Exemple C#

1. Prérequis

Pour tester l'exemple C#, il est nécessaire d'installer un environnement de développement intégré IDE : Microsoft Visual Studio. Lien de téléchargement : https://visualstudio.microsoft.com/fr/

2. Création d'un projet Visual Studio

Dans Visual Studio, créez un nouveau projet et choisissez « Application console (.NET Core). Cliquez sur « suivant », puis entrez un nom et choisissez un emplacement pour votre projet, puis cliquez sur « créer ».

Ensuite, dans « Explorateur de solutions », clic droit sur votre projet ➔Modifier le fichier projet, et remplacez la ligne :

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

Par :

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

3. Inclure les DLL

Dans l'explorateur de solutions à droite, faites clic droit sur votre projet ➔ Ajouter ➔ Nouveau dossier ➔ appeler le dossier « libs ».

Copiez-collez les DLL présentes dans le dossier KalimaCSharpExample/libs dans le vôtre. La plupart de ses DLL proviennent de ikvm :

https://github.com/jessielesbian/ikvm

IKVM est une implémentation de java pour Microsoft .NET, si vous avez besoin de plus de fonctionnalités java, veuillez installer ikvm et inclure les DLL nécessaires dans votre projet.

Ensuite, dans visual studio, faites clic droit sur votre projet ➔ Ajouter ➔ Reference de projet ➔ Parcourir ➔ Sélectionnez toutes les DLL que vous avez ajouter dans votre dossier libs.

4. KalimaClientCallBack

Créez une nouvelle classe sous le nom « KalimaClientCallBack » :

- ⇨ Clic droit sur votre projet
- ⇨ Ajouter
- ⇨ Nouvel élément
- ⇨ Sélectionnez « classe » et entrez « KalimaClientCallBack.cs » dans le champ « nom »
- ⇨ Ajouter

Ci-après le code complet de la classe (le namespace peut changer suivant le nom de votre projet) :

A ce stade, le projet comporte des erreurs qui seront corrigées plus tard.

```
using System;
using org.kalima.kalimamq.message;
using org.kalima.kalimamq.nodelib;
using org.kalima.kalimamq.netlib;
using org.kalima.cache.lib;
using java.util.concurrent.atomic;
using java.nio.channels;

namespace KalimaCSharpExample
{
  public class KalimaClientCallBack : ClientCallback
  {

    private Client client;
    private Logger logger;
    private Node node;

    public KalimaClientCallBack (Client client)
      {
        this.client = client;
        this.node = client.getNode ();
        this.logger = node.getLogger ();
      }

    public void putData(SocketChannel ch, KMessage msg) {
      KMsg kMsg = KMsg.setMessage (msg);
      client.getClone ().set (kMsg.getCachePath(), kMsg, true, false);
Console.WriteLine("putData cachePath=" + kMsg.getCachePath() + " key=" +
kMsg.getKey() + " body=" + System.Text.Encoding.Default.GetString(kMsg.getBody()));
    }

    public void onConnectionChanged(int status, NioClient nioClient){
      logger.log_srvMsg ("ExampleClientNode", "KalimaClientCallBack",
Logger.DEBUG, "onConnectionChanged status=" + status);
      client.getClone().onConnectedChange(status==Node.CLIENT_STATUS_CON
NECTED) ? new AtomicBoolean(true) : new AtomicBoolean(false), nioClient,
false);
    }

    public void onCacheDeleted (string cacheSubPath){
      logger.log_srvMsg ("ExampleClientNode", "KalimaClientCallBack",
Logger.DEBUG, "onCacheDeleted cacheSubPath=" + cacheSubPath);
    }

    public void putRequestData(SocketChannel ch, KMessage msg) {}

    public void onNewVersion(int majver, int minver) {}

    public void onNewCache(String cachePath) {}
      }
}
```

5. Client

Enfin, renommez votre classe principale (par défaut Program.cs) en Client.cs ➔ Clic droit sur Program.cs ➔ Renommer.

