Robot Stäubli TX2-40 / contrôleur Cs9 : Communication par Ethernet à l'aide de sockets TCP

Ce document porte sur le fonctionnement d'une connexion par Ethernet *via* l'utilisation de sockets TCP entre un serveur implémenté (en Python) dans un PC et un client (un seul suffisant à illustrer le fonctionnement de la connexion) implémenté dans le contrôleur Cs9 d'un robot TX2-40.

Dans un premier temps, le client sera émulé dans le logiciel SRS (Stäubli Robotics Suite) 2022 (installé[1] dans le même PC que celui où est implémenté le serveur), puis le client sera transféré dans le contrôleur (réel) Cs9. Aussi, ce document se scinde en deux parties (A et B) :

A) Lorsque le client et le serveur sont locaux (c'est-à-dire, sont implémentés sur le même PC), en considérant trois étapes :

- la construction du client émulé dans le logiciel SRS,
- l'utilisation d'un serveur Python (version 3.8, le code du programme sera fourni) implémenté dans le PC où est émulé le client,
- le test de la communication entre le client (émulé dans SRS) et le serveur Python,

B) Lorsque le client et le serveur sont distants, en considérant trois étapes :

- les modifications à effectuer sur le client et le serveur lorsqu'ils étaient locaux,
- le transfert du client dans le contrôleur (réel) Cs9, un câble Ethernet reliant le port Ethernet J204 du contrôleur Cs9 à celui du PC (où est implémenté le serveur),
- le test de la communication entre le client (implémenté dans le contrôleur Cs9) et le serveur.

Bref rappel sur la communication par Ethernet :

Une fois la connexion entre le client et le serveur établie, des données peuvent être échangées entre eux. Plus précisément, le serveur, une fois activé, se met en écoute, au sens où il attend que le client lui envoie des requêtes. Lorsque le client envoie une requête au serveur, le serveur y répond (ou éventuellement pas). Il en résulte que le client a besoin de connaître l'adresse IP et le numéro de port du serveur afin qu'il puisse s'adresser au serveur (rappelons qu'un seul serveur peut être présent au sein du réseau contrairement au client)[2]. La connexion reste ouverte tant que le client ou le serveur n'y mette pas fin.

Déroulement des échanges entre le client et le serveur dans l'exemple considéré dans le document :

Après établissement de la connexion entre le client et le serveur, il est prévu que les échanges entre eux se déroulent comme suit :

Côté client	Côté serveur
Le client envoie au serveur le message 'DemandeDonnees@' (attention les accents ne sont pas autorisés dans les messages). Le caractère '@' sert à indiquer la fin du message.	
	Une fois reçu le message, le serveur envoie deux messages au client, à savoir '12.34@' et '56.789@'.

[1] La version 4.1 de SRS 2022 est téléchargeable <ici> (en exécutant setup.exe).

[2] Notons que le serveur n'a pas besoin de connaître *a priori* l'adresse IP et le numéro de port du client qui l'a contacté car ces informations sont contenues dans le message envoyé par le client.

Suite à la réception de chacun de ces messages, le message est affiché sur le *Teach Pendant.* Une fois ces deux messages convertis en valeurs numériques, leur somme est affichée sur le *Teach Pendant*.	
Le client envoie au serveur le message 'FIN@'.	
	Une fois le message reçu, le serveur se déconnecte (ce qui a pour effet de désactiver la communication entre le client et le serveur).

Description des étapes à suivre :

A) Cas où le client et le serveur sont locaux

A.1) Opérations à effectuer concernant le client

A.1.1) Création d'une cellule

A.1.2) Création d'une application au sein du contrôleur de la cellule

A.1.3) Construction d'une socket Ethernet

A.1.4) Liaison d'une variable VAL3 à la socket Ethernet

A.1.5) Codage du programme **start()** de l'application client

A.2) Opérations à effectuer concernant le serveur

A.3) Dialogue entre le client et le serveur

A.3.1) Mise en route des applications du client et du serveur

A.3.2) Résultats attendus durant l'exécution des applications du client et du serveur

A.3.3) Arrêt de l'exécution des applications du client et du serveur

B) Cas où le client et le serveur sont distants

B.1) Changements à opérer sur les applications du client et du serveur réalisées précédemment

B.2) Procédure permettant l'exécution des applications du client et du serveur

---

A) Cas où le client et le serveur sont locaux

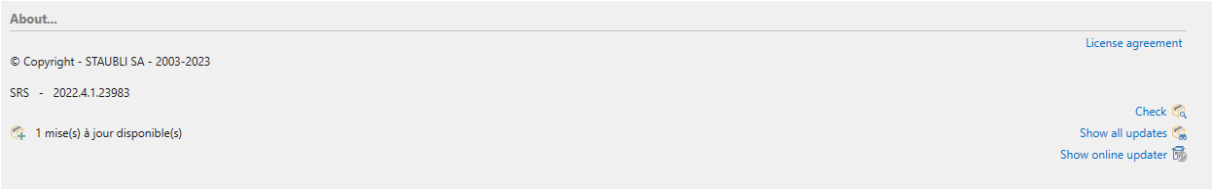
A.1) Opérations à effectuer concernant le client

Il s'agit d'émuler un client TCP dans SRS 2022 à même de se comporter comme décrit dans le tableau ci-dessus. Afin de permettre, par la suite (dans la partie B), une communication par Ethernet lorsque le client et le serveur sont distants, les caractéristiques du robot Stäubli que nous allons considérer correspondent à celles du robot acquis récemment par Polytech Angers, à savoir, un bras TX2-40 et un contrôleur Cs9.

A.1.1) Création d'une cellule

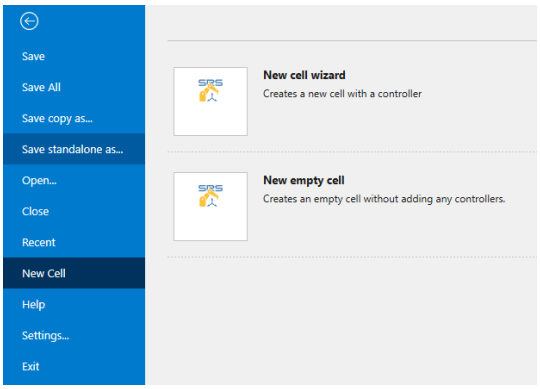
La cellule, intitulée '**Client_TCP_Example'**, que nous allons créer est constituée d'un bras de robot TX2-40 et d'un contrôleur Cs9 dont la version est 's8.15.3Cs9_BS3337' (à l'image de celle du contrôleur réel).

- Vérifiez si la version 's8.15.3Cs9_BS3337' du contrôleur Cs9 est bien installée dans SRS. Pour cela cliquez sur '**Help**' puis '**Show online updater**' (situé à droite dans la fenêtre décrite dans la figure qui suit).



  Dans la fenêtre (intitulée 'UpDates') qui apparaît, sélectionnez l'onglet '**Installed versions**' afin de visualiser les versions installées du contrôleur Cs9. Si la version 's8.15.3Cs9_BS3337' n'apparaît pas, téléchargez là en sélectionnant l'onglet '**Not installed**' puis le bouton '**Download**' relatif à la version à installer (bien sûr une fois trouvé cette version dans la liste proposée). Fermez alors la fenêtre 'UpDates' à l'aide du bouton ✕ situé en haut à droite de la fenêtre.

- Sélectionnez, dans la fenêtre décrite ci-dessous, '**New Cell>New cell wizard**' (pour accéder à cette fenêtre (si ce n'est pas le cas), sélectionnez '**File**' dans la fenêtre principale de SRS).

