Exemple Java

1. Add the Kalima Jar in the project

To test the Java example, it is necessary to have some prerequisites:

• An integrated development environment such as Eclipse. Download link: https://www.eclipse.org/downloads/packages/

• A JDK Java Development Kit. You need at least version 9 of the JDK for the example to work.

To start, you must include the jar Kalima.jar in to your project. For exemple, under Eclipse, place the jar somewhere in your project, then ➔ right clic on the jar ➔ Build Path ➔ Add to Build Path.

You have now access to Kalima API in your project

2. Initialisations

The code below allows you to initialize a certain number of objects and to connect to the Blockchain.

```
clonePreferences = new ClonePreferences(args[0]);
logger = clonePreferences.getLoadConfig().getLogger();

node = new Node(clonePreferences.getLoadConfig());
clone = new Clone(clonePreferences, node);

serverCallBack = new KalimaServerCallBack(this);
clientCallBack = new KalimaClientCallBack(this);

try {
      node.connect(serverCallBack, clientCallBack);
} catch (IOException e) {

      logger.log_srvMsg("ExampleClientNode", "Client", Logger.ERR,
      "initComponents initNode failed : " + e.getMessage());
}

for(Map.Entry<String, KCache> entry : clone.getMemCaches().entrySet()) {
      clone.addListnerForUpdate(new
      ChannelCallback(entry.getValue().getCachePath()));
}
```

The Node will be responsible of the connexion with the Blockchain

Le clone is responsible of the synchronisation of the data in cache memory.

The clientCallBack allows you to react to the addition of new transactions in the Blockchain (see next chapter).

We can see that we have to pass args[0] during the creation of the clonePreferences. Indeed, you have to launch your client by passing the path of a configuration file. We will see the contents of this file later.

3. Callbacks

As we have seen previously, we have to pass two callback classes to the Node object. The serverCallback is not useful for an ordinary node. So you can simply create a simple class that inherits ServerCallback, without putting anything in the methods.

CustomerCallBack is more important, and requires a few mandatory lines. It will allow you to react to the arrival of new transactions. To start, create a class that inherits ClientCallback, then add the missing methods : On Eclipse, click on the error (on the left, next to the line numbers) ➔ « Add unimplemented methods ».

The putData function will be called at each new transaction received, you must at least add the code below, and then customize your code according to the desired behaviour.

```
KMsg kMsg = KMsg.setMessage(msg);
client.clone.set(kMsg.getCachePath(), kMsg, true, false);
```

The onConnectionChanged function will be called at every connection / disconnection with one of the Notary Nodes. You must at least insert the code below, and you can add code as needed.

```
client.clone.onConnectedChange( (status==Node.CLIENT_STATUS_CONNECTED) ? new
AtomicBoolean(true) : new AtomicBoolean(false), nioClient, false);
```

The onNewCache function is called every time a new Cache is created in our Node. All caches will be created at the beginning of the connection, during synchronization. We can create callbacks for each Cache. In this example, a callback has been created to manage smarts contracts. This callback is subscribed to in the onNewCache function:

```
client.getClone().addListnerForUpdate(new
SmartContractCallback(cachePath, client, contractManager));
```

4. Smarts Contracts (SmartContractCallback)

Smarts contracts are stored on git but validated by the Kalima blockchain. All the management of these smarts contracts is integrated in the Kalima API. To be able to execute smarts contracts from our Node, you just have to provide the login information (login, password) of an authorized account on the git directory where the smarts contracts are stored.

