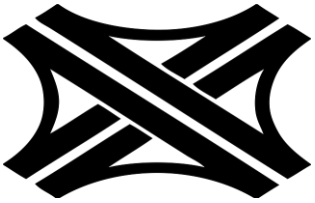
Copyright (c) 2018 Genesis Technologies AG Switzerland, all rights reserved

Network Propeller (GTSNC4ZNETPROP)



Disclaimer

Genesis Technologies does provide this driver as it is. There is no commitment to support, nor does Genesis Technologies assure a decent reaction time in case of a problem. Genesis Technologies can refuse all support, discontinue support, end the driver lifetime at any time. Genesis Technologies will not refund in any case licenses

Notwithstanding the foregoing, Genesis Technologies is not liable to licensee for any damages, including compensatory, special, incidental, exemplary, punitive, or consequential damages, connected with or resulting from this license agreement or licensee's use of this software. Licensee's jurisdiction may not allow such a limitation of damages, so this limitation may not apply.

By using this driver, you accept that you have read and accepted the terms above!

What is the Network Propeller driver? What is it used for?

Named after a motorway junction in Bern this driver does the same for TCP and UDP network connections as a motorway junction for cars would do: Allow traffic in all directions on multiple lines.

In the world of network this can be achieved using proper server and client communication.

Therefore, this driver provides:
- Ability to add up to 10 independent TCP server instances on different ports.
- Ability to add up to 10 independent TCP/ UDP client instances on different/same ports
- Automatic keep alive functions for TCP/UDP clients, ability to set automated keep alive answers in TCP servers
- Receive from TCP server clients and as well from TCP/UDP clients, match strings in Programming
- Send using TCP/UDP client connection.
- Automated conversion from any non-ASCII character into standard $xx notation and vice versa
- Events for online, offline and data received, variable which contains the strings for programming

Software installation Control4
Assuming that you have already downloaded the drivers and placed them in %USERPROFILE%\Control4\Drivers.

Start Composer, go to *System Design* and select the tab *Search* in the right top window. Search for *Network Propeller* and drag and drop the *Network Propeller* driver into the wished room. Click on it and find the Properties page.

Properties
- -> Info: Does show the driver health info

- Driver name: The product identification of the driver
- Activation key: Fill in the activation key.
- -> Log / Lua window: Selects the log level printed out in Lua window from Composer. Use the level *Debug* to see what the driver is working on
- -> Log / Online log server: If a persistent problem appears that cannot be solved from the installer support may ask you to activate a decent level. Do not use until support ask you to do so
- Log period (h): The time frame the log should output
- Log auth: Developer can give you a code to see more logs. Usually not interesting, only for in deep debug

Programming
Do not start any servers or clients on the Event "When the project is loaded".
Or at least add 60 seconds delay before you do so!

Programming Actions

Common to all Programming Actions is the parameter *Instance.* It does define which Instance number of the TCP/UDP server/client should do the action. Do not mess up the instances, note for yourself which instance does what or put this info into the Property *Purpose of servers* and *Purpose of clients*

Open TCP server:
- Opens a TCP server instance on the port specified, always use a delimiter. On arrival of the delimiter the server instance will push the received string onto the variable *TCP_SERVER_INSTANCE_x_DATA* and fire the event *TCP server instance x received data*

Close TCP server:
- Close the server and drop all clients connected to

Set keepalive responses for TCP server
- Once the server instance has received the string inserted in *Listen for* it will reply to the client with the *Answer*

Open TCP client:
- Opens a TCP client connection to the *IP Address* and *Port* specified. As well set a *delimiter* according to your protocol definitions. On arrival of the delimiter the client instance will push the received string onto the variable *TCP_CLIENT_INSTANCE_x_DATA* and fire the event *TCP client instance x received data*

Set keepalive for TCP client:
- Automated keep alive messages will be sent to the client ether to have them happy or to check the connection. *Command* and *Interval* are mandatory fields

Close TCP client:
- Closes the client connection.

Send command using TCP client:
- Sends the *Command* and *Supplement* concatenated together to the client. It's perfectly fine to just to fill in the complete command into *Command* and leave *Supplement* empty.

