emBRICK

emBRICK® - EPC

CouplingBrick Starterkit-1
Remote Bus Coupling via …
MSVC, CODESYS, LabVIEW, Gamma
Python, Node-RED

Starter Kit - Remote-Bus

Rev. 14

emBRICK® is developed and supported by

IMACS GmbH
Alfred-Nobel-Straße 2
D – 55411 Bingen am Rhein
www.imacs-gmbh.com
www.embrick.de

support@embrick.de
Hotline: +49 (0) 7154 80 83 - 15

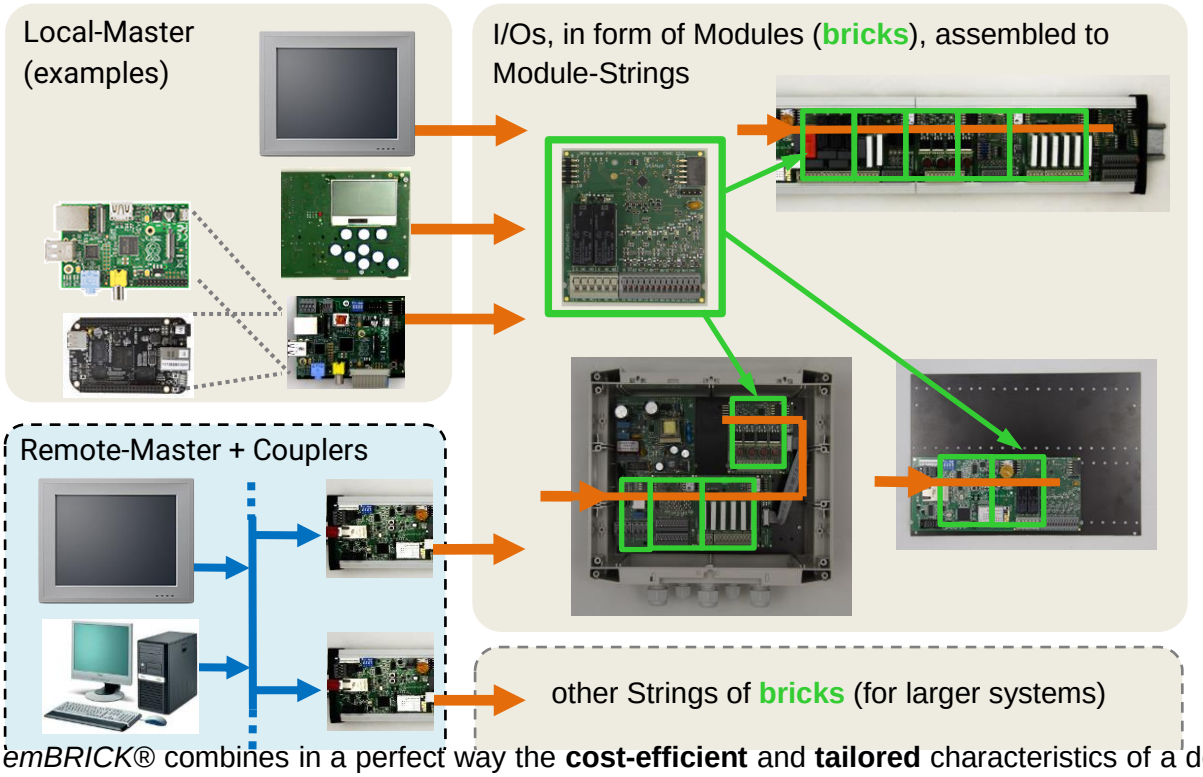
Content

1 The emBRICK® Mission

The mission of *emBRICK*® is an **open** and **free** I/O system to ...

> build **compact** and **industrial suited** electronic **control systems**
> by **assembling** small **existing/own** embedded **boards** (**bricks**) ...

... via a SPI-based **local interface** and optional **remote buses** (LAN, WLAN, CAN, RSxxx, ...).
We call this new class of controllers simple **EPC** (= **E**mbedded **P**atch-board **C**ontroller).



emBRICK® combines in a perfect way the **cost-efficient** and **tailored** characteristics of a dedicated embedded system with the **ready to use** and **flexibility** of a PLC system.

To ensure a high acceptance, it is an open and free system. I.e. besides buying existing devices, everyone can develop his own components to realize easily his individually tailored, cost-efficient and industrial-suited measure and control system.

1.1 Typical Applications

- Small, medium and large size **measure and control systems**
- **Sectoral purpose**, with direct sensor/actor interface
- **Autonomous single box** control solutions i.e. with HMI and communication interfaces
- **Rapid hardware prototyping** system for control and measuring applications
- **PLC replacement** (i.e. with a Soft-PLC, IPC or an embedded controller)
- Medium and large size **distributed IO-systems** (i.e. building automation)
- Physical front-end for **IoT** (Internet of Things)

For more details see *Product_Catalogue (eB_Products)* and *Application_Manual (eB_Applications)*.

1.2 Basic Characteristics

- **free** - also for commercial use in own appliances (for pure EMS with a license fee)
- **open** - supplying reference schematics, protocol source code, samples and starter kits
- **adaptable** to all systems, using common, low cost standard µCs/components
- **half ... third price** compared to common control systems (complete system view)
- scalable local and remote topologies, **1 ... >1000 I/Os**, up to **1ms update**, deterministic
- **low own power** consumption, average 50mW/slave module in operation (outputs inactive)
- global and sector specific modules for **direct connection** of various **sensors and actors**
- **easy installation**, no configuration necessary, simple plug modules together and use
- works with / programmable by **various established**, well known **platforms / languages**

1.3 Available Hardware Products

Beside own developments, currently the following components are available from IMACS:

Slave-Modules	> 50 different modules for the sectors: General Purpose, Building Automation, Process Control (Safety, Medical/Analytics planed)
Master boards	Core: Cortex-M3/4, ARM9/11, PIC24/32; HMI: 128x64 ... WVGA
Adaption boards	for LAN, WLAN, CAN, RSxxx, Raspberry Pi, Beaglebone Black
Appliances / Enclosures	ready Single Box Controller for and top-het rail and wall mounting
Starterkits	for MSVC, CODESYS, Raspberry Pi, Beaglebone Black

1.4 Available Host Platforms, Connectivity

emBRICK® can be adapted to all platforms with almost every footprint/performance. For master units, currently the following system implementations are available (others planed):

Computer platforms	PC, Embedded-PC, Module-PC, Raspberry Pi, Beaglebone Black
µController platforms	ARM-Ax, ARM-Cortex-Mx, Microchip PIC24 / PIC32
Host Interfaces	Ethernet, CAN, RS232, RS485
Wireless Interfaces	WLAN

1.5 Available Programming Platforms

emBRICK® can be programmed by various systems, languages and IDEs (integrated development interface). Currently for master units the following systems are available (others planed):

OS / RTOS	Windows, Linux, FreeRTOS, proprietary
Programming languages	C, C++, IEC61131, Model-based (by implementing UML)
Model-based / Soft-PLC	CODESYS, radCASE, Enterprise Architect
C/C++ IDEs	MSVC, Cocox (GCC), MPLab (Microchip), Geany (Raspberry Pi), every other C/C++ IDE

-modules, actual Microchip PIC16/24 is used. Others (i.e. Cortex-M0) are planned.

