

Example C

Table of Contents

1.	Library (lib)	2
2.	Config file (etc/cfg)	2
3.	Makefile	3
4.	Main project (src + inc)	3
1.	Main	3
2.	Callback	4
5.	Running the project	6
1.	Authorization	6
2.	Launch	6
6.	Possible issues	7

1. Library (lib)

This library contains all the headers of the library, as well as the static archive lib_Kalima-NodeLib.a. This archive will be used when creating the executable of the project. The headers will be used to call the methods that will be useful to us in our example.

2. Config file (etc/cfg)

```
1  LedgerName=KalimaLedger
2  NODE_NAME=Node Example
3  privachain=org.kalima.tuto
4  FILES_PATH=Files/
5  BLOCKCHAIN_PUBLIC_KEY_FILE=RSA/public.pem
6  SerialID=louisTuto
```

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.
- Privachain -> This is the privachain we will be connected into. If you want to connect to another Kalima Blockchain, you will have to change the privachain.
- FILES_PATH -> This is the directory in which all Kalima related files will be created. When you launch your node for the first time, this directory will be created if it doesn't already exist, and you will find inside the files for RSA, DevID and logs.
- BLOCKCHAIN_PUBLIC_KEY -> This is the path for the blockchain public key if you need for some reasons to have it locally. It isn't used for now.
- 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, please contact one of our administrators ([je-rome.delaire@kalima.io](mailto:jerome.delaire@kalima.io), tristan.souillard@kalima.io) to have your SerialID authorized.

3. Makefile

```
HOST_CC=gcc
CC=$(HOST_CC) -pthread
CFLAGS=-W -Wall
LDFLAGS=
EXEC= Exec
INCLUDE=-I./inc -I./lib/inc/KalimaCUtil -I./lib/inc/MQ2/netlib -I./lib/inc/MQ2/message -I./lib/inc/MQ2/nodelib -I./lib/inc/NodeLib
LIBRARY=libKalimaNodeLib.a

all: $(EXEC)

Exec:
$(CC) $(INCLUDE) -o callback.o -c src/callback.c $(CFLAGS)
$(CC) $(INCLUDE) -o main.o -c src/main.c $(CFLAGS)
$(CC) -o main.run *.o -L. lib/$(LIBRARY) $(LDFLAGS)
rm -rf *.o

clean:
rm -rf *.o
rm -rf main.run
if [ -d Files/log ]; then rm -rf Files/log; fi

mrproper: clean
rm -rf $(EXEC)
```

The makefile will allow us to create the executable to run the project. First it will create the objects from source file main. Finally, it will create the executable main.run from the object created before and the archive "lib_KalimaNodeLib.a" located in the lib directory.

To execute the makefile, you just must type "make" in your terminal (you have to be in the project path in your terminal).

To clear the objects, the executable and the logs, you just need to type "make clean".

4. Main project (src + inc)

1. Main

```
Node *node = create_Node("etc/cfg/config.txt", NULL);
if(node == NULL){
    printf("Error creating Node. Please verify the config file\n");
    return 0;
}
printf("Config loaded\n");
Connect_to_Notaries(node, NULL);
```

This is the start of the main function. It contains the Kalima Node initialization functions.

Here is the explanation on the useful Kalima function:

- `create_Node` : Will create a Kalima Node with information from the config file. The node is the main component which will allow communication with the Blockchain. The NULL parameter is because that's where you put the smart contract list when you are using smart contracts. To see how it works, use the C smart contract example.
- `Connect_to_Notaries` : Will start the connection to the Blockchain with the Node and Callback as a parameter.

When creating the node, we will also create a random deviceID that will be encrypted and written in the "DeviceID" directory that will be created in the location where you start the executable from. If this encrypted file already exists, it will not be recreated. The node will just decrypt the file and use the decrypted deviceID. This deviceID, along with the SerialID in your config file, will allow the Blockchain to identify our node and allow us to send data. An RSA directory will also be created containing a public key and a private key used to encrypt communications with the blockchain.

