AARESULT_ARQC**	2	ARQC result for AARESULT (EMV standard).	`public static final int VALUE_RESULT_AARESULT_ARQC`
static final int	**VALUE_RESULT_PAYPASS_MAG_ARQC**	302	ARQC result for PayPass magnetic stripe transactions.	`public static final int VALUE_RESULT_PAYPASS_MAG_ARQC`
static final int	**VALUE_RESULT_PAYPASS_EMV_ARQC**	303	ARQC result for PayPass EMV transactions.	`public static final int VALUE_RESULT_PAYPASS_EMV_ARQC`

static final String	**KEY_ARQC_DATA_String**	"ARQC_DATA"	Stores ARQC data.	`public static final java.lang.String KEY_ARQC_DATA_String`
static final String	**KEY_REVERSAL_DATA_String**	"REVERSAL_DATA"	Stores reversal data.	`public static final java.lang.String KEY_REVERSAL_DATA_String`

Nested Class Summary:

The `onTransactionResult` class, along with its nested classes data and result, is responsible for managing and processing the results of a transaction after the transaction has been processed, either online or offline.

Modifier and Type	Class and Description
class	**onTransactionResult**

Constructor Summary:

The constructor `onTransactionResult()` initializes an instance of the `onTransactionResult` class. This class is responsible for handling and interpreting the result of a transaction, whether it is successful, declined, or encounters an error during the process.

Constructor	Constructor Detail
onTransactionResult()	public onTransactionResult()

Nested Class Summary:

There are two nested classes within `onTransactionResult` that are used to handle and store the result of a transaction, providing a structured way to capture both the transaction details and the status.

Modifier and Type	Class and Description
class	**data**

class	**result**

Nested Class Summary:

The `data` class is used to store the details of a transaction. It contains information that is needed to record or report on the transaction.

Modifier and Type	Class and Description
class	**data**

Constructor Summary:

The `data` class has a simple constructor, this constructor initializes an instance of the `data` class.

Constructor	Constructor Detail
data()	public data()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `data` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final String	**KEY_TC_DATA_String**	"TC_DATA"	A key used for storing or retrieving the Transaction Code (TC) data.	`public static final java.lang.String KEY_TC_DATA_String`
static final String	**KEY_REVERSAL_DATA_String**	"REVERSAL_DATA"	A key used for storing or retrieving Reversal Data (e.g., information related to cancelled or reversed transactions).	`public static final java.lang.String KEY_REVERSAL_DATA_String`

static final String	**KEY_ERROR_String**	"ERROR"	A key used for storing or retrieving Error Information (e.g., error codes or messages related to transaction failures).	`public static final java.lang.String KEY_ERROR_String`

Nested Class Summary:

The `result` class holds information about the outcome of the transaction. It is created to store the status or result of the transaction.

Modifier and Type	Class and Description
class	**result**

Constructor Summary:

This is the default constructor for the `result` class. It initializes an instance of the `result` class but given that the class primarily deals with static constants.

Constructor	Constructor Detail
result()	public result()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `result` class:

Modifier and Type	**Field Name**	Value	**Description**	Field Detail
static final int	**EMV_COMPLETE**	9	Represents the completion of an EMV transaction (chip card).	`public static final int EMV_COMPLETE`

static final int	**AARESULT_TC**	0	Represents a transaction result with TC.	`public static final int AARESULT_TC`
static final int	**AARESULT_AAC**	1	Represents a transaction result with Application Authentication Completed (AAC).	`public static final int AARESULT_AAC`
static final int	**EMV_CARD_BIN_CHECK_FAIL**	24	Represents a failure during the Card Bank Identification Number (BIN) Check in an EMV transaction.	`public static final int EMV_CARD_BIN_CHECK_FAIL`
static final int	**EMV_MULTI_CARD_ERROR**	26	Represents an error when multiple cards are detected during EMV transaction processing.	`public static final int EMV_MULTI_CARD_ERROR`

2.3.9 CTLSKernelID

Package: com.vfi.smartpos.deviceservice.constdefine.CTLSKernelID

Overview:

The `CTLSKernelID` class is part of the POS system software, specifically within the package. It plays a important role in defining kernel identifiers (Kernel IDs) for contactless transactions, used for handling NFC payments like contactless cards, smartphones, and other RFID-enabled devices.

Class Declaration:

```
public class CTLSKernelID extends java.lang.Object
```

Main Class Summary:

The `CTLSKernelID` class is part of the POS system used to define various kernel IDs for contactless transactions. These kernel IDs represent the contactless payment protocols for different payment brands. Additionally, it includes a custom kernel ID for proprietary systems.

Modifier and Type	Class and Description
class	**CTLSKernelID**

Constructor Summary:

The `CTLSKernelID` class has a simple constructor, and it initializes an instance of the `CTLSKernelID` class. Since the class is mostly used to define static constants, this constructor does not contain complex logic.

Constructor	Constructor Detail
CTLSKernelID()	public CTLSKernelID()

Static Public Attributes:

The following table lists Modifier and Type, Field, Value, Description and Field Detail for the `CTLSKernelID` class:

Modifier and Type	**Field Name**	Value	Description	Field Detail
static final int	CTLS_KERNEL_ID_01_VISA	1	Kernel ID for Visa contactless transactions.	`public static final int CTLS_KERNEL_ID_01_VISA`
static final int	CTLS_KERNEL_ID_02_MASTER	2	Kernel ID for Mastercard contactless transactions.	`public static final int CTLS_KERNEL_ID_02_MASTER`
static final int	CTLS_KERNEL_ID_03_VISA	3	Kernel ID for Visa contactless transactions (alternative kernel).	`public static final int CTLS_KERNEL_ID_03_VISA`

static final int	CTLS_KERNEL_ID_04_AE	4	Kernel ID for American Express contactless transactions.	`public static final int CTLS_KERNEL_ID_04_AE`
static final int	CTLS_KERNEL_ID_05_JCB	5	Kernel ID for JCB contactless transactions.	`public static final int CTLS_KERNEL_ID_05_JCB`
static final int	CTLS_KERNEL_ID_06_DISCOVER	6	Kernel ID for Discover contactless transactions.	`public static final int CTLS_KERNEL_ID_06_DISCOVER`
static final int	CTLS_KERNEL_ID_07_UNIONPAY	7	Kernel ID for UnionPay contactless transactions.	`public static final int CTLS_KERNEL_ID_07_UNIONPAY`
static final int	CTLS_KERNEL_ID_PURE	13455445	Custom or proprietary contactless payment kernel ID.	`public static final int CTLS_KERNEL_ID_PURE`

3. Logging

To enable logging in Girgit, ensure the following configurations and permissions are properly set:

1. Configuration Files

Girgit requires specific configuration files, namely `**GirgitJ_log.conf**` and `**GirgitN_log.conf**`.

Described below are the sample json values for these files.

```
{
  "schema_version": "1.0",
  "enabled": true,
  "mask": 255,
  "verbosity": 7,
  "output": "LOGAPI_ALL"
}
```

NOTE

These config files will be provided as .zip. Refer to Section 1.2.3 and 1.2.2.3 for more details on the installation process.

The following table lists the masks that can be set based on the prod/debug devices. For prod device, set 15 or 31 mask value.