2 Introduction

2.1 About this Manual

This manual leads systematically from the hardware mounting and software installation, to start up the emBRICK® adapter starter kit, running the delivered sample application and create your own applications.

Furthermore, it is used as an open reference platform for the brickBUS® local-master protocol stack.

2.2 References / Manual Overview

For *emBRICK*® and *brickBUS*® the following documents are available. Before reading this document, it is recommended to read them in the given order:

System Manual	(*eB_System.pdf)* ... the basic system manual that contains the idea, the intention and the basic technical concept of *emBRICK*®/ *brickBUS*® like mechanics, electronics and communication protocol. It includes the glossary for all other documents.
Application Examples	(*eB_Applications.pdf)* ... overview of typical *emBRICK*® device configurations and sample constellations for different industrial applications. It gives an idea how to use *emBRICK*® as an alternative to a normal PLC or an individual PCB / embedded system.
Product Catalogue	(*eB_Products.pdf)* ... contains the overviews and detailed datasheets of all IMACS-available *emBRICK*® components and products. This includes electrical and mechanical characteristics, terminal assignment and notes about their usage.
Programmers Manual	(*eB_Programmer.pdf)* ... is the manual for application software programmers when using established programing systems like Embedded-IDEs, Soft-PLCs, CASE-Tools but also native C/C++-coding.
FAQ Manual	(*eB_FAQs.pdf)* ... contains answers to the most frequently asked questions about *emBRICK*® and its usage.
Developers Manual	is the manual for system developers, who like to create their own slave modules or master adaptions. It includes all technical details specifications of *brickBUS*® and also sample schematics and code samples of the software stacks. This document is only available on request from IMACS GmbH and needs the agreement on the *emBRICK*® free license conditions. Please contact support@embrick.de.

2.3 Homepage

The official website is www.embrick.de. It will contain the upper named manuals, detailed datasheets and drawings, third party components, distributors, FAQs, schematics and source code, license condition for EMS.

2.4 Forum

Planed

2.5 Roadmap

Currently the following products/enhancements are planed (partners are welcome):

- Wireless Connections
- PoE coupled master for more compact building solutions
- emBRICK-LE, a low energy version with different sleep-mode
- additional mechanical module formats like a box to clip on a top-hat rail

2.6 Package contend

The starter kit CouplingBrick Starterkit-1 contains:

- *Coupling-Master* (Z-CouplingBrick-02)
- *Slave-module* (CAE_P-2Rel4Di2Ai-01)
- Carrier Board (CAE_Y-CHBoc200)

Furthermore, via download from the internet:

- Different board support packages (BSP) with sample application (see The Software)
- Other *emBRICK*® manuals (see References / Manual Overview)

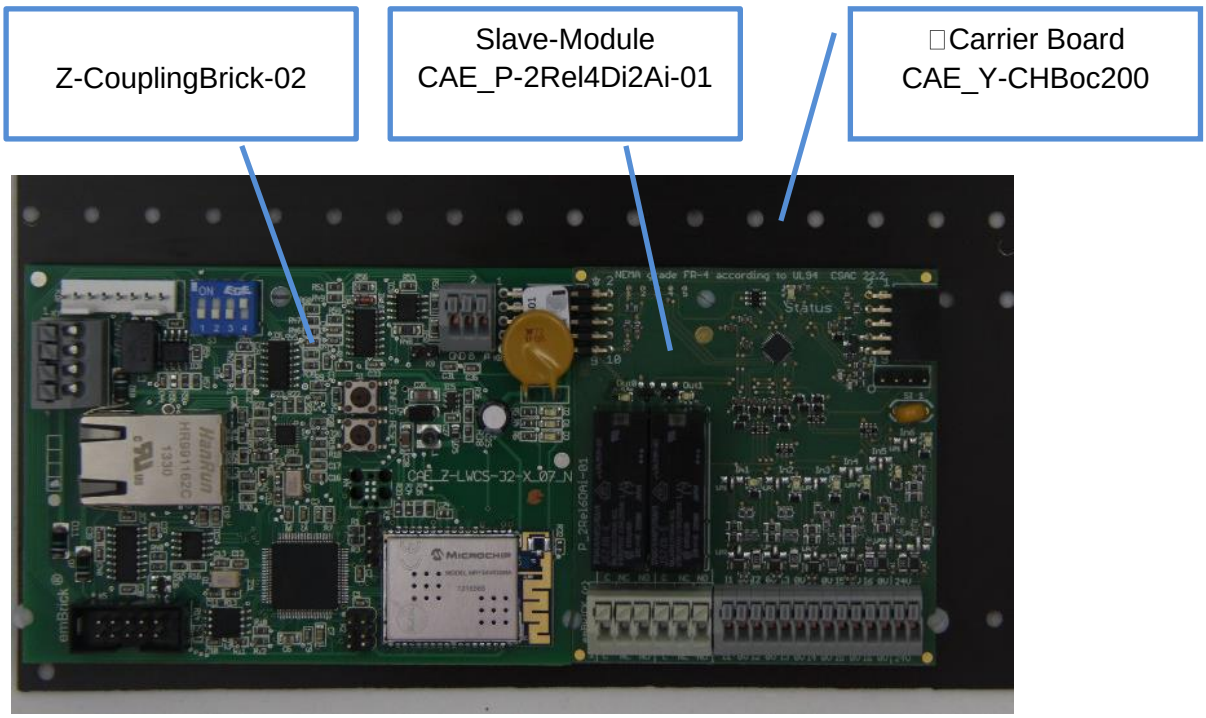
2.7 Separate required components

To operate the starter kit, following separate items are required:

- Computer with Windows XP/7/8/10.
- DC power supply 24V, > 500mA.
- Network cable to connect the *coupling-master* with the PC (no crossover cable).
- Some electronic components and wires for own experiments (if needed).
- Recommended: A second network adapter or a USB to LAN adapter, to keep your primary LAN adapter free for normal use.

