Using the Modbus TCP Client Interface of the UR Robot

(Document version 1.3 – article 16363)

This article provides a complete understanding of the Universal Robots robot controller Modbus TCP client interface (using the robot as a client to access some other Modbus server). For information regarding the robots *Modbus server* please consult document 16377. This article was written assuming the Modbus functionality provided with Polyscope version 5.2 and 3.8 (eSeries and CB3 robots respectively). For older versions of Polyscope there might be some differences.

This article also provides a quick example of using the robots Modbus client interface to expand the robots IO through a Phoenix Contact IO block that behaves as a Modbus/tcp server.

1. Introduction

a. Understanding Modbus client versus server

The UR robot can behave as Modbus client or server. The Modbus TCP protocol is a client/server type of architecture so it's important when determining how to use the Modbus interface and setup communication with other Modbus devices to understand **which** device is the client and which is the server. It's also important to keep the terminology straight. Client = master and server = slave. The client is making requests of the server. The server is passively listening for requests and the server is never initiating communications on the bus on its own (unsolicited).

b. What is a Modbus TCP interface?

The Modbus TCP protocol is the evolution of Modbus serial protocol (developed by Modicon in 1979) implemented under an Ethernet TCP/IP network communications stack. Some people might refer to the Modbus TCP client as a fieldbus communications 'driver'. When using the UR robot Modbus TCP client interface it can be used to communicate to other robots, PLC's, HMI's or IO devices (**when the IO device is functioning as Modbus server**).

To understand more of the history of this protocol and different uses this web site is a good resource :

http://www.simplymodbus.ca/index.html

c. When to use Server vs Client?

i. Client

If your goal is to expand the IO of the robot and no external PLC is involved in your system design then one way to do this is to use an external fieldbus style IO module and most fieldbus style IO modules (Phoenix Contact, WAGO, etc) only function as a modbus server ***hence the robot must be the client***. Within the Universal Robots user manual for CB3 version 3.8 section 13.11 (Polyscope: Installation -> Modbus Client IO Setup) discusses the modbus client module configuration in detail. For eSeries robots this is detailed in section 16.4.1 of the user manual (Polyscope: Installation -> Fieldbus -> Modbus Client IO Setup).

ii. Server

If the external device ONLY behaves as a Modbus client then in this situation the robot would have to behave as the server. If you want the robot to be the passive recipient in your communication scheme then it might be more appropriate to use it as the server. If you had two UR robots that needed to share a lot of data (as an example for some type of coordination) then it might make sense to make one the server (slave) and the other the client (master) but of course the UR robot CAN function as client and server at the same time. Using the robot as the server is going to use less CPU resources on the robot so if that is a concern then it might be better to let the external device (PLC or HMI) be the client. Most HMI’s and PLC’s will function as a modbus client and if that is the case then it’s better to designate the robot as the server in this communication scheme.

Conveyor Tracking – If you are attempting something like conveyor tracking where the conveyor counts are coming from an encoder via Modbus one option is to have the conveyor signals being read by the PLC (because the PLC is more deterministic) and then have the PLC communicate the encoder counts via Modbus/tcp as the client and use the Modbus server of the robot. This makes the PLC the master and it is responsible for getting the conveyor encoder counts to the robot in a timely fashion.

HOWEVER – If the encoder is capable of behaving as a Modbus server then the robot can be the client and there is an existing article on the UR support portal that discusses this scenario : HowTo article 18426 ; Conveyor tracking using encoder that outputs a Modbus register.

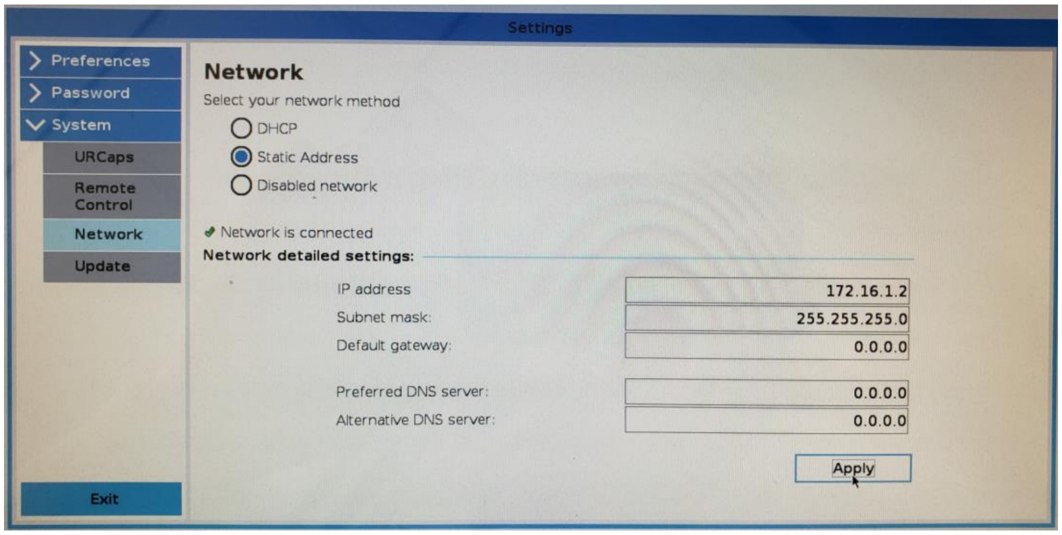
https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/conveyor-tracking-using-encoder-that-outputs-a-modbus-register-18426/

2. Quick Start (robot as Modbus client communicating to some device acting as server)

The following example shows setting up a connection to a Phoenix Contact IO block that behaves as a Modbus server. The Phoenix module is p/n 2832962 (ILB ETH 24 DI16 DIO16-2TX). This is known as an Inline Block IO module with a built-in Modbus/TCP buscoupler.

a. Polyscope Setup
   i. Network Address – The robot needs to have an IP address that is compatible with the IP address of the device you are trying to communicate with. The network address setup on the robot is shown in **section 15.5 of the Universal Robots CB3 User Manual**. For an eSeries model robot these network IP setting are in a different place within the Polyscope GUI (Settings > System > Network). When you are setting it up as a static IP (or DHCP, etc) then make sure you have an IP address and subnet mask for the robot that matches the device. As an example, if the robot has an IP address of 172.16.1.2 then the device should be something like 172.16.1.199 and they should both have a subnet mask of 255.255.255.0

