# Light Example

## Library (lib)

This library contains all the headers of the library, as well as the static archive lib_KalimaMQC.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.

## 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 ID allow your node to be authorized on the Kalima Blockchain which you can obtain here: https://inscription.tuto.kalimadb.com/airdrop (you have 10 serialId received by email)

## Makefile

```
CC=gcc -pthread
CFLAGS=-W -Wall
LDFLAGS=
EXEC= Exec
INCLUDE=-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 main.o -c src/main.c $(CFLAGS)
    $(CC) $(INCLUDE) -o main.run main.o -L. lib/$(LIBRARY) $(LDFLAGS)
    rm -rf main.o

clean:
    rm -rf *.o
    rm -rf main.run
    if [ -d log ]; then rm -rf 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_KalimaMQC.a" located in the lib directory.

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

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

## Main project (src + inc)

```
if (argc > 1){
    node = create_Node(argv[1], NULL);
    if(node == NULL){
    printf("Error creating Node. Please verify the config file\n");
    return 0;
}

    Connect_to_Notaries(node, NULL);
    printf("%s Loaded\n", argv[1]);
}
else{
    node = create_Node(pre_config, NULL);
    if(node == NULL){
        printf("Error creating Node. Please verify the config file\n");
        return 0;
    }
    Connect_to_Notaries(node, NULL);
    printf("%s Loaded\n", pre_config);
}
```

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 (by default "etc/cfg/config.txt", you can also use a different one by putting it as a parameter when launching the executable). 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 a default message

```
void send_default_message(Node *node){
    char str[100];
    char temp;
    printf("Message to send to Blockchain :\n");
    scanf("%c", &temp);          // Clear buffer
    fgets(str, sizeof(str), stdin);      // Get input
    List *kProps = new_propList();
    set_prop(kProps, "ttl", 3, "-1", 2); // will give ttl=-1, meaning the message will stay in blockchain (ttl = time to live)
    KMsg *kmsg = getMessage(node->devid, UUID_SIZE, get_encoded_Type("PUB"), "/sensors", 8, "Default message", 15, 1, str, strlen(str), kProps);
    send_KMessage(node, kmsg->Kmessage);
    printf("Message sent\n");
    sleep(2); // Waiting for response (in logs)
}
```

This choice sends the message put by the user to the Blockchain address "/sensors" with the key "Default message". The time to live (ttl) is put as "-1" which means that the message will stay in the Blockchain memcache.

- 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. 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.

# Running the project

## 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 just register here: https://inscription.tuto.kalimadb.com/airdrop. Once it is done you will receive an email explaining every step to be successfully authorized on the blockchain. Basically, you just need to take one of the 10 serialID received by email on the config file and launch the program, it will create RSA keys and a crypted devID in the directory specified as "FILES_PATH" in the config file. Those files are very important and mustn't be deleted.

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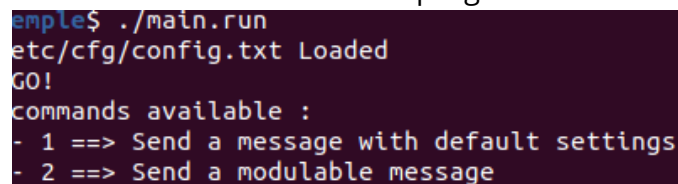
If for some reason your directories got deleted, you changed your system or you changed your serialID, the authorization process must be done again.

## 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.

```
emple$ ./main.run
etc/cfg/config.txt Loaded
GO!
commands available :
- 1 ==> Send a message with default settings
- 2 ==> Send a modulable message
```

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.

You are now free to send data to the Blockchain using either of the options seen previously.

## 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 other problems, you can look at the logs when the program has problems and contact us.