2.8 The Hardware

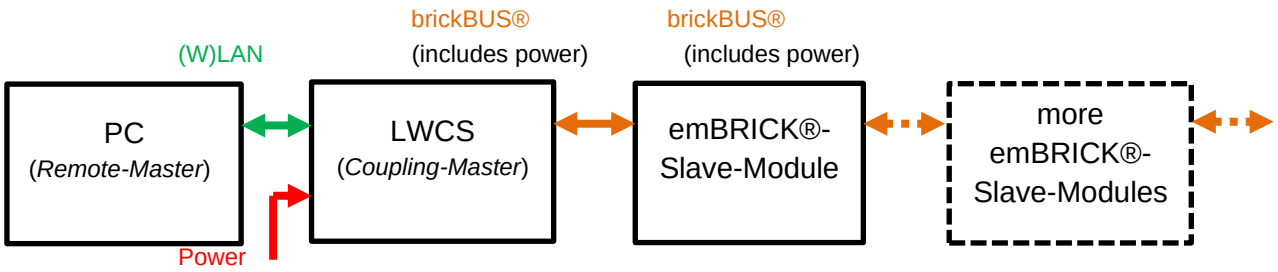
The *coupling-master* is only a coupler between a PC (and other hosts) and an emBRICK®-*String*. The *String* consists of one or several *slave-modules*. *Slave-modules* receives the commands/data from the *brickBUS*® and controls their I/Os.



picture 1

2.8.1 Communication structure

The application on the PC is the *remote-master* that sends/receives commands/data to the *coupling-master* via LAN. The *coupling-master* is only a coupler. It translates the received data from LAN to a *String* of slave-modules. Each *slave-module* receives the translated commands via *brickBUS*® and controls the I/Os.



For a detail information, please read the description of the modules in the System Manual.

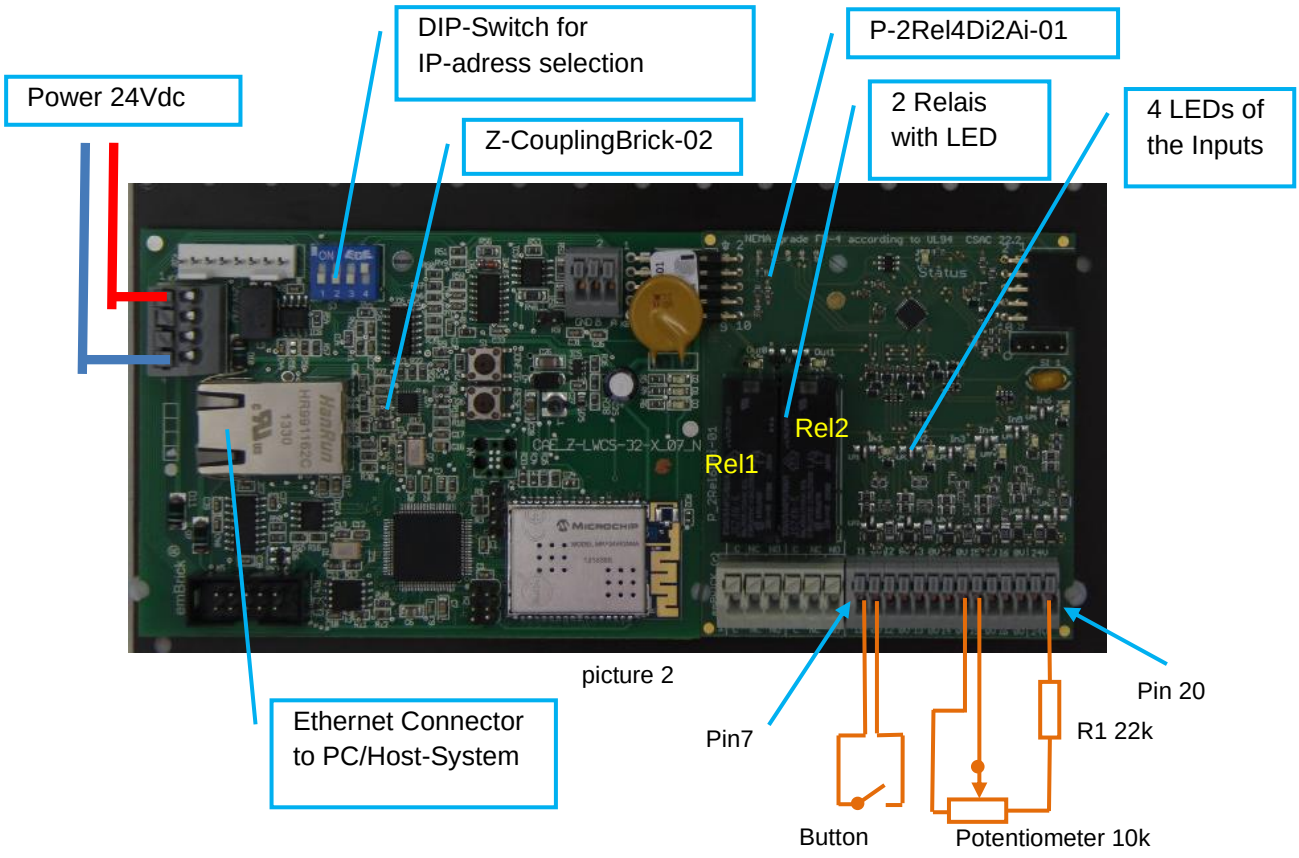
2.9 The Software

For this starter kit diverse Board Support Packages (BSP) for different software platforms/IDE's are available. They include all drivers and all libraries in source code.

TDB_eB-STK-C1-MSVC	for PC (Windows / MAC) with MSVC
TDB_eB-STK-C1-CODESYS	for PC (Windows / Linux) with CODESYS
TDB_eB-STK-C1-LabVIEW	for PC (Windows / MAC / Linux) with NI LabVIEW
TDB_eB-STK-C1-Gamma	for PC with Gamma (in realization)

3 Mounting and wiring

Please connect the *coupling-master* to the *slave-module* in the order shown in the picture below. For more detailed information about the components itself (terminal assignment, electrical data etc.) refer to the Product Catalogue.



picture 2

1. Connect the *coupling-master* (Z-CouplingBrick-02) with the *slave-module* (P-2Rel4Di2Ai-01) in the shown order.
2. Mount the boards onto the carrier board.
3. Connect the 24Vdc and the network cable to the *coupling-master* (see *Product Catalogue*, search for “Z-CouplingBrick-02 and “P-2Rel4Di2Ai-01”). Plug the slave-module into the carrier board.

Set only the fourth DIP-Switch ON, the others have to be OFF. So you have set the *coupling-master* to the IP address 192.168.3.10. Connect it to your PC via LAN cable and configure it as specified here “0

4. Configure your PC Network Adapter". Other possible IPs for the *coupling-master* documented in the *Product Catalogue* search for "Z-CouplingBrick-##".
5. Recommended: With three additional electronic parts, you can test the starter kit functionality.
   a) Connect a potentiometer (10kOhm) via a series resistor (22kOhm) on pin 14 (ground), pin 15 (analog input), pin 19 (24V).
   b) Connect a button on I/Os pin 7 (digital input) and pin 8 (ground).