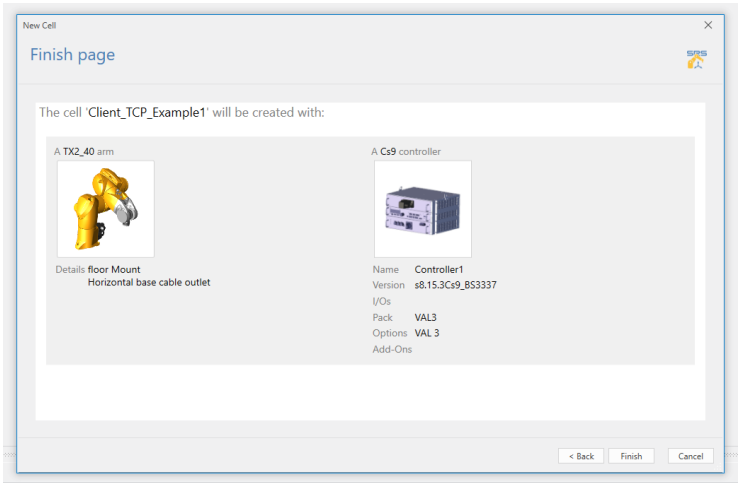


- Dans la fenêtre (intitulée 'Customize your cell') qui apparaît, tapez comme nom de cellule '***Client_TCP_Example***' dans le champ '**Name**'. Le fait de valider le contenu de la fenêtre (en cliquant sur '**Next**') fait apparaître une autre fenêtre.

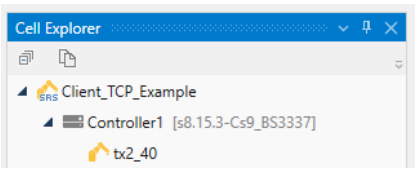
- Dans cette fenêtre (intitulée 'Select controller type'), sélectionnez : '**Add a local controller**'. Une autre fenêtre apparaît alors.

- Dans cette fenêtre (intitulée 'Select arm'), sélectionnez le bras '**TX2_40 / floor Mount / Horizontal base cable outlet**'. Le fait de valider le contenu de la fenêtre fait apparaître une autre fenêtre.

- Dans cette fenêtre (intitulée 'Controller options for arm TX2_40'), sélectionnez la version '**s8.15.3Cs9_BS3337**' du contrôleur, intitulé (par défaut) '**Controller1**'. Valider le contenu de la fenêtre en cliquant sur '**Next**' ce qui provoque l'ouverture d'une fenêtre, intitulée 'Finish page', laquelle permet de vérifier la conformité des caractéristiques du robot, voir la figure qui suit. Cliquez sur '**Finish**' afin de validez ces informations.



Il en résulte l'ouverture de la fenêtre principale de SRS dans l'onglet '**Home**' avec une fenêtre '**Cell Explorer**' située à droite, laquelle permet d'indiquer que la cellule '**Client_TCP_Example**' contient un contrôleur '**Controller1**' (avec une version 's8.15.3-Cs9_BS3337') et un bras TX2_40, voir la figure qui suit.

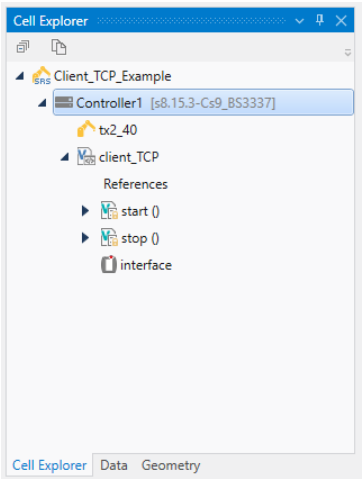


A.1.2) Création d'une application au sein du contrôleur de la cellule

- Cliquez droit sur le contrôleur '**Controller1**', situé dans la fenêtre 'Cell Explorer', afin de sélectionner '**New Application**'.

- Dans la fenêtre qui apparaît (intitulée 'New Application'), tapez '***Client_TCP***', en tant que nom de l'application, dans le champ '**Name**', validez le contenu de la fenêtre en cliquant sur '**OK**'.

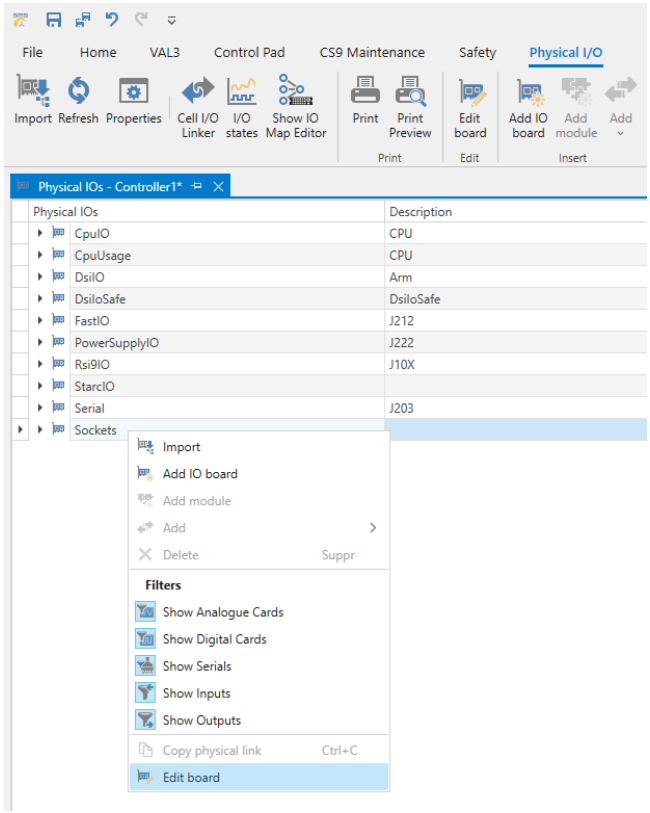
- Le fait de déplier l'application '**Client_TCP**' fait apparaître le programme '**start()**' lequel va contenir le code correspondant au client TCP, voir la figure qui suit.



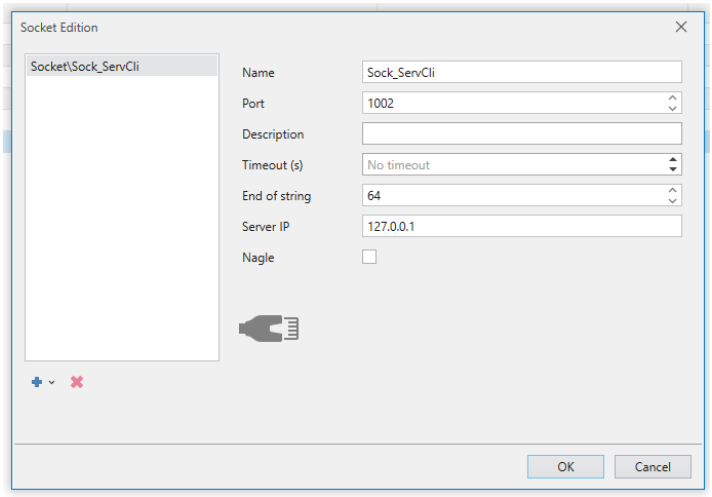
A.1.3) Construction d'une socket Ethernet

Avant de s'intéresser au codage proprement dit du client, il s'agit de créer une socket Ethernet, intitulée '**Sock_ServCli**', permettant la communication entre le client et le serveur (dans le chapitre A.1.4, cette socket Ethernet sera reliée à une variable utilisable dans le programme '**start()**').

