ADVANCED MOBILE PAYMENT INC.

AMP POS 6 AND 8 SERIES

Developer Guide

v 1.9

PDV-8000-1.9-E

PROPRIETARY NOTICE

All pages of this document contain information proprietary to Advanced Mobile Payment (AMP) Inc. This document shall not be duplicated, transmitted, used, or otherwise disclosed to anyone other than the organization or specific individuals to which this document is delivered. This restriction is applicable to all sheets of this document. AMP reserves the right to have the recipient return all copies of this document at any time. AMP reserves the right to change the content of this document without prior notice.

DOCUMENT PROPERTIES

INFORMATION

ID	PDV-8000-1.9-E
Title	AMP POS 6 and 8 Series - Developer Guide
Document Portal Path	/Development/AMP POS/AMP 6,8 Series
Category	AMP POS – 6,8 Series
Access Level	Development
NDA Required	Yes ☑ No☐

VERSION CONTROL

Template Version	1.5

Doc. Version	Date	Summary of Change	Updated by
1.0	Sep 2017	Initial version	R.Vafaie
1.1	May 2018	Update for Android Studio 3.0.0	R.Vafaie
1.1	Jul 2018	Added Application Debugging Section Added PIN Entry Mode Added AMP 6500 Anti-Removal Feature Added COSU Implementation	S. Tabios
1.2	Aug 2018	Added Appendix D. MDB Interface Usage	S. Tabios
1.3	May 2019	Updated Section 1.2 to include the updated contents of the latest SDK package Added instruction to add ATMS Agent Library to the Project Added Section 3 Rudimentary Concepts to AMP POS Development Updated Section 4 Application Debugging	S. Tabios
1.4	Jul 2019	Updated Section 4.2 by adding note for AMP 6500 Changed Appendices A – D to Section 5 - 8 Updated Section 6 Replaced Appendix A	S. Tabios K. Concha
1.5	Jul 2019	Updated Section 5 Added Section for Light and Proximity Sensor Usage	S. Tabios
1.6	Nov 2019	Added Section 2.3.1, 3.1.1~3.1.2 Added Section 8 Terminal Printing	S. Tabios

		Updated description for Section 7 Updated Section 2.1 to highlight proper versions of software need to be used	
1.7	Jan 2020	Updated Section 2.1 for minimum compatible version Updated Section 2.2 for new project creation Updated Section 2.5 for locating SDKXXXX.jar Updated Section 7 to add Sample Activity	C. Longshore, S. Tabios
1.8	Aug 2020	Modified Section 2.1 and Section 2.2 to emphasize some important notes and verification of some steps in Android and Android NDK installation Updated Section 7.2 to include sample code to disable COSU Mode on application exit	S. Tabios, C. Longshore
1.9	Feb 2021	Modified Section 7 and sub sections to include the correct API function to be used in disabling status bar and added other API options needed in COSU mode	S. Tabios

SUPPORTED HARDWARE & SOFTWARE

Doc. Version	Software Title	Release	Supported Hardware Model
1.0+	AMP POS 8 Series FW	3.1.22 – 3.2.23	AMP POS 8 Series
1.0+	AMP POS 6700 FW	6.1.9 – 6.2.5	AMP 6700
1.0 – 1.8	AMP POS 6500 FW	1.0.12 – 1.0.14	AMP 6500
1.9	AMP POS 6500 FW	1.0.20	AMP 6500

EXTERNAL REFERENCES

URLs	Downloading Android Studio https://developer.android.com/studio/archive Creating a new project https://developer.android.com/training/basics/firstapp/creating-project.html Migrating to Android Studio https://developer.android.com/studio/intro/migrate.html
E-Mail Addresses	--
Phone Numbers	--
Documents	--
Knowledge Base Articles	--

TABLE OF CONTENTS

Proprietary Notice i

Document Properties ii

Table of Contents iv

Table of Figures vi

1 About AMP POS 6 and 8 Series 1

1.1 Extracting the Package 1

1.2 The AMP 6 and 8 Series SDK Folder 2

1.2.1 AMP SDK Library Folder 3

1.2.2 Guides & Manuals Folder 4

1.2.3 Workspace 4

2 Development Environment Setup 5

2.1 The Android Studio IDE Setup 5

2.1.1 Important Notes 5

2.2 The SDK Tools Installation 5

2.2.1 Android Version 6

2.2.2 Android NDK 6

2.2.3 CMake 8

2.3 Creating an Android Project 9

2.4 Opening Sample Projects in Android Studio 10

2.4.1 DemoAppEMV Project 10

2.5 Importing/Adding External JAR Files to Android Studio 10

2.5.1 Adding AMP POS 6 and 8 Series Libraries to the Project 10

2.5.2 Adding the ATMS Agent Library to the Project 13

3 AMP POS Development Basic Concepts 15

3.1 The AMP Payment Application Software Architecture 15

3.1.1 Android Components (Java/Android) 16

3.1.2 Native Components (C/C++) 16

3.2 Using Java Native Interface (JNI) 17

3.3 Java AMP SDK Initialization 20

4 Application Debugging 21
4.1 Android Log Functions 21
4.1.1 Native Side Debug Function 21
4.1.2 Java-Debug Function 22
4.2 Debugging via USB 22
4.3 Debugging via TCP/IP 23
5 AMP 6500 Device Features 24
5.1 Anti-Removal Feature 24
5.2 MultiDrop Bus (MDB) Interface 26
5.3 Light and Proximity Sensor 29
6 PIN Entry Modes 29
7 COSU (Corporate-Owned, Single-Use) Implementation 31
7.1 The COSU Mode Procedure 31
7.1.1 Disabling status bar and bottom navigation bar 31
7.1.2 Setting the application for launch after terminal boot 33
7.2 COSU Mode Implementation Code Sample 33
8 Terminal Printing 35
Appendix A 37

TABLE OF FIGURES

Figure 1 - Extracting the ZIP File Package. 1
Figure 2 - The Target Directory for the SDK Package. 2
Figure 3 - The SDK Folder Content. 3
Figure 4 - SDK Manager Icon. 6
Figure 5 - Android API Level Window. 6
Figure 6 - "ndk-bundle" Contents. 7
Figure 7 - Android SDK Location Window. 7
Figure 8 - Android NDK Version. 8
Figure 9 - CMake Version Window. 9
Figure 10 - “external-libs” Folder under “apps” Directory. 11
Figure 11 - Copying the API Library to the "external-libs" Folder. 11
Figure 12 - Adding Compile-Time Dependency to the "build.gradle" File. 12
Figure 13 - Syncing the Project with the Gradle Files. 12
Figure 14 - Declaring the "uses-library" Element in the Manifest File. 13
Figure 15 - ATMS New Module Selection. 13
Figure 16 - Import JAR/AAR Package Selection. 14
Figure 17 - AAR File Selection. 14
Figure 18 – The AMP Payment Application Software Architecture. 15
Figure 19 - EMV L2 Java Callback Interface Functions. 17
Figure 20 - EMV L2 Java Callback Static Method Implementation. 18
Figure 21 - EMV L2 Java Interface. 18
Figure 22 - Method call to Java Native Interface. 19
Figure 23 - JNI Command Console Execution. 19
Figure 24 – Native Side Function Definition of Java Native Interface Method. 19
Figure 25 - EMV L2 Sample Java Callback Method Native Call. 20
Figure 26 - Selecting the Deployment Target Screen. 23
Figure 27 - The Android Logcat Window. 23
Figure 28 - AMP 6500 Plexiglass Bracket 25
Figure 29 - AMP 6500 Switch Press for Anti-Removal. 26
Figure 30 - MdbUtils Class from API-DOC. 27
Figure 31 - The Directions Corresponding to the Numbers. 30
Figure 32 - AMP 8 Series Download Cable. 37
Figure 33 - AMP 6 Series Download Cable. 37

1 ABOUT AMP POS 6 AND 8 SERIES

The AMP POS 6 and 8 Series are Android-based smart Point of Sale (POS) terminals that accept all payment modes like swipe, chip or tap.

The AMP 6500 features a robust hardware for the most demanding unattended environments. The AMP 6700 has a 7" color touch display with signature capture and an appealing video capability. The AMP 8 Series is ideal for an easy and portable POS operation as it has a 5" screen with Magstripe, Chip & PIN, NFC and virtual wallet payment acceptance.

AMP supplies a complete package for the development environment in a zip file that must be extracted to a local machine.

Additionally, a Sign Tool package is needed to sign POS applications, for production purposes. The AMP account manager should be contacted for receiving such package.

This document is a developer's guide for AMP 6 and 8 Series using Java Android Programming with C/C++ Native Interfacing. As such, it is expected that the users of this document are equipped with at least basic to intermediate knowledge of the Java programming language implemented in an Android platform.

1.1 EXTRACTING THE PACKAGE

The AMP 6 and 8 Series SDK zip file can be extracted using the WinRAR application and can be downloaded using the following link: WinRAR Download.

The package names are AMP6700SDKvX.X.X.zip, AMP6500SDKvX.X.X.zip and AMP8SeriesSDKvX.X.X.zip for AMP 6700, AMP 6500 and AMP 8000/AMP 8200 respectively.

The AMP 6 and 8 Series SDK zip file contains libraries that contain symbolic links. Therefore, this package cannot be successfully extracted by unzipping the file using the conventional way. An Administrator account is required for zip file extraction. This can be done by right clicking the WinRAR application or its shortcut and selecting "Run as administrator".

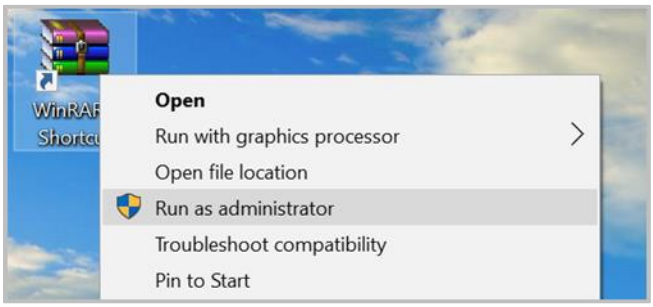

Figure 1 - Extracting the ZIP File Package.

When launching the WinRAR application, browse for the zip file package and extract it to the desired directory.

Note: It is mandatory to extract the package in the top directory (example, “C:\”) since the package contains long filenames and folder directory structures.

