Integration between JCC's financial application and third-party applications for Ingenico AXIUM DX8000 devices

Contents

Introduction

The Ingenico AXIUM DX800 POS device is operating under a customized Android 10 version Operating System. This gives us the capability to integrate our main financial application with other external applications that can be installed on the device and exchange data between them.

The integration is made through a Library Module called VAS Library (VAS – Value Added Service). A common version of VAS library must be imported on both applications, the two applications can exchange data in a safe way through encryption of the data in the messages under a common key using AES cryptographic method.

In this document we demonstrate a showcase for this integration, examples of flows, examples of covered cases and the steps to implement the functionality in any third-party native android application.

Possible examples of integration

The fact that the Ingenico AXIUM DX8000 is running Android 10 gives us more options regarding integration with other applications that may be installed alongside our main financial application solution on the DX8000.

In this document the case we demonstrate is about a coffee store and how the process of product selection, ordering and payment by card can happen on the terminal itself.

We consider the case that merchant has already a customized application with the company's provided services and options.

The case we have as Demo handles the cases of:

1) Selection of available products (on the side of the Demo Coffee Application)
2) Customization of selected products (on the side of the Demo Coffee Application)
3) Payment of an order (on the side of the Mellon Fintech Application)
4) After payment completed successfully there is a slip printed with the transaction details (on the side of the Mellon Fintech Application)
5) After successful or unsuccessful transaction (on the side of the Mellon Fintech Application) we get back to our Demo Coffee Application ready for the next order

In this way, a merchant can make use of the power that DX8000 and Android OS gives us and integrate his/her application with Mellon's Financial Application. The variety of different type of services that can be covered are:

1) Simple shop type (simple sale financial action)
2) Restaurant type (sale with extra TIP option or other related features)
3) Hotel type services (pay for accommodation services options to show room and departure date on final receipt)
4) Supermarket cases (with customized receipt)
5) Rent a car cases (extra options for contract number information and date of payment)

The above cases are covered through parametrization of the DX8000 device depending on the selected industry type of each merchant business. Ui and receipts of the financial application are customized based on the active industry code.

VAS (Value Added Service) library specifications and information

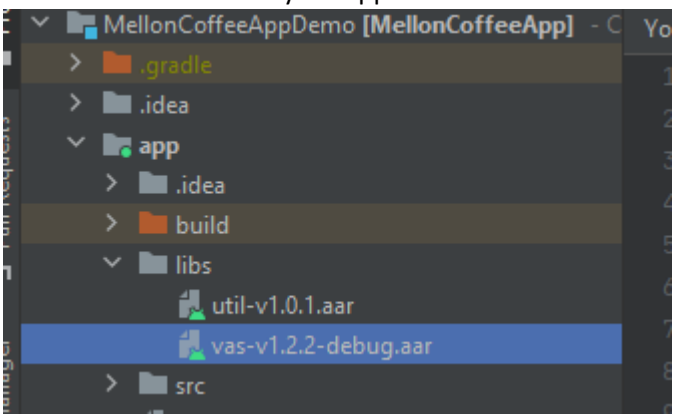
VAS library is provided by Ingenico and can be integrated on an external(third-party) application. VAS enables a third-party application installed on DX8000 terminal to call the financial application to perform a transaction. Common flow of integration with VAS Library would be:

1) The third-party application initiates the transaction
2) Control is transferred to the financial application
3) User performs the transaction (presenting the card, entering PIN where appropriate etc.)
4) Finally, the transaction is finalized (Approved or rejected). Transaction result is returned to the calling application and control is transferred back there.

Technical guides and steps for an external project to be able to communicate with our financial application

Prerequisites for integration of VAS services into your merchant application are:

1) You have an updated Android Studio version (4.0 or newer) installed.
2) vas-v1.2.2-debug.aar (or equivalent, provided by Mellon) library that you will place inside the libs folder of your application.



3) Adding the required dependency for libs directory on your project's build.gradle module inside the block dependencies{}, as shown below.

```
plugins {
    id 'com.android.application'
}
dependencies {
    repositories {

        flatDir {
            dirs 'libs'
        }
    }
}
```

4) Also adding the required dependency for libs directory on your project's build.gradle module (app level) inside the block dependencies{}, as shown below.

```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.aar"])

    implementation 'androidx.appcompat:appcompat:1.5.1'
```

5) Now you can begin working on your application.
6) Additionally, we can provide a demo project as an example for your own implementations.