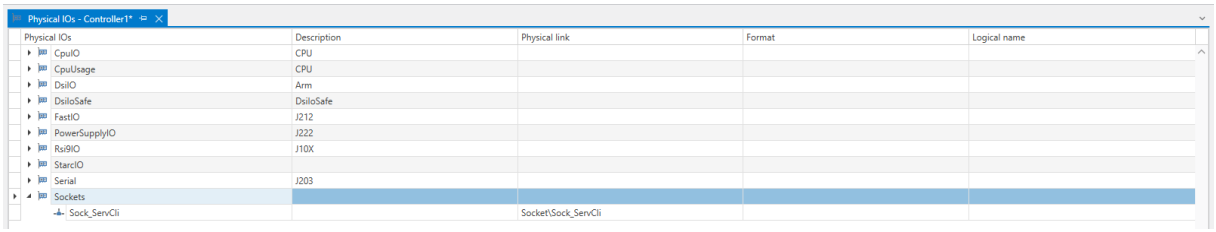
- Une fois sélectionné le contrôleur '**Controller1**', situé dans la fenêtre 'Cell Explorer', sélectionnez '**Home>Physical IOs**' ce qui a pour effet d'afficher le panneau de configuration du contrôleur '**Controller1**' (voir la fenêtre de l'onglet '**Physical IOs – Controller1**' décrite dans la figure qui suit).



- Cliquez droit sur la ligne '**Sockets**' pour sélectionner '**Edit board**'. La fenêtre qui apparaît (intitulée 'Socket Edition') va permettre de créer une socket Ethernet pour un client TCP (notamment en renseignant le numéro de port et l'adresse IP du serveur, notez que ces informations ne seraient pas demandées si la socket était destinée à un serveur TCP). Pour cela, cliquez sur le bouton '**+**' (situé en bas à gauche de la fenêtre), puis sur '**Tcp client**'. Remplissez les champs '**Name**', '**Port**', '**End of string**', '**Server IP**', comme indiqué dans la figure qui suit. Ainsi la socket Ethernet s'intitule '**Sock_ServCli**', le numéro de port et l'adresse IP du serveur sont '**1002**' et '**127.0.0.1**' (ce qui correspond au *localhost* et permet à l'ordinateur de communiquer avec lui-même (et non pas avec un autre) sans avoir besoin d'un réseau physique) et le caractère '@' (dont le code ASCII en valeur décimale est 64) est le caractère de fin de message (un message étant de type ***string***).



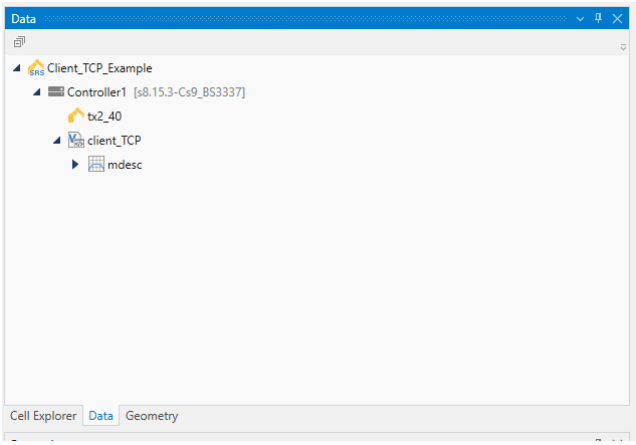
- Faites apparaître la socket Ethernet (‘**Sock_ServCli**’) en dépliant la ligne ‘**Sockets**’, voir la figure qui suit.



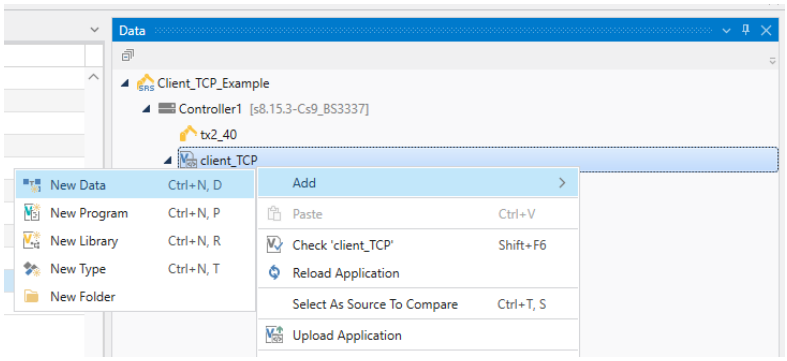
A.1.4) Liaison d’une variable VAL3 à la socket Ethernet

Avant de s’intéresser (dans le chapitre A.1.5) au codage du client dans le programme ‘**start()**’, on effectue la liaison entre une variable, intitulée ‘**siClient**’, et la socket ‘**Sock_ServCli**’, ce qui va permettre d’utiliser cette variable dans le programme ‘**start()**’.

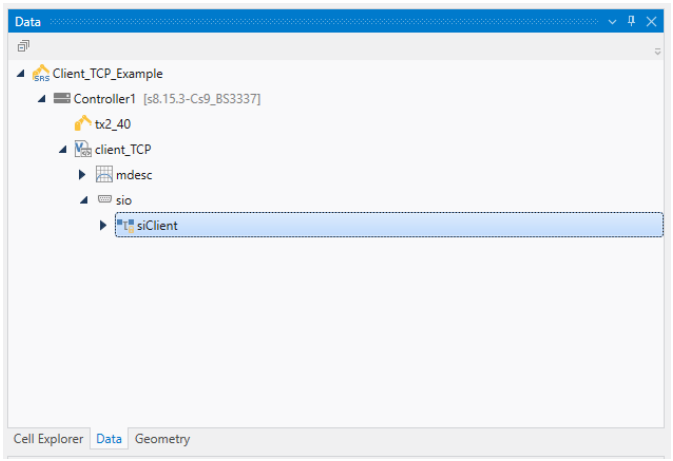
- Dans un premier temps, la variable ‘**siClient**’ va être déclarée comme étant de type *sio*, ce type étant utilisé pour lier une variable VAL3 à une connexion Ethernet *via* une socket (dans notre cas, la socket ‘**Sock_ServCli**’). Pour cela, sélectionnez, dans la fenêtre ‘Cell Explorer’ l’onglet ‘**Data**’ (vous noterez que la fenêtre s’intitule à présent ‘Data’, et non plus ‘Cell Explorer’) ; puis dépliez la cellule ‘**Client_TCP_Example**’ jusqu’à obtenir ce qui est décrit dans la figure qui suit.



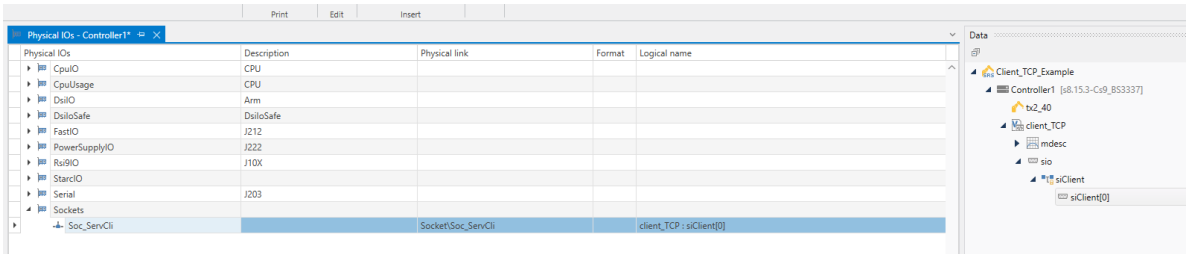
- Le fait de cliquer droit sur l’application ‘**Client_TCP**’ permet de sélectionner ‘**Add>New Data**’ (voir la figure qui suit), ce qui va permettre d’initialiser la variable ‘**siClient**’.