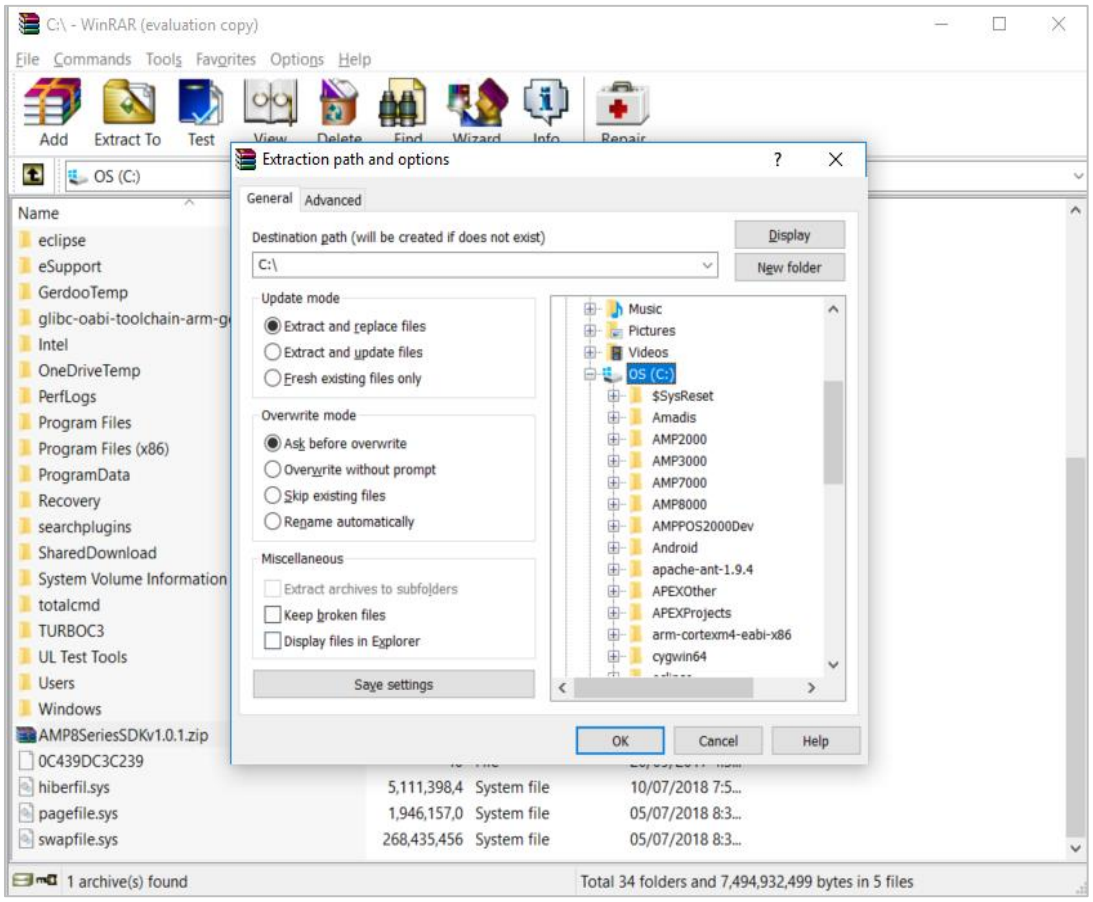

Figure 2 - The Target Directory for the SDK Package.

1.2 THE AMP 6 AND 8 SERIES SDK FOLDER

The zip file package should be fully extracted to the desired directory. The content of the package is intended for a specific terminal type, AMP 8 Series SDK package works only for AMP 8 Series terminals, AMP 6500 SDK package for AMP 6500, etc. The following image shows the correct directory structure when extracted. Moreover, the following sections describe the contents of each folder.

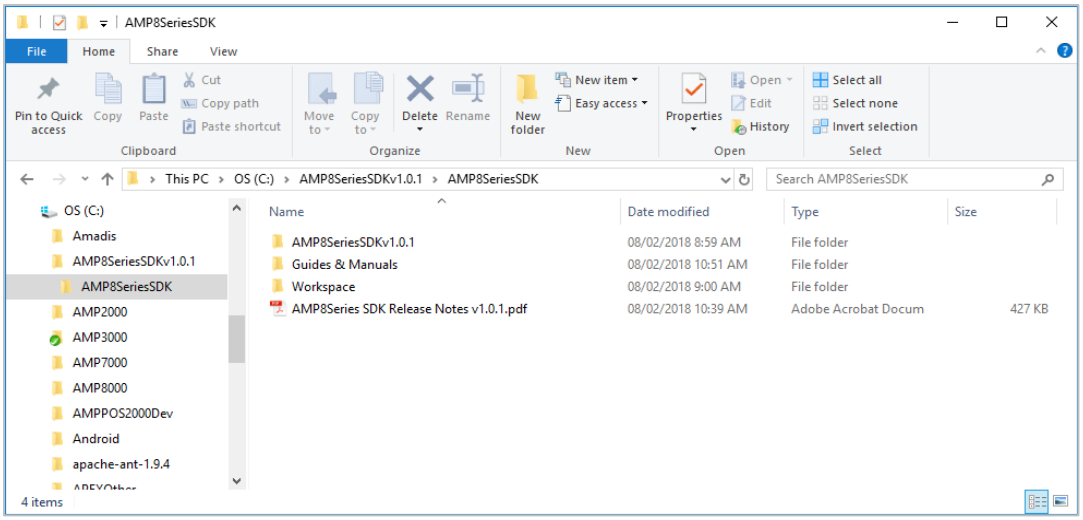

Figure 3 - The SDK Folder Content.

1.2.1 AMP SDK LIBRARY FOLDER

This folder's name follows the AMPXXXXSDKvX.X.X pattern. It contains libraries that are used in the development of AMP Android applications.

This folder is working for a specific version of application in the "workspace" folder. Therefore, it is important not to mix the contents of this folder with other applications outside the version indicated in this package, unless advised to do so. Below are the subfolders:

Folder name	AMPCMPNT
Full name	AMP Component Library
Description	This folder contains application-specific utility functions interlinked with AMPEMVL2 to perform EMV tasks

Folder name	AMPEMVL2
Full name	AMP EMV Level 2 Library
Description	This folder contains functions to perform EMV contact and contactless transactions.

Folder name	AMPPOSLibs
Full name	AMP Point-of-Sale Library
Description	This folder contains low-level libraries used to access the AMP POS terminal components: comms, printer, AMP Terminal Management System (ATMS), etc. It also contains AMP POS SDK JAR library file used in development.

1.2.2 GUIDES & MANUALS FOLDER

This folder contains the necessary documents for developing, compiling, building, and downloading a payment application. It also contains release notes for the version management of source codes. It contains the following files:

- API_DOC folder that contains manual for SDK API functions
- AMP POS 8 Series Demo Application - Release Notes.pdf
- ATMS Agent Library - Release Notes.pdf
- AMP EMV Level 2 Developer Guide.pdf
- AMP POS 6 and 8 Series Developer Guide.pdf

1.2.3 WORKSPACE

This folder contains projects and demo applications that help developers jumpstart into AMP Android application development. These projects are:

Folder name	AMP POS XXXX Demo Application Vx.y.z
Full name	AMP POS Demo Application
Description	This project demonstrates the capabilities of the AMP Android terminals' hardware components such as chip detection, NFC detection, printer usage, etc.
Naming Convention	AMP 8 Series: AMP POS 8 Series Demo Application Vx.y.z AMP 6500: AMP POS 6500 Demo Application Vx.y.z AMP 6700: AMP POS 6700 Demo Application Vx.y.z

Folder name	ATMSSampleCode
Full name	AMP Terminal Management System Sample Source Code
Description	It demonstrates the usage of AMP Terminal Management System (ATMS) API functions.

Folder name	DemoAppEMV
Full name	AMP EMV Demo Application
Description	This project serves as the template for the Android payment application. It contains a demonstration of the EMV contact and contactless transactions processing. It must be noted that the functionality, such as flow and application decisions based on card responses, manifested in this demonstration does not assume the actual development scenario.

	Therefore, it is imperative not to compile this application as a library just to extract the essential payment application functions.
Naming Convention	AMP POS vXX.XX.XXXDEM

2 DEVELOPMENT ENVIRONMENT SETUP

2.1 THE ANDROID STUDIO IDE SETUP

Android Studio provides the prime tools for building apps on every class of Android device. This IDE's world-class code editing, debugging, performance tooling, flexible build system and instant build/deploy system allow the creation of unique, high quality applications.

The official Android IDE and its necessary SDK tools can be downloaded from the link provided below:

https://developer.android.com/studio/archive

Subsequently, the downloaded .exe file should be launched, and the setup wizard must be closely followed for installing Android Studio and any necessary SDK tools, related to Android Studio.

2.1.1 IMPORTANT NOTES

1) Only Android Studio version 3.4.2 or lower must be used for development in AMP Android terminals. There are compatibility issues with the later versions of the IDE because of some dependencies on the set configurations of the provided sample applications. Should there be any changes in the future, this document will be updated accordingly.
2) To proceed with the installation of the required tools, starting from **Section 2.2 - The SDK Tools Installation**, the user should run Android Studio and select either "Start a new Android Studio project" or "Open an existing Android Studio project". Then, user should access SDK Manager and install the necessary files.
3) Do not select "Configure" button from "Welcome to Android Studio" window to install the SDK tools as it has a known issue of not reflecting the downloaded tools.

2.2 THE SDK TOOLS INSTALLATION

Pay specific attention to the following versions of specific Android SDK tools to prevent any compilation or runtime issues. Take note that Android SDK Tools can be installed or modified once the Android Studio was successfully launched and a new project is made for the first time.

Note: In cases where Android Studio IDE requires the user to install plug-ins for some issues, please follow the steps, as instructed, to resolve them.

2.2.1 ANDROID VERSION

Version required	API level 22 (Android Lollipop 5.1)
Link	Downloadable via SDK Tools Manager

Make sure to include the specific Android SDK Platform version to the project. To do that:

1) Click the Android SDK Manager Tool.

Figure 4 - SDK Manager Icon.

2) Check the checkbox corresponding to the API level 22 (Android 5.1 Lollipop).

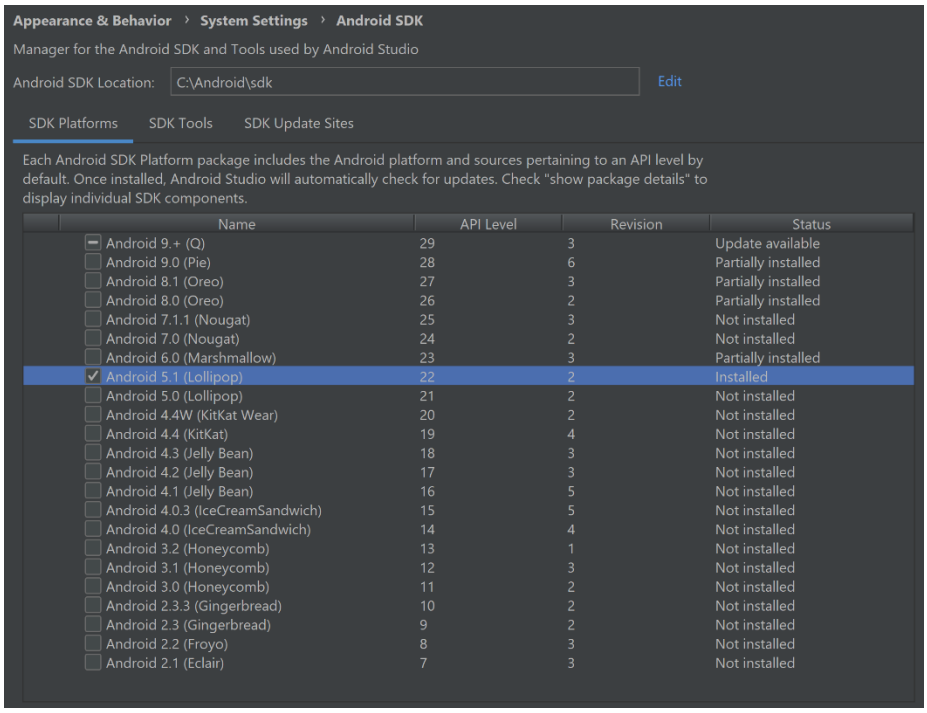

Figure 5 - Android API Level Window.