In the example, the identifiers are requested at the start of the application.

```
contractManager.loadContract(GIT_URL, GIT_USERNAME, password,
kMsg.getKey(), kMsg.getBody());
```

Once loaded, a smart contract can be executed:

```
private void runScript(KMsg kMsg) {
		String scriptPath = logger.getBasePath() + 
"/git/KalimaContractsTuto" + kMsg.getCachePath() + ".js";
		try {
			String result = (String) 
contractManager.runFunction(scriptPath, "main", logger, kMsg, 
client.getClone(), client.getNode());
			logger.log_srvMsg("ExampleClientNode", "TableCallback", 
Logger.INFO, "script result=" + result);
		} catch (Exception e) {
			logger.log_srvMsg("ExampleClientNode", "TableCallback", 
Logger.ERR, e);
		}
	}
```

In this example, the runScript function is called when the node receives a new data. And this node will run the script named as the cache path. For exemple, if the node receive a data in /alarms/fire, so he will run the script KalimaContractsTuto/alarms/fire.js, if exist.

Bindings are used to pass objects to the scripts. We pass the logger, the kMsg (the message received), the clone and the node.

5. Configuration file

Here is an example of configuration file :

```
LedgerName=KalimaLedger
NODE_NAME=Node Client Example

NotariesList=62.171.131.154:9090,62.171.130.233:9090,62.171.131.157:9090,144.91
.108.243:9090
FILES_PATH=/home/rcs/jit/ClientExample
SerialId=PC1245Tuto
```

- LedgerName ➔ Is not yet used in the current version
- NODE_NAME ➔ You can put something that allows you to recognize your node
- NotariesList ➔ Comma-separated list of notary's addresses and ports
- FILES_PATH ➔ This is the path where will be stored the files useful to Kalima, as well as the logs.
- serialId ➔ It is an identifier that will allow the authorization on the blockchain at the first launch of the client node (provided by Kalima Systems in the case of a test on our Notary).

6. Code execution

To test your project, you can run the code from Eclipse, or from a command line console. You just have to pass as a parameter, the path of the configuration file.

Execution from Eclipse :

In Run ➔ Run Configurations ➔ right clic on « Java Application » ➔ New Configuration.

⇨ Choose a name for the configuration
⇨ under « Project » clic on « Browse » and choose your project

- ⇨ under « Main class » clic on « Search » and select the class including the Main method that you will launch (here : Client.java)
- ⇨ In the "Arguments" tab, under "Program arguments", give the path of the config file (here: etc/cfg/node.config)

Your configuration is ready, you can execute it.

Command line execution

You can also generate the jar and then execute it on the command line. On Eclipse, right click on your project ➔ Export, chosse Java ➔ Runnable Jar File ➔ Next.

In the "Runnable JAR File Export" window, choose your configuration under "Launch Configuration", and choose a destination for your jar (ex: /Documents/git/KalimaTuto/TutoClient/etc/jar/TutoClient.jar), then click on "Finish".

then, from the console:

```
cd /Documents/git/KalimaTuto/TutoClient/etc/
java -jar jar/TutoClient.jar cfg/node.config
```