Log level	**Bitmask**	**Log level description**
LOGAPI_EMERG	1	Emergency / system is unusable
LOGAPI_ALERT	2	An immediate action must be taken
LOGAPI_CRIT	4	Critical conditions

LOGAPI_ERROR	8	Error reporting
LOGAPI_WARN	16	Warning reporting
LOGAPI_NOTICE	32	Normal, but significant condition
LOGAPI_INFO	64	Regular info message
LOGAPI_TRACE	128	High verbosity messages

NOTE

The log mask values mentioned in the above table are added. For example, if LOGAPI_EMERG (1) and LOGAPI_ERROR (8) are needed then the mask value becomes 9 (1+8).

2. Android Permissions

Android 13 and up need to request MANAGE_EXTERNAL_STORAGE permission to function properly. Add the permission in the Android Manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
  <manifest ...>
  ...
  <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"/>
  ...
  </manifest>
</xml>
```

Also, this permission has to be requested programmatically for it to be granted (adding this permission in the android manifest is not enough). Code similar to the following should be used to request those permissions:

```
private static final int STORAGE_PERMISSION_CODE = 23;
```

```
public boolean checkStoragePermissions() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
        //Android is 11 (R) or above
        return Environment.isExternalStorageManager();
    }
}

private void requestForStoragePermissions() {
    //Android is 11 (R) or above
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
        try {
            Intent intent = new Intent();
            intent.setAction(Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION);
            Uri uri = Uri.fromParts("package", this.getPackageName(), null);
            intent.setData(uri);
            this.startActivity(intent);
        } catch (Exception e) {
            Intent intent = new Intent();
            intent.setAction(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION);
            this.startActivity(intent);
        }
    }
}
```

Appendix A: Supporting Classes

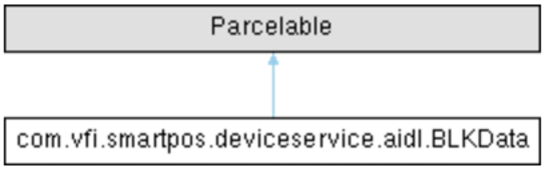
1. BLKData

Package: com.vfi.smartpos.deviceservice.aidl.BLKData

Overview:

The `BLKData` class implements the Parcelable interface, allowing it to be serialized and passed between different Android components. It encapsulates the data related to a card block and its corresponding transaction serial number.

Inheritance diagram:



NOTE

BLKData is a java class and is called using the constructor `BLKData(byte[] cardblk, byte sn).`

Constructor:

BLKData(byte[] cardblk, byte sn)

This parameterized constructor initializes an instance of the `BLKData` class with the specified card block data and transaction serial number.

Parameters

`cardblk`	The byte array representing the card block data.
`sn`	The transaction serial number associated with the card block.

Member Functions:

Modifier and Type	Method
byte[]	**getCardblk** ()
byte	**getSn** ()

Member Function Documentation:

getCardblk()

This method retrieves the byte array representing the card block data. This could include information related to a payment card or some other data block.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.BLKData.getCardblk()
```

Parameters

None.

Return Values

A byte array that contains the card block data.

getSn()

Retrieves the transaction serial number associated with the card block. This is used for tracking or processing the transaction.

Prototype

```
byte com.vfi.smartpos.deviceservice.aidl.BLKData.getSn()
```

Parameters

None.

Return Values

A byte array that contains the serial number.

2. CandidateAppInfo

Package: com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo

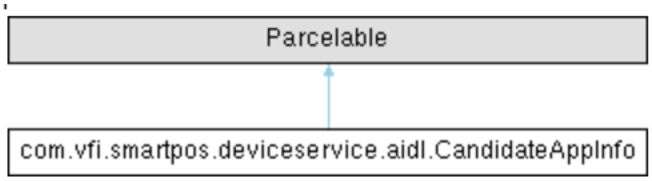
Overview:

This class is constructed to encapsulate and manage information about candidate applications of EMV cards. It includes methods to set and retrieve various attributes like Application Identifier (AID), Application Label, Application Priority Number (APN), Application Priority Indicator (APID), language preferences, and Issuer Country Code Index (CT Index).

Key Responsibilities:

- Application Metadata: It manages various metadata associated with the application, such as the Application Label, APN, and APID.
- Language Preferences: The class handles language preferences, ensuring that the application can adapt to different linguistic settings based on user or system preferences.
- Issuer Country Code Index: It stores and retrieves the Issuer CT Index and its associated flag, which are critical for applications that need to manage country-specific data or operations.

Inheritance diagram:



Public Member Functions:

Modifier and Type	Method
void	**setAID** (byte[] aid)
byte[]	**getAID** ()
void	**setAppLabel** (byte[] appLabel)
byte[]	**getAppLabel** ()
void	**setAPN** (byte[] apn)
byte[]	**getAPN** ()
void	**setAPID** (byte apid)
byte	**getAPID** ()
void	**setAPIDFlag** (byte flag)
byte	**getAPIDFlag** ()
void	**setLangPref** (byte[] langPref)
byte[]	**getLangPref** ()
void	**setIssCTIndex** (byte index)

byte	**getIssCTIndex** ()
void	**setIssCTIndexFlag** (byte flag)
byte	**getIssCTIndexFlag** ()

Member Function Documentation:

setAID()

This method is used to set the AID for the candidate application. The AID is a unique identifier crucial for identifying a specific application on a card.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.setAID(byte[] aid)
```

Parameters

aid	A byte array representing the AID of the application.

Return Values

```
void
```

getAID()

This method is used to retrieve the AID of the candidate application.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.getAID()
```

Parameters

None.

Return Values

A byte array representing the AID of the application.

setAppLabel

This method is used to set the application label for the candidate application. This label is used to identify or describe the application of a card.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.setAppLabel(byte[]
appLabel)
```

Parameters

`appLabel`	A byte array that contains the label of the application.

Return Values

```
void
```

getAppLabel()

This method is used to retrieve the application label for the candidate application.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.getAppLabel()
```

Parameters

None.

Return Values

A byte array representing the application label.

setAPN()

This method is used to set the APN for the candidate application. The APN is crucial for establishing a data connection to a mobile network. This method allows you to configure the APN settings required for the application to communicate over cellular networks.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.setAPN(byte[] apn)
```

Parameters

`apn`	A byte array representing the APN.

Return Values

```
void
```

getAPN()

This method is used to retrieve the APN for the candidate application.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.getAPN()
```

Parameters

None.

Return Values

A byte array representing the APN.

setAPID()

This method is used to set the APID for the candidate application. This identifier helps prioritize applications during selection.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.setAPID(byte[] apid)
```

Parameters

`apid`	A byte representing the APID.

Return Values

```
void
```

getAPID()

This method is used to retrieve the APID for the candidate application.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.getAPID()
```

Parameters

None.

Return Values

A byte representing the APID.

setAPIDFlag()

This method is used to set the flag associated with the APID. This flag is used to indicate whether the application is active or to represent other relevant states.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.setAPIDFlag(byte[]
flag)
```

Parameters

`flag`	A byte representing the APID flag.

Return Values

```
void
```

getAPIDFlag()

This method is used to retrieve the flag associated with the APID.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.getAPIDFlag()
```

Parameters

None.

Return Values

A byte representing the APID flag.

setLangPref()

This method is used to set the language preference for the candidate application. This is useful in cards supporting multiple languages, ensuring that the application will display content in the correct language for the user.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.setLangPref(byte[]
langPref)
```

Parameters

`langPref`	A byte array representing the language preference.

Return Values

```
void
```

getLangPref()

This method is used to retrieve the language preference for the candidate application.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.getLangPref()
```

Parameters

None.

