FAST MOVING TECHNOLOGY

STÄUBLI

socket TCP-IP

Technical documentation

White paper

TSS.0000.009.04-A - 20/06/2019

A "readme.pdf" document may be delivered on the robot's DVD. It contains the documentation addenda and errata.

Table of Contents

History

Revision	Modification	Date (yyyy-mm-dd)	By
A	Initial release	2019 06 20	A.RUSSO
B			
C			

Version

That document has been tested with :

- SRS 2019 4.1
- SRC : s8.8.2
- Safety : 1.000
- SYCON.net : 1.500 Build 180125

Keyword

socket, tcp-ip, ethernet comunication, …

1 Preliminary

DANGER

Instructions drawing the reader's attention to the risks of accidents that could lead to serious bodily harm if the steps shown are not complied with. In general, this type of indication describes the potential danger, its possible effects and the necessary steps to reduce the danger.

It is essential to comply with the instructions to ensure personal safety..

SAFETY

Instructions drawing the reader's attention that its responsibility is engaged if the steps shown are not complied with.

It is essential to comply with the instructions to maintain the robot safety level.

Caution

Instructions directing the reader's attention to the risks of material damage or failure if the steps shown are not complied with. It is essential to comply with these instructions to ensure equipment reliability and performance levels.

ELECTRICAL risk

Instructions drawing the reader's attention to the risks of electrical shock.

It is essential to comply with the instructions to ensure personal safety..

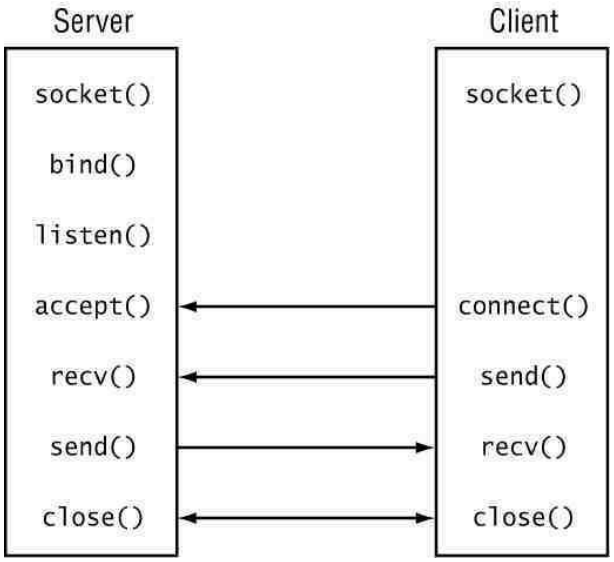
Information

Supplies further information, or underlines a point or an important procedure. This information must be memorized to make it easier to apply and ensure correct sequencing of the operations described.

2 What is a socket connection

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite

TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network.



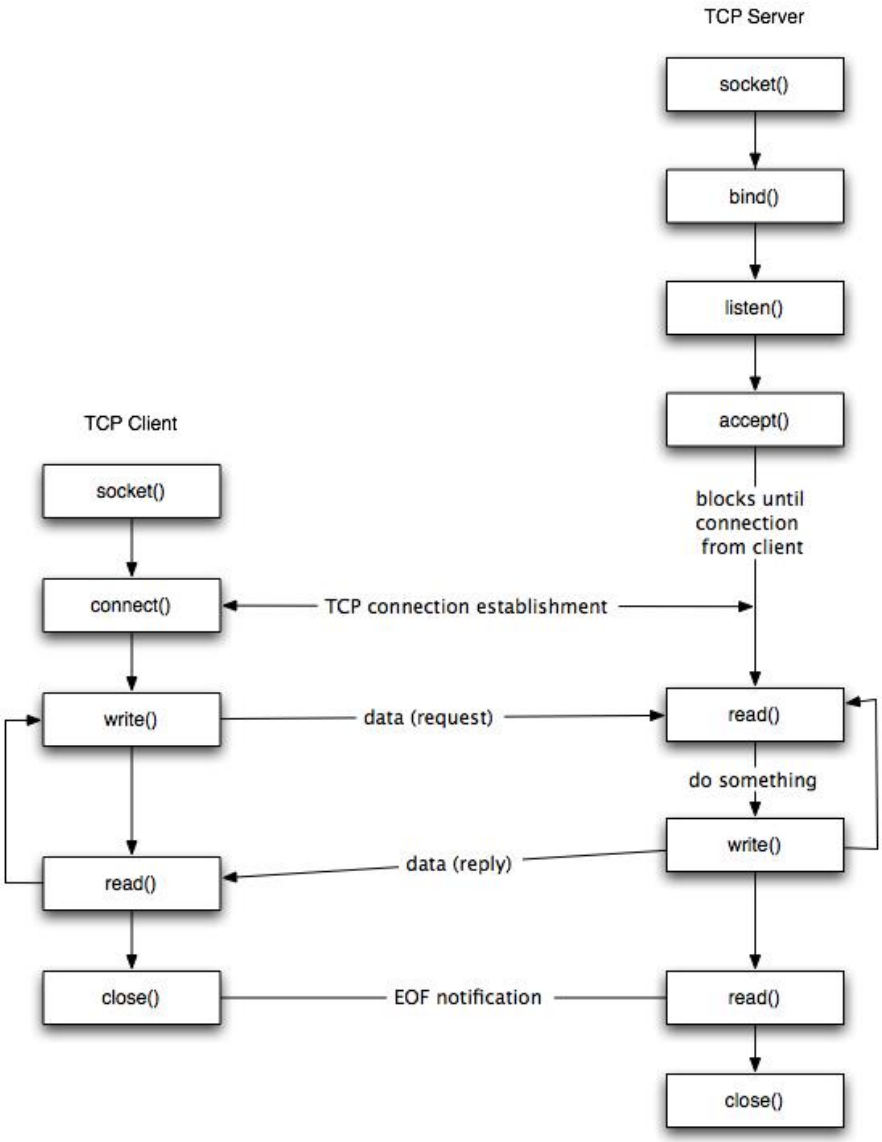
Major internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

2.1 What is TCP/IP and where did it come from?

TCP/IP stands for “Transmission Control Protocol / Internet Protocol”. It is basically a network protocol that defines the details of how data is sent and received through network adapters, hubs, switches, routers and other network communications hardware.

The TCP/IP protocol was also placed in the public domain so that any software company could develop networking software based on the protocol. Because it is the primary protocol used on the Internet, and it is in the public domain, it has become the most popular networking protocol throughout the world and is therefore well supported by almost all computer systems and networking hardware.

2.2 TCP/IP Client and Server Connections



TCP/IP connections work in a manner similar to a telephone call where someone has to initiate the connection by dialing the phone.

At the other end of the connection, someone has to be listening for calls and then pick up the line when a call comes in. In TCP/IP communications, the IP Address is analogous to a telephone number and the port number would be analogous to a particular extension once the call has been answered.

The “**Client**” in a TCP/IP connection is the computer or device that “dials the phone” and the “Server” is the computer that is “listening” for calls to come in. In other words, the

Client needs to know:

- the IP Address
- the port number

of the Server

Information

The Server only has to listen for connections and either accept them or reject them when they are initiated by a client

Information

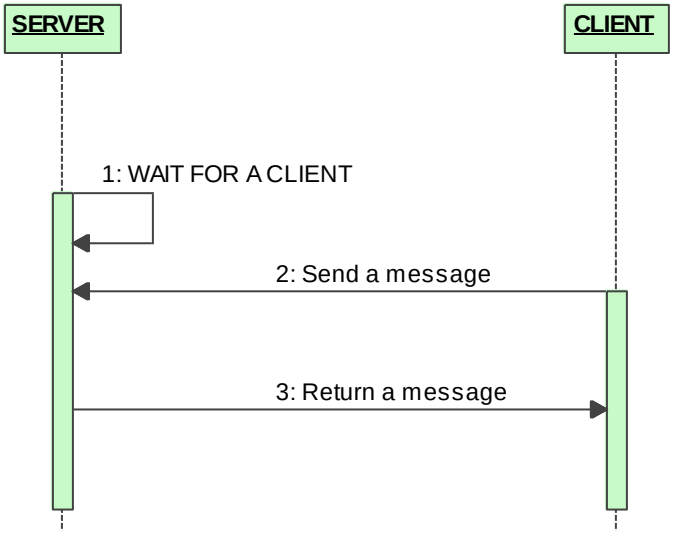
Once a connection through a TCP/IP port has been established between a TCP/IP client and a TCP/IP server, data can be sent in either direction

The connection between a Client and a Server remains open until either the client or the server terminates the connection (i.e. hangs up the phone).