This example when launched will offer the user two options:

- Sending 10 default messages

```
void send_10_messages(Node *node){
    for(int i=0 ; i<10 ; i++) {
        uint8_t body_size = get_int_len(95+i);
        char body[body_size];
        snprintf(body, body_size+1, "%d", 95+i);
        char key[13];
        snprintf(key, 13, "%s%d", "temperature", i);
        put_msg_with_ttl(node->clone, "/sensors", 8, key, 12, body, body_size, 10);
        printf("Sending value %s to key %s on address : /sensors\n", body, key);
        sleep(1);
    }
}
```

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

```
void send_modulable_message(Node *node){
    char temp;
    char choice[10] = {}, address[100] = {}, key[100] = {};
    scanf("%c", &temp); //Clear buffer
    while(strncmp(choice, "a", 1) != 0 && strncmp(choice, "d", 1) != 0){
        printf("Do you want to add (a) or delete (d) ?\n");
        fgets(choice, sizeof(choice), stdin);
    }
    printf("Type the address you want to interact with :\n");
    fgets(address, sizeof(address), stdin);
    strtok(address, "\n");
    printf("Type the key of you choice :\n");
    fgets(key, sizeof(key), stdin);
    strtok(key, "\n");

    if(strncmp(choice, "a", 1) == 0){
        char msg[100];
        printf("Type the message of you choice :\n");
        fgets(msg, sizeof(msg), stdin);
        strtok(msg, "\n");
        put_msg_default(node->clone, address, strlen(address), key, strlen(key), msg, strlen(msg));
    }
    if(strncmp(choice, "d", 1) == 0){
        remove_msg(node->clone, address, strlen(address), key, strlen(key));
    }
}
```

Here we build the message from scratch.

1. First, 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.

2. Callback

The callback is useful to allow the user to do some actions based on what is happening inside functions in the library without having to modify the said library.

In the callback, we have 5 possible actions located in different parts of the library:

- c_send : Isn't really used for now.
- onNewAddress : Same.
- onAddressSynchronized : Same.
- onReject : Will write a log if the join request is refused by the BlockChain.
- PutData : Is the useful callback function for now. It will act based on data received from the Blockchain. You can change it to do whatever you want with the data received. We'll explain what is being done in this example.

```

void putData(void* client_ptr, KMessage* Kmessage){
    KMsg *kmsg = setMessage(Kmessage);
    char* address = (char*)getAddress(kmsg);
    if(client_ptr == NULL){}

    if(getAddressSize(kmsg)==0)
        return;

    if(check_Type(kmsg->Kmessage, "SNAPSHOTRESP", 0) == 0){
        char* key = (char*)getKey(kmsg);
        char* body = (char*)getBody(kmsg);
        printf("Message received on address : %s / Key : %s / Body : %s\n", address, key, body);
        free(key), free(body);
    }
    free(address);
}

```

This is the putData function, which will allow us to interact with the data received from the Blockchain. Here we will first create the KMsg based on the KMessage received from the Blockchain. Then we get the data address from the KMsg. We only want to interact with the new data, so we must check that the data is not from the data received at the time of connection (SNAPSHOTRESP). We then get the key and body from the KMsg and print the data usefull information.

5. Running the project

1. Authorization

Before launching the project, it is necessary that you do the correct procedure to be authorized on the Blockchain.

For this, it is simple, you need to contact us and send the serialID you put in your config file. Once you are authorized, you have 5 minutes to launch the project to be fully authorized. Once it is done your node should be properly launched and the directoried "RSA" and "DevID" should have been created.

The next time you launch the project, if the directories created stay the same and the serialID is also the same, everything should still work.

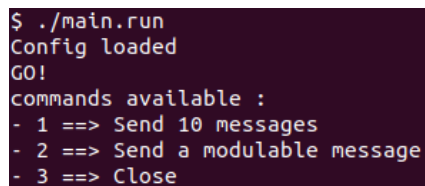
If for some reason your directories got deleted, you changed your system or you changed your serialID, the authorization process must be done again.

2. Launch

To run the example, simply open a terminal and move to the main folder (where your makefile is).

Then simply type "make" to launch the makefile seen above. This command will create the executable file "main.run" (as well as the DevID directory and the RSA directory) that will allow us to launch the program.

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

A terminal window with a dark background and light-colored text. The text shows the execution of the command './main.run'. The output includes 'Config loaded', 'GO!', and a list of available commands: '1 ==> Send 10 messages', '2 ==> Send a moduable message', and '3 ==> Close'.

```
$ ./main.run
Config loaded
GO!
commands available :
- 1 ==> Send 10 messages
- 2 ==> Send a moduable message
- 3 ==> Close
```

When you reach this point, and you are sure you did all the correct steps to be authorized by the Blockchain, it means your Node has been successfully launched.

It also means that you received all the memcache data of the Blockchain and the Lua smart contract has been downloaded to the directory you specified in the main file (the contract directory will be created if it does not already exist).

You are now free to send data to the Blockchain using either of the options seen previously. If you receive a data with the criteria of your smart contracts, they will be loaded and used.

6. Possible issues

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 directory, the new calculated files will be different. It will therefore be necessary to redo the authorization process.
- If you have made changes on the hand that are not considered, make a "make clean" before redoing "make" so that all the changes are considered.
- If you have other problems, you can look at the logs when the program has problems and contact us.