3) Click “Apply” then “OK” to reflect the changes. Make sure that the checkbox is checked after the installation. Otherwise, the specific Android version is not installed correctly.

2.2.2 ANDROID NDK

Description	This is a toolset that allows users to implement parts of the app in native code, using languages such as C and C++.
Version required	Revision 13b

Link	https://developer.android.com/ndk/downloads/older_releases.html Android NDK, Revision 13b (October 2016)

Follow these steps to setup the Android NDK:

1) Download the package from the link above.

2) Once done, rename the extracted folder from “android-ndk-r13” to “ndk-bundle”. After this step, it is expected that the contents inside “ndk-bundle” folder contains the following:

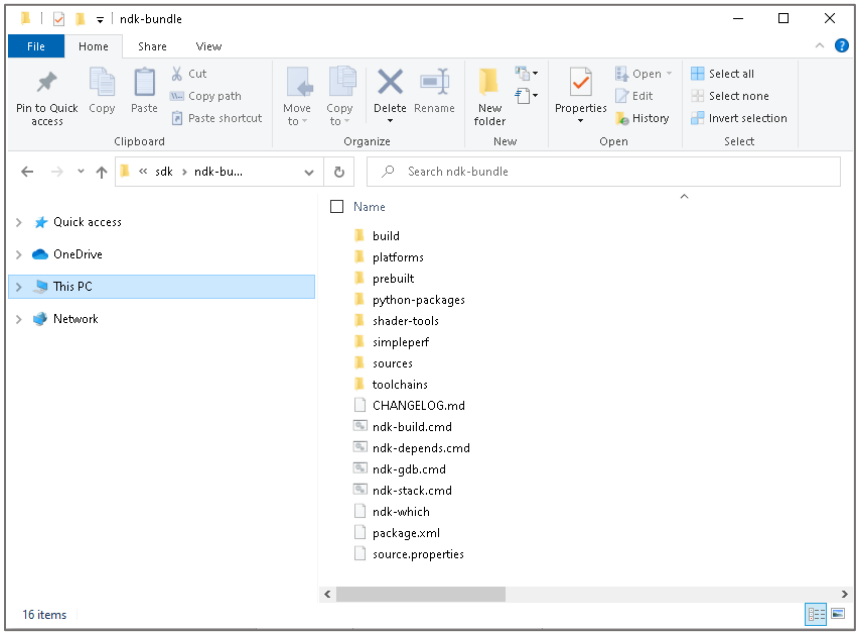

Figure 6 - "ndk-bundle" Contents.

3) Copy (or replace, if existing) the entire “ndk-bundle” folder to Android SDK Location. To verify the location of Android SDK folder:

    a) Click the Android SDK Manager Tool.
    b) Check the path for the location:

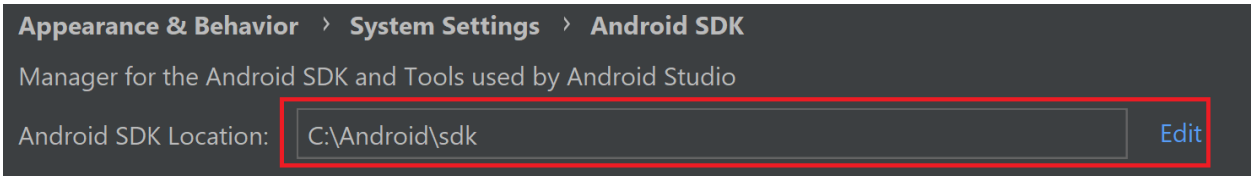

Figure 7 - Android SDK Location Window.

4) To verify the application is installed, go to Android SDK Manager, then select the “SDK Tools” tab and make sure the version on “Android NDK” row is 13.1.3345770.

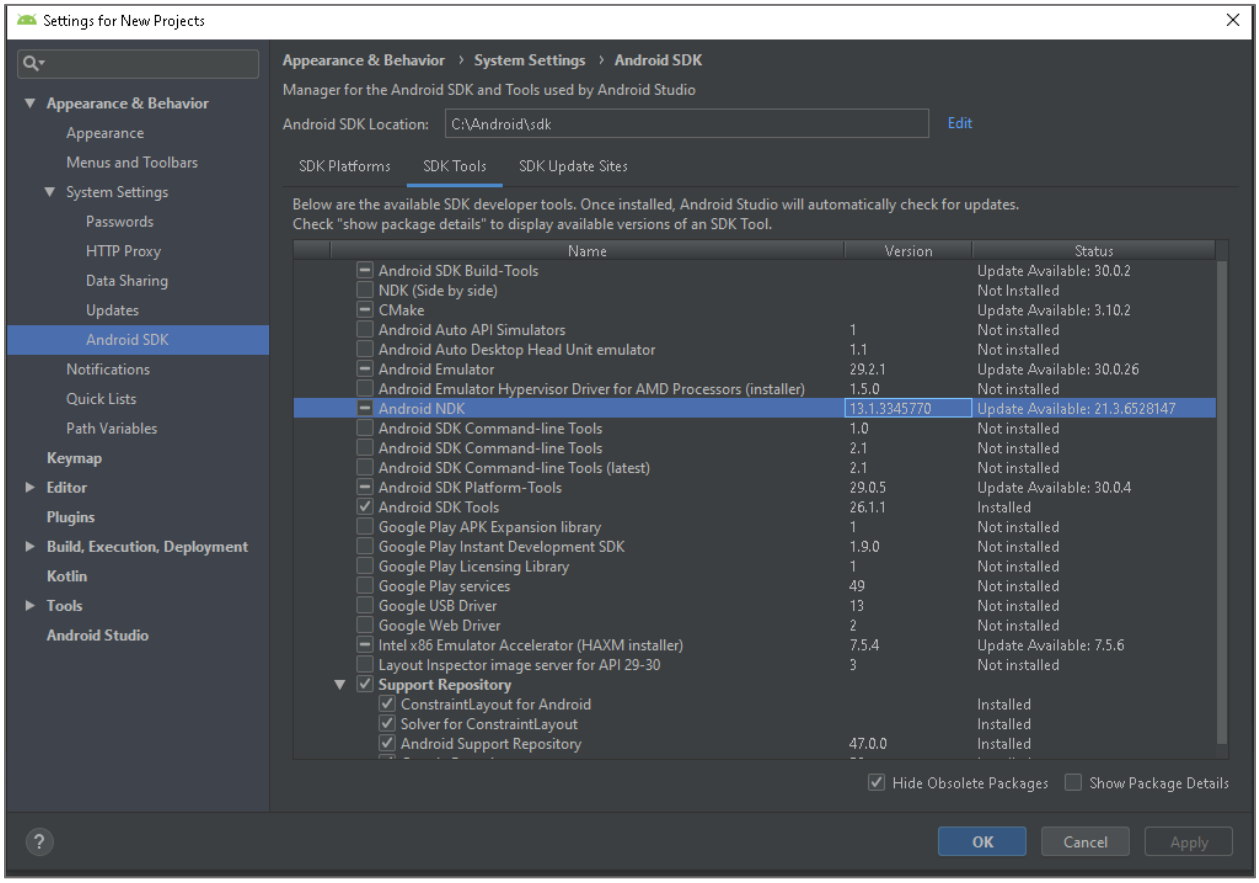

Figure 8 - Android NDK Version.

2.2.3 CMAKE

SDK Tool	CMake
Description	This tool is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native make files and workspaces that can be used in the compiler environment of your choice
Version required	3.6.4111459

Make sure to include the specific Android SDK Platform version in the project. To do that:

1) Click the Android SDK Manager Tool.

2) Check the checkbox corresponding to the indicated CMake version (3.6.4111459).

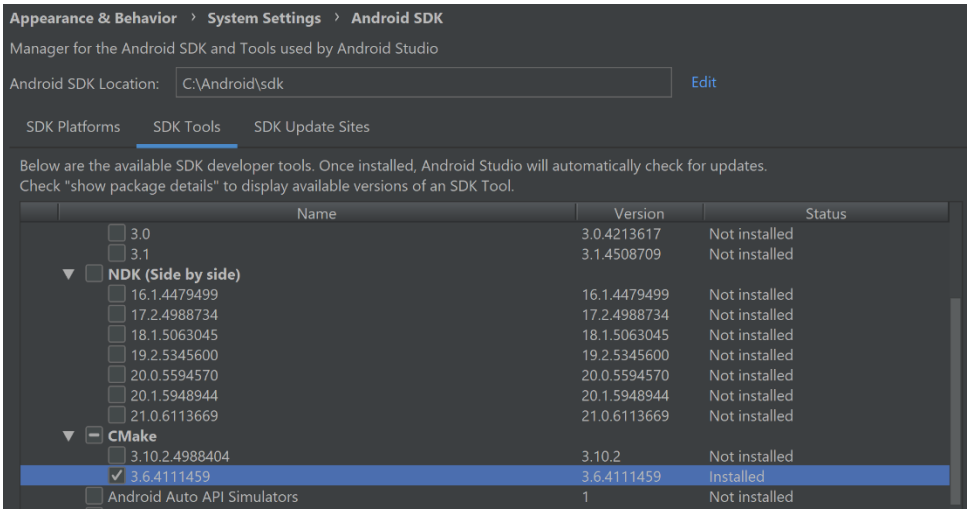

Figure 9 - CMake Version Window.

2.3 CREATING AN ANDROID PROJECT

The following steps should be adhered to when creating a new Android Studio project:

1) Create a new project in Android Studio. If no current project is opened, the user should use the “Welcome to Android Studio” window to choose the “Start a new Android Studio project” option. If a project is already opened, the user should then select “New Project” from the “File” tab.

2) In the “Choose Your Project” screen, the “Empty Activity” option must be selected.

3) In the “Configure Your Project” screen, the following values should be entered:

   - Application Name: "My First Application"
   - Package Name: "com.example.myfirstapp"
   - Language: Java
   - Minimum API level: API 22 (Android 5.1 Lollipop)

4) Click Finish.

The project’s location can be altered based on the user’s preference. However, other options should remain unchanged.

After processing, the Android Studio opens the IDE. The user is advised to take some time to review important files such as “build.gradle” and “AndroidManifest.xml”.

The following link can be used to obtain further information related to the required setups for building the application:

https://developer.android.com/training/basics/firstapp/creating-project.html

2.4 OPENING SAMPLE PROJECTS IN ANDROID STUDIO

To open an existing project, go to "File" → "Open" and then select the desired project to be opened. Refer to Section **1.2.3 - Workspace** for the available sample projects. Take note that the project will take a while to load and will need user confirmation about the settings to be applied to the project. Once the project is opened it will be automatically compiled. Any errors indicating an outdated or missing plugin would require user action to click on the provided links. Please follow the instructions accordingly.