Here is an example of how that network setup looks on the eseries network setup screen (notice that it indicates Network is Connected if robot is plugged into a valid hub/switch – this only indicates a valid electrical connection to an Ethernet device, not that it is communicating to anything yet) :



Network

Select your network method

DHCP [✓ Static Address] Disabled network

✓ Network is connected

Network detailed settings:

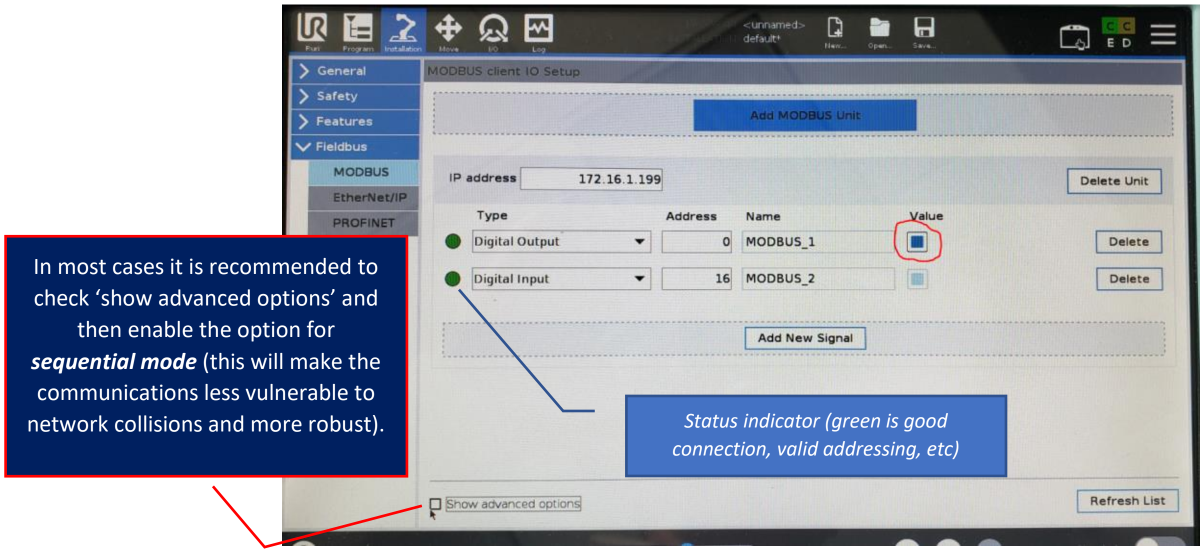
IP address	172.16.1.2
Subnet mask:	255.255.255.0
Default gateway:	0.0.0.0
Preferred DNS server:	0.0.0.0
Alternative DNS server:	0.0.0.0

For network connectivity robustness the best approach is to have the robot controller and Modbus device plugged in to an Ethernet switch/hub. In addition you can have a laptop plugged into this network and use the ping command from the PC to make sure the robot and device both have valid IP addresses.

ii. MODBUS Client I/O Setup

1. On the robot the Modbus client interface is setup within the Installation settings. In CB3 robots (Polyscope 3.x) this is accessible from Installation > Modbus Client. On the eSeries you can get to this from within Polyscope, Installation > Fieldbus > Modbus. To set this up you need to know the IP address of the device and from the device manufacturers documentation you need to understand where the data you want to access resides in terms of it's address offset.
2. As an example see the image below where we have configured a Phoenix ILB module for IP address of 172.16.1.199. For the Phoenix ILB module, when mapping to digital outputs as coils (discrete bits), the addressing starts at 0. For digital input (contacts) the addressing start at 16 (this may seem odd but for the Phoenix module this is a special case and is explained in more detail below). Every manufacturer does this differently. See the screenshot below as an example of how to map a DO (at address 0) and a DI on the Phoenix module.

Figure : Modbus client IO setup



IP address 172.16.1.199

	Type	Address	Name	Value
●	Digital Output	0	MODBUS_1	■
●	Digital Input	16	MODBUS_2	□

☐ Show advanced options

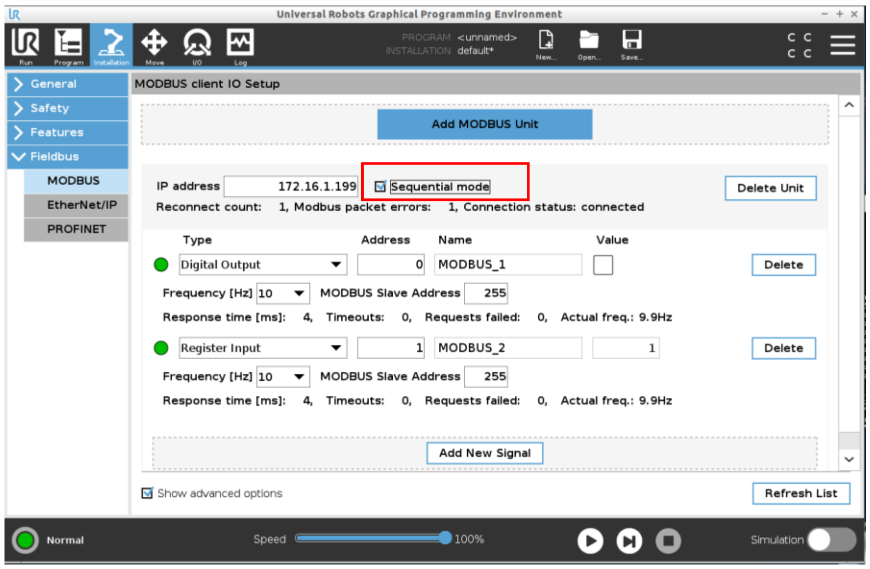
In most cases it is recommended to check 'show advanced options' and then enable the option for ***sequential mode*** (this will make the communications less vulnerable to network collisions and more robust).

Status indicator (green is good connection, valid addressing, etc)

Selecting the Value checkbox will toggle that output ON. In this example we have the DO wired to the DI so the DI comes on at the same time. The green circle to the left of the 'Digital Output' field shows the status of that signal as to whether the robots Modbus client driver is able to communicate to it or not. Green is good, gray is no connection, yellow would indicate some type of error.

To get more detailed information about Modbus client I/O setup see section 13.11 of the CB3 user manual or section 16.4.1 of the eSeries user manual. Sequential mode is described in more detail there as well as the errors you might see for signal connectivity status.