Return Values

A byte array representing the language preference.

setIssCTIndex()

This method is used to set the IssCTIndex. This index represents the geographical region or country of the application’s issuer.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.setIssCTIndex(byte[]
index)
```

Parameters

`index`	A byte representing the IssCTIndex.

Return Values

```
void
```

getIssCTIndex()

This method is used to retrieve the IssCTIndex.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.getIssCTIndex()
```

Parameters

None.

Return Values

A byte representing the IssCTIndex.

setIssCTIndexFlag()

This method is used to set the flag associated with the IssCTIndex. This flag is used to indicate whether the country code index is valid for the current setting.

Prototype

```
void
com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.setIssCTIndexFlag(byte[]
index)
```

Parameters

`index`	A byte representing the flag for the IssCTIndex.

Return Values

```
void
```

getIssCTIndexFlag()

This method is used to retrieve the flag associated with the IssCTIndex.

Prototype

```
byte[]
com.vfi.smartpos.deviceservice.aidl.CandidateAppInfo.getIssCTIndexFlag()
```

Parameters

None.

Return Values

A byte representing the flag for the IssCTIndex.

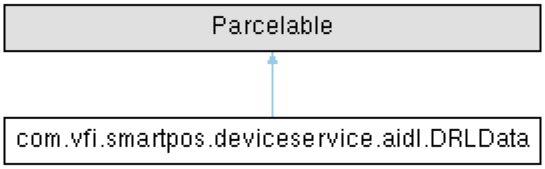
3. DRLData

Package: com.vfi.smartpos.deviceservice.aidl.DRLData

Overview:

The `DRLData` class is created to encapsulate information about the Dynamic Reader Limit (DRL) configuration and any relevant data about the transaction or device-related limits. This class enables you to retrieve, set, and monitor dynamic limits on the card reader or terminal.

Inheritance diagram:



Constructor:

DRLData(byte[] drlID, byte[] clssFloorLimit, byte[] clssTransLimit, byte[] cvmRequiredLimit)

This is the parameterized constructor for the `DRLData` class. It is used to initialize an instance of the class, with the specified dynamic reader limits for DRL ID, CLSS floor limit, CLSS transaction limit, and CVM required limit.

Parameters

`drlID`	A byte array representing the DRL ID, which uniquely identifies the set of dynamic limits.
`clssFloorLimit`	A byte array representing the floor limit for contactless transactions.
`clssTransLimit`	A byte array representing the transaction limit for contactless transactions.
`cvmRequiredLimit`	A byte array representing the CVM required limit.

Public Member Functions:

Modifier and Type	Method
byte[]	**getDrlID** ()
byte[]	**getClssFloorLimit** ()
byte[]	**getClssTransLimit** ()
byte[]	**getCvmRequiredLimit** ()

Member Function Documentation:

getDrlID()

This method is used to retrieve the DRL ID, which is a unique identifier for a particular set of dynamic reader limits.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.DRLData.getDrlID()
```

Parameters

None.

Return Values

The method returns a byte array representing the DRL ID.

getClssFloorLimit()

This method retrieves the CLSS Floor Limit, which is the minimum transaction amount, that can be processed using Contactless transactions.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.DRLData.getClssFloorLimit()
```

Parameters

None.

Return Values

Returns a byte array that represents the floor limit value for CLSS transactions in a format (e.g., as a BigDecimal or an integer encoded as bytes).

getClssTransLimit()

This method retrieves the CLSS Transaction Limit, which is the maximum transaction amount allowed for Contactless (CLSS) transactions.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.DRLData.getClssTransLimit()
```

Parameters

None.

Return Values

Returns a byte array representing the transaction limit for contactless transactions.

getCvmRequiredLimit()

This method retrieves the CVM Required Limit. This represents the limit above which CVM is required for the transaction to proceed.

Prototype

```
 byte[] com.vfi.smartpos.deviceservice.aidl.DRLData.getCvmRequiredLimit()
```

Parameters

None.

Return Values

Returns a byte array that encodes the limit above which CVM is required.

4. IExternalSerialPort

Package: com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort

Overview:

This interface provides methods for interacting with external serial ports on a device. It is creäted to manage communication through serial ports, which can be used to connect various peripherals such as Pinpads, barcode scanners, printers, or other serial-based devices.

Public Member Functions:

Modifier and Type	Method
int	**setExtPinpadPortMode** (int portMode)
boolean	**isExternalConnected** ()
boolean	**openSerialPort** (int portNum, in **SerialDataControl** dataControl)

int	**writeSerialPort** (int portNum, in byte[] writeData, int dataLength)
int	**readSerialPort** (int portNum, out byte[] readData, int dataLength)
int	**safeWriteSerialPort** (int portNum, in byte[] writeData, int Length, long timeoutMs)
int	**safeReadSerialPort** (int portNum, out byte[] readData, int Length, long timeoutMs)
void	**closeSerialPort** (int portNum)

Member Function Documentation:

setExtPinpadPortMode()

This method is used to set or configure the mode of the Pinpad port. The Pinpad port can operate in different modes, such as:

- Transparent Transmission Mode.
- External PP1000V3 Pinpad Mode.
- External PP1000V3 Non-contact Mode.

The default mode is transparent transmission, and the chosen mode will remain active even after the device is powered off. However, when one mode is set, it will disable other function interfaces.

NOTE

Switching modes will require calling this function again.

Prototype

```
int
com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort.setExtPinpadPortMode(int
portMode)
```

Parameters

`portMode`	Specifies the function mode for the Pinpad port. If an undefined value is provided, the mode will not be changed, and the current mode will be returned instead.

Return Values

Returns the current function mode of the Pinpad port.

isExternalConnected()

This method checks whether the POS system is connected to an external base device. If the base device is successfully connected, it returns true; otherwise, it returns false.

Prototype

```
boolean
com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort.isExternalConnected()
```

Parameters

None.

Return Values

A Boolean value:

true: If the POS system is connected to the base device.

false: If the POS system is not connected to the base device.

openSerialPort()

This method opens the specified serial port and configures its data control properties, including baud rate, data bits, stop bits, and parity. If the port is already open, it will return true without performing any further actions.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort.openSerialPort
(int portNum, in SerialDataControl dataControl)
```

Parameters

`portNum`	Specifies the port to be opened. 0: Refer to the Pinpad port (with 5V power supply). 1: Refer to the RS232 (standard serial port);
`dataControl`	Defines the data control settings for the serial port, such as baud rate, data bits, stop bits, and parity.

Return Values

A Boolean value:

true: The serial port was successfully opened.

false: The attempt to open the serial port failed. In this case, read/write operations will return 0.

writeSerialPort()

This method is used for writing data to a specified serial port. It initiates non-blocking data transmission to devices via serial communication.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort.writeSerialPort
(int portNum, in byte[] writeData, int dataLength)
```

Parameters

`portNum`	Specifies which serial port to use for the communication. 0: Refer to the Pinpad port (with 5V power supply). 1: Refer to the RS232 (standard serial port).
`writeData`	The cache of data to be transferred.
`dataLength`	The length of data to be transferred.

Return Values

The function returns the actual length of the data that was transmitted. A return value of 0 indicates that no data was sent. If the return value is a negative number, it signifies an error.

readSerialPort()

This method is used to read the available data from the specified serial port and returns the length of the data that was successfully read. It facilitates non-blocking reads from a serial port.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort.readSerialPort
(int portNum, out byte[] readData, intdataLength)
```

Parameters

`portNum`	Specifies which serial port to use for the communication. 0: Refer to the Pinpad port (with 5V power supply). 1: Refer to the RS232 (standard serial port).
`readData`	Holds the buffer in which the data must be read.
`dataLength`	The length of the data user wants to read (cannot exceed the buffer size).