2.4.1 DEMOAPPEMV PROJECT

To start with AMP POS development, users are more likely to utilize the AMP EMV Demo Application, DemoAppEMV, as a reference. It is important to note that after importing the project, the application is expected to compile and run without modifying any part of its source code if the AMP POS SDK package folder is placed inside "C:/" directory. Otherwise, the developer must specify the directories in "CMakeList.txt" file and point it to the desired directory. For more information related to the flow of DemoAppEMV as a project, refer to Section **3 - AMP POS Development Basic Concepts**.

2.5 IMPORTING/ADDING EXTERNAL JAR FILES TO ANDROID STUDIO

A JAR file is a Java Archive Package file, which comprises several Java files to achieve a specific task. Meta data and resources such as text and images are also contained within the file, so that they can be distributed as library files. The JAR files are archived files, built with a *.zip format. These files can be added to the "libs" folder, available under the main "app" folder for providing the project with additional functionality.

2.5.1 ADDING AMP POS 6 AND 8 SERIES LIBRARIES TO THE PROJECT

These steps must be adhered to, for adding the device's API library to an existing project in Android Studio.

1) Under the "app" folder, create a new folder and name it "external-libs", as indicated by the following figure.

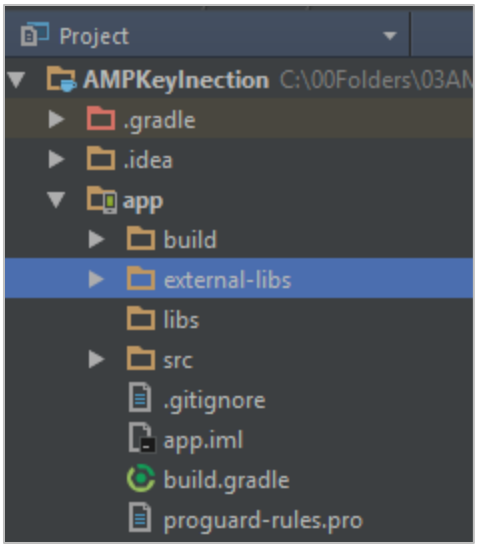

Figure 10 - "external-libs" Folder under "apps" Directory.

2) Browse for the location of SDKXXXX.jar (refer to the "DemoAppEMV" folder inside the "Workspace" folder). The SDKXXXX.jar file should then be copied to the "external-libs" folder that was previously created. The filenames are SDK8Series.jar for AMP 8 Series, SDK6700.jar for AMP 6700 and SDK6500.jar for AMP 6500, respectively.

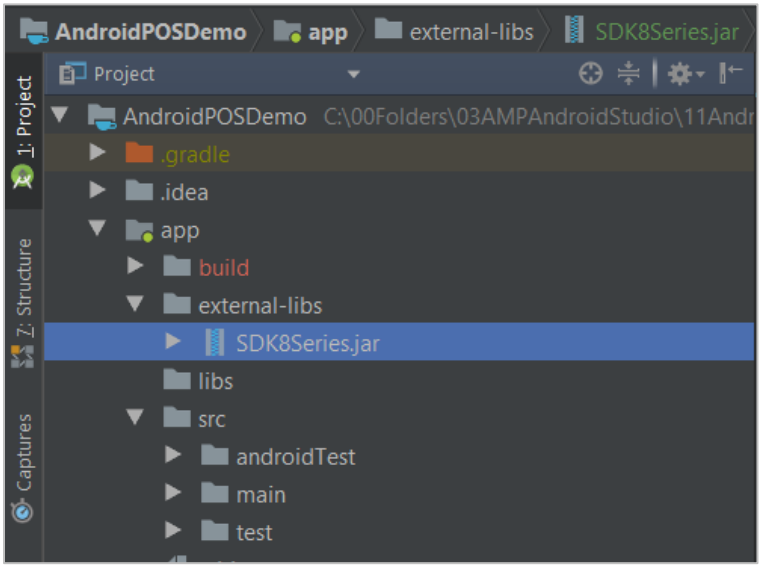

Figure 11 - Copying the API Library to the "external-libs" Folder.

3) To include the file during application compilation, the user should modify the "build.gradle" file and add the "compileOnly files" option as illustrated in the following sample code:

```
compileOnly files ('external-libs/SDK8Series.jar')
```

Note that the parameter in the "compileOnly files" option depends on the JAR file name. Refer to the figure below for the actual code snippet for this step.

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }


}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    androidTestImplementation('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    testImplementation 'junit:junit:4.12'
    api 'com.android.support:recyclerview-v7:27.1.1'
    api 'com.android.support:cardview-v7:27.1.1'

    compileOnly files('external-libs/SDK8Series.jar')

}
```

Figure 12 - Adding Compile-Time Dependency to the "build.gradle" File.

4) The project should then be synced with the Gradle files by clicking the "Sync Project with Gradle file" button as illustrated in the following figure:

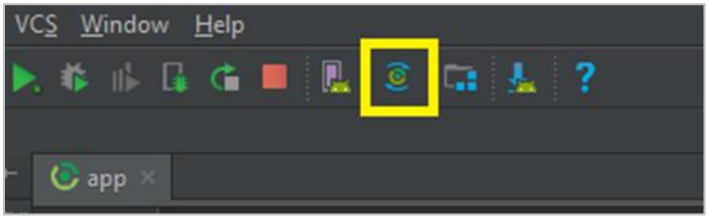

Figure 13 - Syncing the Project with the Gradle Files.

5) Finally, for a successful importing procedure, the "uses-library" tag should be added to the application's manifest file as shown in the following figure.

```
<uses-library android:name="com.pos.device"></uses-library>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.amobilepayment.android.ampkeyinjection">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="AMP Key Injection "
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <uses-library android:name="com.pos.device"></uses-library>

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figure 14 - Declaring the "uses-library" Element in the Manifest File.

2.5.2 ADDING THE ATMS AGENT LIBRARY TO THE PROJECT

The following steps demonstrate how to import the ATMS Agent Library to the project:

1) Select “Project View”, then right-click the application folder. Select “New”, then “Module”.

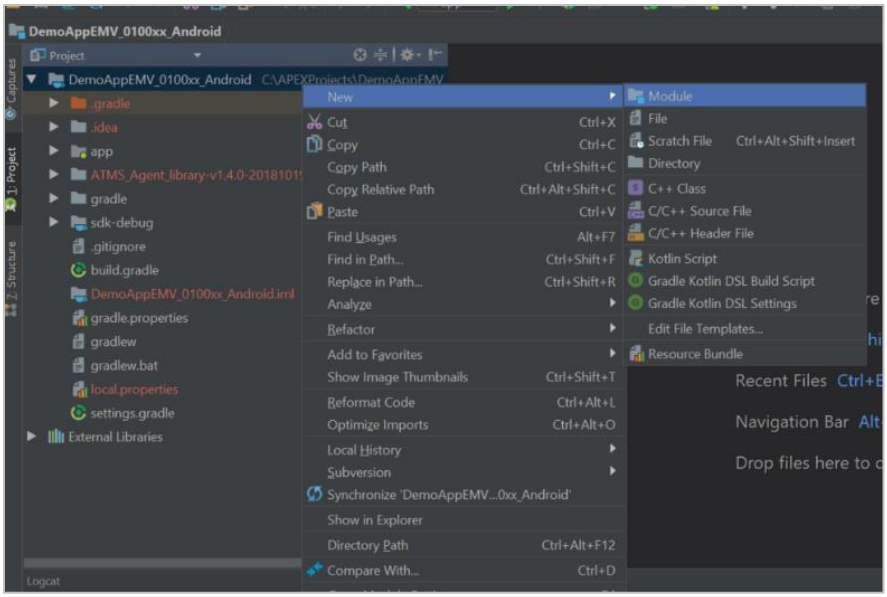

Figure 15 - ATMS New Module Selection.

2) The “Create New Module” dialog will appear. Select “Import JAR/AAR Package”, then click “Next”.

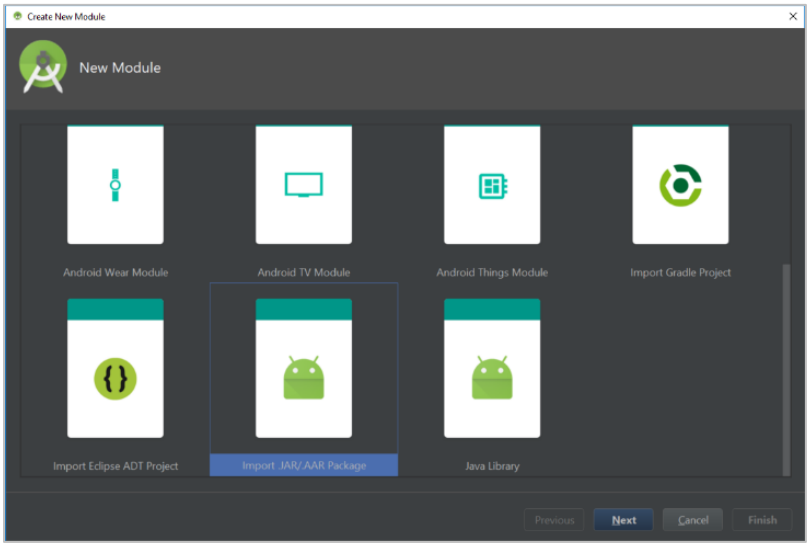

Figure 16 - Import JAR/AAR Package Selection.

3) Browse for the location of ATMS_Agent_library-vX.X.X (refer to the “ATMSSampleCode” folder inside the “Workspace” folder), then click “Finish”.

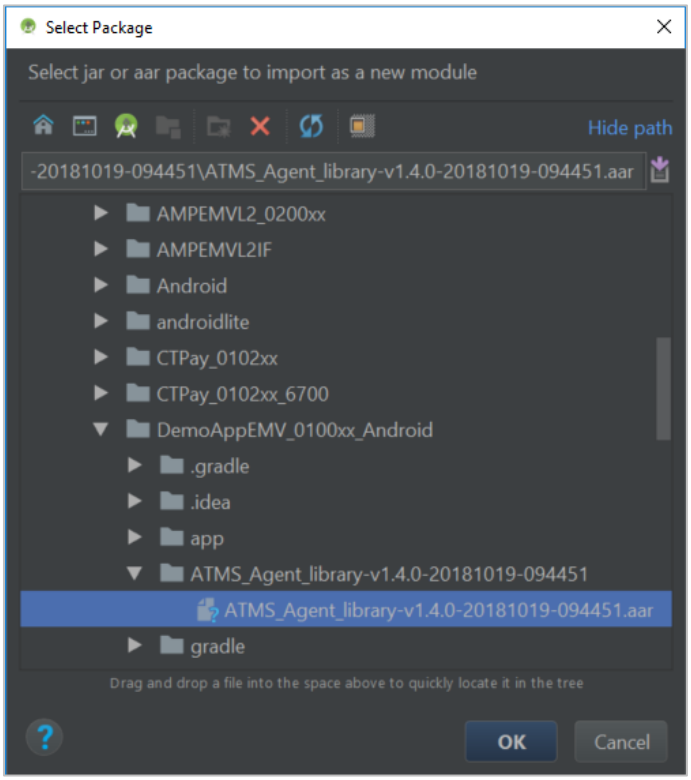

Figure 17 - AAR File Selection.

