Example C

Table of Contents

[1. Library (libs/KalimaLib) 2](#_Toc106784794)

[2. Config file (etc/cfg) 2](#_Toc106784795)

[3. Main project (src) 3](#_Toc106784796)

[a. Makefile 3](#_Toc106784797)

[b. Project Main 3](#_Toc106784798)

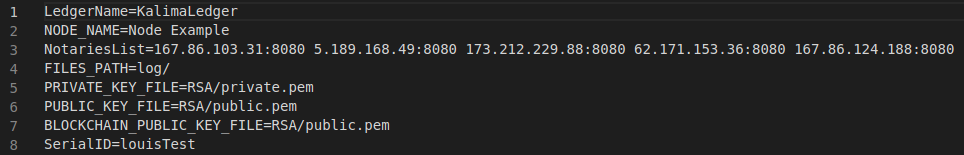
[4. Program Delivery 5](#_Toc106784799)

[5. Possible problems 6](#_Toc106784800)

# Library (libs/KalimaLib)

This library contains all the headers of the library, as well as two static archives lib\_KalimaNodeLib.a and lib\_KalimaNodeLib32.a. These archives will be used when creating the executable in the project (src). The headers will be used to call the methods that will be useful to us in our example.

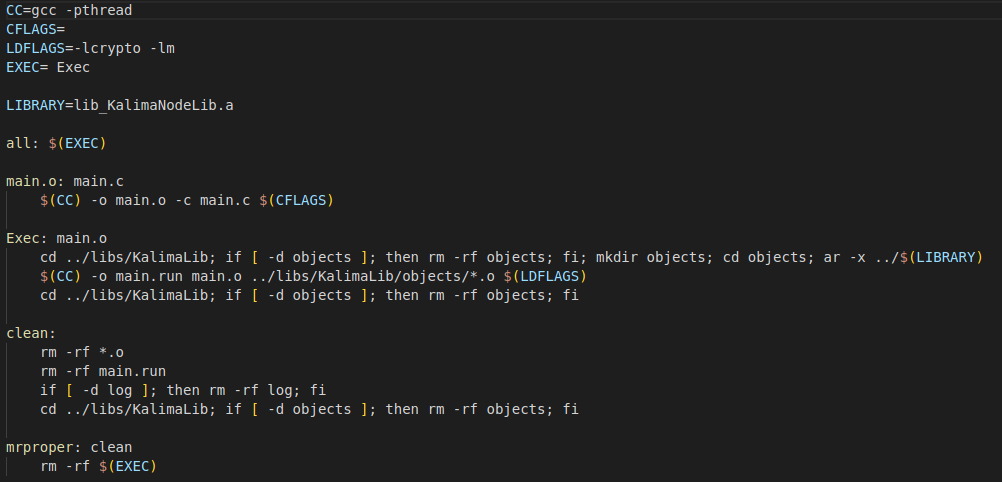
# Config file (etc/cfg)

Let's see the usefulness of the different configurations:

* LedgerName -> This is the name of the Ledger we are going to connect to. For the example it is not very useful.
* NODE\_NAME -> This is the name of the node we create. It is also not very useful for our example.
* NotariesList -> This is the list of nodes to which we need to connect to communicatewith the blockchain tutorial. If you want to connect to another blockchain, just change the notaries here. Between each node, you have to put a space.
* FILES\_PATH -> This is the directory in which you can find the log files. Din our example, the log folder will be created at launch in the directory
* KEY\_FILES -> These are the paths of the different RSA encryption files. These files are necessary to communicate with the Blockchain. They will also be created automaticallyat the start of the project.
* SerialID -> This SerialID will serve as our identification with the blockchain. It must be authorized by the blockchain. This will probably be the only line you will have to modify to put the ID you want.

# Main project (src)

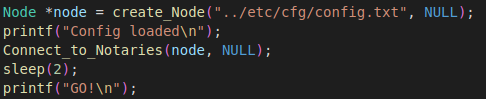
## Makefile



The makefile allows us to create an object for our hand. Then, from this object and the archive of the library, we can create our executable "main.run".

Here, the default library used is "lib\_KalimaNodeLib.a" which was compiled on a 64-bit linux architecture. It is also possible to replace with "lib\_KalimaNodeLib32.a" to create the executable on a 32-bit architecture.

## Project Main



Here we create our node. By default, the node will use the "config.txt" file as the config file. Here it is placed in the etc/cfg folder (.. used to return to the previous folder).

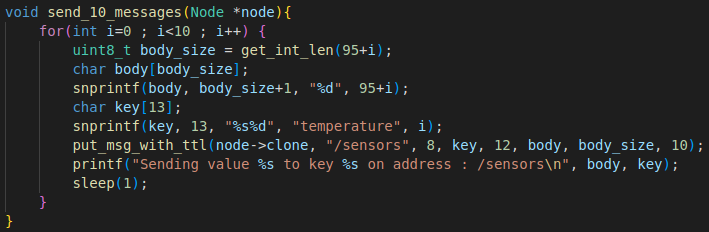
When creating the node, we will also create a random deviceID that will be encrypted and writtenin the "DeviceID" file that will be created in the src folder. If this encrypted file already exists, it will not be recreated. The node will just decrypt the file and use the recovered deviceID. This deviceID, in connection with the SerialID, will allow the block chain to identify our node and allow us to write to it. An RSA folder will also be created containing a public key and a private key used to encrypt communications with the blockchain.