One extremely nice benefit of the TCP/IP protocol is that the low level drivers that implement the sending and receiving of data perform error checking on all data so you are guaranteed that there will be no errors in any data that you send or receive.

3 Communication schema

This is the schema for the communication between the server and the client



4 What's the Difference Between TCP and UDP?

TCP/IP is a suite of protocols used by devices to communicate over the Internet and most local networks.

It is named after two of its original protocols—

1. the Transmission Control Protocol (TCP)
2. and the Internet Protocol (IP).

TCP IP

TCP provides apps a way to deliver (and receive) an ordered and error-checked stream of information packets over the network.

UDP

The User Datagram Protocol (UDP) is used by apps to deliver a faster stream of information by doing away with error-checking..

When configuring some network hardware or software, you may need to know the difference.

4.1 What they Have In Common

Both TCP and UDP are protocols used for sending bits of data—known as packets—over the Internet.

Both protocols build on top of the IP protocol.

In other words, whether you're sending a packet via TCP or UDP, that packet is sent to an IP address.

These packets are treated similarly, as they're forwarded from your computer to intermediary routers and on to the destination.

4.2 How TCP Works

TCP is the most commonly used protocol on the Internet.

When you request a web page in your browser, your computer sends TCP packets to the web server's address, asking it to send the web page back to you. The web server responds by sending a stream of TCP packets, which your web browser stitches together to form the web page. When you click a link, sign in, post a comment, or do anything else, your web browser sends TCP packets to the server and the server sends TCP packets back.

TCP is all about reliability—packets sent with TCP are tracked so no data is lost or corrupted in transit. This is why file downloads don't become corrupted even if there are network hiccups. Of course, if the recipient is completely offline, your computer will give up and you'll see an error message saying it can't communicate with the remote host.

TCP achieves this in two ways. First, it orders packets by numbering them. Second, it error-checks by having the recipient send a response back to the sender saying that it has received the message. If the sender doesn't get a correct response, it can resend the packets to ensure the recipient receives them correctly.

TCP guarantees the recipient will receive the packets in order by numbering them. The recipient sends messages back to the sender saying it received the messages. If the sender does not get a correct response, it will resend the packets to ensure the recipient received them. Packets are also checked for errors..

4.3 How UDP Works

The UDP protocol works similarly to TCP, but it throws out all the error-checking stuff. All the back-and-forth communication introduces latency, slowing things down.

When an app uses UDP, packets are just sent to the recipient. The sender doesn't wait to make sure the recipient received the packet—it just continues sending the next packets. If the recipient misses a few UDP packets here and there, they are just lost—the sender won't resend them. Losing all this overhead means the devices can communicate more quickly.

UDP is used when speed is desirable and error correction isn't necessary. For example, UDP is frequently used for live broadcasts and online games.

For example, let's say you're watching a live video stream, which are often broadcast using UDP instead of TCP. The server just sends a constant stream of UDP packets to computers watching. If you lose your connection for a few seconds, the video may freeze or get jumpy for a moment and then skip to the current bit of the broadcast. If you experience minor packet-loss, the video or audio may be distorted for a moment as the video continues to play without the missing data.

This works similarly in online games. If you miss some UDP packets, player characters may appear to teleport across the map as you receive the newer UDP packets. There's no point in requesting the old packets if you

missed them, as the game is continuing without you. All that matters is what's happening right now on the game server—not what happened a few seconds ago. Ditching TCP's error correction helps speed up the game connection and reduces latency.

When using UDP, packets are just sent to the recipient. The sender will not wait to make sure the recipient received the packet — it will just continue sending the next packets. If you are the recipient and you miss some UDP packets, too bad — you cannot ask for those packets again. There is no guarantee you are getting all the packets and there is no way to ask for a packet again if you miss it, but losing all this overhead means the computers can communicate more quickly.

4.4 What is the difference between TCP and UDP?

TCP	UDP
Reliable	Unreliable
Connection-oriented	Connectionless
Segment retransmission and flow control through windowing	No windowing or retransmission
Segment sequencing	No sequencing
Acknowledge segments	No acknowledgement

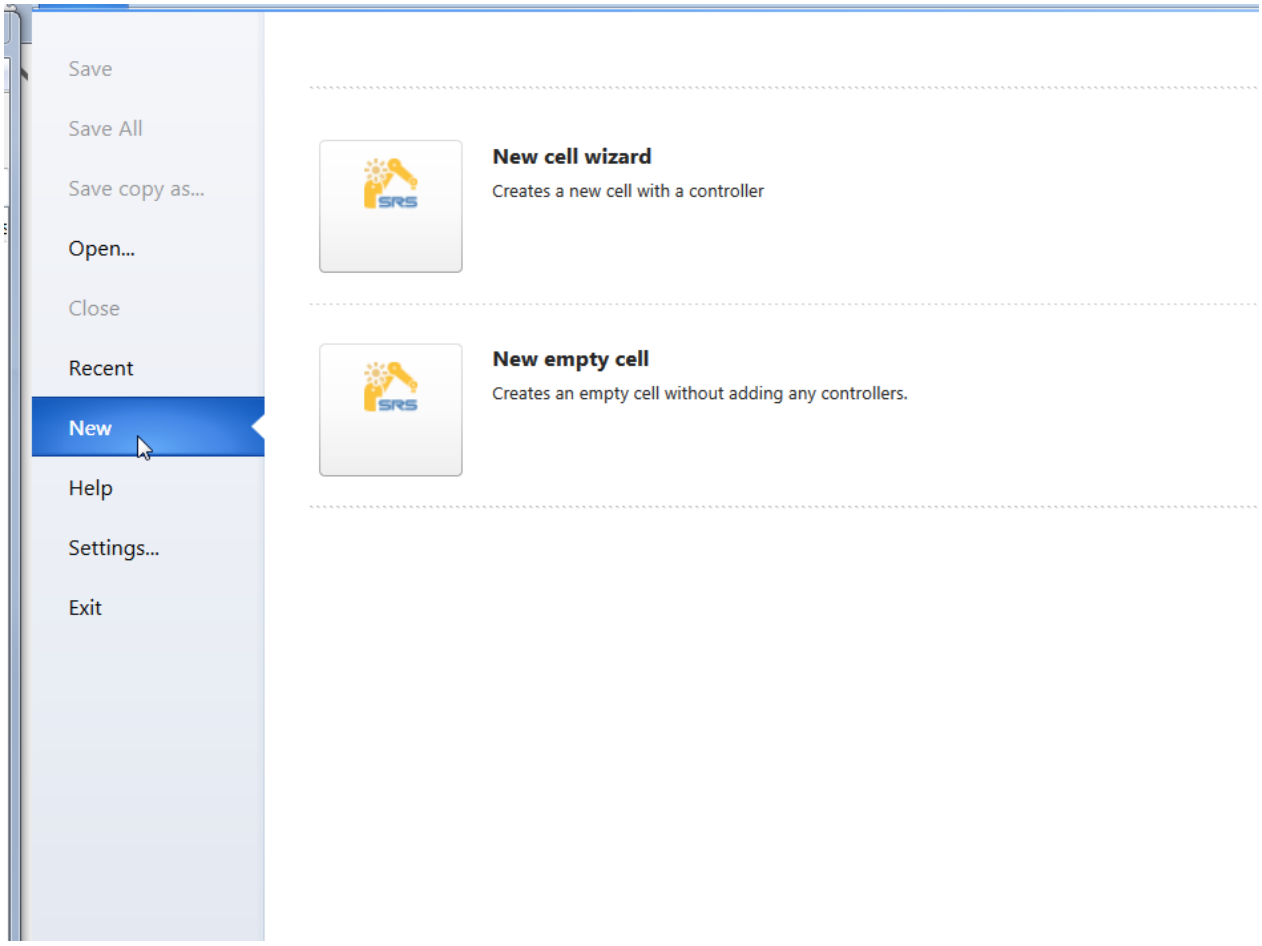
Both TCP and UDP are protocols used for sending bits of data — known as packets — over the Internet. They both build on top of the Internet protocol. In other words, whether you are sending a packet via TCP or UDP, that packet is sent to an IP address. These packets are treated similarly, as they are forwarded from your computer to intermediary routers and on to the destination.

TCP and UDP are not the only protocols that work on top of IP. However, they are the most widely used. The widely used term "TCP/IP" refers to TCP over IP. UDP over IP could just as well be referred to as "UDP/IP", although this is not a common term.