Here is a quick view of where/how sequential mode is enabled :



PROGRAM <unnamed>
INSTALLATION default*

MODBUS client IO Setup

IP address 172.16.1.199 ☑ Sequential mode

Reconnect count: 1, Modbus packet errors: 1, Connection status: connected

Type	Address	Name	Value
● Digital Output	0	MODBUS_1	☐
● Register Input	1	MODBUS_2	1

Frequency [Hz] 10, MODBUS Slave Address 255
Response time [ms]: 4, Timeouts: 0, Requests failed: 0, Actual freq.: 9.9Hz

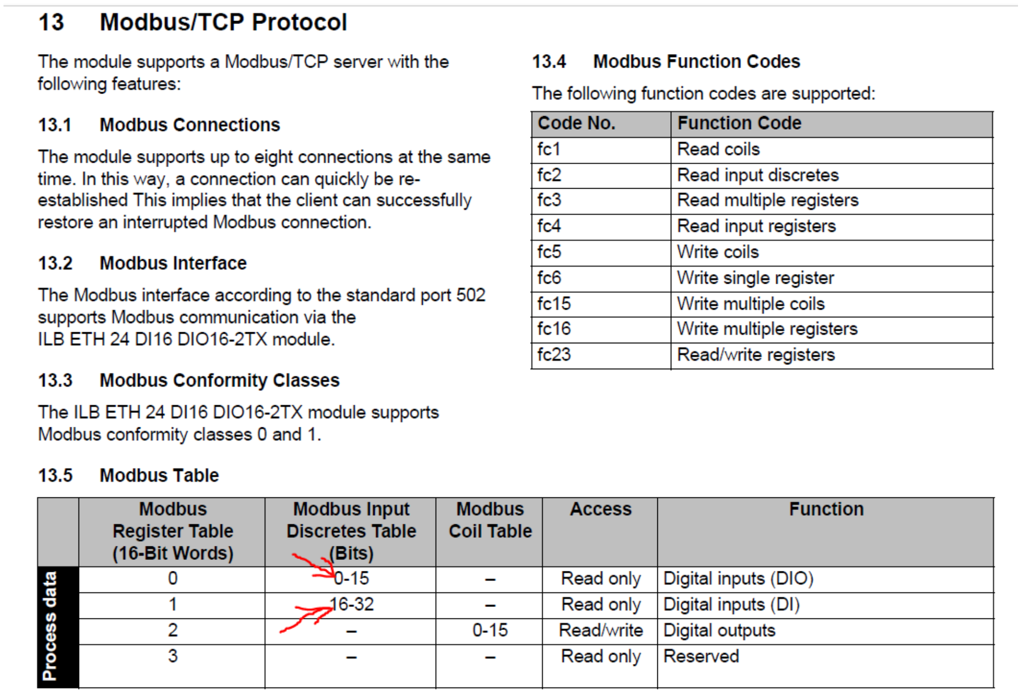
Frequency [Hz] 10, MODBUS Slave Address 255
Response time [ms]: 4, Timeouts: 0, Requests failed: 0, Actual freq.: 9.9Hz

☑ Show advanced options

Speed 100%

SPECIAL CASE OF ADDRESS MAPPING FOR PHOENIX IO MODULE

The address of IO within the Modbus table for the Phoenix module has the DI's of the Phoenix modules starting at address 16 as indicated above. This might seem strange but this information can be derived from the Phoenix documentation (7046_en_02.pdf). This module is a special case because the first 4 slots are DIO modules which means the first 16 bits are usable as DI or DO and depending on which way you are using them dictates how they should be mapped (addressed). In our example we are using the first 16 bits as DO's and therefore they start with addressing at 0 and the next 16 bits are ONLY DI's and their addressing starts at 16 according to this table from the Phoenix document :



13 Modbus/TCP Protocol

The module supports a Modbus/TCP server with the following features:

13.1 Modbus Connections

The module supports up to eight connections at the same time. In this way, a connection can quickly be re-established This implies that the client can successfully restore an interrupted Modbus connection.

13.2 Modbus Interface

The Modbus interface according to the standard port 502 supports Modbus communication via the ILB ETH 24 DI16 DIO16-2TX module.

13.3 Modbus Conformity Classes

The ILB ETH 24 DI16 DIO16-2TX module supports Modbus conformity classes 0 and 1.

13.4 Modbus Function Codes

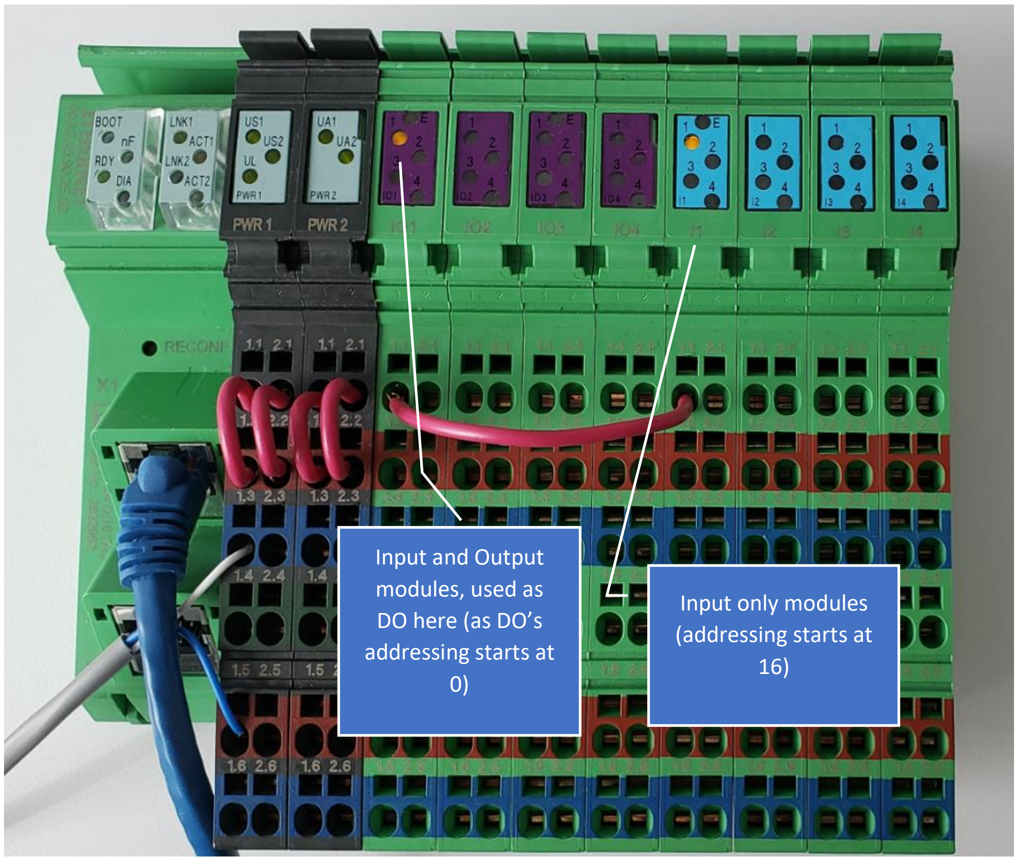
The following function codes are supported:

Code No.	Function Code
fc1	Read coils
fc2	Read input discretes
fc3	Read multiple registers
fc4	Read input registers
fc5	Write coils
fc6	Write single register
fc15	Write multiple coils
fc16	Write multiple registers
fc23	Read/write registers

13.5 Modbus Table

	Modbus Register Table (16-Bit Words)	Modbus Input Discretes Table (Bits)	Modbus Coil Table	Access	Function
Process data	0	0-15	–	Read only	Digital inputs (DIO)
	1	16-32	–	Read only	Digital inputs (DI)
	2	–	0-15	Read/write	Digital outputs
	3	–	–	Read only	Reserved

The physical Phoenix ILB module looks like this :


Input and Output modules, used as DO here (as DO's addressing starts at 0)
Input only modules (addressing starts at 16)
BOOT
RDY
nF
DIA
LNK1
ACT1
LNK2
ACT2
US1
US2
UL
UA1
UA2
PWR 1
PWR 2
RECONF

If the Modbus IO module is configured properly (watchdogs disabled, etc, more on that below) then IF you disconnect the Ethernet cable connecting the module to your network the status indicator on the left will go gray and when you reconnect the cable the status should go back to green. In other words, if the module is setup properly (as per the manufacturers instructions) it should automatically recover from most normal network faults. This should be the goal with ANY fieldbus IO modules.

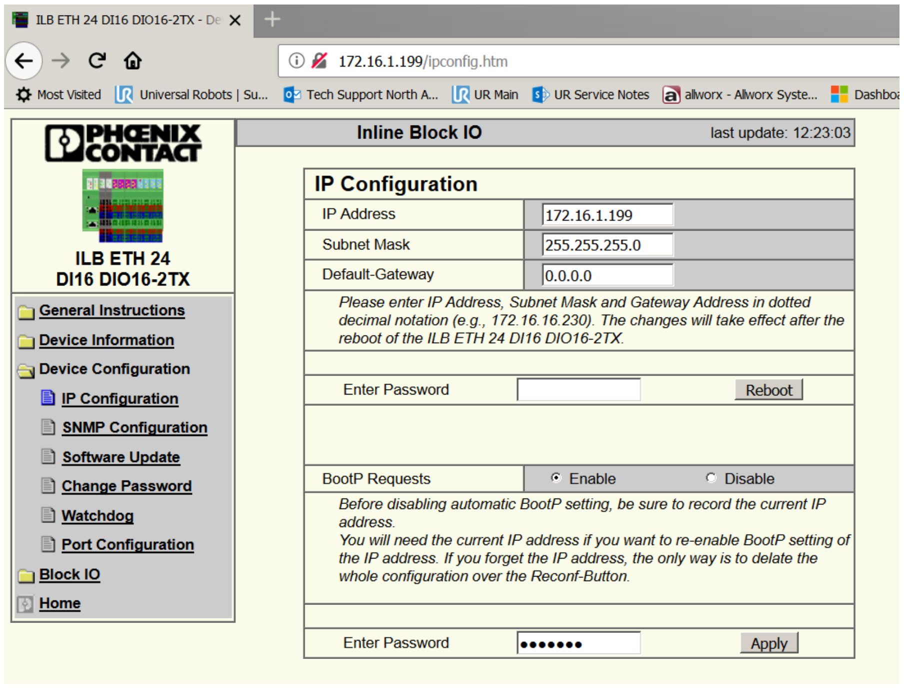
b. Modbus Device Setup (Phoenix Contact ILB as example)

Here we are discussing a specific example of setting up the Phoenix Contact ILB Modbus/TCP module (see image above). We do not go in to every detail but only try to highlight the configuration options that will ensure the most success. In general, you should always consult the manufacturers instructions for more specific details.

i. Network Setup

1. To setup the IP address of the Phoenix module for the first time you need to use a MS Windows utility from Phoenix (this utility uses a bootp protocol, it's called IP_Assign.exe and it incorporates an IP assignment wizard). Please consult the Phoenix user manual for the use of that utility. Once the IP address is setup you can get access to the configuration pages of the module. Most modules like this have a built-in web server with configuration pages that allow you to do things like disable watchdog functions (which is really useful).
2. Phoenix IP configuration

Once the Phoenix module has an IP address that is compatible with a PC on the same network, you can access the module through it's web server pages. From here you can adjust some settings such as changing some component of the IP address or the watchdog. In the image below we have opened a browser and entered the IP address of the Phoenix module (172.16.1.199) and we get this configuration page :



ILB ETH 24 DI16 DIO16-2TX

Inline Block IO — last update: 12:23:03

IP Configuration

IP Address	172.16.1.199
Subnet Mask	255.255.255.0
Default-Gateway	0.0.0.0

Please enter IP Address, Subnet Mask and Gateway Address in dotted decimal notation (e.g., 172.16.16.230). The changes will take effect after the reboot of the ILB ETH 24 DI16 DIO16-2TX.