The "NULL" parameters that can be seen here correspond to the fact that we can allow the node to use LuaJIT scipts with a callback. Since in this example we do not use any, we must put NULL.

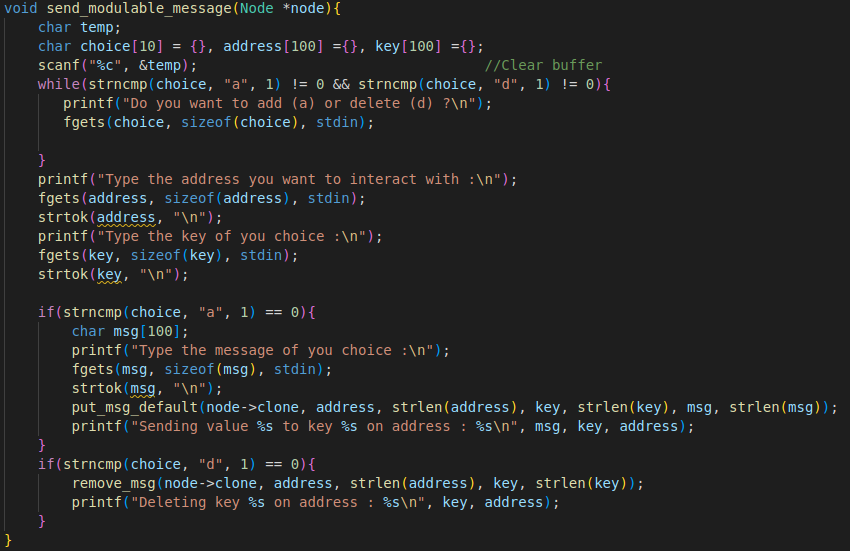
Finally, it will connect directly to the nodes written in the config file.

Next to the creation of the node, the example offers two choices to the user:

* Sending 10 messages by default

This choice sends 10 predefined messages to the blockchain on the address "/sensors". The 10 messages will have a key ranging from temperature0 to temperature9 and a value ranging from 95 to 104. Each message will remain in the blockchain for 10 seconds before being automatically deleted

* Fully configurable message



Here we build ourselves the message of 0.

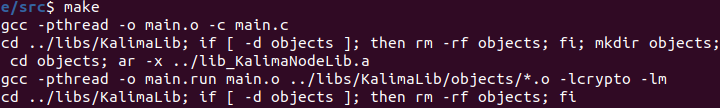
First of all, the user is offered to choose between adding or deleting a message. Then we retrieve the address, the key and the message from the user's terminal. Finally we send the message. Unlike the previous example, the message will remain here indefinitely.

# Program Delivery

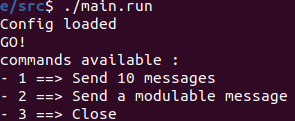
To run the example, simply open a terminal and move to the "src" folder.

Then simply type "make" to launch the makefile seen above. This command will create the executable file "myin.run" that will allow us to launch the program (as well as a main.o file that can be deleted, the DevID folder and the RSA folder).

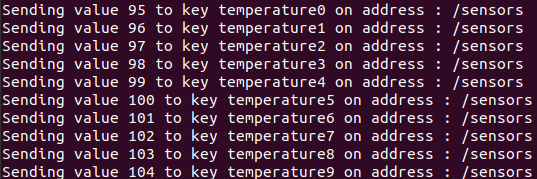
You can also launch with the 32-bit library by typing "make LIBRARY= "lib\_KalimaNodeLib32.a"" if you have not already modified the makefile.



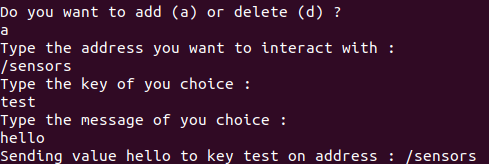
Finally, you must write "./main.run" to launch the program.



Once the program is launched, here's what should appear. Here we see that the config file is loaded for node creation. At this level, the node is created and connected to the blockchain. By typing "1", "2" or "3" you can run one of the options seen above.

Here's what we get with the first option:

And here's what we have with the second option:



Here we decided to send the value "hello" for the key "test" on the address "/sensors"

The third option just serves to close the program.

# Possible problems

If your message does not appear in the blockchain, here are some possible errors:

* Verify that the SerialID in the config file is the same as the one entered in the blockchain.
* If you delete a DeviceID or RSA file, the new calculated file will be different. It will therefore be necessary to redo the SerialID because the associated DeviceID will no longer be the same.
* If you have made changes on the hand that are not taken into account, make a "make clean" before redoing "make" so that all the changes are taken into account.
* If you have other problems, you can look in the logs when the program has problems and contact us.