3 AMP POS DEVELOPMENT BASIC CONCEPTS

3.1 THE AMP PAYMENT APPLICATION SOFTWARE ARCHITECTURE

The following details comprise the general components of the AMP payment application. It must be taken into consideration that this architecture is just a visualization of the used technologies and may or may not be applicable for the development of other applications. This software structure is specifically designed for the AMP payment application. This illustration helps developers to maneuver the software development process for similar projects.

DemoAppEMV (AMP POS vXX.XX.XXXDEM) is provided for reference.

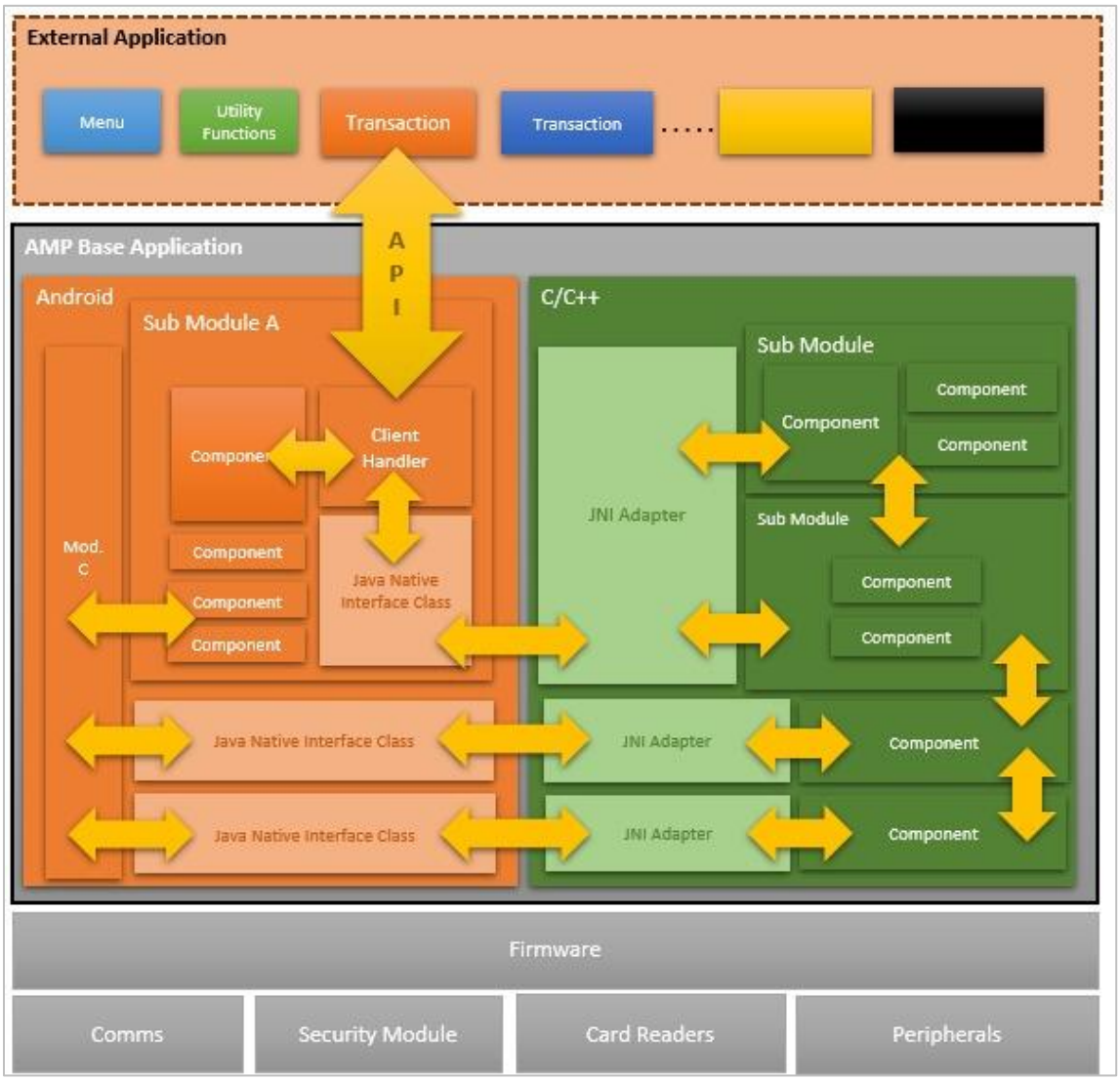

Figure 18 – The AMP Payment Application Software Architecture.

The preceding diagram illustrates the software architecture of the AMP payment application. This is designed to adapt the source code which is native (C/C++) in nature to a platform compatible with Android devices. It consists of the following major components:

3.1.1 ANDROID COMPONENTS (JAVA/ANDROID)

This component contains the necessary functions to interpret the business logic from the native component, which is compatible with specific Android devices. The payment application initialization and menu structuring are also handled by this component, then interlinked with the Java Native Interface (JNI) to cross over technologies. This process is designed to perform tasks which are native in nature. Moreover, it has a direct call to Java AMP SDK for accessing hardware-related functions such as Printer, PED, etc. Android components are organized into the following:

1) Pure Java Functional Sub-components (represented by orange boxes): These components include the user interface, printer, PED, menu, and others, used to control the Android device's hardware-specific functions.

2) Java Native Interface Sub-components (represented by pale orange boxes): This component serves two purposes, as a receiver of the native function calls that are processed as Java callback functions, and as an invoker of the required native functions, such as transaction flow. More details of this component are highlighted in the next section.

3.1.2 NATIVE COMPONENTS (C/C++)

This component contains features related to financial transaction flow, packets, database, and the EMV library and its related components. More importantly, this component contains the business logic / model of the entire application. Native components are organized into the following:

1) Pure Native Functional Sub-components (represented by green boxes): These components provide the application's business logic and overall transaction flow. It is developed using native C/C++ programming language.

2) Java Native Interface Adapter Sub-components (represented by pale green boxes): This component serves two purposes, as a receiving end for Java triggered function calls bridged through the Java Native Interface (JNI), and as an invoker for the Java callback functions that are used to perform specific functions which the Java side can only do such as graphics display, PED, or printer. More details of this component are provided in the next section.

3.2 USING JAVA NATIVE INTERFACE (JNI)

The primary wonder of the AMP POS development for Android application lies in this technology. Its capability to seamlessly cross over software development platforms promises better portability and maintainability of the application.

The Basic JNI flow for an AMP POS payment application is summarized in the steps below:

1) Making Java Callback Object Reference. Before the potential call of a java method from the native side, it is essential that an object reference is passed from the java application. This serves as the reference of the native side to call java methods. To achieve this, the following must be done:

   a) Make Java Callback Interface methods. The JNI processes the native side calls as callback functions. As such, all functions interlinked with Java must be placed in a callback interface.

```
public interface AmpEmvCBIF {
    public void AgnosCBDisplayMessage(String strMessage, int beep, int timeout);
    public int AgnosCBSelectItemFromList(String Title, String[] Items);
    public void AgnosCBGpiSwitchLED(byte Led, byte Flag, byte Colour);
    public int AgnosCBOnlinePinEntry (byte[] Pan, int PanLen);
    public int AgnosCBOfflinePinEntry (byte bEnciphered, int Language, /*byte VerifyOnl
    public byte[] AgnosCBGetEMVResponseCode();
    public void AgnosCBSetTrackData(byte[] track1, int track1sz, byte[] track2, int tra
    public byte[] AgnosGetPinBlock();

    //Agnos Keys
    public int kbGetKey_9210();
    public int kbhit_9210();
    public void kbFlush_9210();
    public void AndroidBeep(int frequency, int durationMs);
}
```

Figure 19 - EMV L2 Java Callback Interface Functions.

   b) Make Java Callback Static method implementation. First, the Java Callback Interface must be implemented in a class to serve its purpose of performing a java specific task. Note that the `static` keyword must be used since invocation is independent of the java class on which it is implemented.

```
public static AmpEmvCBIF listener = new AmpEmvCBIF() {

    Queue<String> qeue_key = new LinkedList<>();
    PinInfo info = null;

    @Override
    public synchronized void AgnosCBDisplayMessage(String strMessage, int beep, int timeout) {
        Log.d(TAG,  msg: "AgnosCBDisplayMessage [START] strMessage=" + strMessage);

        Message message = new Message();
        message.what = MSG_CB_DISPLAY_PROMPT;
        message.obj = strMessage;
        mHandler.sendMessage(message);
        try {
            Thread.sleep( millis: 1200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Log.d(TAG,  msg: "AgnosCBDisplayMessage [END]");
    }

    @Override
    public int AgnosCBSelectItemFromList( String strTitle, String[] astrItems) {
        Log.d(TAG,  msg: "AgnosCBSelectItemFromList [START] Title=" + strTitle);

        for(int i=0; i<astrItems.length; i++)
            Log.d(TAG,  msg: "AgnosCBSelectItemFromList astrItems[" + i + "]= " + astrItems[i]);

        int iRetVal;
        Message message = new Message();
        message.what = MSG_CB_DISPLAY_LIST;
        message.obj = astrItems;
        mHandler.sendMessage(message);
```

Figure 20 - EMV L2 Java Callback Static Method Implementation.

c) Pass the object reference to the native side via the Java Native Interface. Use the `native static` keyword for interfacing since the native call is not dependent on a class.

```
public class AmpEmvL2IF {

    //To get track data for magstripe
    private static byte[] Track1;
    private static byte[] Track2;
    private static byte[] Track3;

    private static byte[] TagValue;

    public native static void InitAmpEmvL2(AmpEmvCBIF obj, String AgnosInitPath, String AgnosConfigPath, String AgnosLibPath);
    public native static int AmpEmvCardEntryPolling(long TimeOut);
    public native static void AmpEmvSetTagCollection(byte[] TagList, int TagListLen);
    public native static byte[] AmpEmvGetTagColData();
    public native static byte[] AmpEmvGetTagData(long TagName);
    public native static byte[] AmpEmvGetAIDInfo();
    public native static byte[] AmpEmvGetTransCategoryCode();
```

Figure 21 - EMV L2 Java Interface.

```
AmpBaseInterface.Initialize(AmpBaseCB.baseCBIF, amp.sdk.lib.AmpCmpnt.AmpCmpntCBIF, context: AmpApplication.this);
```

Figure 22 - Method call to Java Native Interface.

