# Smart Contracts

## 1. Prerequisite

To create a Smart Contract it is necessary to have a code editor and/or a text editor :

- Example of a code editor : Visual Studio Code, download link : https://code.visualstudio.com/
- Example of a text editor : notepad++, Gedit (Ubuntu).

## 2. Definition of a Smart Contract

A Smart Contract is a computer program that automates the execution of a set of pre-defined instructions when certain conditions are met.

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

## 3. Use of Smart Contracts in the Kalima blockchain

The Kalima Blockchain uses Smart Contracts developed by standard and open tools like JavaScript and Python, these open tools allow its simple interconnection with other blockchains, other databases and other applications.

In these Smart Contracts, we can make AI inference thanks to models created from the data collected preferentially by Kalima, as it is immutable, safe and complete.

In our case the smart contracts are stored on Git but validated by the Kalima blockchain. The management of these Smart Contracts is integrated in the Kalima API. To be able to "execute" Smart Contracts from our Node, you just have to provide the connection information (login and password) of an authorized account on the git directory where the Smart Contracts are stored.

## 4. Creation of a JavaScript Smart Contract

**Notes :**

- This part will allow you to create your proper Smart Contracts and test them on the Kalima blockchain.
- To use the Smart Contracts that you will create on the Kalima blockchain, you have to share them on Git in a personal directory on which you will be authorized.

A JavaScript Smart Contract is made up mainly of the following parts :

**Required instruction :**

To be sure that your Smart Contract JavaScript works, it is necessary to add the following line at the beginning of your program :

```
load("nashorn:mozilla_compat.js");
```

**Packages importation :**

This part allows you to import the Java packages that you will need to use in your Smart Contract JavaScript.

To import a Java package you can use the "importPackage()" function as shown in the following example :

```
importPackage(Packages.java.util);
importPackage(org.kalima.kalimamq.nodelib);
```

**Java objects creation :**

To proceed to the Java packages importation, you will need to declare the Java objects to be used in your Smart Contract like the following example :

```
var JString = Java.type("java.lang.String");
var KMsg = Java.type("org.kalima.cache.lib.KMsg");
```

**The functions :**

The purpose of using functions is to simplify a program to make it more readable, to avoid having to retype the same lines of code several times in the same program, to avoid errors.

In a Smart Contract it is possible to define one or more functions in order to break it down into small reusable elements. These functions also allow to provide variables to the Smart Contract during its execution.

The code below shows the example of the main function of a Smart Contract :

```
function main(logger, kMsg, clone, node) {

      var body = new JString(kMsg.getBody());
      var value = parseInt(body, 10);
      if(value == null)
            return "NOK";

      if(value >= 100) {
            var kMsg1 = new KMsg(0);
            node.sendToNotaryNodes(kMsg1.getMessage(node.getDevID(),
KMessage.PUB, "/alarms/fire", kMsg.getKey(), kMsg.getBody() , new
KProps("-1")));
      }

      return "OK";
}
```

## 5. Examples of Smart Contracts

**Note :**

- To organize the data in the Kalima blockchain, we use cache paths. This can be seen as a file system with folders and files that will contain the data.

There are some examples of Smart Contracts shared on Github in the following path : https://github.com/Kalima-Systems/Kalima-Tuto/tree/master/KalimaSmartContracts. These Smart Contracts are called from the Java example of the Kalima blockchain client and they allow to execute specific tasks whenever the conditions are met.

**The Smart Contract « sensor.js » :**

The functioning of this Smart Contract is based on the following part :

```
if(value == null)
            return "NOK";

      if(value >= 100) {
            var kMsg1 = new KMsg(0);
            node.sendToNotaryNodes(kMsg1.getMessage(node.getDevID(),
KMessage.PUB, "/alarms/fire", kMsg.getKey(), kMsg.getBody() , new
KProps("-1")));
      }
```

This Smart Contract performs a test on the values of new data added in the cache path " /sensor ". Each time a value exceeds the set threshold (for example a threshold of 100), it writes an alarm in the "/alarms/fire" cache path.

**The Smart Contract « fire.js » :**

The functioning of this Smart Contract is based on the following part :

```
for(i=body.length ; i>=0; i--) {
            reverseString += body.charAt(i);
      }
```

The Smart Contract " fire.js " is executed each time there is a new data to add in the cache path " /alarms/fire ". It allows to invert the value of this new data.

## 6. Securisation of Smart Contracts

It is important to note that the execution of Smart Contracts is secured by the Kalima blockchain : only validated Smart Contracts will be executed by the Kalima blockchain.

## 7. Execution of Smart Contracts

The execution of Smart Contracts on the Kalima blockchain is done from a Java node. To know how to create the Java node and/or how to execute these Smart Contracts it is necessary to consult the documentation " JavaExample.pdf " shared in the following path : https://github.com/Kalima-Systems/Kalima-Tuto/blob/master/etc/doc/en/JavaExample.pdf