# Exemple C

## Table des matières

# 1. Librairie lib_KalimaMQC

Cette librairie contient tous les headers de la librairie, ainsi qu'une archive statique lib_KalimaMQC.a. Cette archive nous servira lors de la création de l'exécutable dans le projet (MainProject). Les headers nous serviront à appeler les méthodes qui nous seront utiles dans notre exemple.

# 2. Projet principal

## a. Fichier config

```
LedgerName=KalimaLedger
NODE_NAME=Node Example
NotariesList=167.86.124.188:9090 62.171.130.233:9090 62.171.131.157:9090 144.91.108.243:9090
FILES_PATH=log/
SerialID=LouisVM
```

Voyons l'utilité des différentes configurations :

- LedgerName ➔ C'est le nom du Ledger auquel nous allons nous connecter. Pour l'exemple ce n'est pas très utile.
- NODE_NAME ➔ C'est le nom du node que nous créons. Ce n'est pas non plus très utile pour notre exemple
- NotariesList ➔ C'est la liste des nodes auxquels nous devons nous connecter pour communiquer avec la blockchain tuto. Si vous souhaitions nous connecter à une autre blockchain, il suffit de changer les notaries ici. Entre chaque node, il faut mettre un espace.
- FILES_PATH ➔ C'est le directory dans lequel on pourra trouver les fichers de logs ainsi que le fichier qui contiendra le deviceID crypté. Dans notre exemple, le dossier log se créera au lancement dans le directory MainProject.
- SerialID ➔ Ce SerialID nous servira d'identification auprès de la blockchain. Il doit correspondre au nom d'un device de la Blockchain.

## b. Makefile main.mk

```
CC=gcc -pthread
CFLAGS=-W -Wall
EXEC= Exec

all: $(EXEC)

NodesChatC.o: main.c
	$(CC) -o main.o -c main.c $(CFLAGS)

Exec: main.o
	$(CC) -o main.run main.o -L. ../lib_KalimaMQC/lib_KalimaMQC.a $(CFLAGS)

clean:
	rm -rf *.o
	rm -r log/

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

Le makefile nous permet de créer un objet pour notre main. Ensuite, à partir de cet objet et de l'archive de la librairie, nous pouvons créer notre exécutable « main.run ».

### c. Main du projet

```
Node *node;
if(argc>1){
    node = create_Node(argv[1]);
    printf("%s Loaded\n", argv[1]);
}
else{
    node = create_Node(pre_config);
    printf("%s Loaded\n", pre_config);
}
printf("GO!\n");
Menu();
```

Ici, nous créons notre node. Par défaut, le node utilisera le fichier « config.txt » comme fichier de config. Cependant, si nous rentrons un fichier de config comme paramètre d'exécution du programme, il sera utilisé à la place. Le node sera créé avec les paramètres du fichier config.

Lors de la création du node, nous allons également créer un deviceID aléatoire qui sera crypté et écrit dans le fichier « DeviceID » qui sera créé dans le dossier du « FILES_PATH » (par défaut « log/ »). Si ce fichier crypté existe déjà, il ne sera pas recréé. Le node va juste décrypter le fichier et utiliser le deviceID récupéré. Ce deviceID, en lien avec le SerialID, permettra à la blockchain d'identifier notre node et de nous permettre d'écrire dans celle-ci.

Enfin, il se connectera directement aux nodes écrits dans le fichier config.

Après la création du node, l'exemple propose deux choix à l'utilisateur :

- Message par défaut

```
char str[100];
char temp;
printf("Message to send to Blockchain :\n");
scanf("%c", &temp);                                  //Clear buffer
fgets(str, sizeof(str), stdin);                      //Get input
send_Message_to_Notaries(node, str);                 //Send message
printf("Message sent\n");
sleep(2);                                            //Waiting for response (in logs)
```

Ce choix propose d'entrer un message dans le terminal. Ce message sera ensuite envoyé dans la blockchain au cachepath « /sensors » avec la clé « key ». Le « sleep(2) » de fin permet juste d'attendre 2 secondes pour avoir le temps de recevoir la réponse de la blockchain. Nous pourrons voir la réception dans les logs.

- Message entièrement configurable