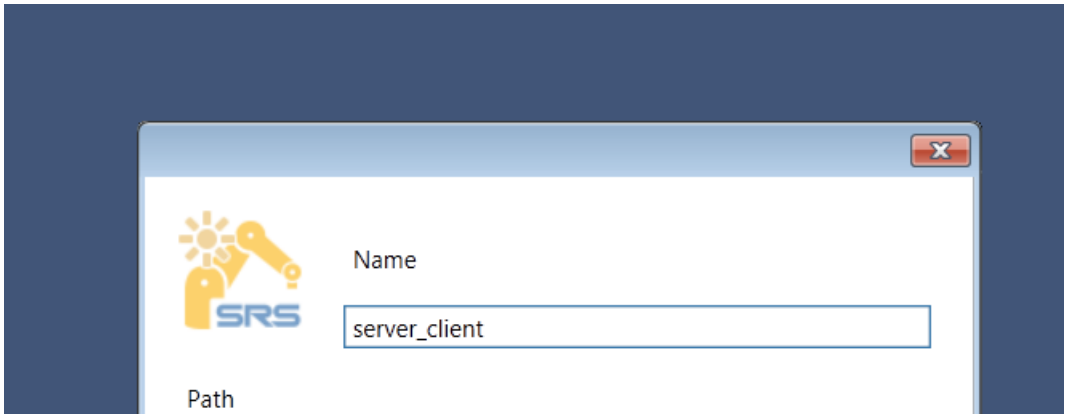
5 TCP-IP socket using the val3

5.1 Create the Communication using SRS

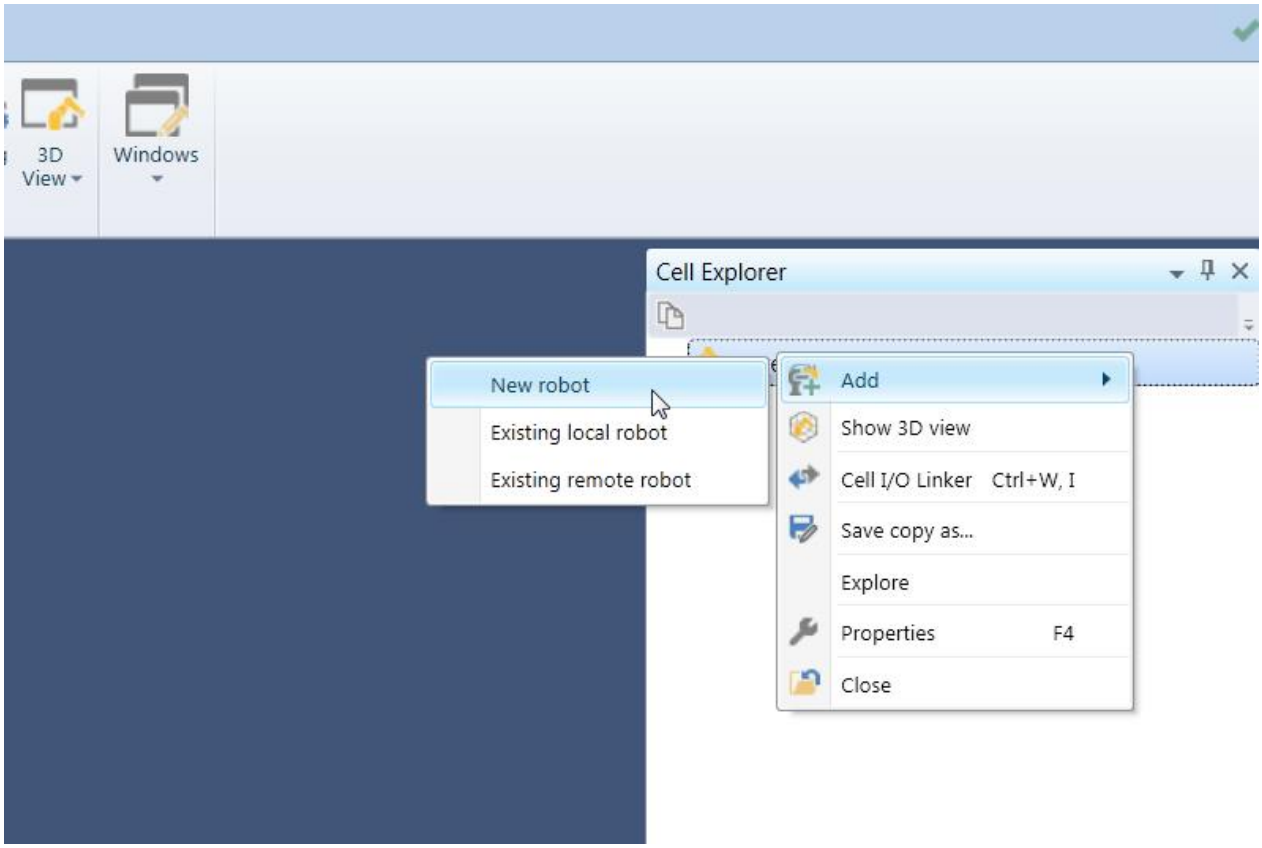
Create a new empty cell with SRS



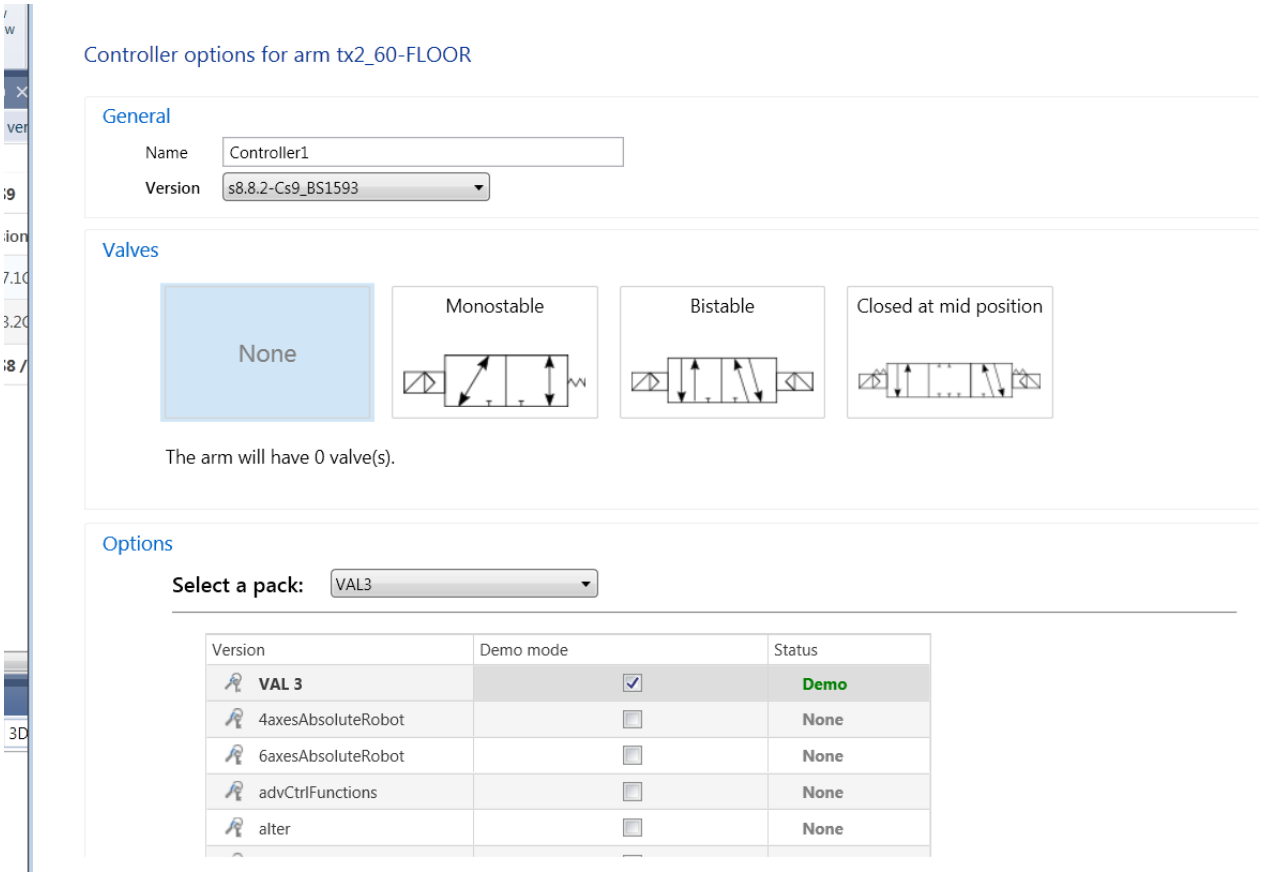
Define the name of the cell as server_client