Enter Password [ ] Reboot

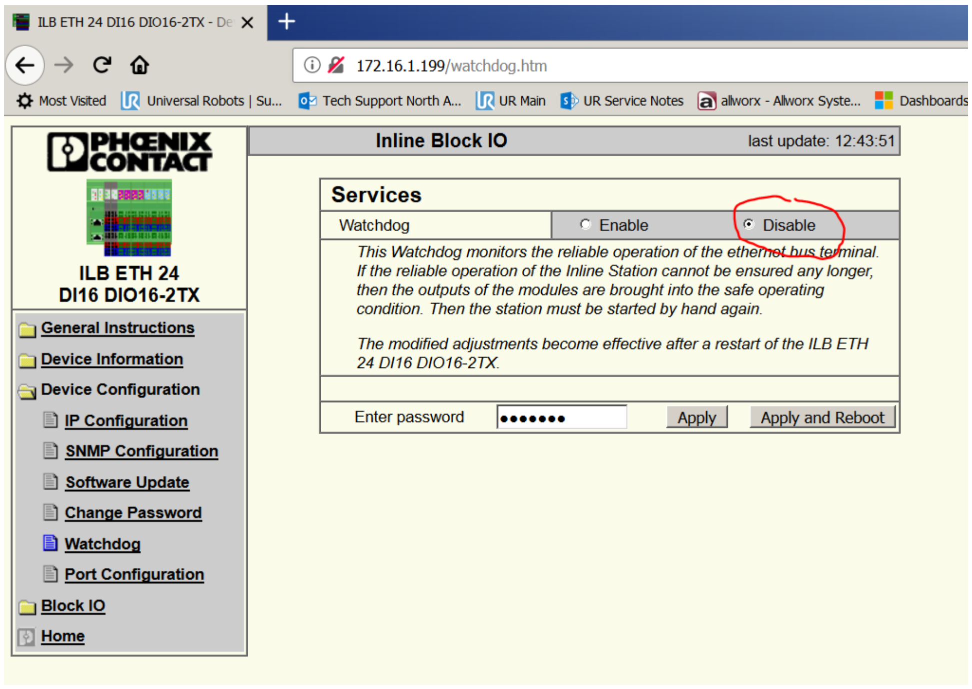
BootP Requests [✓ Enable] Disable

Before disabling automatic BootP setting, be sure to record the current IP address.
You will need the current IP address if you want to re-enable BootP setting of the IP address. If you forget the IP address, the only way is to delate the whole configuration over the Reconf-Button.

Enter Password ••••••• Apply

ii. Watchdog settings of IO module

1. Most Modbus servers (as IO modules in particular) have some type of watchdog function. The purpose of the watchdog is to give the user some way to control the failsafe state of the IO such that IF the network were to go down, or the Modbus client (the robot controller in this scenario) were to quit communicating with the module, that it would assert its digital outputs to a certain state. The problem with this watchdog functionality typically is that when the network connection is broken, and the watchdog is tripped, the only way to recover is to cycle power on the IO module (this functionality is vendor specific). This is really inconvenient for many reasons. So if your goal is to have a robust network connection from the robot controller to the IO device that will recover automatically from network anomalies then disabling the watchdog of the IO module is a good idea. Here is how that is done on the Phoenix ILB module :



Inline Block IO last update: 12:43:51

Services

Watchdog: Enable [✓ Disable]

This Watchdog monitors the reliable operation of the ethernet bus terminal. If the reliable operation of the Inline Station cannot be ensured any longer, then the outputs of the modules are brought into the safe operating condition. Then the station must be started by hand again.

The modified adjustments become effective after a restart of the ILB ETH 24 DI16 DIO16-2TX.

Enter password: •••••••

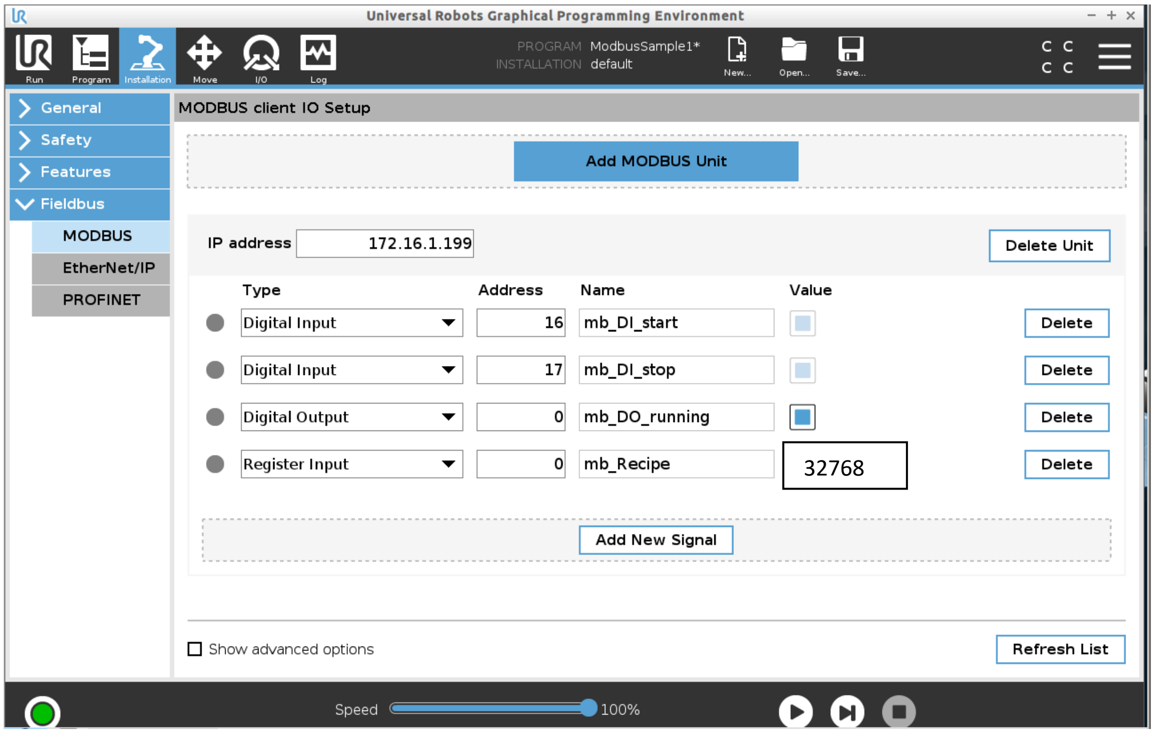
After disabling the watchdog you need to Apply and Reboot to save this setting in non-volatile storage on the module so it is retained after a power cycle of the module.