- Dans la fenêtre (intitulée 'Add New Data') qui apparaît, sélectionnez le type ***sio***, puis tapez '***siClient***' (plutôt que 'sSio') dans le champ '**Name**'. A l'issue de la validation (bouton '**OK**'), '**siClient**' apparaît en tant que variable de type ***sio***, comme décrit dans la figure qui suit.



- La liaison entre la socket '**Sock_ServCli**' et la variable '**siClient**' s'effectue en faisant un ***Glisser et Déposer*** (*Drag and Drop*) de la variable '**siClient**' dans la ligne listant la socket '**Sock_ServCli**', ce qui donne lieu à ce qui est décrit dans la figure qui suit.



A.1.5) Codage du programme start() de l'application client

Le programme '**start()**' est appelé lorsqu'une application VAL3, dans le cas présent '**Client_TCP**', est lancée, alors que le programme '**stop()**' (dans le cas présent sans instruction) est appelé lors de l'arrêt de l'application.

> **N.B. :** Dans le langage de programmation VAL3, il est à noter que les premières lettres d'une variable sont importantes au sens où (en plus de faciliter la reconnaissance par le programmeur du type d'une variable), elles permettent à l'environnement de développement de SRS de proposer *a priori* le type adapté à une variable lors de sa création.
> La convention relative aux premières lettres d'une variable est la suivante :
> - **bVariable** est une variable de type ***bool***,

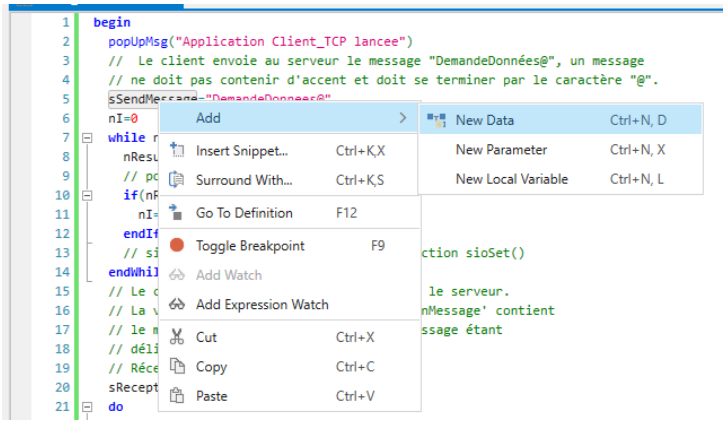
- **nVariable** est une variable de type ***num***,
- **sVariable** est une variable de type ***string***,
- **diVariable** est une variable de type ***dio*** relatif aux entrées et sorties digitales du système,
- **aiVariable** est une variable de type ***aio*** relatif aux entrées et sorties analogiques du système,
- **siVariable** est une variable de type ***sio***, à l'image de la variable **siClient** créée précédemment dans A.1.4 ; ce type, associé à une variable, indique que la variable est interfacée à un port série ou à une connexion Ethernet *via* une socket.

- Dans la fenêtre '**Cell Explorer**' (située à droite et accessible en sélectionnant l'onglet '**Cell Explorer**' plutôt que '**Data**' ou '**Geometry**'), double-cliquez sur '**start()**' afin d'ouvrir une fenêtre dont l'onglet s'intitule '**Client_TCP-start**' et dans laquelle il va être possible d'écrire (entre les lignes '**begin**' et '**end**') le code relatif au programme '**start()**'. Le code à introduire est décrit ci-dessous.

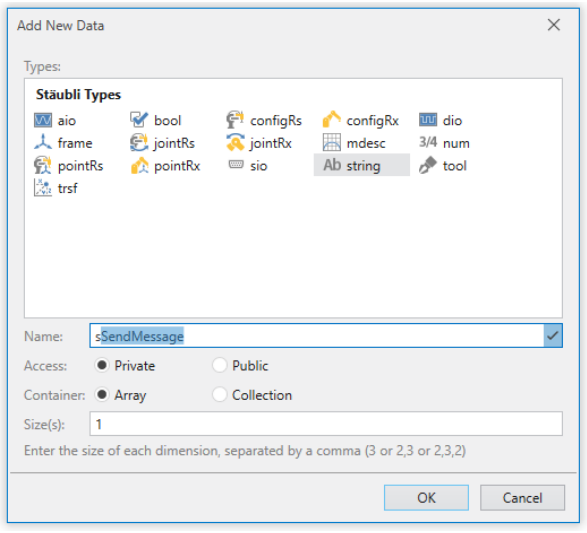
```
begin
    popUpMsg("Application Client_TCP lancee")

    // Le client envoie au serveur le message "DemandeDonnées@", un message
    // ne doit pas contenir d'accent et doit se terminer par le caractère "@".
    sSendMessage="DemandeDonnees@"
    nI=0
    while nI<len(sSendMessage)
        nResultat=sioSet(siClient,asc(sSendMessage,nI))
        // popUpMsg(toString("3",nResultat))
        if(nResultat>0)
            nI=nI+nResultat
        endIf
        // si nResultat=-1 on réexécute l'instruction sioSet()
    endWhile

    // Le client attend 2 messages envoyés par le serveur.
    // La variable (de type string) 'sReceptionMessage' contient
    // le message envoyé par le serveur, ce message étant
    // délimité par le caractère "@".
    // Réception du premier message
    sReceptionMessage=""
    do
        nResultat=sioGet(siClient,nReceptionOctet)
        sReceptionMessage=sReceptionMessage+chr(nReceptionOctet)
    until nReceptionOctet==asc("@",0)
    popUpMsg("serveur -> client premier message : "+sReceptionMessage)
    toNum(sReceptionMessage,nValeur1,bOK)
    // Réception du deuxième message
    sReceptionMessage=""
    do
        nResultat=sioGet(siClient,nReceptionOctet)
        sReceptionMessage=sReceptionMessage+chr(nReceptionOctet)
    until nReceptionOctet==asc("@",0)
    popUpMsg("serveur -> client 2eme message : "+sReceptionMessage)
    toNum(sReceptionMessage,nValeur2,bOK)

    // Affichage de nValeur1 + nValeur2
    popUpMsg("nValeur1 + nValeur2 = "+toString(".3",nValeur1+nValeur2))

    // Le client envoie au serveur le message "FIN@" afin que le serveur
```

```
    // se déconnecte.
    sSendMessage="FIN@"
    nI=0
    while nI<len(sSendMessage)
        nResultat=sioSet(siClient,asc(sSendMessage,nI))
        // popUpMsg(toString("3",nResultat))
        if(nResultat>0)
            nI=nI+nResultat
        endIf
        // si nResultat=-1 on réexécute l'instruction sioSet()
    endWhile

    wait(false)
end
```

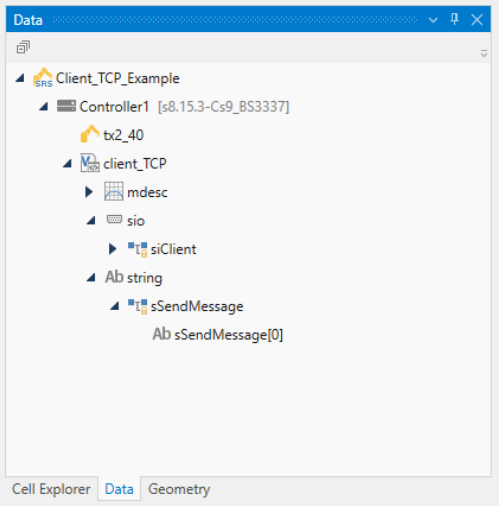
- Une fois le code rentré dans le programme (en effectuant un *copier-coller* du code listé ci-dessus), il faut veiller, pour chaque variable utilisée la première fois dans le code, à déclarer son type (ce travail étant facilité si la convention relative aux premières lettres d'une variable est respectée). Par exemple, concernant la variable '**sSendMessage**' listée dans la ligne (6) du programme, cliquez droit sur cette variable afin de sélectionner '**Add>New Data**', comme décrit dans la figure qui suit.