Return Values

Returns the length of the data that is read. A return value of 0 indicates that no data was sent. If the return value is a negative number, it signifies an error.

safeWriteSerialPort()

This method is used for writing data to a specified serial port in a blocking manner. This means the method will wait until the data transmission is either completed successfully or the operation times out. The function returns the length of the data that was successfully transmitted.

Prototype

```
int
com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort.safeWriteSerialPort
(int portNum, in byte[] writeData, int Length, long timeoutMs)
```

Parameters

`portNum`	Specifies which serial port to use for the communication. 0: Refer to the Pinpad port (with 5V power supply). 1: Refer to the RS232 (standard serial port).
`writeData`	The cache of data to be transferred.
`dataLength`	The length of data to be transferred.
`timeoutMs`	Timeout period in milliseconds.

Return Values

Returns the length of the data that occurred. A return value of 0 indicates that no data has been sent out after the timeout. If the return value is a negative number, it signifies an error.

safeReadSerialPort()

This method is used for blocking reads from a serial port. It attempts to read data from the specified port, waiting until data is available or the timeout period expires. The function then returns the length of the data that was successfully read.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort.safeReadSerialPort
(int portNum, out byte[] readData, int Length, long timeoutMs)
```

Parameters

`portNum` Specifies which serial port to use for communication.

0: Refer to the Pinpad port (with 5V power supply).

1: Refer to the RS232 (standard serial port).

`readData` Holds the buffer in which the data must be read.

`dataLength` The length of the data user wants to read (cannot exceed the buffer size).

`timeoutMs` Timeout period in milliseconds.

Return Values

Returns the length of the data that is read. A return value of 0 indicates that no data has been sent out after the timeout. If the return value is a negative number, it signifies an error.

closeSerialPort()

This method is used to close a specified serial port, effectively turning off or disabling the communication on that port.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IExternalSerialPort.closeSerialPort
(int portNum)
```

Parameters

`portNum`	Specifies which serial port to close. 0: Refer to the Pinpad port (with 5V power supply). 1: Refer to the RS232 (standard serial port).

Return Values

```
void
```

5. ILed

Package: com.vfi.smartpos.deviceservice.aidl.ILed

Overview:

The `ILed` interface provides methods to control the LED lights on a device. It allows for operations such as turning LEDs on, turning them off, and controlling their states.

Public Member Functions:

Modifier and Type	Method
void	**turnOn** (int led)

void	**turnOff** (int led)
void	**ledControl** (byte led, byte status)

Member Function Documentation:

turnOn()

This method is used to turn on a specific LED on the device, based on the provided LED index.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.ILed.turnOn (int led)
```

Parameters

`led`	The index of the LED that needs to be turned on. 1: Blue LED. 2: Yellow LED. 3: Green LED. 4: Red LED.

Return Values

```
void
```

turnOff()

This method is used to turn off a specific LED on the device, based on the provided LED index.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.ILed.turnOff (int led)
```

Parameters

`led`	The index of the LED that needs to be turned off. 1: Blue LED. 2: Yellow LED. 3: Green LED. 4: Red LED.

Return Values

```
void
```

ledControl()

This method is used to control the state of a specific LED on the device, allowing user to turn it on or off based on the provided status.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.ILed.ledControl (byte led,
byte status)
```

Parameters

`led`	The index of the LED to control. The values correspond to specific LED colors:

0x01:	Blue LED.
0x02:	Yellow LED.
0x03:	Green LED.
0x04:	Red LED.

`status` The status of the LED.

0:	Turn the LED off (close).
1:	Turn the LED on (open).

Return Values

`void`

6. IMagCardReader

Package: com.vfi.smartpos.deviceservice.aidl.IMagCardReader

Overview:

This interface provides methods to interact with a magnetic card reader device. These methods allow user to initiate card searches, stop searches, and enable specific tracks for reading magnetic data from cards.

Public Member Functions:

Modifier and Type	Method
void	**searchCard** (int timeout, **MagCardListener** listener)
void	**stopSearch** ()

void	**enableTrack** (int trkNum)

Member Function Documentation:

searchCard()

This method is used to search for a magnetic card in a non-blocking manner, allowing the system to detect the presence of a card while continuing other operations.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IMagCardReader.searchCard (int
timeout, MagCardListener listener)
```

Parameters

`timeout`	Specifies the timeout duration in seconds for searching the card. If a card is not detected within this period, the search will stop.
`listener`	A callback listener that will be triggered when a card is swiped. The listener handles the events related to the card swipe. Refer to `MagCardListener`.

Return Values

```
void
```

stopSearch()

This method is used to stop the ongoing card search process.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IMagCardReader.stopSearch ()
```

Parameters

None.

Return Values

```
void
```

enableTrack()

This method is used to enable a specific track for reading on the magnetic card reader.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IMagCardReader.enableTrack (int
trkNum)
```

Parameters

`trkNum`	Specifies which track data of the magnetic card must be read. It can be either of the following: Track 1 data Track 2 data

Return Values

```
void
```

7. UPCardListener

Package: com.vfi.smartpos.deviceservice.aidl.UPCardListener

Overview:

This interface is used for managing events related to Uni-Pay mobile cards. It handles both successful card read operations and various error conditions, assisting developers in creating robust card processing applications.

Public Member Functions:

Modifier and Type	Method
void	**onRead** (out Bundle data)
void	**onError** (int error, String message)

Member Function Documentation:

onRead()

This method is used to handle the successful reading of card data, typically after a smart card (such as an EMV, NFC, or chip card) has been inserted into the reader and the reader has successfully processed the card.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.UPCardListener.onRead (out Bundle data)
```

Parameters

`data` A Bundle object containing key-value pairs with the card information that was successfully read. The data typically includes the following:

PAN(String)	The PAN card number.
TRACK2(String)	Track 2 data.
TRACK3(String)	Track 3 data.
CARD_SN(String)	The serial number of the card.

EXPIRED_DATE(String)	The expiration date of the card.
TLV_DATA(String)	The TLV format used to structure the card data. Common tags include: DF32, DF33, DF34.

Return Values

`void`

onError()

This method is invoked when an error occurs during the card reading process. It provides error details that can be used to understand and handle the issue.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.UPCardListener.onError (int error,
String message)
```

Parameters

`error` An integer value that represents the error code, indicating the type of error that occurred. The error codes include:

ERROR_DETECT_CARD (1)	An error occurred while detecting the card.
ERROR_READ_SN (2)	An error occurred while reading the SN of the card.
ERROR_READ_TRACK (3)	An error occurred while reading the track information from the card.
ERROR_SERVICE_CRASH (4)	The service crashed.

ERROR_NULL_DRIVER (5)	An error occurred when the contactless driver is null or missing.

`message` A description of the error, providing additional context for troubleshooting.

Return Values

```
void
```

8. MagCardListener

Package: com.vfi.smartpos.deviceservice.aidl.MagCardListener

Overview:

This interface defines the callbacks for managing the outcomes of swiping a magnetic card on a device. It assists in handling successful swipe operations, errors during card reading, and timeouts.

