Smart contract Java

Prerequisites

- Android Studio\Installation: https://developer.android.com/studio/install
- Install JAVA 11\Installation: JAVA 11
- Install Android 31 API on Android Studio

To use the Android Kalima API, it is recommended to have previously read the documentation Android_Example and to have installed the Java JDK in version 11 at least.

A complete Android smart contract example is available on our public GitHub: KalimaAndroidExample.

To create an Android smart contract, you just need the **kalima-android-lib-release.aar**,API, so you should implement this API in your project to be able to create your Java smart contract

1- Smart contracts?

A Smart Contract is a computer program that automates the execution of a set of predefined instructions when prerequisites are met.

During its execution, all validation steps are recorded at the blockchain level. These records ensure the security of all data by preventing its modification or deletion.

Android smarts contracts == " KalimaCacheCallback " which we will see later

Android smart contracts are in the form of an instance of the class KalimaCacheCallback. This callback is implemented to react to certain events such as the creation of a new cache or the arrival of new transactions.

Kalima Cache Callback

The object KalimaCacheCallback will allow us to react in an event-driven way to interactions with the blockchain. Below is an example of instantiation:

```
kalimaCacheCallback = new KalimaCacheCallback.Stub() {
  @Override
  // Callback for incoming data
  public void onEntryUpdated(String cachePath, KMsgParcelable kMsgParcelable)
  throws RemoteException {
    Log.d("onEntryUpdated", "cachePath="  + cachePath + ", key=" +
    kMsgParcelable.getKey());
  }

  // Callback for deleted data
  @Override
  public void onEntryDeleted(String cachePath, String key) throws RemoteException
  {
    Log.d("onEntryDeleted", "cachePath="  + cachePath + ", key=" + key);
  }

  @Override
  public void onConnectionChanged(int i) throws RemoteException {
  }
};
```

As we can see, 3 callbacks are available:

- onEntryUpdated➔ This callback will be called whenever data is added /modified in cache. The callback has two parameters: The address that allows you to know where the address the data arrived, and the message in the form of KMsgParcelable (which allows you to retrieve the information of the message). An easy example of using this Callback would be updating a message list. A message has been added or changed, so you update your list.
- onEntryDeleted➔ This callback will be called every time a message is deleted in cache. It has two settings: The address on which the message was deleted, and the key of the deleted message. A simple example of using this callback would be updating a message list. A message has been deleted, so you'llkeep it off your display list.
- onConnectionChanged➔ This callback will be called each time a disconnection or reconnection occurs with one of the Master Nodes. This Callback can for example be useful to warn the user that he is no longer connected to the Blockchain, or that one of the notary no longer responds.

To work, we must switch to the service the implementation of our Callback, when connecting our activity and the service (which will be detailed just after), with the help of the following line:

```
kalimaServiceAPI.addKalimaCacheCallback(kalimaCacheCallback);
```

Similarly, when our activity is no longer in the foreground, we will want to delete the implementation of the callback, thanks to the following line, which we can for example use in the function onPause of the activity life cycle:

```
kalimaServiceAPI.unregisterKlimaCacheCallback(kalimaCacheCallback);
```