Result: Now the hardware is mounted and wired to start the installation of software.

3.1 Configure your PC Network Adapter

In this step you prepare your PC to communicate via LAN with the coupling-master. For this we recommend to use a separate network card or an USB-Ethernet adapter (to keep your primary LAN connection untouched).

To setup IP-address of the LAN-Adapter follow these steps (for Windows 7):

-> in Control Panel

-> Network and Internet –> Network and Sharing Centre –> Change adapter settings

-> Choose your network connection (click right mouse button) –> Properties

-> Internet Protocol Version 4 (TCP/IPv4) (double click)

-> check box „Use the following IP address: “

-> enter the *IP address* (i.e. 192.168.3.250) and the *Subnet mask* 255.255.255.0

This configuration will be later tested with a tool.

Now you can start with the software package you prefer:

-> jump to chapter 4 Hands on Software - with MSVC

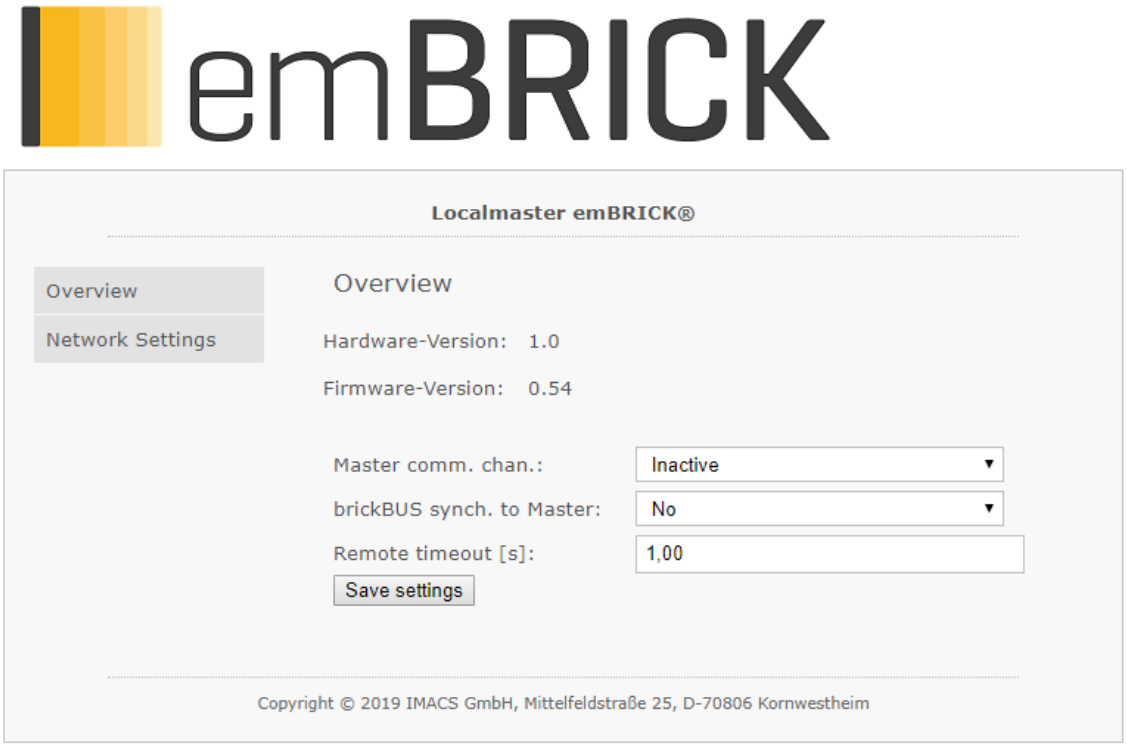
-> jump to chapter 5 Hands on Software - with CODESYS

3.2 Main configuration with integratet webpage

CouplingBricks (patBridge / uniBridge and airBridge) with preinstalled software versions 0.53 and later have an integrated webinterface, on which you can configure main settings of the brick.

To open the configuration page, you must connect your CouplingBrick with your computer like explained in the previous chapter a call in a webbrowser the IP-address you have set up. The pre-configured IP-address is 192.168.3.10.

When the page loads up in your browser you will see the following overview:



Before you can communicate with the remote Target, you must set the “Master comm. Chan.”. If you want to use the CouplingBrick over an ethernet communication, you will change the “Master comm. Chan.” To “LAN/WLAN”. If you want to communicate over Modbus Large Block (RTU or

TCP), you will set up the setting to “ModBUS LB”. After changing the communication channel, you must restart your system. Now you can communicate with your remote Master.

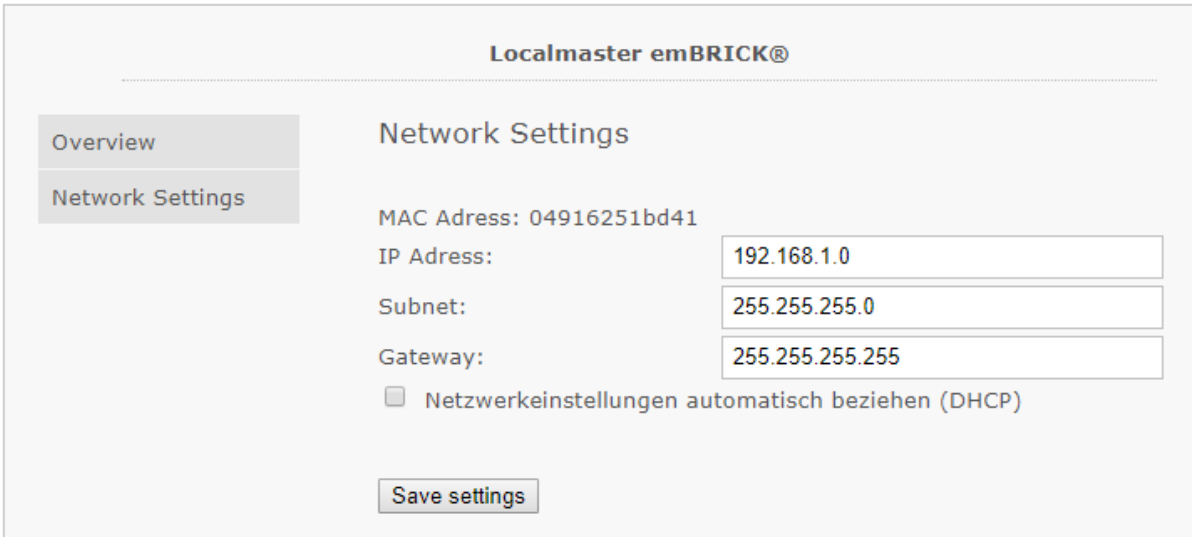
You can also change with “brickBUS synch. To Master” whether your CouplingBrick will update the brickBUS synchronous or asynchrosous with the incoming data from the remote master. “brickBUS synch. To Master: No” means, that the brickBUS will be updated asynchronous and continguous.