Open UDP client:
- Opens a UDP client connection to the *IP Address* and *Port* specified. As well set a *delimiter* according to your protocol definitions. On arrival of the delimiter the client instance will push the received string onto the variable *UDP_CLIENT_INSTANCE_x_DATA* and fire the event *UDP client instance x received data.* Use the setting *Mirror Port* if the UDP connection should use the same port to send out as defined in *Port.* Normally this is set to false then C4 chooses a free port

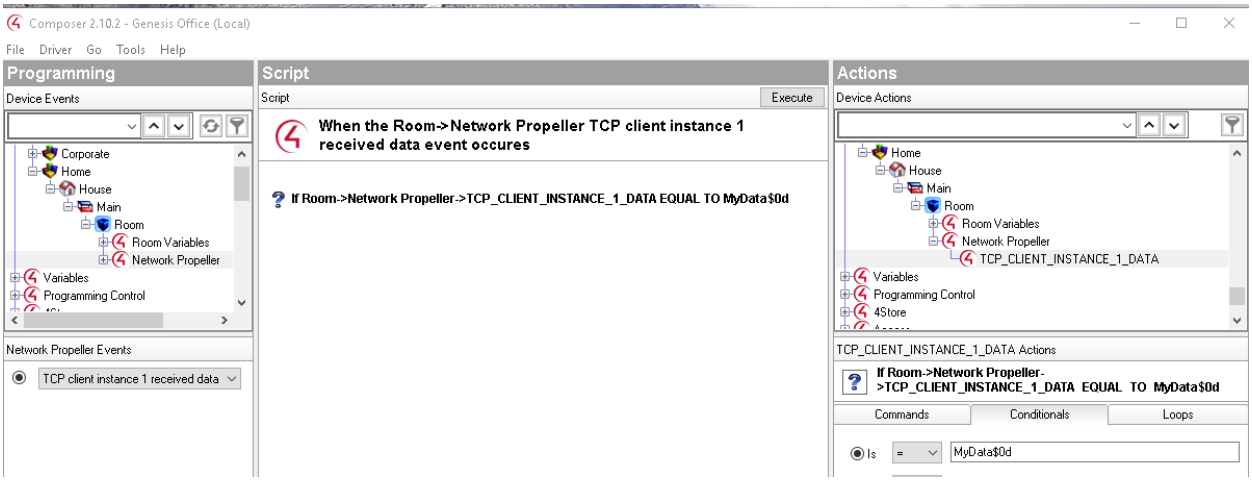
Set keepalive for UDP client:
- Automated keep alive messages will be sent to the client ether to have them happy or to check the connection. *Command* and *Interval* are mandatory fields

Close UDP client:
- Closes the client connection.

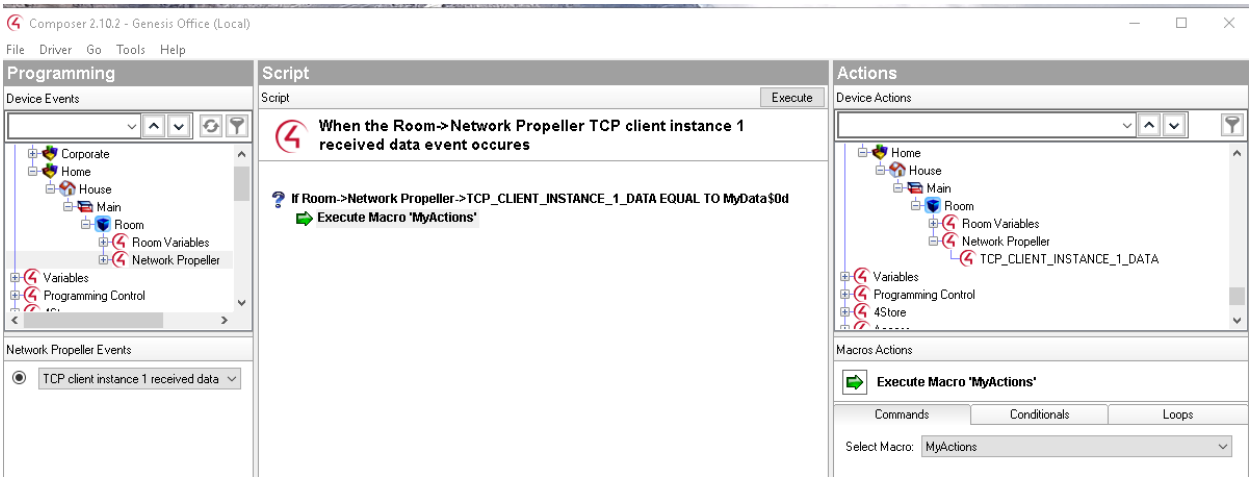
Send command using UDP client:
- Sends the *Command* and *Supplement* concatenated together to the client. It's perfectly fine to just to fill in the complete command into *Command* and leave *Supplement* empty.