In this case, the `AmpBaseCB.baseCBIF` object is to be passed through the InitAmpEMVL2 Java Interface method.

d) The Java Native Interface class must be compiled in the Java console (`javac` and `javah` commands) for making a native side representation of the Java Native Interface methods.

```
> javac HelloJNI.java
> javah HelloJNI
```

Figure 23 - JNI Command Console Execution.

e) After successfully executing the mentioned commands, it is expected that a header file containing the Java method native function prototype will appear and therefore, the function definition in the native side can be made. Note that the first two elements are supplied by the system. The actual arguments start to be realized in the third parameter. Remember to save the `m_obj` in a global buffer and use it whenever a JNI call from a native function is executed.

```
//! Initialize AMPEMVL2 Framework
JNIEXPORT void JNICALL Java_AmpEmvL2Android_AmpEmvL2IF_InitAmpEmvL2
    (JNIEnv *env, jclass, jobject m_obj, jstring AgnosIniPath, jstring AgnosConfigPath, jstring)
{

    const char *pAgnosIniPath = env->GetStringUTFChars(AgnosIniPath, 0);
    const char *pAgnosConfigPath = env->GetStringUTFChars(AgnosConfigPath, 0);
    const char *pAgnosLibPath = "";

    InitAmpEmvL2(pAgnosIniPath, pAgnosConfigPath, pAgnosLibPath, &gAgnosCB);
    env->GetJavaVM(&g_JavaVM);
    g_obj = env->NewGlobalRef(m_obj);

    //Agnos library
    jni_init(env, m_obj);
    setListenerClass("AmpEmvL2Android/AmpEmvCBIF");

    AndroidReleaseString(env, AgnosIniPath, pAgnosIniPath);
    AndroidReleaseString(env, AgnosConfigPath, pAgnosConfigPath);
}
```

Figure 24 – Native Side Function Definition of Java Native Interface Method.

2) Calling Java Callback method from the native side. A Java function can also be triggered from the native side. This enables the developer to perform a specific function on the Java side seamlessly, as if calling a function from the native side. The following is an illustration of a sample function that is made using this approach.

```
unsigned short  PerformAgnosCBDisplayMessage(const char *szMessage, int beep, int timeout)
{

    JNIEnv *env;
    g_JavaVM->GetEnv((void**)&env, JNI_VERSION_1_6);

    jstring strMessage = env->NewStringUTF(szMessage);
    jclass myClass =    env->FindClass("AmpEmvL2Android/AmpEmvCBIF");
    jmethodID myMethod = env->GetMethodID(myClass, "AgnosCBDisplayMessage", "(Ljava/lang/String;II)V" );

    env->CallVoidMethod(g_obj, myMethod, strMessage, beep, timeout);
    AndroidDeleteObject(env, strMessage);
    AndroidDeleteObject(env, myClass);

    return admNO_ERROR;
}
```

Figure 25 - EMV L2 Sample Java Callback Method Native Call.

For more details related to the syntax of the demonstrated function definition, refer to this link: https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html

This approach is sophisticated in such a way that missing any part of the code will result to a fatal error. Debugging the fatal error is specified in sections **4.2 - Debugging via USB** and **4.3 - Debugging via TCP/IP.**

3.3 JAVA AMP SDK INITIALIZATION

AMP POS SDK offers API functions which can be used to perform specific tasks. These topics are discussed in the following sections.

However, it is important to note that prior to using any of these functions, SDK Manager must first be initialized. To do this, inject the source below in the application:

```
// Initialization of AMP POS SDK
SDKManager.init(context, new SDKManagerCallback()
{
      // function called when the initialization is finished
      @Override
      public void onFinish()
      {
```

```
                // Insert code here
            }
        });
```

4 APPLICATION DEBUGGING

The debugging process is important in the root cause analysis of any developer. AMP POS through Android platform offers APIs that developers can use to trace and fix errors in the application. Before proceeding to the sections below, make sure that the debug cable is properly detected by the workstation with Android Studio IDE when it connects to the AMP POS terminal. For more information on the cables used, refer to **Appendix A**.

4.1 ANDROID LOG FUNCTIONS

4.1.1 NATIVE SIDE DEBUG FUNCTION

This function must be used to log a specific line of code in the native side (C/C++):

Prototype name	`__android_log_print(int prio, const char *tag, const char *fmt, ...)`	
Description	Writes a formatted string to the log, with priority prio and tag tag.	

Input Parameters	prio	Android log priority values, in increasing order of priority
		android_LogPriority{
		ANDROID_LOG_UNKNOWN = 0,
		ANDROID_LOG_DEFAULT,
		ANDROID_LOG_VERBOSE,
		ANDROID_LOG_DEBUG,
		ANDROID_LOG_INFO,
		ANDROID_LOG_WARN,
		ANDROID_LOG_ERROR,
		ANDROID_LOG_FATAL,
		ANDROID_LOG_SILENT
		}

	tag	Source of Log Message.
	fmt	String to be printed, the formatting details are the same as the printf() function in C.
Output Parameters	none	
Return Value	The number of bytes printed; if an output error is encountered, a negative value is returned.	
Support	AMP 8 Series / 6500 / 6700	

4.1.2 JAVA-DEBUG FUNCTION

For Java, this function is used in logging:

Prototype name	`public static int d (String tag, String msg)`	
Class	Log	
Description	Sends a DEBUG log message.	
Input Parameters	tag	String: Used to identify the source of a log message. It usually identifies the class or activity where the log call occurs.
	msg	String: The message to be logged.
Output Parameters	none	
Return Value	The number of bytes printed; if an output error is encountered, a negative value is returned.	
Support	AMP 8 Series / 6500 / 6700	

4.2 DEBUGGING VIA USB

The following steps must be followed to successfully log information for application debugging purposes:

1) Connect the USB cable from the AMP terminal to the PC that runs Android Studio. Refer to **Appendix A** for details on the proper USB cables to use for different terminal models.
   - Note: For AMP 6500, ensure that the proper USB connection is selected. Settings can be found in "Settings → PosSettings → USB Mode Switch → USB Connection", the "USB device" must be selected.
2) Click on the Run/Play button (Shift + F10).
3) The "Select Deployment Target" screen will appear. Select the target device that will be used to run the application and click the "OK" button to proceed with the following step.

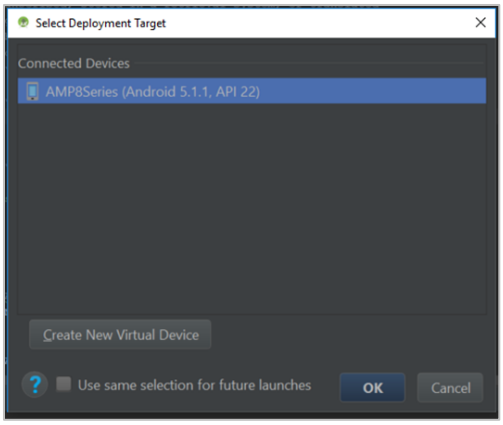

Figure 26 - Selecting the Deployment Target Screen.

4) Go to the Logcat window in Android Studio, the logs will appear once they pass through.

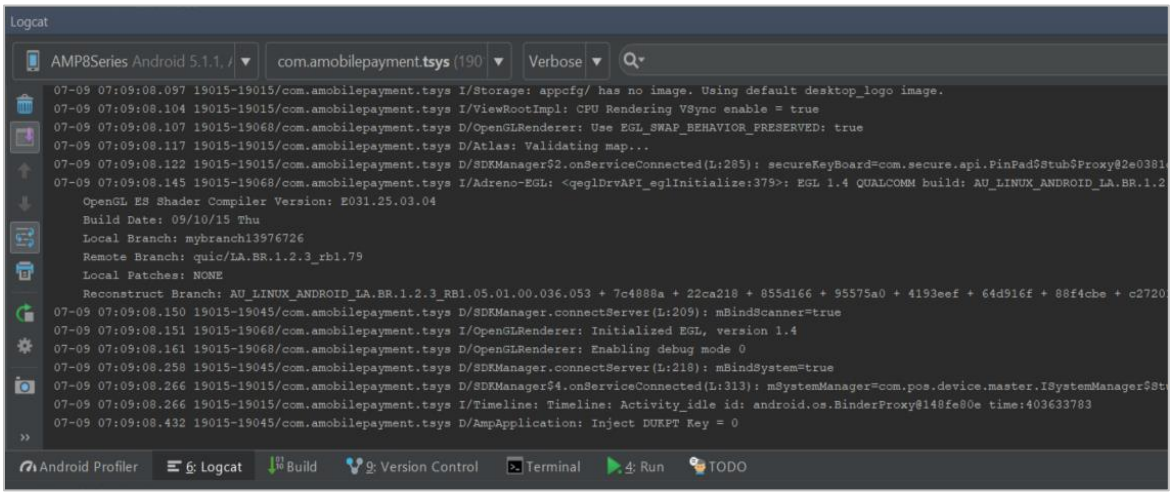

Figure 27 - The Android Logcat Window.

4.3 DEBUGGING VIA TCP/IP

For instances that the USB port is not available for debugging, TCP/IP mode can be used as an alternative debugging mode for devices with WiFi or ETHERNET connection. The following steps summarize debug via TCP/IP:

1) Connect the USB cable from the AMP terminal to the PC that runs Android Studio. Refer to **Appendix A** for details on the proper USB cables to be used for different terminal models.

Note: For AMP 6500, ensure that the proper USB connection is selected. Settings can be found in "Settings → PosSettings → USB Mode Switch → USB Connection", the "USB device" must be selected.

2) Run the following command in the "terminal" tab of Android Studio. This shall enable the connection to restart in TCP/IP mode:

```
adb tcpip 5555
```

3) Check the IP address of the terminal in "About POS" >> "Status" >> "IP address". Use this IP address to connect the terminal via TCP/IP. Below is the command:

```
adb connect <ip_address>
```

4) Remove the USB cable from the terminal. The terminal must now be able to connect to Android Studio and the debug logs should be displayed, without using the cable.

5 AMP 6500 DEVICE FEATURES

AMP 6500 is a robust weatherproof terminal with an Android operating system intended to perform self-service / unattended transactions. It is equipped with a variety of peripherals to support payment transactions under extreme conditions. Below is an overview of some of the vital features of the terminal:

- **5.1 - Anti-Removal Feature**
- **5.2 - MultiDrop Bus (MDB) Interface**
- **5.3 - Light and Proximity Sensor**

These features are furthermore explained in the succeeding sub-sections.

5.1 ANTI-REMOVAL FEATURE

AMP 6500 unattended terminal utilizes an anti-removal detection mechanism to protect it against unauthorized removal after fixing it in its place.

This mechanism comprises of two small tamper switches on top and bottom of front bezel underneath the silicon gasket.

After installation of the terminal on the panel or any other fixture, these switches become under pressure through the gasket pads and any attempt to remove the terminal from its fixture or panel will release the pressure and causes the terminal to be tampered and all sensitive information will be wiped out instantly including application keys and Transport Key. In this case, a "Device is attacked" message will be displayed on the screen.

To clear the tamper and re-inject the Transport Key (TK or TK-1), the Factory Key Loading Device (FKLD) must be used. Please contact AMP support regarding this process.

If a panel or fixture is unavailable to install the AMP 6500, a plexiglass bracket can be used during development and testing.

Figure 28 - AMP 6500 Plexiglass Bracket

This feature can be switched on/off to control its restriction. This is particularly important to eliminate the inconvenience of the terminal being tampered when transporting the device from one location to another. A tempered terminal can no longer be used.