```
proplist kProps = Initiate_prop_List();
kProps = set_prop(kProps, "ttl", 3, "-1", 2); // will give ttl=-1, meaning the message will stay in blockchain (ttl = time to live)
char type = encode_Type(PUB); //Publish
char cachePath[100], key[100], msg[100];
char temp;
printf("Type the cachePath you want to interact with :\n");
scanf("%c", &temp);                                 //Clear buffer
fgets(cachePath, sizeof(cachePath), stdin);
strtok(cachePath, "\n");
printf("Type the key of you choice :\n");
fgets(key, sizeof(key), stdin);
strtok(key, "\n");
printf("Type the message of you choice :\n");
fgets(msg, sizeof(msg), stdin);
strtok(msg, "\n");
KMsg *kmsg = getMessage(node->devid, UUID_SIZE, type, cachePath, strlen(cachePath), key, strlen(key), 1, msg, strlen(msg), kProps);
send_to_Notary_Nodes(node, kmsg->Kmessage);  //Send message
printf("Message sent\n");
sleep(2);                                   //Waiting for response (in logs)
```

Ici nous construisons nous même le message de 0.

Les kProps ici que nous créons sont des paramètres que l'on transmet à la blockchain pour notre message. Ici par exemple, on paramètre le « time to live »de notre message. Lorsqu'il est mis à « -1 » comme ici, on signale à la blockchain que le message doit rester sans disparaître. Cependant, si on le met à « 5 », il restera 5 secondes avant d'être supprimé automatiquement.

Ensuite on récupère le cachepath, la clé et le message depuis le terminal de l'utilisateur.

Après cela, nous créons un KMessage avec nos paramètres et on envoie ce KMessage au notary nodes.

Le « sleep » est encore là pour attendre la réponse.

## 3. Exécution du programme.

Pour exécuter l'exemple, il suffit d'ouvrir un terminal et de se placer dans le dossier « MainProject ».

Il suffit ensuite simplement de taper « make -f main.mk » pour compiler le makefile vu précédemment. Cette commande créera le fichier exécutable « main.run » qui nous permettra de lancer le programme (ainsi qu'un fichier main.o qui peut être supprimé).

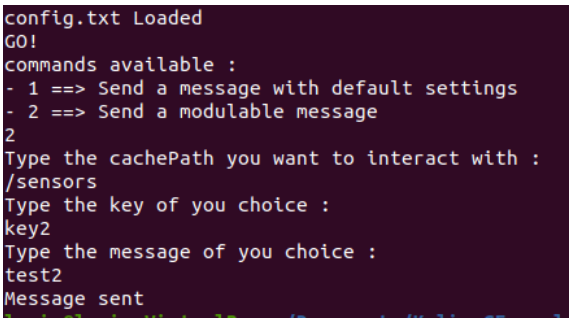
Enfin, il faut écrire « ./main.run » pour lancer le programme.

```
config.txt Loaded
GO!
commands available :
- 1 ==> Send a message with default settings
- 2 ==> Send a modulable message
```

Une fois le programme lancé, voici ce qui devrait apparaître. Nous voyons ici que le fichier « config.txt » est chargé pour la création du node. À ce niveau-là, le node est créé et connecté à la blockchain. En tapant « 1 » ou « 2 » on peut ainsi exécuter l'une des deux options vues précédemment.

```
config.txt Loaded
GO!
commands available :
- 1 ==> Send a message with default settings
- 2 ==> Send a modulable message
1
Message to send to Blockchain :
test
Message sent
```

Ici, nous avons donc envoyé le message « test » sur le cachepath « /sensors » avec la clé « key ».




Ici, nous avons donc envoyé le message « test2 » sur le cachepath « /sensors » avec la clé « key2 ».

## 4. Problèmes possibles

Si votre message n'apparaît pas dans la blockchain, voici quelques possibles erreurs :

- Vérifiez que le SerialID dans le fichier config et le même que celui entré dans la blockchain.
- Si vous supprimez le fichier DeviceID, le nouveau calculé sera différent. Il faudra donc refaire le SerialID car le DeviceID associé ne sera plus le même.