Public Member Functions:

Modifier and Type	Method
void	**onSuccess** (in Bundle track)
void	**onError** (int error, String message)
void	**onTimeout** ()

Member Function Documentation:

onSuccess()

This method is called when the system successfully reads the data from a magnetic stripe card.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.MagCardListener.onSuccess (in Bundle
track)
```

Parameters

`track` A Bundle object containing the card information that was successfully read. The data includes:

PAN(String)	The PAN card number.
TRACK1(String)	Track 1 data.
TRACK2(String)	Track 2 data.
TRACK3(String)	Track 3 data.
SERVICE_CODE(String)	The service code associated with the card.
EXPIRED_DATE(String)	The expiration date of the card.

Return Values

```
void
```

onError()

This method is triggered when an error occurs during the card swiping process, providing details on the nature of the error.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.MagCardListener.onError (int error,
String message)
```

Parameters

`error` An integer representing the error code, which indicates the type of error that occurred during the magnetic card swipe. Possible error codes include:

SERVICE_CRASH (99)	Indicates that the service has crashed.
REQUEST_EXCEPTION (100)	Represents an exception that occurred during the card read request.
MAG_SWIPE_ERROR (1)	Indicates a failure in swiping the magnetic card.

`message` A description of the error, providing additional context about the error.

Return Values

```
void
```

onTimeout()

This method is called when the card swiping operation fails to complete within a specified timeframe.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.MagCardListener.onTimeout ()
```

Parameters

None.

Return Values

```
void
```

9. RFSearchListener

Package: com.vfi.smartpos.deviceservice.aidl.RFSearchListener

Overview:

This interface is used for managing CTLS card search operations. It provides methods to handle events related to the detection of contactless cards and errors that may occur during the card search process.

Public Member Functions:

Modifier and Type	Method
void	**onCardPass** (int cardType)
void	**onFail** (int error, String message)

Member Function Documentation:

onCardPass()

This method is called when a contactless card is successfully detected during the CTLS card search process.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.RFSearchListener.onCardPass (int
cardType)
```

Parameters

`cardType` An integer indicating the type of card detected. The card types are represented by constants, which define different types of contactless cards. The possible card types:

S50_CARD(0x00)	S50 MIFARE card.
S70_CARD(0x01)	S70 MIFARE card.
PRO_CARD(0x02)	PRO card.
S50_PRO_CARD(0x03)	S50 MIFARE PRO card.
S70_PRO_CARD(0x04)	S70 MIFARE PRO card.
CPU_CARD(0x05)	CPU card (contactless card).

Return Values

```
void
```

onFail()

This method is called when an error occurs during the card search process.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.RFSearchListener.onFail (int error,
String message)
```

Parameters

`error` An integer representing the error code, which provides specific details about the failure that occurred. Possible error codes include:

ERROR_TRANSERR(0xA2)	Error during transaction, such as communication failure.
ERROR_PROTERR(0xA3)	The response from the card is illegal.
ERROR_MULTIERR(0x84)	Multiple cards were found.
ERROR_CARDTIMEOUT(0xA7)	The card timeout.
ERROR_CARDNOACT(0xB3)	The card (PRO, type B) is not active.
ERROR_MCSERVICE_CRASH(0xff01)	The master service has crashed.
ERROR_REQUEST_EXCEPTION(0xff02)	An exception occurred during the request.

`message` A description of the error.

Return Values

`void`

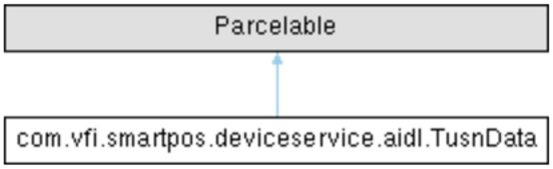
10. TusnData

Package: com.vfi.smartpos.deviceservice.aidl.TusnData

Overview:

This class represents a data structure that stores terminal information, including its type, MAC address, and a unique serial number.

Inheritance diagram:



NOTE

TusnData is a java class and is called using the constructor `TusnData(Parcel source)`.

Constructor:

TusnData(Parcel source)

This constructor initializes a TusnData object from a given Parcel `source`.

Parameters

`source` A Parcel object from which the TusnData is read and constructed.

Public Member Functions:

Modifier and Type	Method
void	**setTerminalType** (int type)
void	**setMac** (String mac)
void	**setTusn** (String tusn)
String	**getMac** ()

String	**getTusn** ()
int	**getTerminalType** ()

Member Function Documentation:

setTerminalType()

This method sets the type of the terminal.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.TusnData.setTerminalType (int type)
```

Parameters

`type`	An integer representing the type of terminal.

Return Values

```
void
```

setMac()

This method sets the MAC address for the terminal.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.TusnData.setMac (String mac)
```

Parameters

`mac`	A string representing the MAC address of the terminal.

Return Values

```
void
```

setTusn()

This method sets a unique serial number for the terminal.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.TusnData.setTusn (String tusn)
```

Parameters

`tusn`	A string representing the unique terminal serial number or identifier.

Return Values

```
void
```

getMac()

This method retrieves the MAC address associated with the terminal.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.TusnData.getMac ()
```

Parameters

None.

Return Values

The MAC address as a string.

getTusn()

This method retrieves the unique terminal serial number (Tusn).

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.TusnData.getTusn ()
```

Parameters

None.

Return Values

The Tusn value as a string.

getTerminalType()

This method retrieves the type of the terminal as previously set.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.TusnData.getTerminalType ()
```

Parameters

None.

Return Values

An integer representing the terminal type.

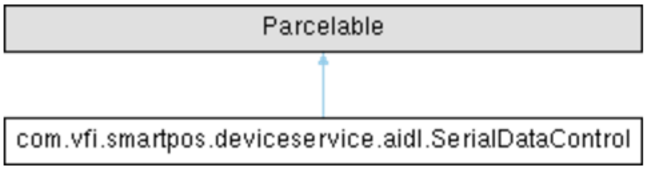
11. SerialDataControl

Package: com.vfi.smartpos.deviceservice.aidl.SerialDataControl

Overview:

This class acts as a configuration entity for establishing serial communication settings on the devices. It encapsulates critical parameters like baud rate, data bits, stop bits, and parity, providing a structured approach to manage serial data transmission settings.

Inheritance diagram:



NOTE

SerialDataControl is a java class and is called using the constructor `SerialDataControl (int baudRate, int dataBits, int stopBits, int serialParity)`

Constructor:

SerialDataControl (int baudRate, int dataBits, int stopBits, int serialParity)

This constructor initializes the `SerialDataControl` object with the specified serial communication parameters. The parameters are crucial for configuring serial communication correctly.

Parameters

`baudRate`	The baud rate determines the speed at which data is transmitted over the serial connection.
`dataBits`	The number of data bits per frame.

`stopBits`	The number of stop bits used to signal the end of a data frame.
`serialParity`	The parity bit used for error checking.

Public Member Functions:

Modifier and Type	Method
int	**getBaudRate** ()
int	**getDataBits** ()
int	**getStopBits** ()
int	**getSerialParity** ()

Member Function Documentation:

getBaudRate()

This method retrieves the baud rate of the serial communication, which defines the speed at which the data is transmitted.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.SerialDataControl.getBaudRate ()
```

Parameters

None.