Ci-après le code complet de la classe :

```
using System;
using org.kalima.kalimamq.nodelib;
using org.kalima.kalimamq.message;
using org.kalima.kalimamq.crypto;
using org.kalima.cache.lib;
using ikvm.extensions;

namespace KalimaCSharpExample
{
  public class Client : KalimaNode
  {

    private Node node;
    private Clone clone;
    private Logger logger;
    private KalimaClientCallBack kalimaClientCallback;
    private ClonePreferences clonePreferences;

    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();
       System.Threading.Thread.Sleep(2000);
       Console.WriteLine("GO");

       for (int i = 0; i < 10; i++)
       {
         String body = "hello" + i;
         KMsg kMsg = new KMsg(0);
         node.sendToNotaryNodes(kMsg.getMessage(node.getDevID(),
KMessage.PUB, "/sensors", "key" + i, body.getBytes(), new
KProps("10")));
         System.Threading.Thread.Sleep(1000);
       }
     }
```

```
    public void initComponents()
    {
      node = new Node(clonePreferences.getLoadConfig());
      clone = new Clone(clonePreferences, node);

      kalimaClientCallback = new KalimaClientCallBack(this);

      node.connect(null, kalimaClientCallback);
     }

     public Node getNode()
     {
      return node;
     }

     public Logger getLogger()
     {
      return logger;
     }

     public Clone getClone()
     {
      return clone;
     }

   }
```

6. Fichier de configuration

Ajoutez un dossier cfg dans votre projet, et placer dedans un fichier « node.config ». Ce fichier de configuration sera passé en paramètre de l'application.

Voici un exemple de fichier de configuration :

```
LedgerName=KalimaLedger
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
```

- LedgerName ➔N'est pas encore utilisé dans la version actuelle
- NODE_NAME ➔ Vous pouvez mettre quelque chose qui permet de reconnaître votre nœud
- NotariesList ➔ La liste des adresses et ports des notary, séparés par des virgules
- FILES_PATH ➔ C'est le chemin ou seront stockés les fichiers utiles à Kalima, ainsi que les logs
- serialId ➔ C'est un identifiant qui va permettre l'autorisation sur la blockchain au premier lancement du node client (fournis par Kalima Systems dans le cas d'un essais sur nos Notary)

7. Exécution du code

Pour tester votre projet, vous pouvez exécuter le code depuis Visual studio, ou depuis une console en ligne de commande. Il suffit de passer en paramètre, le chemin du fichier de configuration.

Exécution depuis Visual Studio :

Dans Déboguer ➔ Propriétés de débogage de … ➔ Dans l'onglet Déboguer ➔ Passer le chemin complet ou relatif du fichier de config dans « Arguments de l'application » (exemple : ../../../cfg/node.config). Enfin, appuyez sur F5 pour lancer l'application.

Exécution en ligne de commande

Tout d'abord dans Visual studio, générez la solution en appuyant sur F6. Si la génération a réussi, l'exécutable et les dll nécessaires ont été copiés dans le dossier bin (dans bin/Debug/net48/ par exemple).

Ensuite depuis la console, dirigez vous dans le dossier de l'exécutable et lancer le en passant le fichier de configuration en paramètre :

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

8. Résultats

Le programme d'exemple se connecte à la Blockchain, puis envoi 10 messages (1/seconde). Le TTL (Time To Live) de ces messages est de 10, ce qui signifie que chaque message sera automatiquement supprimé au bout de 10 secondes (une transaction aura lieu sur la blockchain pour chaque suppression). Ainsi, si votre code est correct, que vous avez correctement configuré le fichier de configuration, et que votre appareil est bien autorisé sur la blockchain, vous devriez avoir quelque chose de similaire dans votre 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=

Explication des résultats :

- Dans la première partie des logs, on peut voir que le programme se connecte à la blockchain, en se connectant en fait à chaque Notary.
- Ensuite une demande de snapshot est faite, ce qui permet à notre client de recevoir les données qu'il est autorisé à recevoir.
- Ensuite, le client va envoyer 10 messages en 10 secondes. Les messages seront reçus par tous les nodes autorisés sur la cache path en question, dont le vôtre. Ainsi, vous devez voir dans les logs une ligne pour chaque message envoyé (lignes commençant par « `StoreLocal` »).
- Enfin, les messages seront supprimés un à un, puisque le TTL a été configuré sur 10 secondes. Vous devez donc voir les transactions dans les logs (lignes commençant par « `StoreLocal remove` »).