Add a new robot for example TX2_60



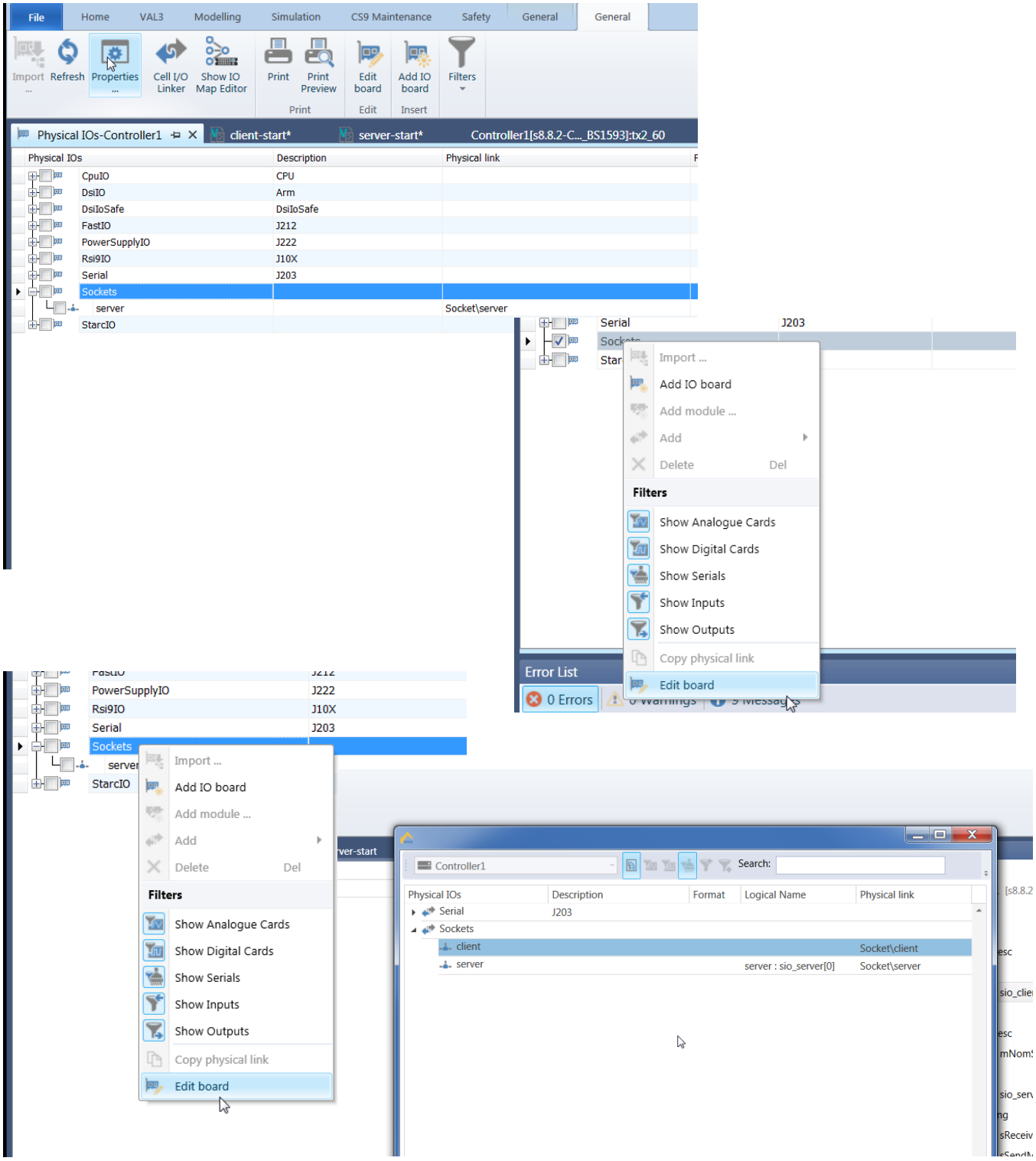
the version val3 selected is the 8.8.2



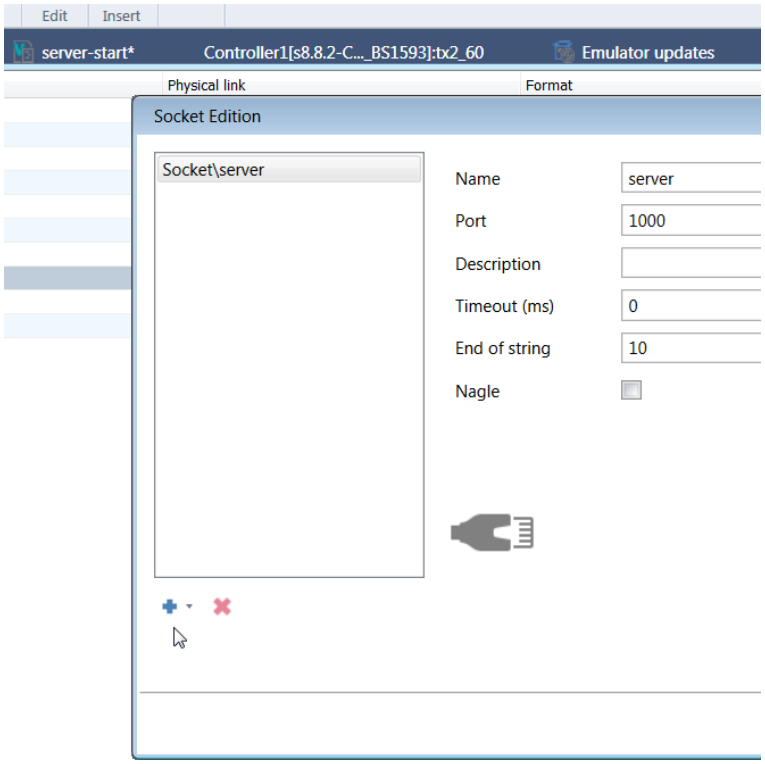
Create 2 new application :

5.2 **server – client controller**

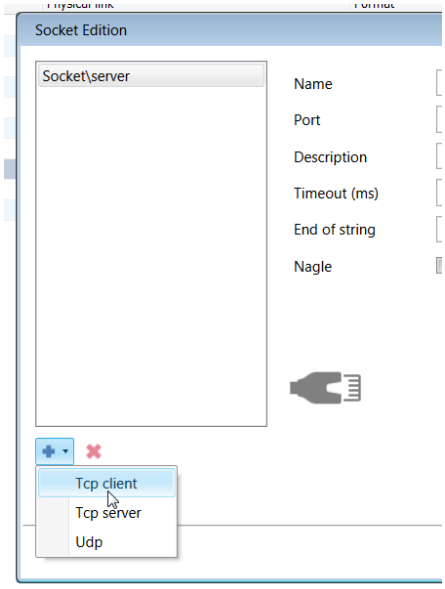
into each controller create one **socket**



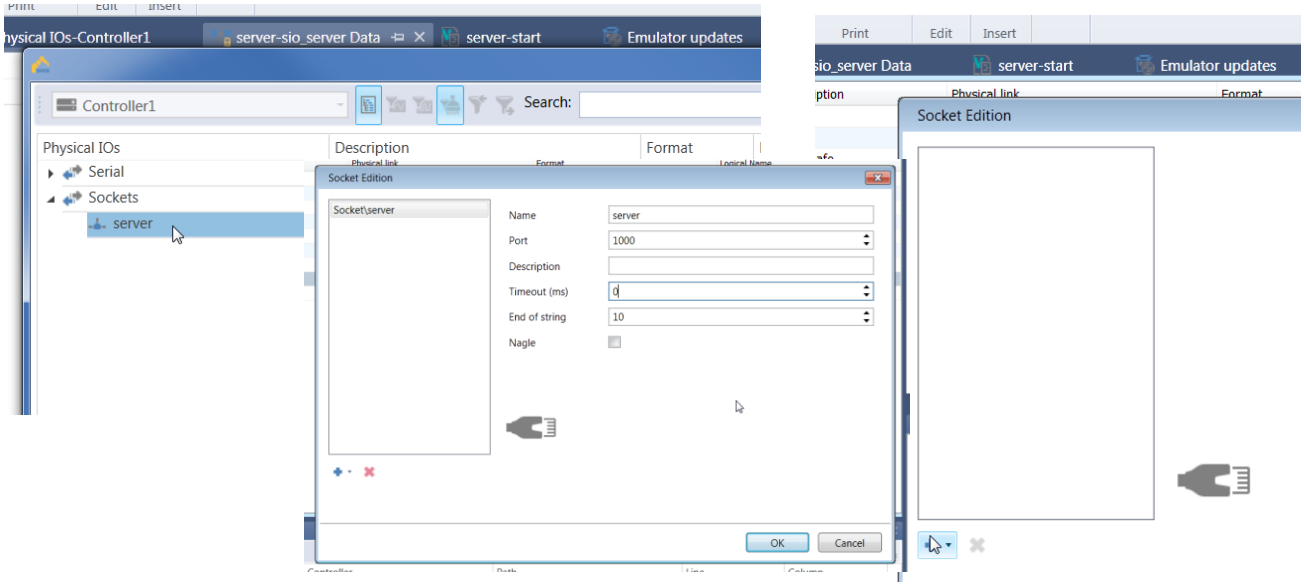
in the server controller create a **server TCP IP**