Return Values

The baud rate as an integer.

getDataBits()

This method retrieves the number of data bits per frame used in the serial data communication.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.SerialDataControl.getDataBits ()
```

Parameters

None.

Return Values

The number of data bits as an integer.

getStopBits()

This method retrieves the number of stop bits used to signal the end of a data frame in the serial data communication.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.SerialDataControl.getStopBits ()
```

Parameters

None.

Return Values

The stop bits setting as an integer.

getSerialParity()

This method retrieves the parity setting used for error checking in the serial communication.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.SerialDataControl.getSerialParity ()
```

Parameters

None.

Return Values

The parity configuration as an integer.

12. ISerialPort

Package: com.vfi.smartpos.deviceservice.aidl.ISerialPort

Overview:

This interface is used for inter-process communication (IPC). It defines methods for managing and interacting with serial ports effectively. It allows applications to open, configure, read from, and write to serial ports, enabling seamless communication with peripherals.

Public Member Functions:

Modifier and Type	Method
boolean	**open** ()
boolean	**close** ()

boolean	**init** (int bps, int par, int dbs)
int	**read** (inout byte[] buffer, int expectLen, int timeout)
int	**write** (in byte[] data, int timeout)
boolean	**clearInputBuffer** ()
boolean	**isBufferEmpty** (boolean input)

Member Function Documentation:

open()

This method opens the serial port, establishing a communication channel with the connected device.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.ISerialPort.open ()
```

Parameters

None.

Return Values

A Boolean value:

true: If the port was successfully opened.

false: If the port opening failed.

See Also

```
close()
```

close()

This method closes the currently opened serial port.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.ISerialPort.close ()
```

Parameters

None.

Return Values

A Boolean value:

true: If the port was successfully closed.

false: If the port closing failed.

See Also

```
open()
```

init()

This method initializes the serial port with the given parameters.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.init (int bps, int par, int dbs)
```

Parameters

`bps`

Baud rate (bits per second), which defines the speed of data transmission over the serial port. Supported values:

- 1200 bps.
- 2400 bps.
- 4800 bps.
- 9600 bps.
- 14400 bps.
- 19200 bps.
- 28800 bps.
- 38400 bps.
- 57600 bps.
- 115200 bps.

`par`

Parity setting for the data transmission. The available parity settings are:

0	No parity checks.
1	Odd parity.
2	Even parity.

`dbs`

The number of data bits used in each byte of transmission.

Return Values

A Boolean value:

- true: If initialization succeeded.
- false: If initialization failed.

See Also

`open()`

read()

This method is used to read data from a device and store it in a provided buffer.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.read (inout byte[] buffer, int
expectLen, int timeout)
```

Parameters

`buffer`	The byte array to store the incoming data.
`expectLen`	The expected length of the data to be read.
`timeout`	The timeout duration in milliseconds.

Return Values

Either of the following values:

Positive integer: The length of data successfully read and stored in the buffer.

-1: If the operation fails.

See Also

`write()`

write()

This method is used to write data to the serial port.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.write (in byte[] data, int timeout)
```

Parameters

`data`	The byte array containing data to be sent.
`timeout`	The timeout duration in milliseconds.

Return Values

Either of the following values:

Positive integer: The length of data successfully written.

-1: If the operation fails.

See Also

`read()`

clearInputBuffer()

This method clears the input buffer of the serial port.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.ISerialPort.clearInputBuffer()
```

Parameters

None.

Return Values

A Boolean value:

true: The input buffer was successfully cleared.

false: Failed to clear the input buffer.

isBufferEmpty()

This method checks whether a specified buffer (input or output) is empty.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.ISerialPort.isBufferEmpty (boolean
input)
```

Parameters

`input` Indicates whether to check the input buffer.

true:	Check the input buffer (used for reading data).
false:	Check the output buffer (used for writing data).

Return Values

A Boolean value:

true: Data is available in the specified buffer.

false: No data is available in the specified buffer.

13. IUsbSerialPort

Package: com.vfi.smartpos.deviceservice.aidl.IUsbSerialPort

Overview:

This class provides an interface for interacting with USB serial ports in the device service. It allows communication with serial devices over USB, facilitating read, write, and buffer management operations for data transfer.

Public Member Functions:

Modifier and Type	Method
boolean	**isUsbSerialConnect** ()
int	**read** (inout byte[] buffer, int timeout)
void	**write** (in byte[] data)

Member Function Documentation:

isUsbSerialConnect()

T This method checks whether a USB-serial device is currently connected to the system via an OTG cable.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IUsbSerialPort.isUsbSerialConnect ()
```

Parameters

None.

Return Values

A Boolean value:

true: If a USB-serial device is available and connected.

false: If no USB-serial device is available or connected.

read()

This method is used to read data from the USB serial device into a provided buffer.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IUsbSerialPort.read (inout byte[]
buffer, int timeout)
```

Parameters

`buffer`	A byte array to store the data read from the USB-serial device.
`timeout`	The timeout duration in milliseconds.

Return Values

Either of the following values:

Positive integer: The length of data successfully read and stored in the buffer.

-1: If the operation fails.

See Also

`write()`

write()

This method sends data to the connected USB-serial device. It writes the contents of the provided data buffer to the device.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IUsbSerialPort.write (in byte[] data)
```

Parameters

`data`	A byte array containing the data to be sent to the USB-serial device.

Return Values

`void`

See Also

`read()`

14. IRFCardReader

Package: com.vfi.smartpos.deviceservice.aidl.IRFCardReader

Overview:

This interface is used for interacting with various types of contactless cards, such as MIFARE, Memory, and CPU cards. It defines a set of methods that allow for the initialization, authentication, and communication with these cards.

Public Member Functions:

Modifier and Type	Method
void	**searchCard** (RFSearchListener listener, int timeout)
void	**stopSearch** ()
int	**activate** (String driver, out byte[] responseData)

void	**halt** ()
boolean	**isExist** ()
byte[]	**exchangeApdu** (in byte[] apdu)
byte[]	**cardReset** ()
int	**authBlock** (int blockNo, int keyType, in byte[] key)
int	**authSector** (int sectorNo, int keyType, in byte[] key)
int	**readBlock** (int blockNo, out byte[] data)
int	**writeBlock** (int blockNo, in byte[] data)
int	**increaseValue** (int blockNo, int value)
int	**decreaseValue** (int blockNo, int value)
Bundle	**getCardInfo** ()
byte	**restore** (byte blockNo)
byte	**transfer** (byte blockNo)
void	**CloseRfField** ()

Member Function Documentation:

searchCard()

This method is used to search for an RFID card within range. The device will attempt to detect a card within the given timeout period.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IRFCardReader.searchCard
(RFSearchListener listener, int timeout)
```

Parameters

`listener`	The callback listener that will handle the result of the card search.
`timeout`	Timeout duration in milliseconds. This should be >= 1 millisecond.

Return Values

`void`

See Also

- Refer to `stopSearch()`
- Refer to `RFSearchListener` under Appendix A.

stopSearch()

This method is used to stop an ongoing search for an RFID card.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IRFCardReader.stopSearch ()
```

Parameters

None.

Return Values

```
void
```

activate()

This method is used to activate an RFID card. It initiates communication with the card and retrieves a response to confirm its activation.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IRFCardReader.activate (String driver,
out byte[] responseData)
```

Parameters

driver	The driver’s name corresponding to the type of card. It includes:

S50	Represents S50 (M1) cards.
S70	Represents S70 (M1) cards.
CPU	Represents CUP cards.
PRO	Represents PRO cards e.g., S50_PRO, S70_PRO.