With “Remote timeout” you can set the delay time of the incoming data between the remote a local master. When data arrived outside the timeout, the CouplingBrick will signal this with an error code (blink code with the three LEDs).

3.2.1 Network settings

3.2.1.1 Switching IP Address with PC-visualization

You can set the IP address of the CouplingBrick in the menue “Network Settings”.



The IP Adress is set after a restart of the coupling master. Before you do this, set the DIP-switches to “0001”. If you don´t set them to the position “0001”, the IP Adress of the DIP-switch is set. You can reboot the system by clicking on the onboard reset button.

3.2.1.2 Switching IP Adress with DIP-switch

The DIP-switch position determines a value between 0..15 as: Sw1 + Sw2 x2 + Sw3 x4 + Sw4 x8.

The standard position of the onboard DIP-switch is “1000” (DIP Switch value 8). In this case the IP 192.168.3.10 is set to your coupling master.

Usage during power on:
DIP switch value 0: .....use DHCP
DIP switch value 1: ……. IP-address set in the Visualisation
DIP switch value 2 - 7: .....unused
DIP switch value 8 ... 15: .....determines fix IP-address (192.168.3.10 ... 17)

Here is a list of the switch positions and the resulting IP-addresses:

Switch-positions	DIP-switches value in the VIS	IP-Address
0000	0	DHCP
0001	1	Software set Adress
1000	8	192.168.3.10
1001	9	192.168.3.11
1010	10	192.168.3.12
1011	11	192.168.3.13
1100	12	192.168.3.14
1101	13	192.168.3.15
1110	14	192.168.3.16
1111	15	192.168.3.17

4 Hands on Software - with MSVC

4.1 Setup the Development Environment MSVC-Express

To write and compile your own applications you need a C/C++ IDE (integrated development environment). Here we use the free Express Version of VisualStudio 2010 (or newer).

Please download the official free express version here ...

http://www.visualstudio.com/downloads/download-visual-studio-vs

... and follow the given instructions

Result: Now you can create and edit applications and compile them.

4.2 Download the Board Support Package

In this step the board support package "Z-CouplingBrick_MSVC_V#" will be downloaded from the web and unziped.

1. Download the software part Z-CouplingBrick_MSVC_V#.zip here.
2. Unzip the Z-CouplingBrick_MSVC_V#zip into any folder on your PC.

The BSP contains also the tool *NetBRICK* to explore the LAN environment and search for connected *coupling-masters* (LWCS-Boards).

Result: Now you are ready to compile and start the sample application.

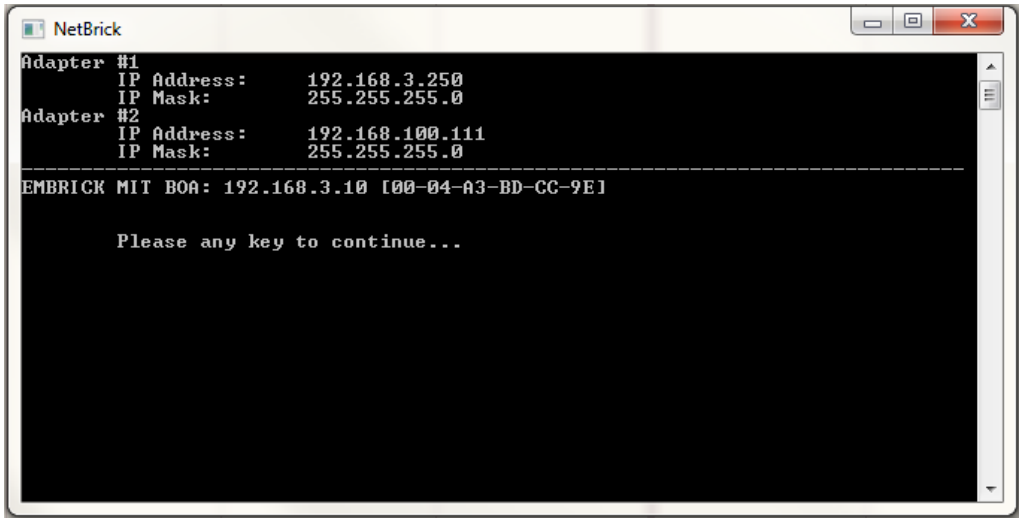
4.3 Check Hardware and LAN-Adapter Settings

Before starting with the software development, check the hardware by switch on the 24V power of LWCS board and starting "NetBrick.exe" (it is delivered inside the main folder of the BSP. NetBrick_new.exe works with firmware versions of the CouplingBricks from V0.37 and the NetBrick_old.exe works with firmware versions V19 and V20).

NetBRICK is a useful tool that checks all network ports from your PC whether there is a *coupling-master* connected with its IP-address. All founded coupling-masters will be listed with their IP addresses. With *NetBRICK* you can simply check ...

- if your PC Ethernet-Adapters is correct configured
- if the *coupling-master* is working and connected/found
- the IP-addresses of the available *coupling-master*

To start, double click on **NetBrick.exe**



picture 3

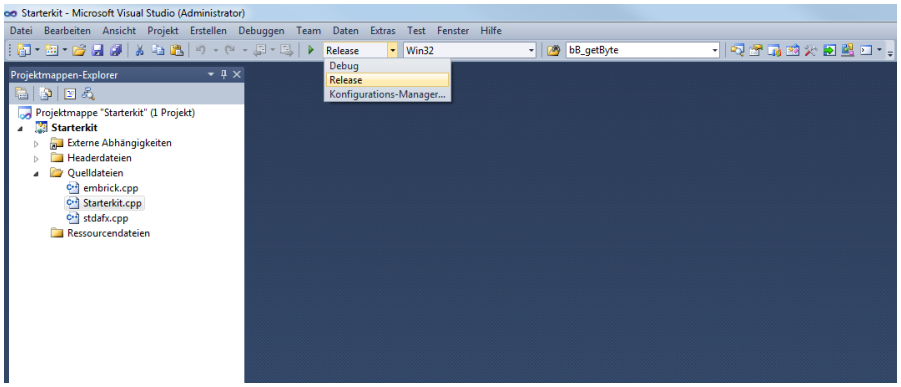
In this picture you can see the output of the *NetBrick* at first all your Ethernet-Adapters are listed and there after comes the detected *coupling-master* with the IP 192.168.3.10.