Une fenêtre (intitulée 'Add New Data') apparaît, validez là (*via* son bouton '**OK**') après avoir vérifier que les données correspondent à celles décrites dans la figure qui suit.

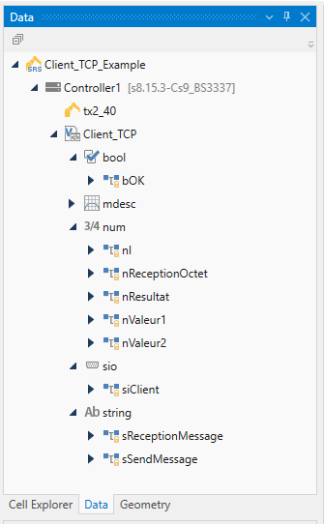


Vérifiez que cette initialisation a bien été effectuée en constatant la présence de la variable en tant que donnée (de type ***string***) du programme '**start()**' : pour cela, dans la fenêtre '**Data**' (située à droite et accessible en sélectionnant l'onglet '**Data**' plutôt que '**Cell Explorer**' ou '**Geometry**'), vous devriez voir apparaître -une fois l'arborescence dépliée- la variable '**sSendMessage[0]**' (cette variable étant équivalente à la variable '**sSendMessage**'), voir la figure qui suit. Vous noterez, également et sans surprise, la présence de la variable (créée dans le chapitre A.1.4) '**siClient**' de type ***sio***.



- Pour récapituler, les variables à initialiser sont les suivantes :
  - '**bOK**' à la ligne (27) de type ***bool***,
  - '**nI**' à la ligne (7), '**nResultat**' à la ligne (9), '**nReceptionOctet**' à la ligne (23), '**nValeur1**' à la ligne (27), '**nValeur2**' à la ligne (35) de type ***num***,
  - '**sSendMessage**' à la ligne (6), '**sReceptionMessage**' à la ligne (21) de type ***string***,
  - '**siClient**' à la ligne (9) de type ***sio***,

  ce qui donne lieu au listing dans la fenêtre 'Data' des variables décrit dans la figure qui suit.



Expliquons à présent chacune des lignes du code, à savoir :

```
(1)         begin
(2)         popUpMsg("Application Client_TCP lancee")
(3)
(4)         // Le client envoie au serveur le message "DemandeDonnées@", un message
(5)         // ne doit pas contenir d'accent et doit se terminer par le caractère "@".
(6)         sSendMessage="DemandeDonnees@"
```

```
(7)         nI=0
(8)         while nI<len(sSendMessage)
(9)             nResultat=sioSet(siClient,asc(sSendMessage,nI))
(10)            if(nResultat>0)
(11)              nI=nI+nResultat
(12)            endIf
(13)            // si nResultat=-1 on réexécute l'instruction sioSet()
(14)        endWhile
(15)
(16)        // Le client attend 2 messages envoyés par le serveur.
(17)        // La variable (de type string) 'sReceptionMessage' contient
(18)        // le message envoyé par le serveur, ce message étant
(19)        // délimité par le caractère "@".
(20)        // Réception du premier message
(21)        sReceptionMessage=""
(22)        do
(23)           nResultat=sioGet(siClient,nReceptionOctet)
(24)           sReceptionMessage=sReceptionMessage+chr(nReceptionOctet)
(25)        until nReceptionOctet==asc("@",0)
(26)        popUpMsg("serveur -> client premier message : "+sReceptionMessage)
(27)        toNum(sReceptionMessage,nValeur1,bOK)
(28)        // Réception du deuxième message
(29)        sReceptionMessage=""
(30)        do
(31)           nResultat=sioGet(siClient,nReceptionOctet)
(32)           sReceptionMessage=sReceptionMessage+chr(nReceptionOctet)
(33)        until nReceptionOctet==asc("@",0)
(34)        popUpMsg("serveur -> client 2eme message : "+sReceptionMessage)
(35)        toNum(sReceptionMessage,nValeur2,bOK)
(36)
(37)        // Affichage de nValeur1 + nValeur2
(38)        popUpMsg("nValeur1 + nValeur2 = "+toString(".3",nValeur1+nValeur2))
(39)
(40)        // Le client envoie au serveur le message "FIN@" afin que le serveur
(41)        // se déconnecte.
(42)        sSendMessage="FIN@"
(43)        nI=0
(44)        while nI<len(sSendMessage)
(46)            nResultat=sioSet(siClient,asc(sSendMessage,nI))
(47)            if(nResultat>0)
(48)              nI=nI+nResultat
(49)            endIf
(50)            // si nResultat=-1 on réexécute l'instruction sioSet()
(51)        endWhile
(52)
(53)        wait(false)
(54)    end
```

- Les lignes (1) et (54) servent à délimiter le code contenu dans le programme '**start()**' ;

- La ligne (2) affiche *via* la fonction '**popUpMsg**' le texte '**Application Client_TCP lancee**' dans l'historique/le journal/ des événements (accompagné de la date et de l'heure) ; cet historique est accessible *via* la touche '**i**' situé en haut à gauche du *Teach Pendant* ;

- Les lignes (4) et (5) permettent de commenter les lignes de code (6) à (14) ;

- Les lignes (6) à (14) permettent l'envoi, à destination du serveur, du message contenu dans la variable *string* 'sSendMessage', à savoir «DemandeDonnees@» (le caractère «@» sera utilisé par le serveur, codé en Python, pour repérer la fin de la chaîne envoyée par le client). Pour cela :

    - L'instruction 'asc(sSendMessage,nI)' renvoie la valeur décimale du code ASCII du caractère d'index 'nI' contenu dans la variable *string* 'sSendMessage' ;

    - L'instruction 'sioSet(siClient,asc(sSendMessage,nI))' écrit dans la variable *sio* 'siClient' le caractère (de type *num*) issu de l'instruction 'asc(sSendMessage,nI)'. L'instruction 'nResultat=sioSet(siClient,asc(sSendMessage,nI))' renvoie dans la variable *num* 'nResultat' le nombre de caractères écrits (dans le cas présent égal à 1 s'il n'y a pas de problème de communication, sinon la valeur retournée est égale -1) ;

- L'instruction 'nResultat=sioSet(siClient,asc(sSendMessage,nI))' est insérée dans une boucle 'while – endWhile' laquelle est parcourue tant que le compteur 'nI' est inférieur strictement à 'len(sSendMessage)' ce qui permet de traiter chacun des caractères contenus dans la variable *string* 'sSendMessage' (sachant que l'instruction 'len(sSendMessage)' renvoie le nombre de caractères contenus dans la variable *string* 'sSendMessage') ;

- Les lignes (16) à (20) permettent de commenter les lignes de code (21) à (35) ;

- Les lignes (21) à (27) permettent la réception, en provenance du serveur, de son premier message, à savoir «12.34@» (le caractère «@» signalant la fin du message (envoyé par le serveur)). Pour cela :