************** END OF QUICK START SECTION **************

3. Modbus signals and variables

When we setup Modbus modules in the Modbus client IO setup section within the Installation settings Polyscope will generate some URscript functions for us (transparent to the programmer) that read and write to these signals. These Polyscope functions (and the Modbus/TCP client driver of the robot) are only capable of reading/writing to Boolean data (bits or coils) and register data (16 bit to be specific). You cannot communicate with Modbus devices that incorporate string data (ascii) or floating point data, the driver does not support this.

a. DIGITAL INPUTS and OUTPUTS - The Modbus client IO setup allows you to read/write digital signals (Boolean). Once these are mapped in the client IO setup Polyscope creates the variable that gets mapped to that signal. In the case of a Modbus digital input that variable will be a Boolean and therefore it can only be True or False (it can contain no other value). In the image below mb_DI_start is an example of such a signal. If the signal is an output then it can only be set via the SET command to Hi (ON) or Low (OFF).
b. REGISTER INPUTS and OUTPUTS - The Modbus register data types that the Modbus client can access are limited to 16 bit registers. The UR controller cannot deal with 32 bit register data. Assuming the Modbus server that the robots client is connected to is serving up 16 bit register data then in the image below we have a register input mapped to mb_Recipe. This variable will contain a 16 bit unsigned integer value. So if we read in a value where the MSB of this word is set such as 1000 0000 0000 0000 (binary) we will see a value of 32768 stored in the variable mb_Recipe.



IP address 172.16.1.199

Type	Address	Name	Value
Digital Input	16	mb_DI_start	☐
Digital Input	17	mb_DI_stop	☐
Digital Output	0	mb_DO_running	■
Register Input	0	mb_Recipe	32768

The equivalent URscript functions that gets called from this setup (what gets contributed to the overall program_name.script file from the installation) are documented in the script manual. Those URscript functions are :

Modbus_add_signal()

Modbus_delete_signal()

Modbus_get_signal_status()

Modbus_send_custom_command()

Modbus_set_digital_input_action()

Modbus_set_output_register()

Modbus_set_signal_update_frequency()

See the *URscript Programming Language* manual to learn more about these functions and their parameters. Of course you need to be careful if using these functions in a program directly and setting up configurations within the 'MODBUS Client IO Setup' screen of the installation as conflicts could arise.

4. Using Modbus signals (variables) in a robot program

When you create a modbus client module and have a signal name defined Polyscope will allow you to use that signal or variable within a program. In the example above, we have various signal names defined and mapped to some external server :

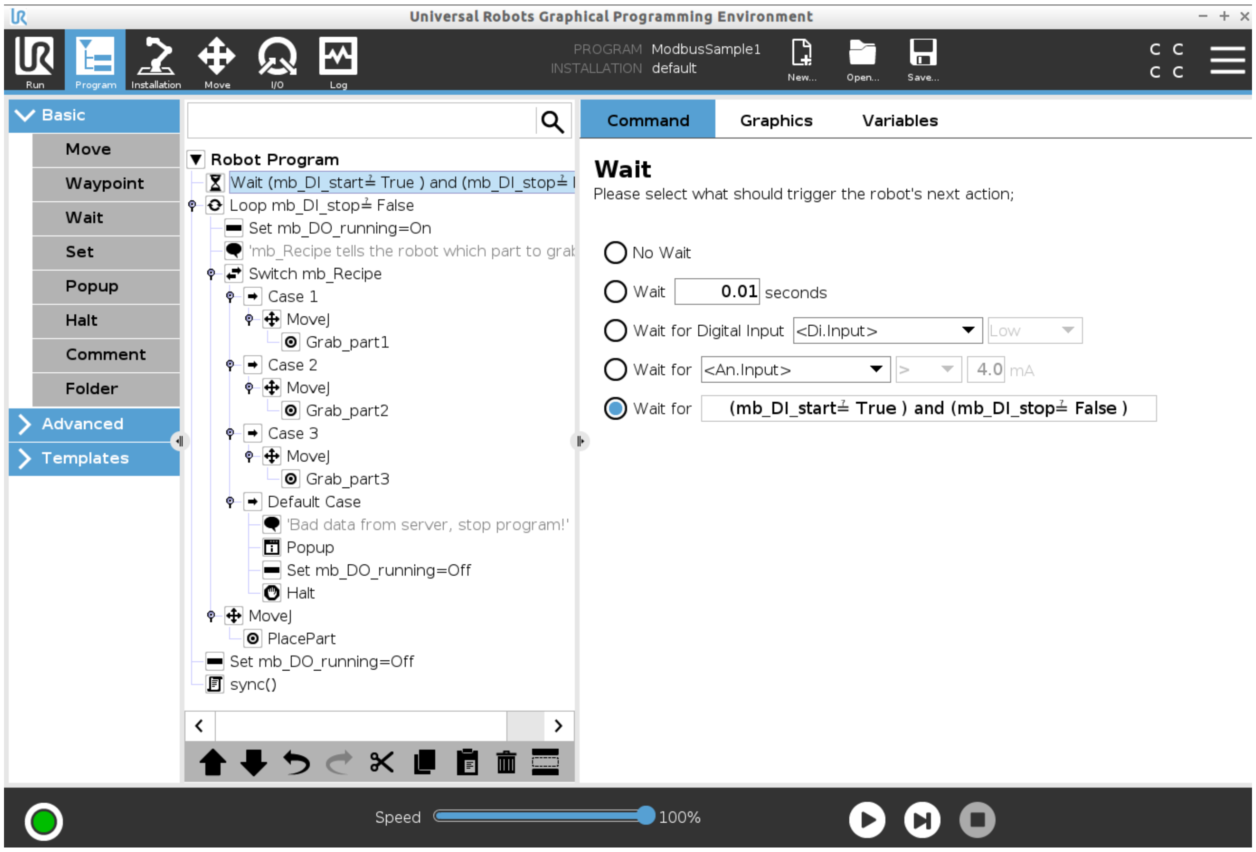
mb_DI_start ; is a digital input telling the program to start

mb_DI_stop ; is a DI telling program to stop

mb_DO_running ; is a DO telling the external equipment that the program is running some sequence

mb_Recipe ; is a register input that tells the program what 'recipe' to run. In other words, which part to pick up and place somewhere.