Result: Now you can access the *coupling-master* and *slave-modules*.

4.4 Load and compile the Sample Application

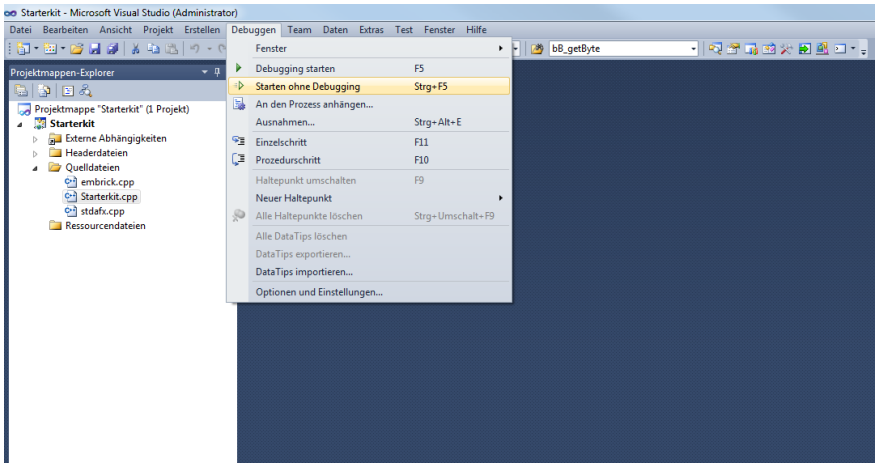
In the folder you have unzipped the “Z-CouplingBrick_MSVC_V#” software package you will find a file named “Starterkit.sln”. When you now double-klick it the Microsoft VisualStudio 2010 on your PC will open up this project. When you use a newer version VisualStudio will ask you whether you want to import it or not – then you choose import.

Now you choose “Release” instead of “Debug”



picture 4

and now go to “Debuggen” -> “Starten ohne Debugging” or you press Strg + F5 for compile and start.



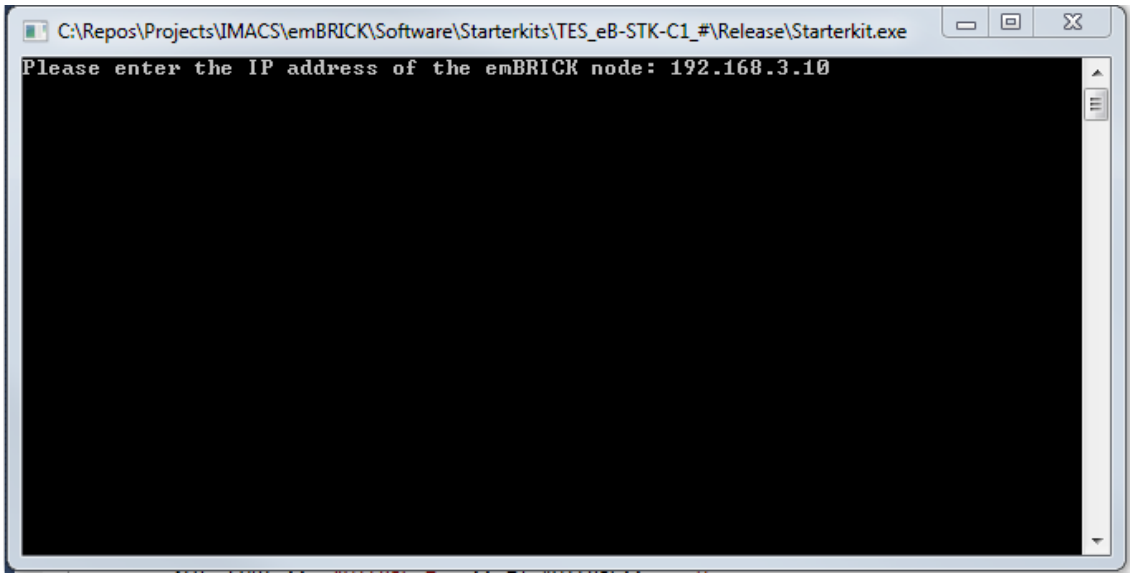
picture 5

Info: On the two pictures you can see on the left side the files that are integrated to the project. The “Starterkit.cpp” includes the functional application.

Result: A window is opened from the application.

4.5 Start and explore the Functionality of "Starter Kit"

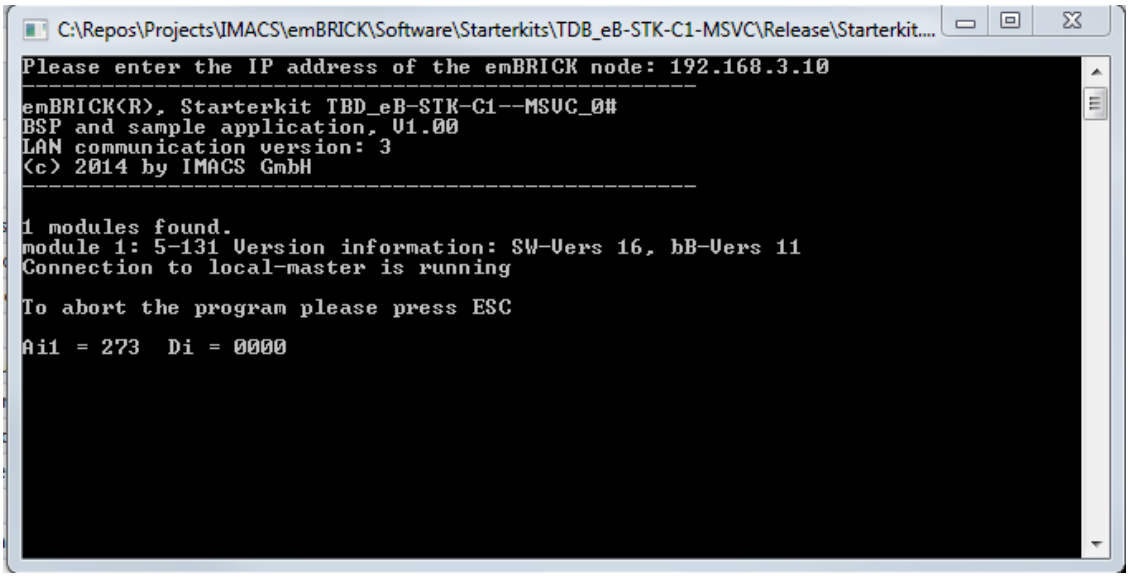
The window now is opened from the application looks like this:



picture 6

Now you have to give in the IP address of the coupling-master, in this case it´s the “192.168.3.10”. This is also the IP which NetBRICK is given back at the Check Hardware and LAN-Adapter Settings.

