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 ajouté dans votre dossier libs.

4. Explications du code

a. Initialisations

Le code ci-dessous permet d'initialiser un certain nombre d'objet et de se connecter à la 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);
}
```

Le Node va être responsable de la connexion avec la Blockchain.

Le clone est responsable de la synchronisation des données en mémoire cache.

Le clientCallBack permet de réagir à l'ajout de nouvelles transactions dans la Blockchain (cf chapitre suivant).

On peut voir que l'on doit passer args[0] lors de la création de clonePreferences. En effet, vous devez lancer votre client en passant le chemin d'un fichier de configuration. On verra plus tard le contenu de ce fichier.

b. Callbacks

Comme on a pu le voir précédemment, on doit passer deux classes de callbacks à l'objet Node. Le serverCallback n'est pas utile pour un nœud ordinaire. On peut donc simplement passer la valeur null.

Le ClientCallBack a plus d'importance en revanche, et nécessite quelques lignes obligatoires. Il vous permettra notamment de réagir à l'arrivée de nouvelles transactions. Dans le code ici, nous avons donc une classe qui hérite de ClientCallback. Les méthodes qui ne nous intéressent pas sont laissées vides. Les fonctions qui nous intéressent pour cet exemple sont putData, onConnectionChanged.

La fonction putData sera appelée à chaque nouvelle transaction reçue. Il doit donc y avoir au minimum le code ci-dessous :

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

Vous pouvez ensuite y ajouter votre code métier pour réagir a de nouvelles transactions.

La fonction onConnectionChanged sera appelée à chaque connexion / déconnexion avec l'un des Notary Nodes. Il doit donc y avoir au minimum le code ci-dessous :

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

La fonction onJoined est appelée a chaque fois que l'établissement d'une connexion cryptée avec un notary node est établie. Il faut au minimum y ajouter ce code :

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

5. 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 :

```
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 ➔ 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) Pour chaque appareil, il faut changer le seialId. Il est donc très important de le vérifier lorsque vous lancez sur un nouvel appareil.
- WATCHDOG ➔ Le watchdog ici est commenté, il ne nous est pas utile pour notre exemple. Le watchdog est utile en cas de connexion prolongée avec la Blockchain. Il correspond à un temps (ici, 600000 = 600 secondes = 10 minutes). Si par exemple le watchdog est configuré sur 10 minutes, alors une transaction sera créée dans /Kalima_alarms si votre nœud perd la connexion avec la blockchain pendant au moins 10 minutes. Cela permet donc d'être averti en cas de problème sur un nœud.
- PRODUCTION ➔ Dans le cadre de ce tuto, nous nous connectons sur une blockchain de test. La méthode de connexion est simplifiée, on doit donc ajouter ce paramètre pour que cela fonctionne

6. 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
```

7. Résultats

Le programme d'exemple se connecte à la Blockchain, et vous propose d'effectuer des essaies de communication avec la blockchain. L'interface vous propose de choisir le cache path que vous voulez utiliser par défaut vous pouvez utiliser le cache path /sensors, ensuite il vous propose soit d'ajouter ou de supprimer une donnée et enfin vous pouvez rentrer les valeurs de la donnée. Puis vous pouvez continuer d'interagir avec la blockchain jusqu'à ce que vous n'en ayez plus besoin et dans ce cas vous pouvez simplement appuyer sur 'E' lorsque la proposition de quitter ou de continuer vous sera envoyé dans la console. 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 :

Choose a cache path (Console)

/sensors (Vous)

Press 'A' to add something to the cache path and 'D' to delete (Console)

A (Vous)

Enter the key of the data you want to add (Console)

testExample (Vous)

Enter the value of your data (Console)

valueExample (Vous)

putData cachePath=/sensors key=testExample body=valueExample (Console)

Press ENTER to continue or 'E' to exit (Console)

E (Vous)

End of the console program… (Console)

Explication des résultats :

- Les données saisis ou supprimés seront actualisées dans le cache path saisis au préalable, à tout moment vous pouvez décider de changer de cache path ou d'arrêter le programme.
- Vous pouvez vérifier ce qui a été envoyé ou supprimer avec la ligne ci-dessus (putData) ou vous avez la valeur du cache path, de la clé et du body de votre donnée.