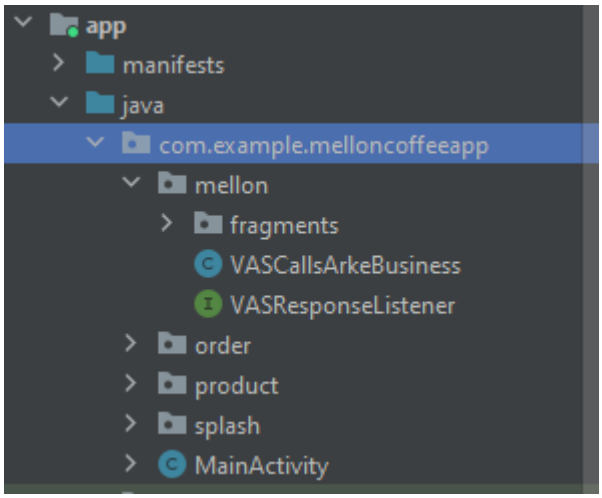
Example Implementation (MellonCoffeeApp Demo)

The first part of the example is our financial application based on our solution for local National Bank of Greece that will be provided as an apk file and will be the project from which the VAS library aar files and versions should be exported for usage in the external (3rd party application). Application is modified for mimic-online transactions meaning that there are no real transactions or communications made. Application is provided for test and demo purposes only. The only thing is needed is on first installation of the app is that the user has to enter a test Terminal Id provided by Mellon to perform parameter download and setup the financial application to support transactions. If agreed, we can have this value set by default or have the financial application and the DX8000 device already initialized from our side regarding the first setup.

The MellonCoffeeApp Demo application represents a coffee store merchant app which allows us to select products, add to basket, go to basket view and perform a SALE transaction (SALE) with payment button click with total amount of added products.

Note that through VAS integration we are also able to void a previously performed SALE transaction (VOID) or to reprint (PRINT_LAST) the last succeeded transaction performed.

The demo application has the following structure.



Notice the two added java classes:

1) VASCallsArkeBusiness class. Provides implementation of calls to the financial application.
2) VASResponseListener interface. A listener interface used to inform the merchant application that the call to financial application is completed.

In our OrderDetailsActivity inside order folder, we define a VASCallsArkeBusiness object called “vas”

```
private VASCallsArkeBusiness vas = null;
```

Inside the onCreate() method of OrderDetailsActivity we call initVas() in order to initialize our VAS object. Below the initVas() method:

```
public void initVas()
{
    if(vas==null)
    {
        vas = new VASCallsArkeBusiness( context: this);
        vas.addListener(this);
    }
}
```

We have to implement VASResponseListener also to handle responses from the financial application

```
public class OrderDetailsActivity extends AppCompatActivity implements VASResponseListener {
```

Below image shows the override of the responseReceived() listener method that will signal our application that the vas call is completed.

```
@Override
public void responseReceived() {
    response = vas.getResponseData();
    Log.i( tag: "response JSON:" , response);
    JSONObject sourceObject;
    try {
        sourceObject  = new JSONObject(response);


        if(sourceObject.get("responseMessage").equals("Transaction succeeded")
                ||sourceObject.get("responseMessage").equals("Επιτυχημένη συναλλαγή"))
        {
            totalBasketProducts = 0;
            totalBasketAmount = 0;
            basketProducts.clear();
            this.finish();
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Calls and method arguments examples

Our VAS related calls are performed using the doTransaction call of VASCallsArkeBusiness. Before the call to doTransaction the call arguments should be set using setSendData (String sendData) method.

The parameters are in JSON string format.

For SALE:

{“amount”:<floating point value>,”orderNumber”:”<order no>”,”needAppPrinted”:<true ή false>}

E.g.

{“amount”:1.00,”orderNumber”:”1”,”needAppPrinted”:true}

For VOID:

{“originalVoucherNumber”:”<voucherNumber>”,”orderNumber”:”<orderNumber>”}

E.g.

{“originalVoucherNumber”:”45”,”orderNumber”:”1”}

Note that VOID implementation is based on the functionality of the financial application. In current demo case void transactions are not performed but we are able to perform them in different implementation by sending the initial orderNumber of a completed sale to the financial application.

For reprint of the last transaction no arguments are required so we can set the parameters as an empty sting “”.

Conclusion

If there is interest from any side for this integration we presented, we can share the following items:

1) The MellonCoffeeApp example as a demo project ready to run and install on the Intenico AXIUM DX8000 terminal through Android Studio.
2) The VAS Library aar file separately for implementation with other applications
3) An apk file of the financial application customised for mock-up transactions to be installed on the Ingenico AXIUM DX8000 device. The installation of apk files on a DX8000 device can be done through a specific software provided by Ingenico named “AXToolKit” and we can share it also with any interested side or have the terminal ready from our side with the financial application already installed and set.

Finally, we can also provide other customized solutions as a demo for integration as we already have examples that can perform Sale, Void and print last transaction receipt actions through the financial application. Options for integration with our financial application are not restricted only to the one case we presented in this document.