C# example :

1. Prerequisite

For this example, you must install an IDE for C#: Microsoft Visual Studio. Download link: https://visualstudio.microsoft.com

2. The creation of a Visual Studio project

In Visual Studio, create a new project and choose “Application console (.NET Core). Click on “Next”, then enter a name and choose a location for your project. Then, click on “Create”.

Afterwards, in « Solution Explorer », right click on your project → Edit the project file and replace the line:

```
<TargetFramework>netcoreapp3.1</TargetFramework>
```

By :

```
<TargetFramework>net48</TargetFramework>
```

3. Include the DLLs

On the right side, in the Solution Explorer, right click on your project → Add → New folder → Name it « libs ».

Copy-paste the DLLs present in the KalimaCSharpExample/libs in yours. Most of the DLLs come from ikvm:

https://github.com/jessielesbian/ikvm

IKVM is an implementation of Java for Microsoft .NET. If you need more Java features, please install ikvm and include the necessary DLLs in your project.

Then, in Visual Studio, right click on your project → Add → Project Reference → Browse → Select all the DLLs that you’ve addded in your libs folder.

4. Code explanation

a. Initialisations

The code below allows you to initialize a certain number of objects and to connect to the Blockchain.

```
public static void Main(string[] args)
{
  try
  {
    Client client = new Client(args);
  }
  catch (Exception e)
  {
    e.printStackTrace();
  }
}

public Client(string[] args)
{
  clonePreferences = new ClonePreferences(args[0]);
  logger = clonePreferences.getLoadConfig().getLogger();
  initComponents();
}

public void initComponents()
{
  node = new Node(clonePreferences.getLoadConfig());
  clone = new Clone(clonePreferences, node);
  kalimaClientCallback = new KalimaClientCallBack(this);
  node.connect(null, kalimaClientCallback);
}
```

The Node will be responsible for the connexion with the Blockchain

The clone is responsible for the synchronisation of the data in cache memory.

The clientCallBack allows you to react to the addition of new transactions in the Blockchain (see CallBacks chapter).

We can see that we have to pass args[0] during the creation of the clonePreferences. Indeed, you must launch your client by passing the path of a configuration file. We will talk later about the content of this file.

b. Callbacks

As we have seen previously, we have to pass two callback classes to the Node object. The serverCallback is not useful for an ordinary node. So you can just pass null value.

The ClientCallBack is more important and requires a few necessary lines. It will allow you to react to the arrival of new transactions. In the code here, we have a class that inherits from ClientCallback. The functions that we are not interested in are left empty. The functions that interest us for this example are putData, onConnectionChanged and onJoined.

The putData function will be called at each new transaction received, you must at least add the code below, and then customize the code according to your needs.

```
KMsg kMsg = KMsg.setMessage(msg);
client.clone.set(kMsg.getCachePath(), kMsg, true, false);
```

The onConnectionChanged function will be called at every connection / disconnection with one of the Notary Nodes. You must at least insert the code below, and you can add more if needed.

```
client.getClone().onConnectedChange( (status==Node.CLIENT_STATUS_CONNECTED) ?
new AtomicBoolean(true) : new AtomicBoolean(false), nioClient, false);
```

The onJoined function is called every time a crypted communication is isteablished with a notary node. You must add a least this code in the onJoined function:

```
client.getClone().onJoined(sc, false);
```

5. Configuration file

Add a cfg folder in your project and place a « node.config » file in it. That configuration file will be passed as a parameter of the application.

See below for an example of a configuration file :

```
NODE_NAME=Node Client Example
NotariesList=62.171.131.154:9090,62.171.130.233:9090,62.171.131.157:9090,144.91
.108.243:9090
FILES_PATH=/home/rcs/jit/ClientExample
SerialId=PC1245Tuto
#Watchdog=60000
PRODUCTION=false
```

- NODE_NAME ➔ You can put something which allows you to recognize your node
- NotariesList ➔ The address and port list of the notary, separated by commas
- FILES_PATH ➔ It is the path where the files useful for Kalima will be stored, as well as the logs
- serialId ➔ It is an ID which will allow the authorization on the blockchain of the first launch of the client node (provided by Kalima Systems in the case of trials on our Notary)
- WATCHDOG ➔ The watchdog here is commented out, it is not useful for our example. The watchdog is useful in case of prolonged connection with the Blockchain. It corresponds to a time (here, 600000 = 600 seconds = 10 minutes). In fact, every 10 minutes, we will check that the connection is still active. You can then for example make a Smart Contract that will send you an email if the connection is lost. It is thus useful to check the state of the network.
- PRODUCTION ➔ For this tutorial, we connect our node on a test blockchain. On this blockchain, the authentication of nodes are simplified, so we must set this parameter to false.

6. Code Execution

To test your project, you can execute the code from Visual Studio, or from an online command console. You need to pass the configuration file as parameter.

Execution from Visual Studio

In Debug ➔ Properties of debug of … ➔ In the Bebug tab ➔ Pass the complete path or relative of the configuration file in “Arguments of the application” (example: ../../../cfg/node.config). Lastly, click on F5 to launch the application.

Command-line execution

First, in Visual Studio, generate the solution by clicking on F6. If the generation works, the executable and the necessary DLLs have been copied in the bin folder (in bin/Debug/net48/ for example).

Then, from the console, go to the executable folder and launch it by passing the configuration file as parameter:

```
cd
Documents\Kalima\git\KalimaTuto\KalimaCSharpExample\KalimaCSharpExample\
bin\Debug\net48
KalimaCSharpExample.exe ..\..\..\etc\cfg\node.config
```

7. Results

The example program connects to the Blockchain, then sends 10 messages (1/second). The TTL (Time To Live) of those messages is of 10 which means that each message will be automatically deleted after 10 seconds (a transaction will happen on the blockchain for each deletion). Thus, if your code is correct, if you have configured the configuration file correctly, and if your device is authorized on the blockchain, you should have something similar to this in your console:

```
GO

putData cachePath=/sensors key=key0 body=hello0

putData cachePath=/sensors key=key1 body=hello1

putData cachePath=/sensors key=key2 body=hello2

putData cachePath=/sensors key=key3 body=hello3

putData cachePath=/sensors key=key4 body=hello4

putData cachePath=/sensors key=key5 body=hello5

putData cachePath=/sensors key=key6 body=hello6

putData cachePath=/sensors key=key7 body=hello7

putData cachePath=/sensors key=key8 body=hello8

putData cachePath=/sensors key=key9 body=hello9

putData cachePath=/sensors key=key0 body=

putData cachePath=/sensors key=key1 body=

putData cachePath=/sensors key=key2 body=

putData cachePath=/sensors key=key3 body=

putData cachePath=/sensors key=key4 body=

putData cachePath=/sensors key=key5 body=

putData cachePath=/sensors key=key6 body=

putData cachePath=/sensors key=key7 body=

putData cachePath=/sensors key=key8 body=
putData cachePath=/sensors key=key9 body=
```

Results explanation :

- In the first part the client will send 10 messages in 10 seconds. The messages will be received by all the nodes authorized on the cache path, including yours. Thus, you will have, in the logs, a line for each message sent (lines starting with « `StoreLocal` »).
- Afterwards, the messages will be deleted one by one, since the TLL has been configured on 10 seconds. You will be able to see the transactions in the logs with an empty body.