To enable or disable the anti-removal feature, please perform the following:

1) Ensure that the switches in the figure below have been properly pressed by the panel, fixture, or bracket, otherwise the device might get tampered due to the slightest movement.

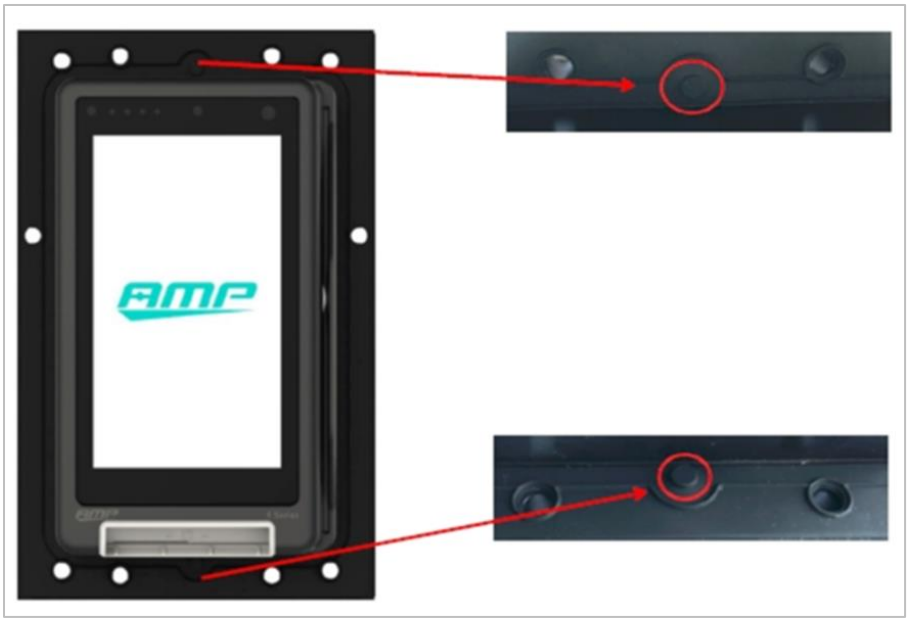

Figure 29 - AMP 6500 Switch Press for Anti-Removal.

2) Select “Settings” from the terminal’s desktop and key in the Admin password. The default Admin password is “1234567”.

3) Select “PosSettings”.

4) Switch “On” or “Off” the toggle button for the “Anti-removal” option. This action enables or disables the terminal’s anti-removal function.

Note: For production terminals, the Anti-removal feature must be switched on. Otherwise, the terminal will have problem executing APIs in the SDK when used in the application, particularly the PIN Entry.

5.2 MULTIDROP BUS (MDB) INTERFACE

The AMP 6500 POS Terminal supports MDB (MultiDrop Bus) for unattended devices, like vending machines and petrol stations, to communicate with the MDB components. Please refer to API-DOC that can be found in the “AMP6500SDKvX.X.X\AMP6500SDK\Guides & Manuals\index.html” directory, for a list of supported functions and their corresponding details.

Prior to using the provided API, SDK initialization must be executed. Refer to section **3.3 - Java AMP SDK Initialization** for details.

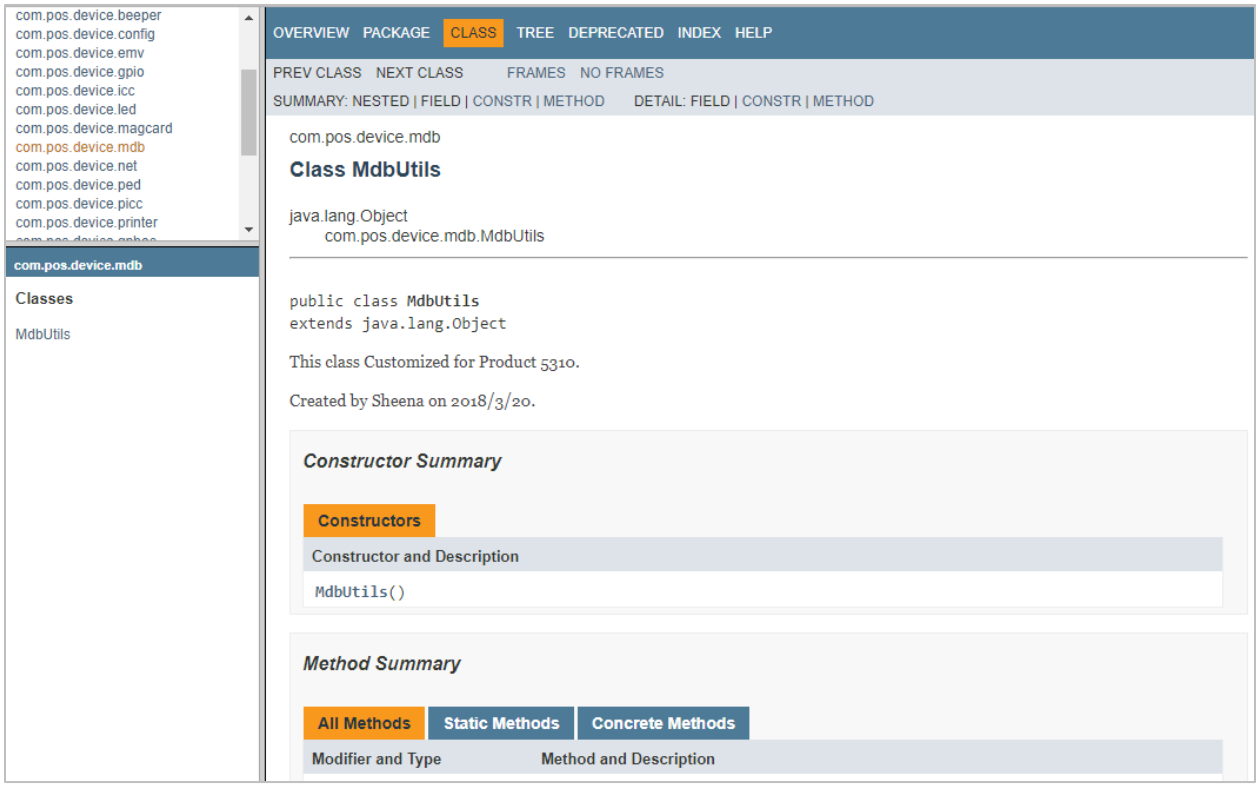

Figure 30 - MdbUtils Class from API-DOC.

To transfer data to a slave device via MDB, use the following function:

```
public static byte[] exchangeFromHostToSlaveMdb(byte addr, byte[]
sendBuf, int timeoutTick;
```

Please refer to the following sample code snippet for using the function:

```
public void salveSendDataTest() {
   int status = 0;
   byte addr = 8;
   byte[] data1 = {0x02, 0x00, 0x01, 0x05, 0x02, 0x00, 0x07,
                   0x01, 0x02, 0x05, 0x19};
   byte[] data2 = {0x41, 0x01, 0x42};
   byte[] data3 = {0x00};
MdbUtils.setMdbMode(1);//slave mode
int result = MdbUtils.setMdbSlaveAddress(addr);
Log.d(TAG, "set slave address result == " + result);

if (result < 0) {
    return;
}
int sendRet = -1;
```

```
while (true) {
   byte[] receiveData = MdbUtils.receiveDataFromHost();
switch (receiveData[0])
{
    case 11:
       switch (status)
         {
          case 0:
          sendRet = MdbUtils.exchangeFromSlaveToHostMdb(receiveData);
          Log.d(TAG, "0, send data result == " + sendRet);
          status = 1;
          break;

          case 1:
          sendRet = MdbUtils.exchangeFromSlaveToHostMdb(data3);
          Log.d(TAG, "1, send data result == " + sendRet);
          break;

          case 2:
          sendRet = MdbUtils.exchangeFromSlaveToHostMdb(data2);
          Log.d(TAG, "2, send data result == " + sendRet);
          data2[1]++;
          data2[2]++;
          break;

          default:
          break;
          }
          break;
          case 9:
          sendRet = MdbUtils.exchangeFromSlaveToHostMdb(data1);
          Log.d(TAG, "9, send data result == " + sendRet);
          status = 2;
          break;

          case 8:
          sendRet = MdbUtils.exchangeFromSlaveToHostMdb(data3);
          Log.d(TAG, "8, send data result == " + sendRet);
          status = 0;
          break;

          default:
          sendRet = MdbUtils.exchangeFromSlaveToHostMdb(data3);
          Log.d(TAG, "default, send data result == " + sendRet);
          status = 0;
          break;
        }
     }
 }
```

5.3 LIGHT AND PROXIMITY SENSOR

AMP 6500 is integrated with a light and proximity sensor that recognizes air gestures and hover manipulations primarily used for processing sleep mode or device locking. It is important to know that AMP 6500 uses VCNL14200 hardware as its sensor, and generic implementations on android devices such as Android Sensor Manager would not work in this case because the SDK has the overall control over its functions.

Prior to using the provided API, SDK initialization must be executed. Refer to section **3.3 - Java AMP SDK Initialization** for details. Otherwise, refer below for the implementation of the feature in AMP terminal:

1) Override the function “onKeyDown()” in the calling activity. The keycode is 227.

2) The sensitivity of the device can be configured in the SDK. Refer below for the class implementation:

```
package com.pos.device.sensor;

public class SensorUtils {
    public SensorUtils() {
    }

    public static int getVcnl4200Sensitivity() {
        throw new RuntimeException();
    }

    public static boolean setVcnl4200Sensitivity(int sensitivity) {
        throw new RuntimeException();
    }
}
```

6 PIN ENTRY MODES

The AMP POS Terminal offers four modes for displaying the PIN Entry screen. Note that for AMP 6500, the Anti-removal feature must be enabled prior to executing any of the provided sample codes. Please refer to **5.1- Anti-Removal Feature** section for details. Also, SDK initialization must be executed. Refer to section **3.3 - Java AMP SDK Initialization** for details. Below are the modes for displaying the PIN Entry screen on Android Terminals:

1) (Default) Numbers are scrambled (shuffled) on the displayed keypad.

2) Numbers are not shuffled on the displayed keypad, but the keypad itself is displayed at random locations on screen, on a smaller scale. The “Confirm” and “Cancel” keys are always displayed at the same location and the same size. To use this mode, refer to the code snippet below:

```
Bundle bundle = new Bundle();
bundle.putInt(Ped.KEYBOARD_VIEW_TEMPLATE,Ped.KEYBOARD_VIEW_TEMPLATE_
DEFAULT);
bundle.putBoolean(Ped.RANDOM_SCALE_LOCATION,true);
Ped.getInstance().setPadViewStyle(bundle);
Ped.getInstance().setPinInputViewRandom(false);
```

3) Numbers are not shuffled on the displayed keypad and the keypad itself is in a fixed position. In other words, the keypad is displayed at a fixed position and all keypad numbers are displayed at a fixed position. The following is the code snippet for using this mode:

```
Bundle bundle = new Bundle();
bundle.putInt(Ped.KEYBOARD_VIEW_TEMPLATE,Ped.KEYBOARD_VIEW_TEMPLATE_
DEFAULT);
Ped.getInstance().setPadViewStyle(bundle);
Ped.getInstance().setPinInputViewRandom(false);
```

4) Support for visually impaired individuals. The keypad is displayed at a fixed position, with fixed position numbers. The cardholder will proceed with the following steps for entering the PIN:

   a) To enter number 5, the user must press one finger anywhere on the screen for 1 second or more.
   b) To enter numbers 1, 2, 3, 4, 6, 7, 8, and 9, the user must enter the number by moving his/her fingers in these directions on the display, as depicted in **Figure 31** below.