`responseData`	An output parameter that stores the response data from the card.

Return Values

Either of the following values:

0: Success. The card was successfully activated.

Non-zero value: Failure. If the return value is not 0, it indicates an error during the activation process.

halt()

This method is used to halt the operation of the card reader.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IRFCardReader.halt ()
```

Parameters

None.

Return Values

`void`

See Also

`stopSearch()`

isExist()

This method is used to check whether an RFID card is currently detected or available.

Prototype

```
boolean com.vfi.smartpos.deviceservice.aidl.IRFCardReader.isExist ()
```

Parameters

None.

Return Values

A Boolean value:

true: This indicates that an RFID card is currently detected.

false: This indicates that no RFID card is detected.

exchangeApdu()

This method is used to send an APDU command to an RFID card and receive the response from the card.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IRFCardReader.exchangeApdu (in
byte[]apdu)
```

Parameters

`apdu`	The APDU command that will be sent to the RFID card.

Return Values

A byte array that contains the card’s APDU response data.

cardReset()

This method is used to reset an RFID card.

Prototype

```
byte[] com.vfi.smartpos.deviceservice.aidl.IRFCardReader.cardReset ()
```

Parameters

None.

Return Values

A byte array that contains the response from the card after the reset operation.

authBlock()

This method is used to authenticate a specific block on the RFID card. This is part of the process for accessing or modifying data stored in that block.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IRFCardReader.authBlock (int blockNo,
int keyType, in byte[] key)
```

Parameters

`blockNo` The block number (index) on the RFID card to authenticate. The block number starts at 0 (first block).

`keyType` The type of key used for authentication.

KEY_A (0) Authentication using key A.

KEY_B (1) Authentication using key B.

`key` A 6-length key used for the authentication process.

Return Values

Either of the following values:

0: Success. The authentication of the block was successful.

Non-zero value: Failure if the return value is not 0.

See Also

`authSector()`

authSector()

This method is used to authenticate a sector on a smart card, when interacting with a contactless smart card reader.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IRFCardReader.authSector (int sectorNo,
int keyType, in byte[] key)
```

Parameters

`sectorNo` This is the sector number starting from 0 that user want to authenticate.

`keyType` The type of key used for authentication.

KEY_A (0) Authentication using key A.

KEY_B (1) Authentication using key B.

`key` The key is used for authentication. The length of the key should be exactly 6 bytes.

Return Values

Either of the following values:

0: Success. The authentication of the sector was successful.

Non-zero value: Failure. An error occurred during the authentication process.

See Also

`authBlock()`

readBlock()

This method is used to read data from a specific block on a smart card.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IRFCardReader.readBlock (int blockNo,
out byte[] data)
```

Parameters

`blockNo`	This is the block number starting from 0 that the user wants to read from.
`data`	This is an output parameter that will hold the data read from the specified block. The length of the array will be 16 bytes.

Return Values

Either of the following values:

0: Success. The block was successfully read.

Non-zero value: An error occurred while reading the block.

See Also

writeBlock()

This method is used to write data to a specific block on a smart card.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IRFCardReader.writeBlock (int blockNo,
in byte[] data)
```

Parameters

blockNo	The block number on the smart card where data should be written. The numbering starts from 0.
data	This is the source data that will be written to the specified block. The array must contain exactly 16 bytes of data, as each block on the card is 16 bytes long. If the data array has a length different from 16, the method will result in an error.

Return Values

Either of the following values:

0: Success. The block was successfully written with the provided data.

Non-zero value: Failure. An error occurred during the write operation.

See Also

`readBlock()`

increaseValue()

This method is used to increase the value stored on a specific block in a smart card.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IRFCardReader.increaseValue (int
blockNo, int value)
```

Parameters

`blockNo`	The block number on the card or device where the value should be increased.
`value`	Indicates the amount by which the value at the specified block should be increased.

Return Values

Either of the following values:

0: Indicates success.

Non-zero value: Indicates Failure.

See Also

`decreaseValue()`

decreaseValue()

This method is used to decrease the value stored on a specific block in a smart card.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.IRFCardReader.decreaseValue (int
blockNo, int value)
```

Parameters

`blockNo`	The block number on the card or device where the value should be decreased.
`value`	Indicates the amount by which the value at the specified block should be decreased.

Return Values

Either of the following values:

0: Indicates success.

Non-zero value: Indicates Failure.

See Also

`increaseValue()`

getCardInfo()

This method is used to retrieve the card information during the `RFSearchListener.onCardPass` callback function, which is triggered when a card is detected by an RFID.

Prototype

```
Bundle com.vfi.smartpos.deviceservice.aidl.IRFCardReader.getCardInfo ()
```

Parameters

None.

Return Values

A Bundle object containing card information. The SN of the card is returned as a ByteArray.

See Also

Refer to `RFSearchListener.onCardPass()` under Appendix A.

restore()

This method is used to restore or reset a specific block on a smart card or RFID card, identified by its block number.

Prototype

```
byte com.vfi.smartpos.deviceservice.aidl.IRFCardReader.restore (byte blockNo)
```

Parameters

`blockNo`	The block number on the card that user wants to restore.

Return Values

A byte value that indicates the result of the restore operation.

0x00: Success: The block was successfully restored.

0x01: BlockNo error: There was an issue with the specified block number.

0x02: Operation failed: The restore operation failed due to an unknown issue with the operation.

0xff: Other error: A generic error occurred during the restore process, indicating an issue not covered by the previous return codes.

transfer()

This method is used to transfer an operation on a specific block on a smart card or RFID card, identified by its block number.

Prototype

```
byte com.vfi.smartpos.deviceservice.aidl.IRFCardReader.transfer (byte
blockNo)
```

Parameters

`blockNo`	The block number on the card that user wants to transfer.

Return Values

A byte value that indicates the result of the operation:

0x00: Success: The transfer operation was successful.

0x01: BlockNo error: There was an issue with the specified block number.

0x02: Operation failed: The transfer operation failed for an unspecified reason.

0xff: Other error: A generic error occurred during the transfer process, which does not fall under the other error categories.

CloseRfField()

This method is used to close or deactivate the RF field on a smart card reader.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.IRFCardReader.CloseRfField ()
```

Parameters

None.

Return Values

```
void
```

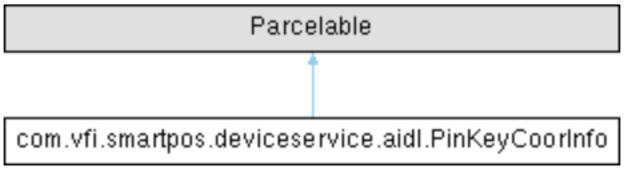
15. PinKeyCoorInfo

Package: com.vfi.smartpos.deviceservice.aidl.PinKeyCoorInfo

Overview:

This class is created to manage data related to the physical key presses or coordinates on a keypad during PIN entry. It is primarily used in secure environments where PIN entry and validation are critical for ensuring transaction security.

Inheritance diagram:



NOTE

PinKeyCoorInfo is a java class and is called using the constructor `PinKeyCoorInfo (String keyName, int coor1_x, int coor1_y, int coor2_x, int coor2_y, int keyType).`

Constructor:

PinKeyCoorInfo (String keyName, int coor1_x, int coor1_y, int coor2_x, int coor2_y, Int keyType)

This constructor initializes the `PinKeyCoorInfo` object with details related to a specific key on a keypad, such as its name, coordinates, and type.

Parameters

`keyName` Represents the name or label of the key on the keypad.

coor1_x The x-coordinate of the first point that defines the key's position on the keypad.

coor1_y The y-coordinate of the first point that defines the key's position.

coor2_x The x-coordinate of the second point that defines the key's boundary.

coor2_y The y-coordinate of the second point that defines the key's boundary.

`keyType` Specifies the type of the key. The possible values are:

0-TypeNum A numeric key.

1-TypeConf A confirmation key.

2-TypeCancel A cancel key.

3-TypeDel A delete key.

Public Member Functions:

Modifier and Type	Method
String	**getKeyName** ()
int[]	**getCoor1** ()
int[]	**getCoor2** ()

int	**getKeyType** ()

Member Function Documentation:

getKeyName()

This method retrieves the name or label of the key.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.PinKeyCoorInfo.getKeyName ()
```

Parameters

None.