  - L'instruction 'nResultat=sioGet(siClient,nReceptionOctet)' réceptionne dans la variable *num* 'nReceptionOctet' la valeur de l'octet en cours codant le message reçu *via* la socket 'siClient' ;

  - L'instruction 'sReceptionMessage=sReceptionMessage+chr(nReceptionOctet)' rajoute dans la variable *string* 'sReceptionMessage' (initialement vide) le caractère correspondant au code ASCII de l'octet en cours réceptionné (sachant que l'instruction 'chr(nReceptionOctet)' renvoie le caractère, de type *string*, correspond au code ASCII de l'octet contenu dans la variable 'nReceptionOctet') ;

  - Les 2 précédentes instructions sont insérées dans une boucle 'do – until' laquelle est parcourue tant que l'octet en cours ne correspond pas à la valeur ASCII du caractère «@» (à savoir 64 en décimale) ;

  - La ligne (26) affiche *via* la fonction '**popUpMsg**' le texte 'serveur -> client premier message :' suivi du message contenu dans la variable 'sReceptionMessage' dans l'historique/le journal/ des événements (accompagné de la date et de l'heure) ;

  - La ligne (27) met *via* la fonction '**toNum**' la valeur numérique contenue dans la variable *string* 'sReceptionMessage' dans la variable *num* 'nValeur1' (notez que l'on aurait pu tester la variable *bool* 'bOK' pour savoir si le message envoyé par le serveur contenait effectivement une valeur numérique, on suppose ici que c'est toujours le cas) ;

- Les lignes (28) à (35) sont analogues aux lignes (21) à (27) et concernent le deuxième message envoyé par le serveur, à savoir «56.789@ », et dont la valeur numérique est stockée dans la variable *num* 'nValeur2' ;

- La ligne (38) affiche *via* la fonction '**popUpMsg**' le texte 'nValeur1 + nValeur2 = ' suivi de la somme des variables 'nValeur1' et 'nValeur2' dans l'historique/le journal/ des événements (accompagné de la date et de l'heure) ;

- Les lignes (42) à (51) sont analogues aux lignes (6) à (14) et permettent l'envoi, à destination du serveur, du message contenu dans la variable *string* 'sSendMessage', à savoir «FIN@».

- La ligne (53) fait en sorte que l'application '**Client_TCP**' se mette en attente (indéfiniment).

A.2) Opérations à effectuer concernant le serveur

Le serveur communique avec le client selon un déroulement décrit dans les pages 1 et 2 du document. Le programme Python correspondant au serveur, intitulé '**serveur_Staubli.py**', est listé ci-dessous avec des commentaires intégrés dans le code afin d'expliquer ce qui est réalisé. Le serveur est supposé communiquer avec un seul client, ce qui est suffisant pour ce que l'on fait dans le cas présent. Ce code est largement inspiré de celui décrit dans la page qui suit : https://python.developpez.com/cours/apprendre-python3/?page=page_20#L20-A

Le code peut être exécuté avec une version 3.8 de Python, il nécessite l'utilisation des modules ***socket*** et ***sys*** (*cf*. instruction 'import socket, sys').

Vous noterez que le serveur a pour adresse IP 127.0.0.1 et pour port 1002, comme cela a été mentionné lors de la création de la socket 'Sock_ServCli' dans le chapitre A.1.3.

```
# Définition du serveur lequel permet de transmettre 2 données au client (préalablement
connecté)

import socket, sys

HOST = '127.0.0.1'
PORT = 1002

# 1) Création de la socket (c-à-d, une interface logicielle permettant d'exploiter les
services d'un
#    protocole réseau) entre le serveur et le client (il n'y a qu'un client) :
mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2) Liaison de la socket à une adresse (c-à-d, à une IP et à un port) précise :
try:
    mySocket.bind((HOST, PORT))
except socket.error:
    print("La liaison du socket à l'adresse choisie a échoué.")
    sys.exit()

# 3) Le serveur attend la requête de connexion du client :
print("Serveur prêt, en attente de requêtes ...")
mySocket.listen(2)

# 4) Etablissement de la connexion :
connexion, adresse = mySocket.accept()
print("Client connecté, son adresse est : IP %s, port %s" % (adresse[0], adresse[1]))

# 5) Dialogue avec le client :
# Le serveur attend le message "DemandeDonnees@" (envoyé par le client) :
msg=""
while True:
    msgClient = connexion.recv(1024).decode("Utf8")
    if msgClient.upper() == "@":
        break
    msg=msg+msgClient
print("client > serveur : ", msg)

# Envoi de 2 messages au client :
msgServeur = "12.34@"
connexion.send(msgServeur.encode("Utf8"))
msgServeur = "56.789@"
connexion.send(msgServeur.encode("Utf8"))

# Attend le message "FIN@" mettant fin à la communication TCP
msg=""
while True:
    msgClient = connexion.recv(1024).decode("Utf8")
    if msgClient.upper() == "@":
        break
    msg=msg+msgClient
print("client > serveur : ", msg)
if msg.upper() == "FIN":
    print("Le serveur a reçu le message 'FIN' de la part du client")
else:
    print("Pb !! Le serveur n'a pas reçu le message 'FIN' de la part du client")
print("Le serveur se déconnecte")
connexion.close
```

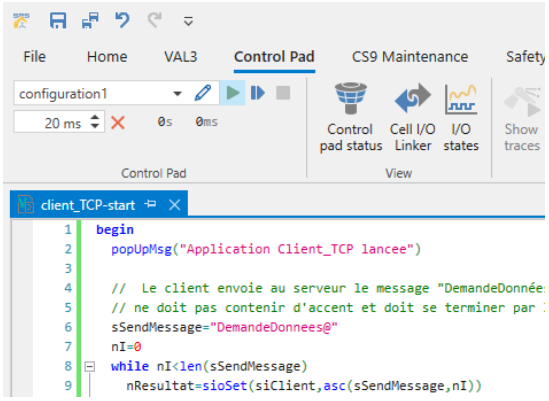
A.3) Dialogue entre le client et le serveur

A.3.1) Mise en route des applications du client et du serveur

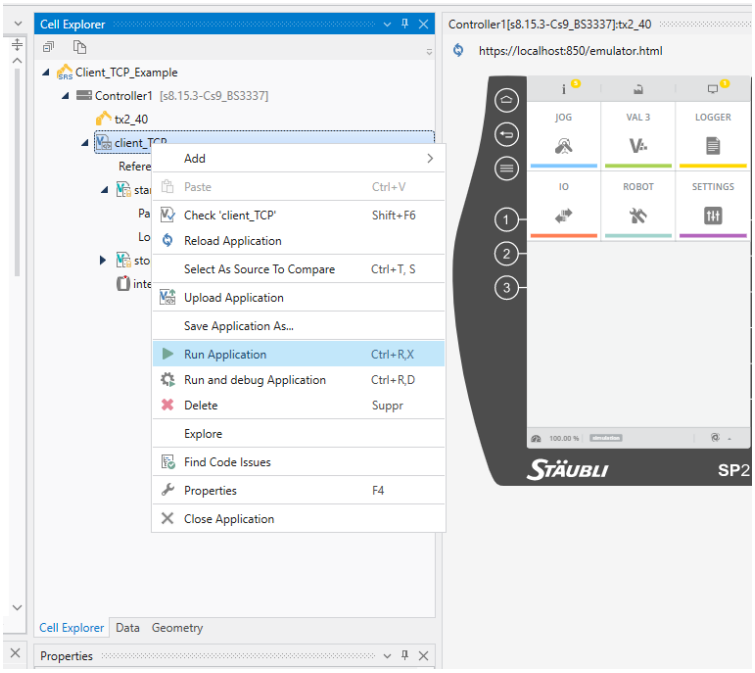
Après avoir téléchargé <<ici>> le fichier '**serveur_Staubli.zip**' lequel contient le programme '**serveur_Staubli.py**', exécuter le programme afin que le serveur se mette en écoute en affichant dans sa fenêtre d'exécution le message « **Serveur prêt, en attentes de requêtes …** ».