Now the application is started and has read out the module ID and it´s software and protocol versions. These are shown in picture 7. At the first start the IP is required, on the second not because it was saved in this file Release/datei.txt



picture 7

Result:

Now the system is running, so there is Relay1 toggling with 1Hz. Input 1 is switching Relay2 and the analogue input of I5 is continuous read out and be shown on the screen. You can manipulate the potentiometer and button to show the changing. The inputs are shown on the screen after the analogue input if no input is active there are four zeros, and each zero switch to one when the input is activated.

The clamp numbers of the IOs are defined in product catalogue search for “P-2Rel4Di2AI”.

To exit, press [ESC] or [Ctrl+C] or you click on the x from the window.

For more details about the hardware, see *Product Calatogue* (search for "Z-CouplingBrick-02", “P-2Rel4Di2Ai-01”).

4.6 Create your own application

This sub chapter describes how create your own application based on the “Sample Application” descripted above.

Preparation:

- Unzip the **Z-CouplingBrick_MSVC_V#.zip** again in another folder, i.e. ***…/example***.
- Open the project files (Starterkit.sln) for a Visual Studio 2010 and open – StarterKit.cpp.

You can change the functional code in the DEFINES and the while loop of the StarterKit.cpp.

Currently, the application controls the Relay1 so that it alternates every second. Now change the application that Relay2 alternates every second too.

1. Create a new definition for the second Relay “#define MY_RELAY2 1,1,0,1”
   The meaning of 1,1,0,1 is explained in chapter 4.7.4.2 IO-Access Functions.
   Add it after near the other defines marked with `/*---DEFINES---*/.`

2. Write the command “***bB_putBit(MY_RELAY2, flash);”*** in it into function
   while (!GetAsyncKeyState(VK_ESCAPE)) after bB_putBit(RELAY1, flash).

3. Start the compilation and execution of your application.

Result: Now the Relay1 and Relay2 alternates every second.

When you want to access the other future of *slave-module* then search for “P-2Rel4Di2AI” in *Product Catalogue*.

4.7 The MSVC Remote-Bus Driver

This driver provides a simple but efficient remote-bus access to the connected emBRICK strings (string = Coupling-Master + n x I/O-modules), further described as “node” via Ethernet (TCP/IP).

It is written in C++ to allow an easy adaption/integration into own code/projects. As of right now the driver is not available as a LIB or DLL, although it is possible to create some of the supported code.

4.7.1 Features

The driver supports:

- Establishing the connection to the node(s) that is (are) connected to your PC via Ethernet.
- Read the configuration data of each node and its connected bricks
- Read and write I/O-data to each emBRICK node (and its bricks)

4.7.2 Mode of operation

Therefore, the actual native SPI update process (local emBRICK Bus) is controlled by the coupling master, the operation via this driver (remote emBRICK bus) contains only a few simply steps.

The actual data exchange is managed via a separate input and output buffer (shared memory). After the initialisation a permanent triggered process have to be called to execute the update function. Parallel to this, a set of simple access functions (in C) allows a synchronized reading/writing access to the I/O-values.

During the initialization the node returns miscellaneous config-data to the driver that are used to locate the start of each I/O-module in the buffer.

4.7.3 Involved Files

The driver consists of two files

embrick.cpp

Contains the functions are called from the application. There are all defines for the size of the buffers and the task with the periodic communication to the *coupling-master*.

embrick.h

In the embrick.h all functions of the embrick.cpp are declared which parameters the functions need and they are given back.

For the sake of completeness, but not part of the driver itself:

Starterkit.cpp

In the Starterkit.cpp is the programmed application with the _tmain() and all the start-up functions for the communications are called once. The application uses simple put and get functions to set/reset an output or to read in the inputs.

4.7.4 Basic implementation

In an own application the following steps have to be implemented:

4.7.4.1 Initializing and Starting

Initializing and starting the I/O-update has been split into two function groups

a) Initialize the driver by **bB_Init(void)**. It connects to the node(s) specified in the command prompt at start-up and receives information about all connected modules. The module information can then be read by the user by various get-functions:

bB_GetConfNumModules(node)

bB_getModulID(node, moduleNumberInString)

bB_getModulSWVers(node, moduleNumberInString)

bB_getModulebBVers(node, moduleNumberInString)

These methods are also described in **embrick.h**.

b) After initializing, the user can call **bB_Start(updateRate).**This function starts a thread that triggers an internal function emBRICKTask() transmits and receives data to and from the node(s) periodically. This thread also takes care of reconnecting to the node in case it lost power or got disconnected briefly.

If your system offers an own time repetition mechanism, it´s also possible to call the emBRICKTask() by this function.

4.7.4.2 IO-Access Functions

The data of the I/O *slave-modules* are organized in a byte buffer for each node (a separate one of in- and out-data). To access this data, you need to define the ...

node number ..... (here always 1 because we have only one node),

module number (1...)..... position of IO-module in the node (emBRICK-*string)*

offset_position(0...) ..... relative position/offset of the data inside the module image. For details of each module refer to *Product Catalogue*, chapter 6.x.x., "process data image"

bit_position (0..7) ..... **only** in case of a bit access, indicates the bit in the selected byte

The actual data access is performed by 6 simple functions and that differ in the direction (reading/writing) and the data width (bit, byte, word). Of course also own functions can be developed to do this.

data reading (from IO-modules to application):

bB_getBit(node, module, offset_pos, bit_pos)

bB_getByte(node, module, offset_pos)

,bB_getShort (node, module, byte_pos)

writing (from the application to the IO-modules):

bB_putBit(node, module, offset_pos, bit_pos)

bB_putByte(node, module, offset_pos)

bB_putShort (node, module, offset_pos)

About their exact parameters and their return value, refer to the comments/description inside the files ***embrick.h*** and ***embrick.cpp***, where they are defined and implemented.

Note: Access to the byte buffer is already buffered and secured by mutexes.

Tipp: To avoid the manual input of the single digits in the function parameter, create a macro definition for each I/O you like to use that contains these digits.

Example:
#define MY_BUTTON 1,1,0,1 // Node 1, Module 1, Byte 0, Bit 1
This allows the coding *bB_getBit(MY_BUTTON)* instead of *bB_getBit(1,1,0,1)*

5 Hands on Software - with CODESYS

5.1 Setup the Development Environment

Download the newest version of the freely available CoDeSys EXE (works only with the 32-Bit Version) from http://www.codesys.com/download.html (tested on version between 3.5.13.20 to 3.5.16.40). If you are new to CODESYS, you might have to register to the CODESYS website to gain access to the download.

Install CODESYS.