Following is an example of how all these signals might be used in a program that is driven by these signals from an external source (perhaps a PLC with a Modbus server module).



```
Robot Program
  Wait (mb_DI_start≟ True ) and (mb_DI_stop≟ I
  Loop mb_DI_stop≟ False
    Set mb_DO_running=On
    'mb_Recipe tells the robot which part to grat
    Switch mb_Recipe
      Case 1
        MoveJ
          Grab_part1
      Case 2
        MoveJ
          Grab_part2
      Case 3
        MoveJ
          Grab_part3
      Default Case
        'Bad data from server, stop program!'
        Popup
        Set mb_DO_running=Off
        Halt
    MoveJ
      PlacePart
  Set mb_DO_running=Off
  sync()
```

Wait

Please select what should trigger the robot's next action;

- No Wait
- Wait 0.01 seconds
- Wait for Digital Input <DI.Input> Low
- Wait for <An.Input> > 4.0 mA
- [✓ Wait for (mb_DI_start≟ True) and (mb_DI_stop≟ False)]

So, the Modbus signals can be used in many different Polyscope commands such as :

- Wait (waiting for digitals or analogs/register inputs)
- Set (setting digital outputs to ON or OFF, setting analog outputs)
- If-Then-Else expression
- Loop
- Assignment (as long as you assign to a similar variable type)
- Switch-Case

5. Accessing bits from script commands

If you had a Modbus register being read by the robot and wanted to access the bit level data within that register (example – to determine if a particular bit was set) there are URscript functions for doing this. If you wanted to read the MSB from the variable mb_Recipe (shown above) then that function would look like this :

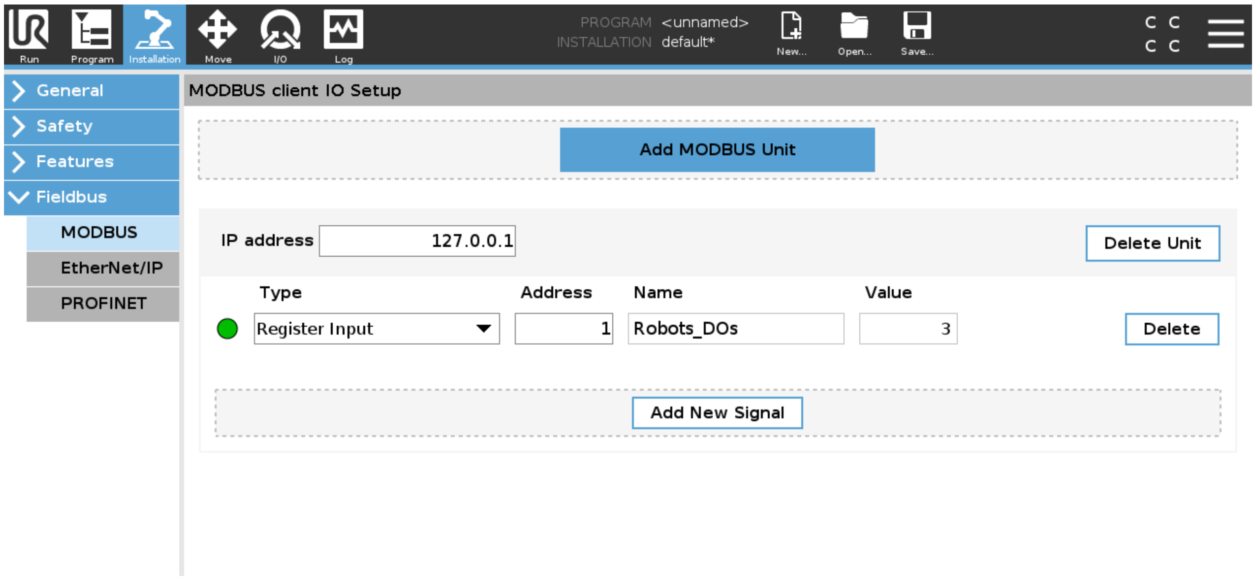
Bin_list:=integer_to_binary_list(mb_Recipe) << using an Assignment command >>

And then we could evaluate 'Bin_list[0] =? True' to determine if that bit was really set.

There is a corresponding function called binary_list_to_integer() which will convert the values in a binary list back to an integer to it can be written back out to a device via a Modbus output register.

6. Using Internal Modbus client to access robots server registers and bits

Any data/status produced by the robots own Modbus server can be accessed by a program running on the robot by using the Modbus client module with an IP address of 127.0.0.1 (the 'loopback address'). As an example, if you wanted to read the state of ALL of the robots physical digital outputs into a register this is what that configuration would look like (in this case DO_0 and DO_1 are set therefore this is an integer value of 3) :



MODBUS client IO Setup

IP address 127.0.0.1

Type	Address	Name	Value
Register Input	1	Robots_DOs	3

7. Modbus Specifications

a. Register update rate

Whatever you set it to, not guaranteed to be updated at that rate necessarily since Modbus communications are non-deterministic and lower priority in the robot controller.

b. Addressing

i. 0-based

ii. Unsigned integer

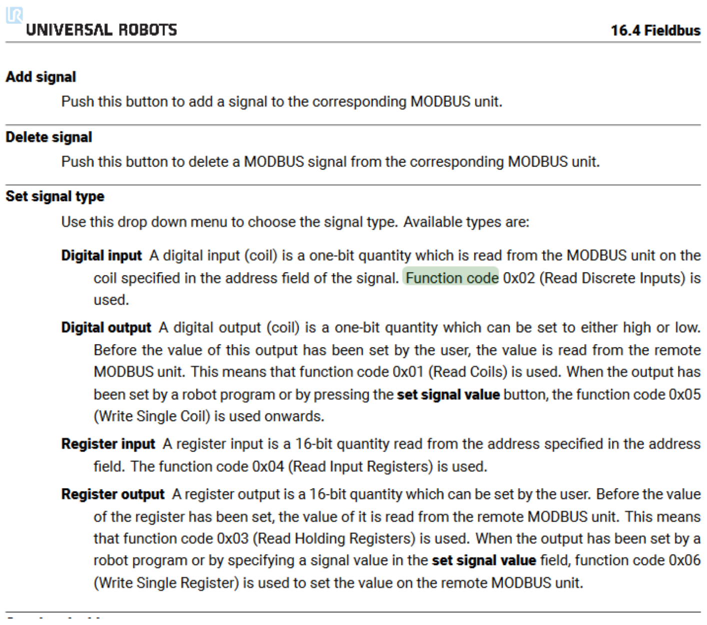
iii. Indexing - Some MODBUS units use designated addresses for each function code, i.e. Siemens use: coils (generally addressed as 0xxxx), contacts (1xxxx), input registers (3xxxx), holding registers (4xxxx). To solve this issue, address "x" is reflected at 10000+"x", 20000+"x", 30000+"x" and 40000+"x" Address 0 is also reflected at address 9999, as some units have an addresses starting at 1

8. Modbus functions available

The Modbus client interface supports Function Codes 0x01 through 0x06. The function code is a low level code that is embedded within the protocol and sent to a server. Below is a description of what these function codes mean and do within the Modbus protocol.

a. 0x01: READ_COILS (read output bits)
b. 0x02: READ_DISCRETE_INPUTS (read input bits)
c. 0x03: READ_HOLDING_REGISTERS (read output registers)
d. 0x04: READ_INPUT_REGISTERS (read input registers)
e. 0x05: WRITE_SINGLE_COIL (write output bit)
f. 0x06: WRITE_SINGLE_REGISTER(write output register)

For more detail on this see section 16.4 of the URx Software Manual. Here is an excerpt:



UNIVERSAL ROBOTS — 16.4 Fieldbus

Add signal
Push this button to add a signal to the corresponding MODBUS unit.