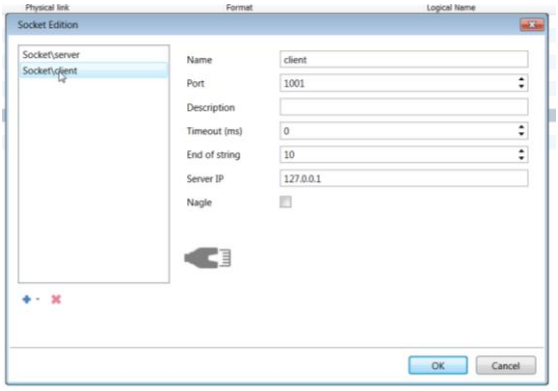
in the client controller create a **client TCP IP**



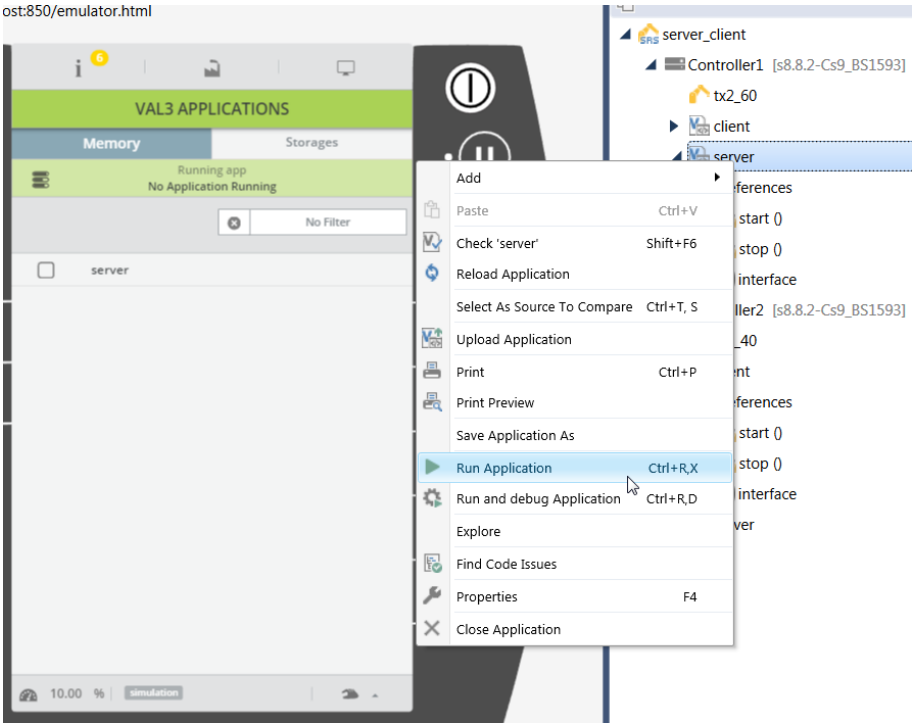
Server declaration



Client declaration

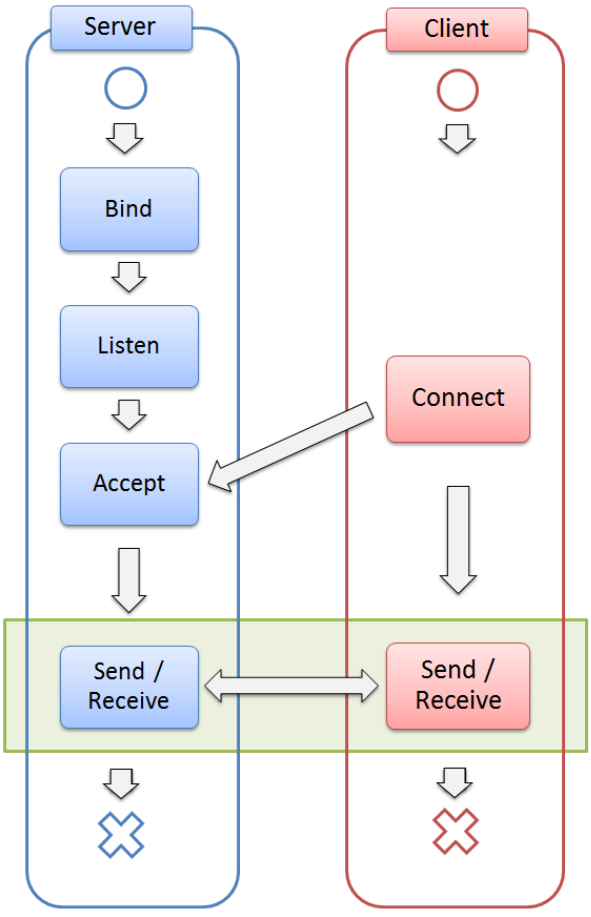


Run the applications



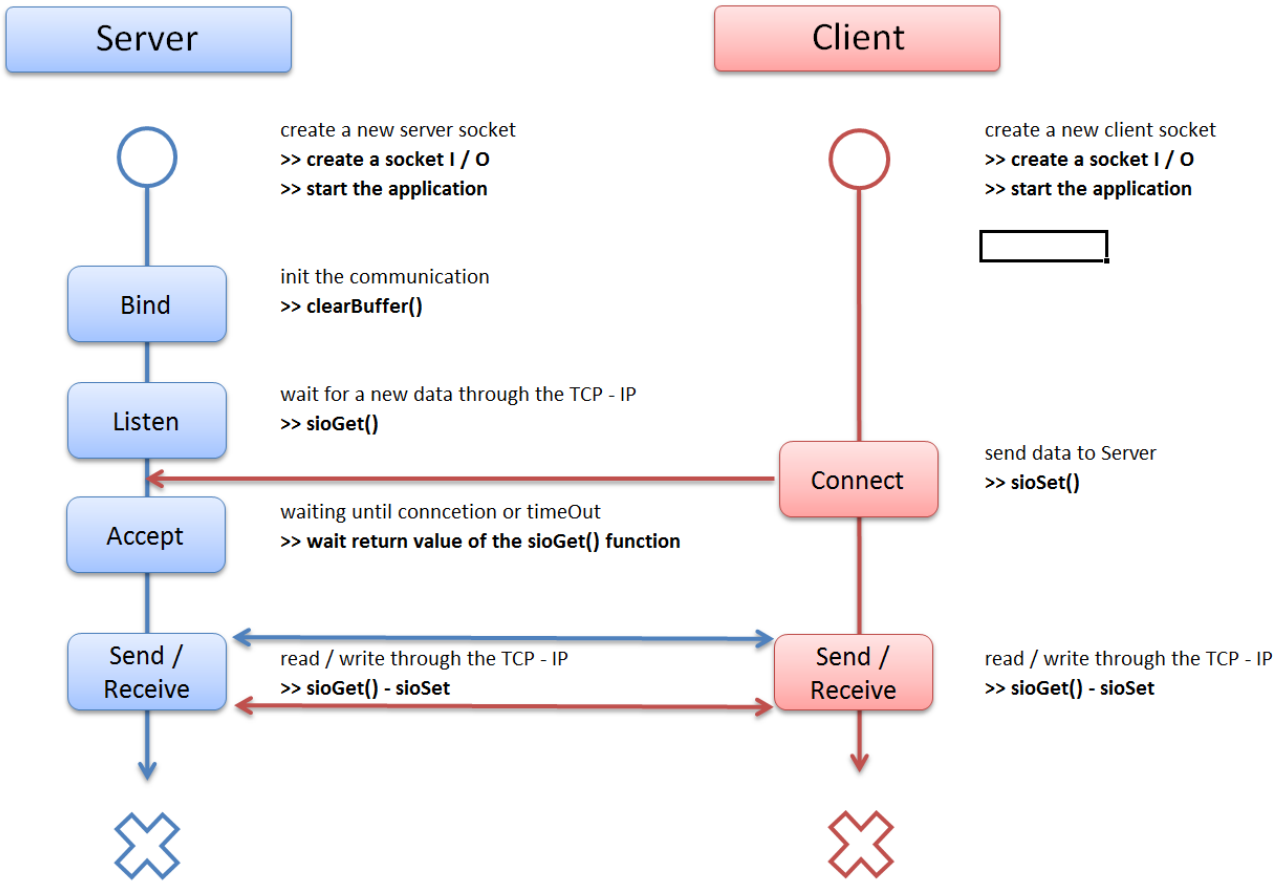
5.3 Initialize the communication

1. the server has to be on “listen”, sioGet or string= sioData
2. the client send Data to the server in order to established a connection, sioSet() or sioData=string
3. the server wait the client has been connected using the timeout:
    - -1: the server will give back a result without waiting :
        - - 1 : no message arrived
        - n char received
    - 0: the server will wait until a message has been received
    - n: the server will wait until n-millisecond and give back a result
        - -1 : no message arrived
        - n char received



5.4 Send Receive

When the communication has been established the server and the client can send and receive some messages or characters, until one of the two socket will be close



5.5 Disconnection

If the sender will close the communication

- the sioGet will retrun -1 even if the timeOut is set to 0
- the string = sioData will stop the current task

If the receiver will close the communications t

- the sioSet() will return -1 even if the timeOut has been set to 0
- the sioData = string will stop the current task

5.6 SioGet - SioSet

Basic programs on val3 how to start the communication on the client – server

- **server**

```
begin
  // clear the buffer
  clearBuffer(sio_server)
  ? "Server is ready.."
  // wait the client has sent one message
  sReceiveMsg=""
  do
    l_nResult=sioGet(sio_server,l_nReceiveByte)
    sReceiveMsg=sReceiveMsg+chr(l_nReceiveByte)
  until l_nResult!=1 or l_nReceiveByte==13
  ? "receive: "+sReceiveMsg
  // write
  ? sSendMsg="Hello I'm the Staubli server "