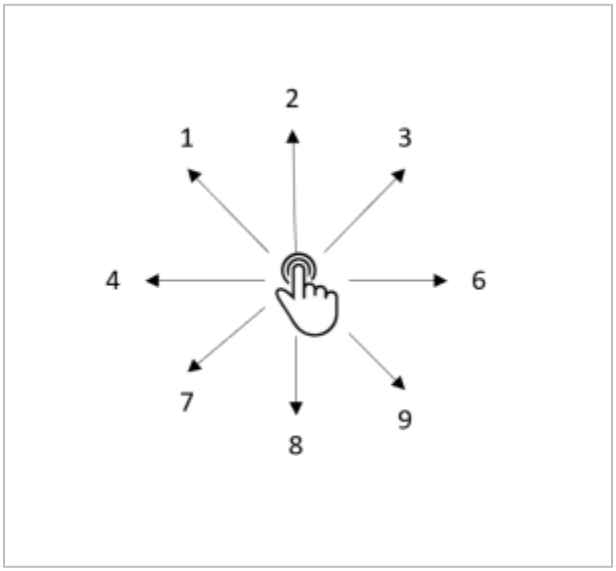

Figure 31 - The Directions Corresponding to the Numbers.

c) To enter number 0, the user should simultaneously press four or five fingers on the screen.
d) To confirm, the user should simultaneously press two fingers on the screen.
e) To cancel, the user should simultaneously press three fingers on the screen.
f) The user can press their finger anywhere on the screen for number entry and for confirmation/cancel button selection.

```
PadView padView = new PadView();
padView.setTitleIcon(BitmapFactory.decodeResource(getResources(),
R.drawable.ic_default_keyboard));
padView.setTitleMsg("Secure PIN PAD(Blind mode)");
Ped.getInstance().setPinPadView(padView);
Bundle bundle = new Bundle();
bundle.putInt(Ped.KEYBOARD_VIEW_TEMPLATE,
Ped.KEYBOARD_VIEW_TEMPLATE_DEFAULT);
bundle.putBoolean(Ped.PIN_ENTRY_FOR_BLIND, true);
Ped.getInstance().setPadViewStyle(bundle);
```

7 COSU (CORPORATE-OWNED, SINGLE-USE) IMPLEMENTATION

The AMP POS supports COSU (Corporate-Owned, Single-Use) implementation, which is designed for corporate-owned devices that fulfill a single use case. This allows developers to further lock down the usage of a device to a single app or a small set of apps. Additionally, it prevents users from enabling other apps or performing other actions on the device.

Note that COSU mode does not completely hide the navigation bar. It only prevents the user from going in and out of the system without authorization (processor password entry), by disabling buttons that will cause the application to exit.

7.1 THE COSU MODE PROCEDURE

Prior to using the provided API, SDK initialization must be executed. Refer to section **3.3 - Java AMP SDK Initialization** for details. For application integration, proceed with the following steps:

7.1.1 DISABLING STATUS BAR AND BOTTOM NAVIGATION BAR

This step can be achieved in two ways:

1) Use setFullScreen API.

To provide COSU mode experience, the user must not be able to see the status bar and bottom navigation bar. Therefore, the screen must be set to a full-screen mode. The following function is used to set the terminal to full screen:

```
public static boolean setFullScreen(boolean enabled);
```

- Parameters:
    - Enabled: true means full screen is enabled; false means it is disabled.

Note that this function is only available in AMP 6500 terminal, firmware versions 1.0.20 or above.

2) Disable the “Home” and “Recent App” buttons and the status bar.

This option can be used when the setFullScreen function is not available. To perform this operation, set the “setHomeRecentAppKeyEnable” function parameters to false. This will prevent the application from exiting after pressing the “Home” or “Recent Apps” buttons. See the syntax provided below.

```
public static void setHomeRecentAppKeyEnable(boolean homeKeyEnable,
   boolean recentAppKeyEnable);
```

- Parameters:
    - homeKeyEnable: true means the home button is enabled; false means it is disabled.
    - recentAppKeyEnable: true means the recent app button is enabled; false means it is disabled.

When enabled, the “Home” and “Recent Apps” buttons will be taken over by the system. This prevents the application from receiving events when button operations are triggered.

Aside from the buttons at the bottom of the screen, the status bar operation must also be disabled to have full control over the device. To achieve this, insert the following code snippet at application start up.

```
public static void setDisableStatusBar(boolean disable);
```

- Parameters:
    - Disable: true means the status bar is disabled; false means it is enabled.

Lastly, to disable the back button functionality, the user can override the function below with an empty implementation:

```
@Override
public void onBackPressed() {
// nothing to do
}
```

7.1.2 SETTING THE APPLICATION FOR LAUNCH AFTER TERMINAL BOOT

Use the "writeLauncherPrioAppInfo" function for enabling the terminal to immediately launch a specific application after terminal boot. The following is the syntax for using the function:

```
public static int writeLauncherPrioAppInfo(java.lang.String pkgName,
      java.lang.String className);
```

- Parameters:
  - pkgName – The application's package name, to be launched after system boot-up.
  - className – The application's class name, to be launched after system boot-up.

It is possible that multiple applications implement COSU mode while calling writeLauncherPrioAppInfo(), setFullScreen(), setDisableStatusBar(), or setHomeRecentAppKeyEnable() functions upon application initialization. This could cause an application to override another application's attempt to set the priority and the behavior of home and recent keys (enabled or disabled). In this case, the last app to execute the two functions takes precedence.

7.2 COSU MODE IMPLEMENTATION CODE SAMPLE

After creating a new project, as outlined in section 2, use the following code to implement COSU Mode in an activity declaration.

Note: COSU Mode needs to be enabled upon opening the application and disabled upon exiting the application. This is required to maintain the functionality of the Home and Recent Apps keys without impacting navigation on the device.

7.2.1 EXAMPLE USING SETFULLSCREEN() API

```
// Full Screen Setting upon opening the application
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        SDKManager.init(this, new SDKManagerCallback() {
            @Override
            public void onFinish() {

SystemManager.writeLauncherPrioAppInfo("com.example.myfirstapp",
                        "com.example.myfirstapp.MainActivity");
                SystemManager.setFullScreen(true);
            }
        });
    }
```

```
// To disable SetFullScreen API upon exiting the application
    @Override
    public void onFinish() {

SystemManager.writeLauncherPrioAppInfo("com.example.myfirstapp",
                "com.example.myfirstapp.MainActivity");
        SystemManager. setFullScreen(false);
            }
        });
    }
}
```

7.2.2 EXAMPLE BY DISABLING NAVIGATION BUTTONS AND STATUS BAR

```
// To disable the Home and Recent App keys upon opening the application
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        SDKManager.init(this, new SDKManagerCallback() {
            @Override
            public void onFinish() {

SystemManager.writeLauncherPrioAppInfo("com.example.myfirstapp",
                        "com.example.myfirstapp.MainActivity");
                SystemManager.setHomeRecentAppKeyEnable(false,
false);
             SystemManager.setDisableStatusBar(true);
            }
        });
    }

// To enable the Home and Recent App keys upon exiting the application
    @Override
    public void onFinish() {

SystemManager.writeLauncherPrioAppInfo("com.example.myfirstapp",
                "com.example.myfirstapp.MainActivity");
        SystemManager.setHomeRecentAppKeyEnable(true, true);
        SystemManager.setDisableStatusBar(fals);

            }
        });
    }
}
```

8 TERMINAL PRINTING

Since receipt printing is necessary in AMP POS terminals due to the nature of its transactions, AMP POS also provides APIs for printing receipt. Prior to using the API provided, SDK initialization must be executed. Refer to section **3.3 - Java AMP SDK Initialization** for details. Otherwise, refer below for the sample implementation of receipts:

```
/**
* SAMPLE CODE FOR PRINTING
*
*/

int printerWidth = 383;
int y = 0;
int textsize = 17;
String text = null;

// Initialize objects needed for printing
Paint paint = new Paint();
PrintCanvas printCanvas = new PrintCanvas();
Printer printer = Printer.getInstance();

// Check if printer is available
if(printer.getStatus() != Printer.PRINTER_OK)
{
Log.d("PRINTER", "Error="+printer.getStatus());
return;
}

// Place items in the canvas

// Left alignment
text = "Sample text left aligned";
paint.setTypeface(Typeface.defaultFromStyle(Typeface.NORMAL));
paint.setTextSize(textsize);
y += textsize;

printCanvas.setX(0);
printCanvas.setY(y);
printCanvas.drawText(text, paint);

// Center alignment
textsize = 30;
text = "Sample text center aligned";
paint.setTypeface(Typeface.defaultFromStyle(Typeface.BOLD));
paint.setTextSize(textsize);
y += textsize;

printCanvas.setX((int)((printerWidth - paint.measureText(text))/2));
```

```
printCanvas.setY(y);
printCanvas.drawText(text, paint);
y += 17; // add offset if font is bold

// Right alignment
textsize = 17;
text = "Sample text right aligned";
paint.setTypeface(Typeface.defaultFromStyle(Typeface.NORMAL));
paint.setTextSize(textsize);
y += textsize;

printCanvas.setX((int)(printerWidth - paint.measureText(text)));
printCanvas.setY(y);
printCanvas.drawText(text, paint);

// Draw image
Bitmap image = null ;

try {
InputStream is =
getApplicationContext().getAssets().open("sample.png");
image = BitmapFactory.decodeStream(is);
is.close();
} catch (IOException e) {
e.printStackTrace();
}

printCanvas.drawBitmap(image, paint);

// Make a printtask and assign a canvas
PrintTask printTask = new PrintTask();
printTask.setPrintCanvas(printCanvas);

// Call print function
printer.startPrint(printTask, new PrinterCallback() {
@Override
public void onResult(int i, PrintTask printTask) {
Log.d(TAG,"startPrint result="+i);
}
});
```

APPENDIX A

AMP USB CABLE LIST

1) AMP 8 Series Download Cable

Figure 32 - AMP 8 Series Download Cable.

Name	AMP 8 Series Download Cable
AMP Part Number	AM-CBL-POS-1014
Description	1.5-meter USB Type A male + USB Micro Type B male
Compatible AMP Models	AMP 2 and 8 Series Cable
Application	Downloading, debugging and key injection

2) AMP 6 Series Download Cable

Figure 33 - AMP 6 Series Download Cable.

Name	AMP 6 Series Download Cable
AMP Part Number	AM-CBL-POS-1013
Description	1.5-meter USB Type B male + USB Type A male
Compatible AMP Models	AMP 6 Series
Application	Downloading, debugging and key injection