L'application '**Client_TCP**' peut alors être exécutée, pour cela :

- cliquez sur le bouton ▶ situé dans l'onglet '**ControlPad**', voir la figure qui suit ;



- cliquez droit sur l'application '**Client_TCP**' (située dans la fenêtre '**Cell Explorer**') afin de cliquer sur '**Run Application**', voir la figure qui suit.



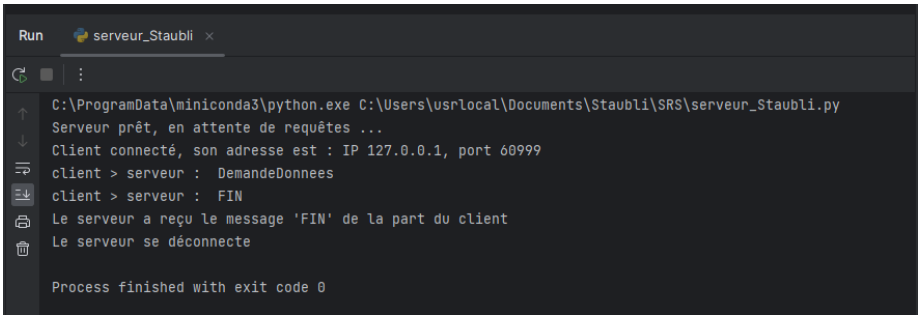
A.3.2) Résultats attendus durant l'exécution des applications du client et du serveur

- **Du côté du client**, les résultats que vous devriez obtenir sont décrits dans la figure qui suit laquelle est une visualisation de l'historique/du journal/ des événements ; cet historique est accessible *via* la touche '**i**' située en haut à gauche du *Teach Pendant*.

Plusieurs informations sont listées dont les messages :

- « Application Client_TCP lancee »,
- « serveur -> client premier message : 12.34@ » sachant que la chaîne « 12.34@ » provient du serveur,
- « serveur -> client 2eme message : 56.789@ » sachant que la chaîne « 56.789@ » provient du serveur,
- « nValeur1 + nValeur2 = 69.129 », 69.129 étant égale à 12.34 + 56.789.

- **Du côté du serveur**, les résultats que vous devriez obtenir sont décrits dans la figure qui suit laquelle correspond à la fenêtre d'exécution du programme 'serveur_Staubli.py'.



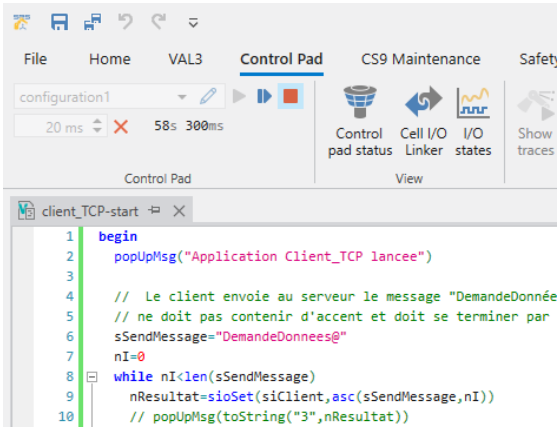
On a notamment comme information le fait que :

- l'adresse IP du client est 127.0.0.1 et son port est 50415,
- le message « client > serveur : DemandeDonnees » sachant que « DemandeDonnees » provient du client,
- le message « client > serveur : FIN » sachant que « FIN » provient du client,

- il s'ensuit une déconnexion du serveur.

A.3.3) Arrêt de l'exécution des applications du client et du serveur

L'arrêt de l'application du client se fait en cliquant sur le bouton ■ situé dans l'onglet '**ControlPad**', voir la figure qui suit.

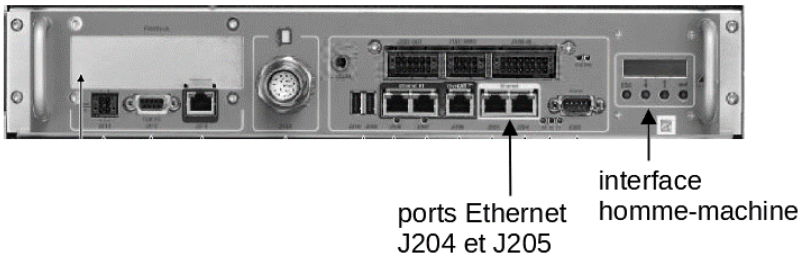


Du côté du serveur, il n'y a rien à faire car le serveur est à l'arrêt suite à sa déconnexion.

B) Cas où le client et le serveur sont distants

B.1) Changements à opérer sur les applications du client et du serveur réalisées précédemment

A présent, un câble Ethernet (croisé ou non) relie le port Ethernet du PC au port Ethernet **J204** du contrôleur Cs9. Notez que l'adresse IP du port Ethernet **J204** est affichée sur l'interface homme-machine du contrôleur Cs9, voir la figure qui suit.



La principale modification à effectuer au niveau des applications '**Client_TCP**' et '**serveur_Staubli.py**' concerne leurs adresses IP (précédemment égale à 127.0.0.1) afin que le contrôleur Cs9 (dans lequel est implémenté le client) et le PC (dans lequel est implémenté le serveur) soient dans le même réseau.

Pour cela :

- **Du côté PC**, on relève l'adresse IP de sa carte Ethernet en exécutant l'instruction « **ipconfig** » dans l'*Invite de commandes de Windows* (tapez « **Invite de commandes** » dans le champ situé en bas à gauche de la fenêtre Windows). A titre d'exemple, on considère par la suite que cette IP est égale à '192.168.2.1' (*cf.* « Adresse IPv4 » de la carte Ethernet), le masque de sous-réseau[3] étant égal à '255.255.255.0'.

[3] Le *masque de sous-réseau* permet de distinguer la partie de l'adresse commune à tous les appareils du sous-réseau et celle qui varie d'un appareil (aussi appelé hôte) à l'autre (Wikipédia).

- **Du côté contrôleur Cs9**, on configure en conséquence l'adresse IP du port Ethernet J204 du contrôleur Cs9 en appuyant sur les touches « **SETTINGS>NETWORK** » du *Teach Pendant*. Sélectionnez alors le port J204 ce qui fait apparaître une fenêtre permettant la modification de l'adresse IP. Relativement à l'exemple considéré, on choisit de fixer l'IP du contrôleur Cs9 à l'adresse '192.168.2.2'.