  for i=0 to len(sSendMsg)-1
    l_nResult=sioSet(sio_server,asc(sSendMsg,i))
    if l_nResult!=1
      i=len(sSendMsg)
      l_bError=true
    endIf
  endFor
  // connection will close
  ? "end"
  wait(false)
end
```

- **client**

```
begin
  // clear the buffer
  ? "Client is ready.."
  // write
  ? sSendMsg="Hello I'm the Staubli client "
  for i=0 to len(sSendMsg)-1
    l_nResult=sioSet(sio_client,asc(sSendMsg,i))
    if l_nResult!=1
      i=len(sSendMsg)
      l_bError=true
    endIf
  endFor
  // send the last char
  l_nEOS=13
  l_nResult=sioSet(sio_client,l_nEOS)
  if l_nResult!=1
    l_bError=true
  endIf
  // wait the server has sent one message
  sReceiveMsg=""
  do
    l_nResult=sioGet(sio_client,l_nReceiveByte)
    sReceiveMsg=sReceiveMsg+chr(l_nReceiveByte)
  until l_nResult!=1 or l_nReceiveByte==13
  ? "receive: "+sReceiveMsg
  // connection will close
  ? "end"
  wait(false)
end
```

5.7 example using directly a sioData

- **server**

```
begin
  // clear the buffer
  clearBuffer(sio_server)
  ? "Server is ready.."
  // wait the client has sent one message
  sReceiveMsg=sio_server
  ? "receive: "+sReceiveMsg
  // write
  sSendMsg="Msg sent >> Hello I'm the server "
  sio_server=sSendMsg
  // connection will close
  ? "end"
  wait(false)
end
```

- **client**

```
begin
  ? "Client is ready.."
  // write
  sSendMsg="Msg sent >> Hello I'm the client "
  sio_client=sSendMsg
  // wait the server has sent one message
  sReceiveMsg=sio_client
  ? "receive: "+sReceiveMsg
  // connection will close
  ? "end"
  wait(false)
end
```

5.8 Message from the console

client:

```
Application 'client' started.
"Client is ready.."
" Hello I'm the staubli client "
"receive: Welcome to my server"
"end"
```

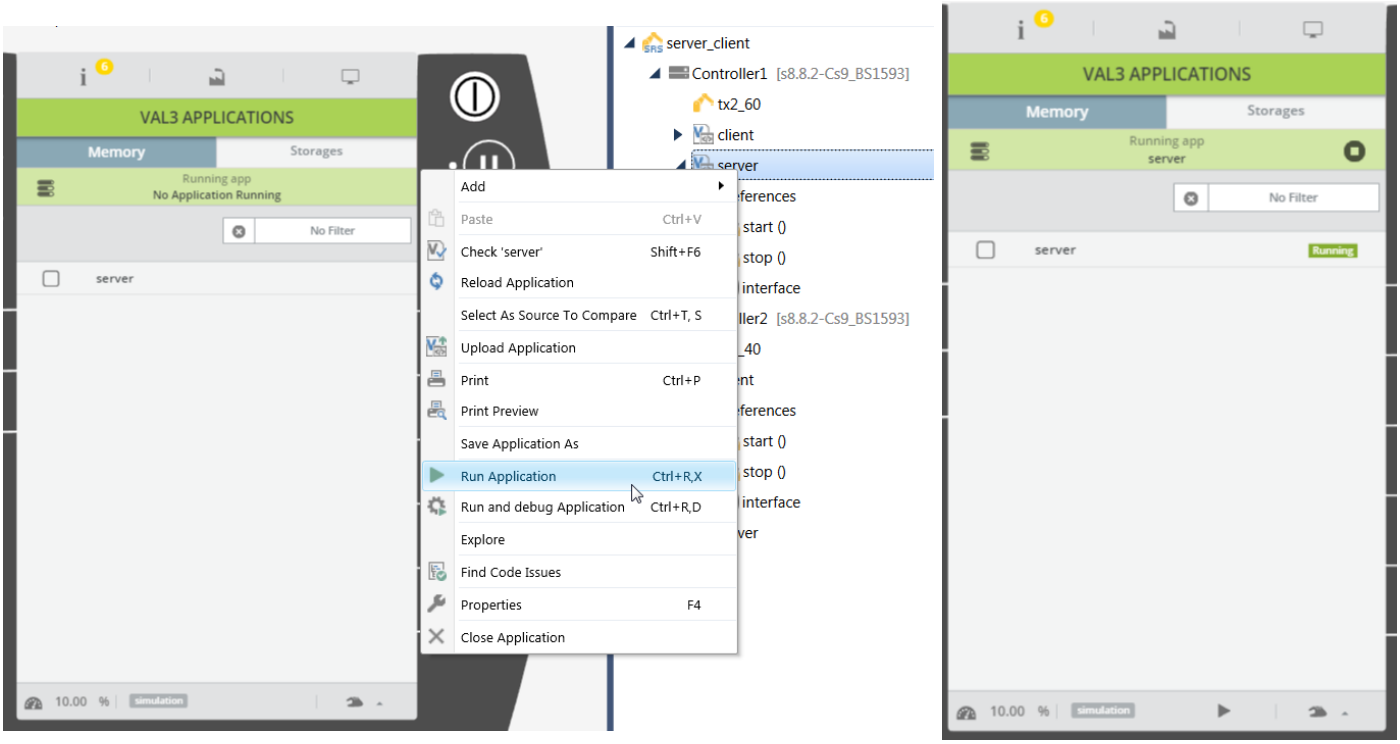
server:

```
Application 'server' started.
"Server is ready.."
"receive: Is anybody there"
"Hello I'm the staubli server "
"end"
```