Delete signal
Push this button to delete a MODBUS signal from the corresponding MODBUS unit.

Set signal type
Use this drop down menu to choose the signal type. Available types are:

Digital input A digital input (coil) is a one-bit quantity which is read from the MODBUS unit on the coil specified in the address field of the signal. Function code 0x02 (Read Discrete Inputs) is used.

Digital output A digital output (coil) is a one-bit quantity which can be set to either high or low. Before the value of this output has been set by the user, the value is read from the remote MODBUS unit. This means that function code 0x01 (Read Coils) is used. When the output has been set by a robot program or by pressing the **set signal value** button, the function code 0x05 (Write Single Coil) is used onwards.

Register input A register input is a 16-bit quantity read from the address specified in the address field. The function code 0x04 (Read Input Registers) is used.

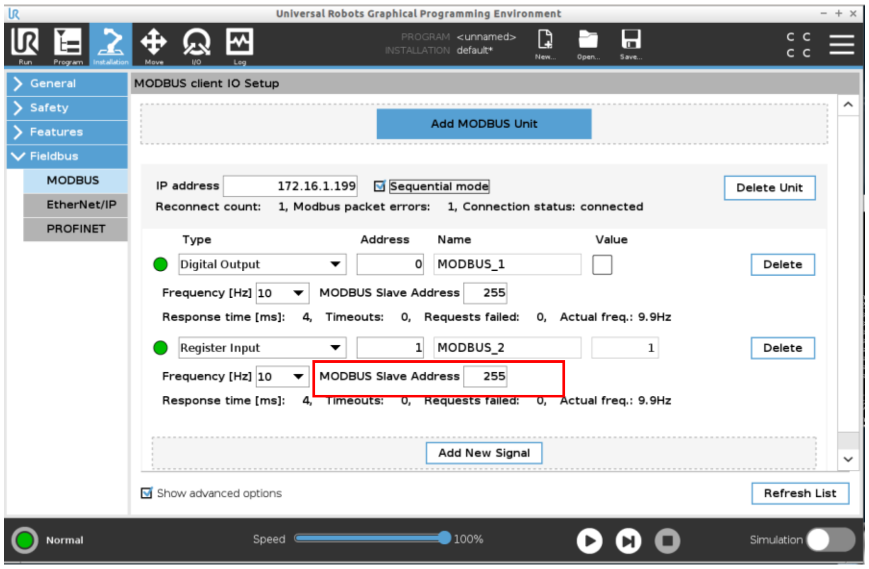
Register output A register output is a 16-bit quantity which can be set by the user. Before the value of the register has been set, the value of it is read from the remote MODBUS unit. This means that function code 0x03 (Read Holding Registers) is used. When the output has been set by a robot program or by specifying a signal value in the **set signal value** field, function code 0x06 (Write Single Register) is used to set the value on the remote MODBUS unit.

9. Modbus error message with exception codes

a) E1 ILLEGAL FUNCTION (0x01) The function code received in the query is not an allowable action for the server (or slave).
b) E2 ILLEGAL DATA ADDRESS (0x02) The function code received in the query is not an allowable action for the server (or slave), check that the entered signal address corresponds to the setup of the remote MODBUS server.
c) E3 ILLEGALDATAVALUE (0x03) A value contained in the query data field is not an allowable value for server (or slave), check that the entered signal value is valid for the specified address on the remote MODBUS server.
d) E4 SLAVE DEVICE FAILURE (0x04) An unrecoverable error occurred while the server (or slave) was attempting to perform the requested action.
e) E5 ACKNOWLEDGE (0x05) Specialized use in conjunction with programming commands sent to the remote MODBUS unit.
f) E6 SLAVE DEVICE BUSY (0x06) Specialized use in conjunction with programming commands sent to the remote MODBUS unit, the slave (server) is not able to respond now.

10. Terminology

a. Modbus TCP – Protocol built upon a TCP/IP stack that implements the Modbus communication protocol on Ethernet.
b. Server – Servers are passive software components that 'serve data requests' to clients. Servers do no initiate communications unsolicited. Server = slave.
c. Client – The client is also known as the master and it is the software component in the communications architecture that is soliciting data from the server. Client = master.
d. Coil – A coil refers to a traditional PLC relay coil but it basically equates in modern day terminology/usage to a digital output.
e. Contact – refers to a digital input.
f. Register – Integer data, usually used to represent analog data or a counter from an encoder or drive.
g. Bit – Could be a digital output or input or some binary element of a word (16 bits) data type.
h. Read – The Modbus client is typically reading input contacts from some device.
i. Write – The Modbus client is typically writing to a coil or register that represents some output data type.
j. Slave ID – Each slave in a network (slaves are under control of the master/client) has a unique unit address from 1 to 247. For most devices under Modbus TCP the slave ID is not used and if that is the case then it is set to 255. The UR Modbus TCP client driver does this by default but it also gives you the ability to change it if needed (see below).



MODBUS client IO Setup

IP address 172.16.1.199 ☑ Sequential mode
Reconnect count: 1, Modbus packet errors: 1, Connection status: connected

Type	Address	Name	Value
● Digital Output	0	MODBUS_1	☐
● Register Input	1	MODBUS_2	1

Frequency [Hz] 10 MODBUS Slave Address 255
Response time [ms]: 4, Timeouts: 0, Requests failed: 0, Actual freq.: 9.9Hz

Frequency [Hz] 10 MODBUS Slave Address 255
Response time [ms]: 4, Timeouts: 0, Requests failed: 0, Actual freq.: 9.9Hz

☑ Show advanced options

100%

11. Related Articles

k. Using a barcode reader directly with the robot – 19450
l. Data Source Matrix For Remote Access – 33474
m. Conveyor tracking with wizard – 21733
n. Switch case – 18861
o. Modbus Register and Digital I/O - 19252
p. Modbus TCP server article - 16377