Return Values

A string representing the name of the key.

getCoor1()

This method retrieves the first pair of coordinates that define the key's position.

Prototype

```
int[] com.vfi.smartpos.deviceservice.aidl.PinKeyCoorInfo.getCoor1 ()
```

Parameters

None.

Return Values

An array of integers representing the first set of coordinates of the key.

getCoor2()

This method retrieves the second pair of coordinates that define the key's position.

Prototype

```
int[] com.vfi.smartpos.deviceservice.aidl.PinKeyCoorInfo.getCoor2 ()
```

Parameters

None.

Return Values

An array of integers representing the second set of coordinates of the key.

getKeyType()

This method returns the type of the key, such as a numeric key (0), a confirmation key (1), a cancel key (2), or a delete key (3).

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.PinKeyCoorInfo.getKeyType ()
```

Parameters

None.

Return Values

An integer representing the key type.

16. PinpadKeyType

Package: com.vfi.smartpos.deviceservice.aidl.PinpadKeyType

Overview:

This class is essential for mapping the physical keys on a terminal's Pinpad to recognizable constants in the software, ensuring that the terminal correctly processes user inputs during financial transactions.

Static Public Attributes:

The following table lists Modifier and Type, Field Name, Value, Description, and Field Detail:

Modifier and Type	Field Name	Value	Description	Field Detail
static final int	**MASTERKEY**	0	Represents the master key used for encryption.	`public static final int MASTERKEY`
static final int	**MACKEY**	1	Represents the MAC (Message Authentication Code) key.	`public static final int MACKEY`
static final int	**PINKEY**	2	Represents the key used for encrypting PINs.	`public static final int PINKEY`
static final int	**TDKEY**	3	Represents the key for Transaction Data encryption.	`public static final int TDKEY`
static final int	**SM_MASTERKEY**	4	Represents the master key for the SM (Secure Module).	`public static final int SM_MASTERKEY`
static final int	**SM_MACKEY**	5	Represents the MAC key for the SM.	`public static final int SM_MACKEY`

static final int	**SM_PINKEY**	6	Represents the PIN key for the SM.	`public static final int SM_PINKEY`
static final int	**SM_TDKEY**	7	Represents the transaction data key for the SM.	`public static final int SM_TDKEY`
static final int	**AES_MASTERKEY**	8	Represents the AES (Advanced Encryption Standard) master key.	`public static final int AES_MASTERKEY`
static final int	**AES_MACKEY**	9	Represents the AES MAC key.	`public static final int AES_MACKEY`
static final int	**AES_PINKEY**	10	Represents the AES key used for PIN encryption.	`public static final int AES_PINKEY`
static final int	**AES_TDKEY**	11	Represents the AES key used for transaction data encryption.	`public static final int AES_TDKEY`
static final int	**DUKPTKEY**	12	Represents the DUKPT key.	`public static final int DUKPTKEY`
static final int	**TEK**	13	Represents the TEK (Transaction Encryption Key).	`public static final int TEK`
static final int	**SM_TEK**	14	Represents the TEK for the SM.	`public static final int SM_TEK`
static final int	**AES_TEK**	15	Represents the AES TEK.	`public static final int AES_TEK`

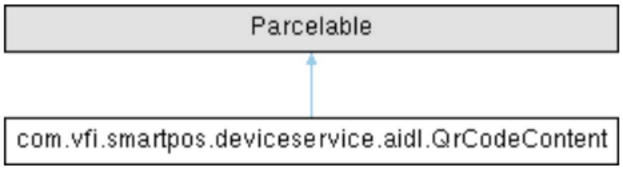
17. QrCodeContent

Package: com.vfi.smartpos.deviceservice.aidl.QrCodeContent

Overview:

This class serves as the container for the data that is encoded in a QR code, which is used in various financial and transaction-related processes in POS systems. It could store information such as the transaction amount, merchant ID, and payment method, allowing for secure and efficient interactions between customers and merchants.

Inheritance diagram:



NOTE

QrCodeContent is a java class and is called using the constructor `QrCodeContent (Parcel in)`.

Constructor:

QrCodeContent (int height, int leftOffset, String barcode)

This is the constructor for the `QrCodeContent` class. It initializes a new `QrCodeContent` object with the given height, left offset, and barcode string.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.QrCodeContent.QrCodeContent (int
height, int leftOffset, String barcode)
```

Parameters

`height`	The height of the QR code.

`leftOffset`	The left offset for positioning the QR code.
`barcode`	The content of the barcode.

Public Member Functions:

Modifier and Type	Method
int	**getHeight** ()
void	**setHeight** (int height)
int	**getLeftOffset** ()
void	**setLeftOffset** (int leftOffset)
String	**getBarcode** ()
void	**QrCodeContent** (int height, int leftOffset, String barcode)

Member Function Documentation:

getHeight()

This method returns the height of the QR code.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.QrCodeContent.getHeight ()
```

Parameters

None.

Return Values

An integer representing the height of the QR code.

setHeight()

This method allows the user to set the height of the QR code.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.QrCodeContent.setHeight (int height)
```

Parameters

`height`	The height to set for the QR code.

Return Values

```
void
```

getLeftOffset()

This method returns the left offset for positioning the QR code.

Prototype

```
int com.vfi.smartpos.deviceservice.aidl.QrCodeContent.getLeftOffset ()
```

Parameters

None.

Return Values

An integer representing the left offset in the same unit as the height.

setLeftOffset()

This method sets the left offset for the QR code.

Prototype

```
void com.vfi.smartpos.deviceservice.aidl.QrCodeContent.setLeftOffset (int
leftOffset)
```

Parameters

`leftOffset`	The value representing the left offset.

Return Values

```
void
```

getBarcode()

This method retrieves the barcode of the QR code as a String.

Prototype

```
String com.vfi.smartpos.deviceservice.aidl.QrCodeContent.getBarcode ()
```

Parameters

None.

Return Values

A String representing the barcode content encoded in the QR code.

Verifone

University Drive
Coral Springs,
FL 33065, USA
Fax: 4545 233
Phone: 001 454 2333

www.verifone.com

Thank you!

We are the payments architects who truly understand commerce.

As payment architects we shape ecosystems for online and in-person commerce experiences, including all the tools you need… from gateways and acquiring to fraud management, tokenization and reporting.

As commerce experts, we are here for you and your business. With our payment devices, our systems & solutions and our support. Everywhere. Anytime. So that your customers feel enabled, recognized and well taken care of, even beyond their expectations.

Verifone. Creating omni-commerce solutions that simply shape powerful customer experiences.