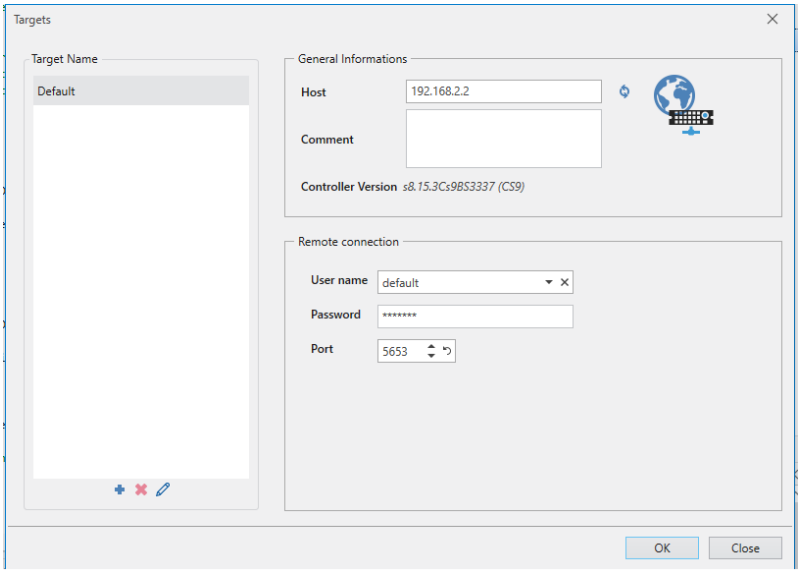
N.B. : On peut tester que l'adresse IP du contrôleur Cs9 est bien reconnue par le PC en exécutant l'instruction '**ping 192.168.2.2**' dans l'*Invite de commandes de Windows*.

Il reste :

- **au niveau de l'application du client à :**

  - modifier la socket Ethernet '**Sock_ServCli'**, voir le chapitre A.1.3 relatif à la construction de la socket, en mettant '192.168.2.1' (au lieu de '127.0.0.1') dans le champ '**Server ip'** ;

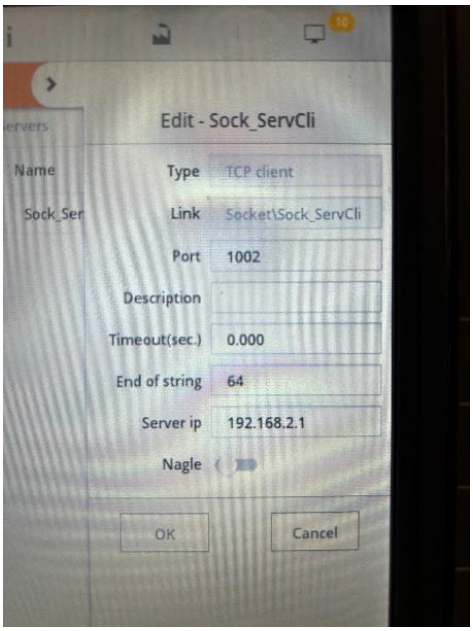
  - transférer l'application du client (jusqu'à présent émulée dans SRS) du PC vers le contrôleur Cs9. Pour cela, sélectionnez le contrôleur '**Controller1'**, puis cliquez droit sur lui afin de sélectionner '**Upload All Applications**' ; ceci va avoir pour effet de transférer dans le contrôleur Cs9 l'ensemble des fichiers nécessaires à l'exécution de l'application '**Client_TCP'**. Il s'ensuit l'affichage d'une fenêtre, voir figure qui suit (Host : 192.168.2.2, Password : default), à valider pour permettre le transfert de l'ensemble des fichiers dans le contrôleur Cs9.



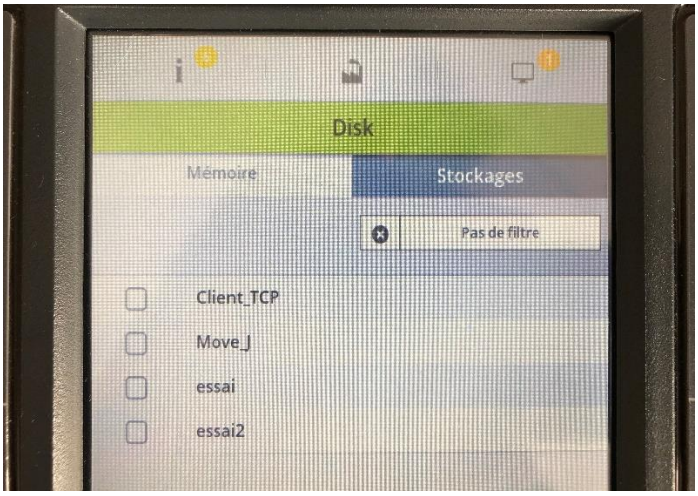
N.B. : Il est important de vérifier que l'application est bien installée dans le contrôleur Cs9. Pour cela, sélectionnez sur le *Teach Pendant* :

- le menu '**Val3>Storages>Disk**' devrait faire apparaître l'application 'Client_TCP'.

- le menu '**IO>Socket>TCP Clients**' devrait faire apparaître la socket '**Sock_ServCli**' (l'affichage de ses caractéristiques s'obtenant en double-cliquant sur '**Sock_ServCli**'). Si la socket n'a pas été transférée au contrôleur (ce qui peut arriver !!), il faut la créer directement sur le *Teach Pendant*. Pour cela, aller dans le menu '**IO>Socket>TCP Clients**' afin d'appuyer sur la touche '**+**' (située en bas à droite) pour permettre de créer la socket comme indiquée dans la figure qui suit.



- charger l'application (pour l'instant sur le disque du contrôleur) en mémoire vive (pour permettre son exécution), pour cela allez dans le menu '**VAL3>Storages**' afin de sélectionner l'onglet '**Disk>Stockages**', voir la figure qui suit.



Le fait de sélectionner l'application '**Client_TCP**' devrait la transférer dans la mémoire vive du contrôleur (l'application apparaît alors dans l'onglet '**Memory**') ;

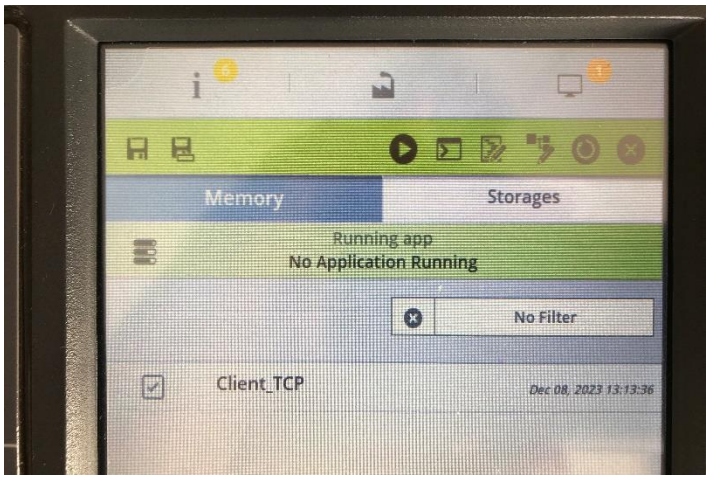
- **au niveau de l'application du serveur** à remplacer l'instruction '**HOST = 127.0.0.1**' par '**HOST = 192.168.2.1**'.

B.2) Procédure permettant l'exécution des applications du serveur et du client

L'exécution des applications du serveur et du serveur (dans cet ordre afin que le serveur soit 'prêt') se fait comme suit :

- **du côté du serveur :** exécutez le programme '**serveur_Staubli.py**' afin que le serveur se mette en écoute (le message « Serveur prêt, en attentes de requêtes … » s'affiche alors dans sa fenêtre d'exécution) ;

- **du côté du client** (*via* le *Teach Pendant*) : une fois l'application 'Client_TCP' sélectionnée (dans l'onglet '**Memory**'), cliquez sur le bouton ▶ situé dans l'onglet '**ControlPad**', voir la figure qui suit.



Le fait de cliquer sur le bouton ■ (situé dans l'onglet '**ControlPad**') permet d'arrêter l'exécution de l'application. Notons que l'exécution de l'application ne nécessite pas de mettre le bras sous puissance.

Les résultats sont affichés à travers le '**LOGGER**' (fenêtre accessible en sélectionnant l'icône '**i**' située en haut à gauche du *Teach Pendant*), voir la figure qui suit.