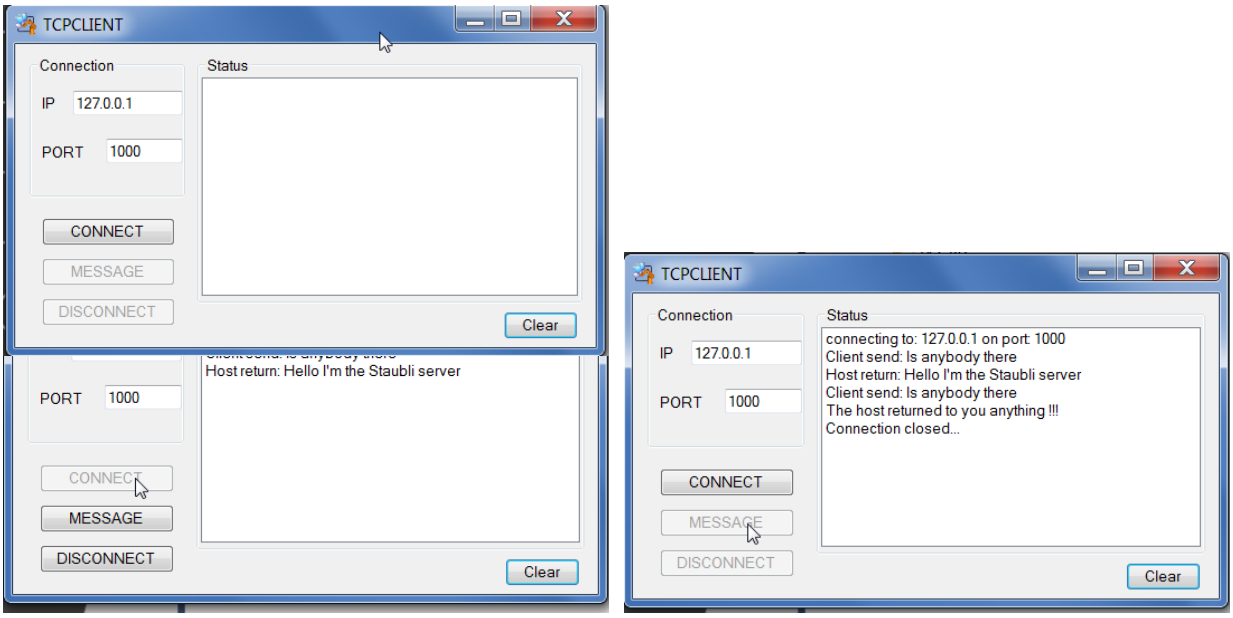
5.9 Test using an external client – server

server val3 ; **client** cSharp

1) run the server application inside the Controller1 of the server-client cell
2) check if the server application is inside the running application on the VAL3 APPLICATIONS menu

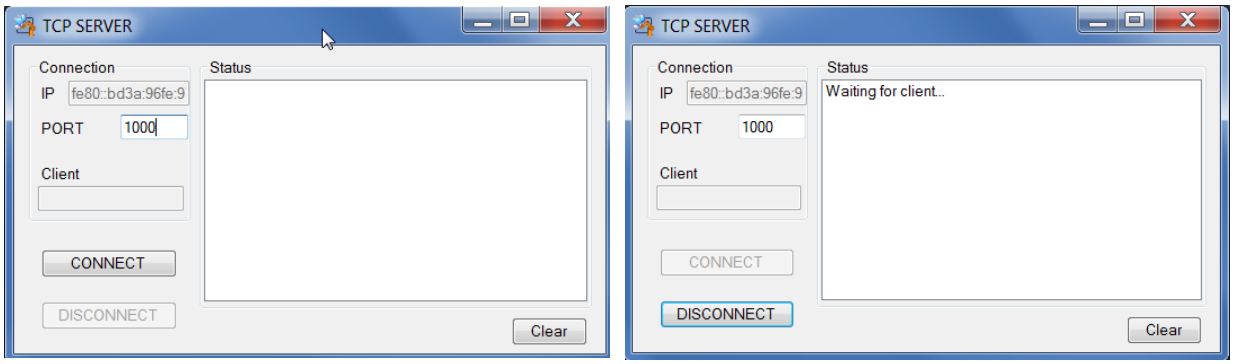


3) execute the TCPCLIENT windows application
4) Insert the IP and the PORT of the server
5) CONNECT the client

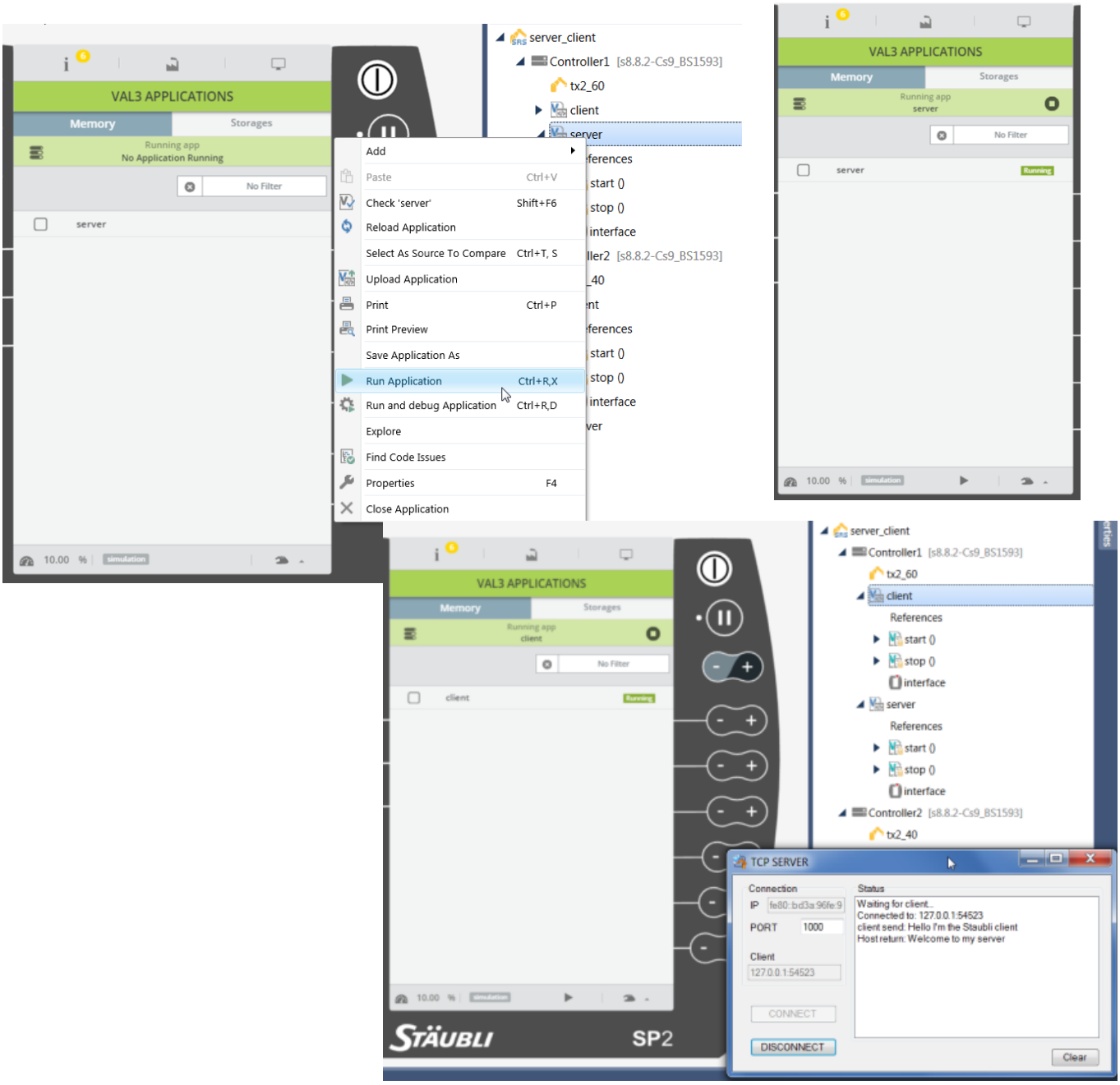


client val3 ; **server** cSharp

1) execute the TCPSERVER windows application
2) Define the PORT of the server
3) CONNECT the server



4) run the client application inside the Controller1 of the server-client cell
5) check if the server application is inside the running application on the VAL3 APPLICATIONS menu