7. Results

The example program connects to the Blockchain and sends 10 messages (1/second). The TTL (Time To Live) of these messages is 10, which means that each message will be automatically deleted after 10 seconds (a transaction will take place on the blockchain for each deletion). So, if your code is correct, you have correctly configured the configuration file, and your device is authorized on the blockchain, you should have something similar in your console after 2 seconds:

```
Do you want use Smart Contracts ? (Y/n)
n
log_srvMsg:KalimaMQ:KeyStore:60:setDevId deviceID=30dbd16c-1e0e-3265-8492-
de8b14f9fb3e
log_srvMsg:KalimaMQ:NioServer:60:NEW SERVER port ServerSocketChannel:9118
log_srvMsg:KalimaMQ:Node:60:[connect new NioClient] 167.86.124.188:9090
log_srvMsg:KalimaMQ:Node:60:[connect new NioClient] 62.171.130.233:9090
log_srvMsg:KalimaMQ:Node:60:[connect new NioClient] 62.171.131.157:9090
log_srvMsg:KalimaMQ:Node:60:[connect new NioClient] 144.91.108.243:9090
log_srvMsg:KalimaMQ:Node:60:[handleConnection add node ] 62.171.131.157:9090
myClients.size=1
log_srvMsg:KalimaMQ:Node:60:[handleConnection add node ] 62.171.130.233:9090
myClients.size=2
log_srvMsg:KalimaMQ:Node:60:[handleConnection add node ] 167.86.124.188:9090
myClients.size=3
log_srvMsg:KalimaMQ:Node:60:[handleConnection add node ] 144.91.108.243:9090
myClients.size=4
log_srvMsg:KalimaMQ:Node:60:Node subscribe
log_srvMsg:KalimaMQ:Node:60:Node subscribe
log_srvMsg:KalimaMQ:Node:60:Node subscribe
log_srvMsg:KalimaMQ:Node:60:Node subscribe
log_srvMsg:KalimaMQ:Node:60:Node getSnapshotFromNotaryNodes
snapshotForAllCaches=true
```

```
log_srvMsg:NodeLib:Clone:60:addCache : /alarms/fire
log_srvMsg:NodeLib:Clone:60:addCache : /alarms/fire.hdr
log_srvMsg:NodeLib:Clone:60:addCache : /alarms/fire.val
log_srvMsg:NodeLib:Clone:60:addCache : /alarms/fire.fmt
log_srvMsg:NodeLib:Clone:60:addCache : /alarms/fire.json
log_srvMsg:NodeLib:Clone:60:addCache : /sensors
log_srvMsg:NodeLib:Clone:60:addCache : /sensors.hdr
log_srvMsg:NodeLib:Clone:60:addCache : /sensors.val
log_srvMsg:NodeLib:Clone:60:addCache : /sensors.fmt
log_srvMsg:NodeLib:Clone:60:addCache : /sensors.json
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Scripts
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/Kalima_Scripts key=Kalima-
Tuto/etc/scripts/reverse_string.js sequence=1
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/Kalima_Scripts
key=KalimaContractsTuto/KalimaExamples/reverse_string.js sequence=3
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Scripts.hdr
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Scripts.val
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Scripts.fmt
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Scripts.json
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_User
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_User.hdr
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_User.val
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_User.fmt
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_User.json
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Password
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Password.hdr
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Password.val
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Password.fmt
log_srvMsg:NodeLib:Clone:60:addCache : /Kalima_Password.json
GO
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key0 sequence=603
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key1 sequence=604
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key2 sequence=605
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key3 sequence=606
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key4 sequence=607
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key5 sequence=608
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key6 sequence=609
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key7 sequence=610
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key8 sequence=611
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key9 sequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key0 sequence=613
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key0
seq=613 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key1 sequence=614
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key1
seq=614 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key2 sequence=615
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key2
seq=615 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key3 sequence=616
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key3
seq=616 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key4 sequence=617
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key4
seq=617 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key5 sequence=618
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key5
seq=618 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key6 sequence=619
```

```
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key6
seq=619 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key7 sequence=620
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key7
seq=620 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key8 sequence=621
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key8
seq=621 HighestRemainingSequence=612
log_srvMsg:NodeLib:MemCache:60:StoreLocal cachePath=/sensors key=key9 sequence=622
log_srvMsg:NodeLib:MemCache:60:StoreLocal remove cachePath= /sensors key=key9
seq=622 HighestRemainingSequence=612
```

At the beginning the program connects to the blockchain and a snapshot request is made, which allows our client to receive the data he is authorized to receive. This is done relatively quickly. In the main class Client.java the program is put on hold for 2 seconds.

The message "Go" is then displayed.

Then the client will send 10 messages in 10 seconds. The messages will be received by all authorized nodes on the path cache in question, including yours. Thus, you must see in the logs a line for each message sent (lines starting with "StoreLocal"). For each message received in /sensors, the reverse_string script is run. If you answered "Y" at the beginning of the program, and that you correctly identified yourself on git afterwards, you must see in the logs the route of the script, which displays the body upside down (ex: 3olleh).

Finally, the messages will be deleted one by one, since the TTL has been set to 10 seconds. So you must see the transactions in the logs (lines starting with "StoreLocal remove").

If nothing happens after « Go » there is sevral possibilities :

- You are not authorized on the blockchain
- You made a mistake on the config file
- You are not connected to internet