Working with variables, matching variables in Programming

All received data will be written into the variable *TCP_CLIENT_INSTANCE_x_DATA, UDP_CLIENT_INSTANCE_x_DATA or TCP_SERVER_INSTANCE_x_DATA.* **Do not use the Event** *When the TCP_CLIENT_INSTANCE_x_DATA / UDP_CLIENT_INSTANCE_x_DATA changes* **on the variable itself**, instead use as indicator that something was received; the Events *TCP server instance x received data, TCP client instance x received data* and *TCP client instance x received data.* **Maybe the same string is received a few times and variable does not change**!
A correct programming for incoming strings is this like: First the conditional



and then the action



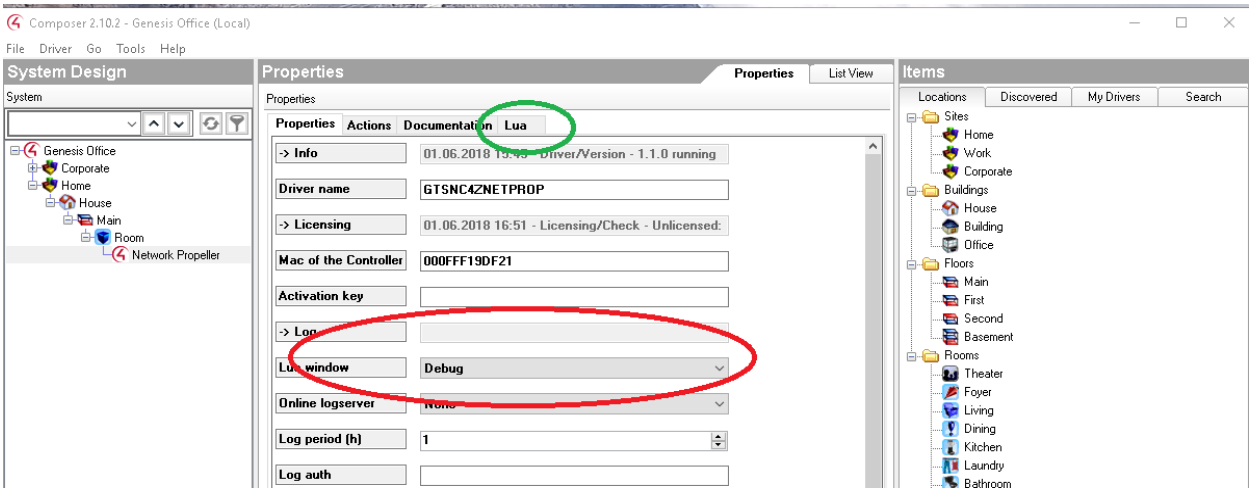
Sending and receiving non-ASCII characters
All non-alphanumeric characters must be sent as $xx and will be received as $xx. Let's assume we send a ping command to our protocol which is defined like this:

- Start character h07
- Command string ("ping")
- Values string ("001")
- Stop characters hb3 h0d h0a

The resulting string with $ masking is: $07ping001$b3$0d$0a

In our example we will then receive back a $07pong001$b3$0d$0a which can be used for sting matching in Programming.

Using Lua window to see communication
Setting up a variable matching, set in Properties the *Lua Window* to *Debug*. Now see the *Lua* tab in Composer to see the exact strings the connected devices do send to the driver. All printed *data* is matching the strings the variables are set to. (Copy and paste may help you)



Known issues
If a UDP connection does not send out data or does not receive anything you may have confused Control4 by setting up and deleting UDP connections. A reboot of the Controller does always help.

Updates / Support
Updates are available on: https://technet.genesis-technologies.ch
Bugs should be reported to: info@genesis-technologies.ch