5.10 Test sending array of bits and floats

server

```
begin
  // clear the buffer
  clearBuffer(sio_server)
  ? "Server is ready.."
  ? "receive: "
  // wait the server has sent one message
  l_nResult=sioGet(sio_server,l_nReceiveByte)
  if l_nResult==size(l_nReceiveByte) and l_nReceiveByte[7]==13
    // 1 byte  >> 8 bit
    // 2 bytes >> 1 word
    // 4 bytes >> 1 float
    if fromBinary(l_nReceiveByte,1,"1",l_nByte)!=1
      l_bError=true
    endIf
    call word2bits(l_nByte,bBits)
    ? bBits[0]
    ? bBits[1]
    ? bBits[2]
    ? bBits[3]
    ? bBits[4]
    ? bBits[5]
    ? bBits[6]
    ? bBits[7]
    ? l_nByte
    if fromBinary(l_nReceiveByte[1],2,"21",nWord)!=2
      l_bError=true
    endIf
    ? nWord
    if fromBinary(l_nReceiveByte[3],4,"4.01",nFloat)!=4
      l_bError=true
    endIf
    ? nFloat
  endIf
  // send it
  ? "write"
  // array of bits
  ? bBits[0]=true
  ? bBits[1]=true
  ? bBits[2]=true
  ? bBits[3]=false
  ? bBits[4]=false
  ? bBits[5]=false
  ? bBits[6]=false
  ? bBits[7]=false
  call bits2word(bBits,nSendMessage)
  ? nSendMessage
  // word double bytes
  ? l_nWord=333
  toBinary(l_nWord,2,"21",nSendMessage[1])
  // float 4 bytes
  ? l_nFloat=987.654321
  toBinary(l_nFloat,4,"4.01",nSendMessage[3])
  // EOS - send the last char
  l_nEOS=13
  nSendMessage[7]=l_nEOS
  l_nResult=sioSet(sio_server,nSendMessage)
  if l_nResult!=size(nSendMessage)-1
    l_bError=true
  endIf
  // connection will close
  ? "end"
  wait(false)
end
```

client

```
begin
  // clear the buffer
  ? "Client is ready.."
  // write
  ? "write"
  // array of bits
  ? bBits[0]=true
  ? bBits[1]=true
  ? bBits[2]=true
  ? bBits[3]=true
  ? bBits[4]=false
  ? bBits[5]=false
  ? bBits[6]=false
  ? bBits[7]=false
  call bits2word(bBits,nSendMessage)
  ? nSendMessage
  // word double bytes
  ? l_nWord=222
  toBinary(l_nWord,2,"21",nSendMessage[1])
  // float 4 bytes
  ? l_nFloat=123.4567
  toBinary(l_nFloat,4,"4.01",nSendMessage[3])
  // EOS - send the last char
  l_nEOS=13
  nSendMessage[7]=l_nEOS
  l_nResult=sioSet(sio_client,nSendMessage)
  if l_nResult!=size(nSendMessage)-1
    l_bError=true
  endIf
  // wait the server has sent one message
  ? "receive: "
  l_nResult=sioGet(sio_client,l_nReceiveByte)
  if l_nResult==size(l_nReceiveByte) and l_nReceiveByte[7]==13
    // 1 byte  >> 8 bit
    // 2 bytes >> 1 word
    // 4 bytes >> 1 float
    if fromBinary(l_nReceiveByte,1,"1",l_nByte)!=1
      l_bError=true
    endIf
    call word2bits(l_nByte,bBits)
    ? bBits[0]
    ? bBits[1]
    ? bBits[2]
    ? bBits[3]
    ? bBits[4]
    ? bBits[5]
    ? bBits[6]
    ? bBits[7]
    ? l_nByte
    if fromBinary(l_nReceiveByte[1],2,"21",nWord)!=2
      l_bError=true
    endIf
    ? nWord
    if fromBinary(l_nReceiveByte[3],4,"4.01",nFloat)!=4
      l_bError=true
    endIf
    ? nFloat
  endIf
  // connection will close
  ? "end"
  wait(false)
end
```

5.11 Message from the console

server

```
Application 'server' started.
"Server is ready.."
"receive: "
true
true
true
true
false
false
false
false
15
222
123.456703
"write"
true
true
true
false
false
false
false
false
7
333
987.654321
"end"
```

client

```
Application 'client' started.
"Client is ready.."
"write"
true
true
true
true
false
false
false
false
15
222
123.4567
"receive: "
true
true
true
false
false
false
false
false
7
333
987.654297
"end"
```

5.12 Run the Controller 3 – 4 together to simulate client – server

In order to execute both:

- client
- server

on val3

you can run

the server on the Controller 3

the client on the Controller 4

- The server application need to start and be in listen mode
- The client start
- The client send one message using some bytes
- The server is listen and convert the bytes[] into bits / word /float
- The server is sending back
  - some bytes
  - one string for the camera trsf
- The client is reading the:
  - bytes and convert into bits / word /float
  - string and convert into a trsf

The dot net tcp client – server

5.13 Execute the TcpClient.exe

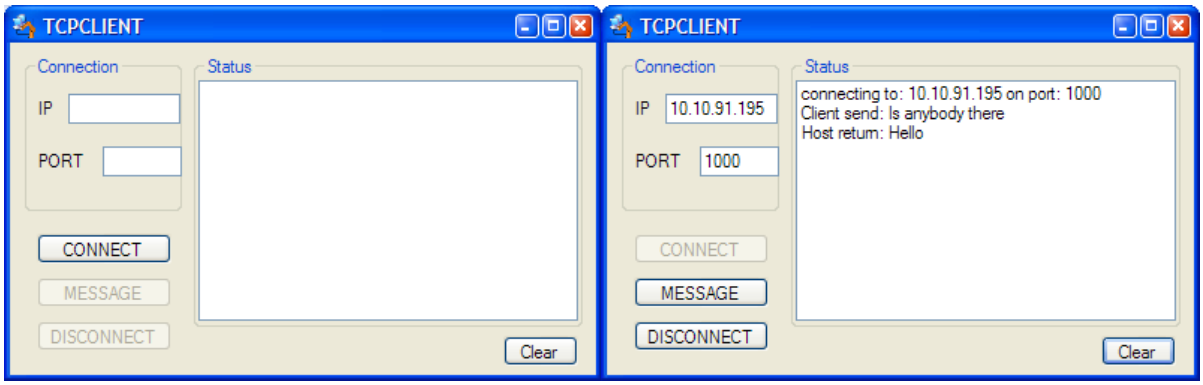
Launch from your pc the TcpClient executable file

5.14 Configure the TcpClient application

In order to Connect to the CS9 server follow the following steps:

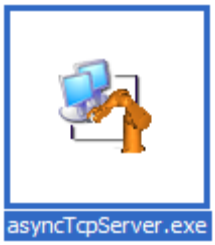
1) insert the IP of the CS9 and the port that we have previously define
2) press the connect button
3) looking in the CS9 we will have also a response
4) to send other message use the Message button

to close the communication use the disconnect button



5.15 **Execute the TcpServer.exe**

Launch from your pc the TcpServer executable file

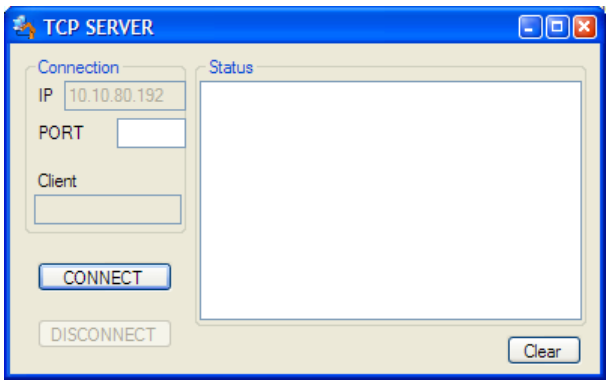


5.16 **Configure the Tcp Server application**

To Start the communication to the CS9 client follow the following steps:

1. insert the port of the CS9 that we have previously define
2. press the connect button

to close the communication use the disconnect button



5.17 **Error note:**

CS9 as Server

- If the test_server library has not been launched from the CS9, and we try to connect the client, we have a connection refuse error.

- If we connect the server and the client and we stop the server on CS9, if we try to send a message the first time we will have a null response and the second time an error of writing reading to the transport connection, because we have lost the communication. To solve the null response it's necessary to not avoid a null message or before to quit send a message to the client and to close by himself.

- If we connect the server and the client and we stop the client on the pc, the val3 will give us an error of writing reading with the code 125, and for this reason in the test_server library there is a parallel task that is supervising the application and resume the task. The test_server application will wait again for a client.

Connection close

- If the server library has not been launched from the computer, and we try to connect the client, the val3 system will wait until the connection is done.

- If we connect the server and the client and we stop the server on the pc, if we try to send a message an error of writing reading with the code 125, because we have lost the communication. The val3 has to restart from the beginning, so there is a supervisor task that will kill and restart the communication task.

- If we connect the server and the client and we stop the client on the CS9, the server application will also close, because it has no response from the client