5.2 Download the Demo-Project

- Download the demo project: "BSP_Z-CODESYS-Brick-V0.13.zip" from the web.
- Unzip the file and save it anywhere to your computer.

5.3 Check Hardware and LAN-Adapter Settings

see 4.3

5.4 Setup the Hardware

Follow the steps below or stick to the instructions in Chapter “**Mounting and wiring**”.

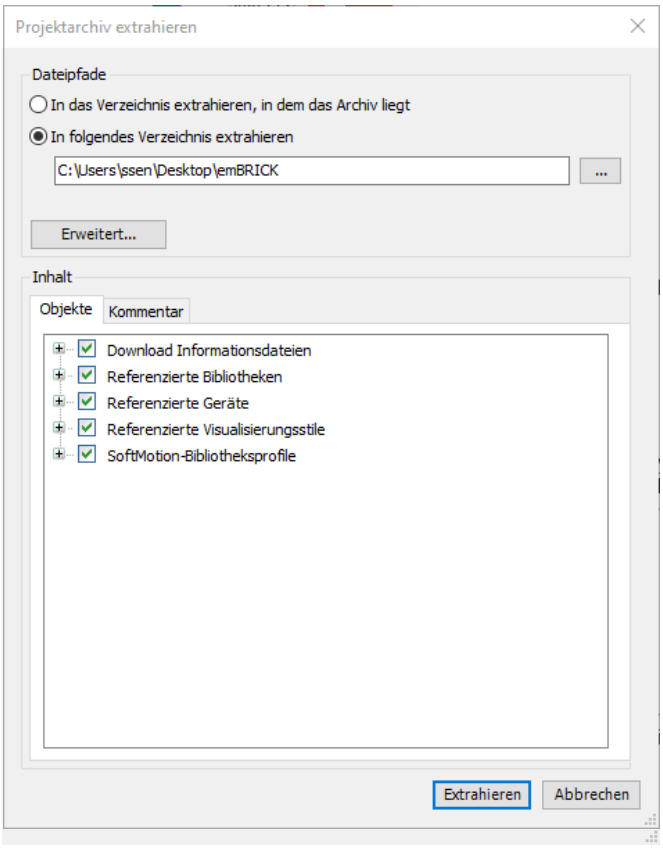
- Connect the *Localmaster* (Z-PadBridgeMx) to the *emBRICK*®-module (P_2Rel4Di2Ai-01)
- Set the DIP-Switch to ‘0001’.
- Supply the *Localmaster* with 24V
- Test the connectivity by sending a ping to 192.168.3.10.
  If necessary, configure your network card for the right subnet. A connection can also be established via a USB-network adapter.

5.5 Starting the Demo Project on a PC

Note: Depending on your operating system and CODESYS version, you might get additional dialogs that are not covered in this guide. As a rule of thumb, press 'Yes' or 'OK' on all other dialogs.

To ease starting the Demo-Project, it has been packed into a projectarchive which is prebuilt and contains all necessary files. To start, open the file "*emBRICK_demo.projectarchive*" you just unzipped. CODESYS will start automatically. After CODESYS finished starting, a window will appear:

Click "Extract" to confirm the prompt.



A prearranged CODESYS project will open in the CODESYS IDE.

The CODESYS IDE has now started. To start the Application, a runtime is needed.

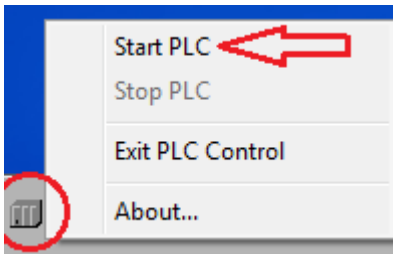
5.5.1 Starting the Runtime

For Windows, 3S provides a free demo runtime that will run for 2 hours.
After a standard installation, the runtime has usually started but is not running.

To check if the CODESYS runtime has started, search for the Control Win V4 icon in your windows statusbar (bottom right corner of your screen).

5.5.1.1 Runtime has started

Now that the CODESYS runtime has started, you can activate it. Do so by rightclicking on the runtime icon in your statusbar and select “Start PLC”.



The runtime icon should change to:

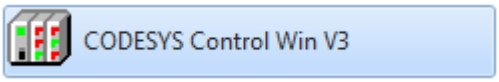
The runtime is now active. Continue to the next chapter.

Note: It will only work for 2 hours after activating, after which it has to be restarted manually.

5.5.1.2 Runtime has not started

If your runtime has not started, it has to be started manually.

The runtime can be found in your CODESYS installation directory. The standard path is “c:\Program Files (x86)\3S CODESYS\GatewayPLC\CODESYSControlService.exe”. You should also find it through the Start Menu (All programs → 3S CODESYS → CODESYS Control Win V3 → CODESYS Control Win V3).



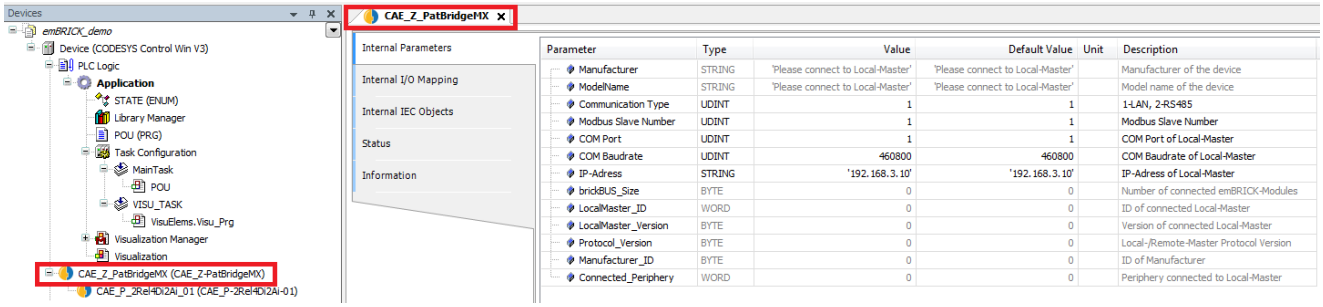
The Runtime should look similar to this.

When starting the runtime manually, you won’t have to activate it.

Note: Without a license it will only work for 2 hours after activating, after which it has to be restarted manually.

5.5.2 Select the Type of Communication

Double click the LocalMaster and the “Internal Parameters” Tab should be visible.



In this tab are all options that are needed to select a communication type.

There are only four options that are changeable:
Communication Type <-- To select over which way the communication should be established. Lan or RS-485?

If RS-485 is selected:
COM Port <-- Which COM Port should be used?
COM Baudrate <-- At which Baudrate should the COM Port operate? (PiBrick: 460800 Baud)
Modbus Slave Number <-- Which Slave Number has the Modbus Slave ?

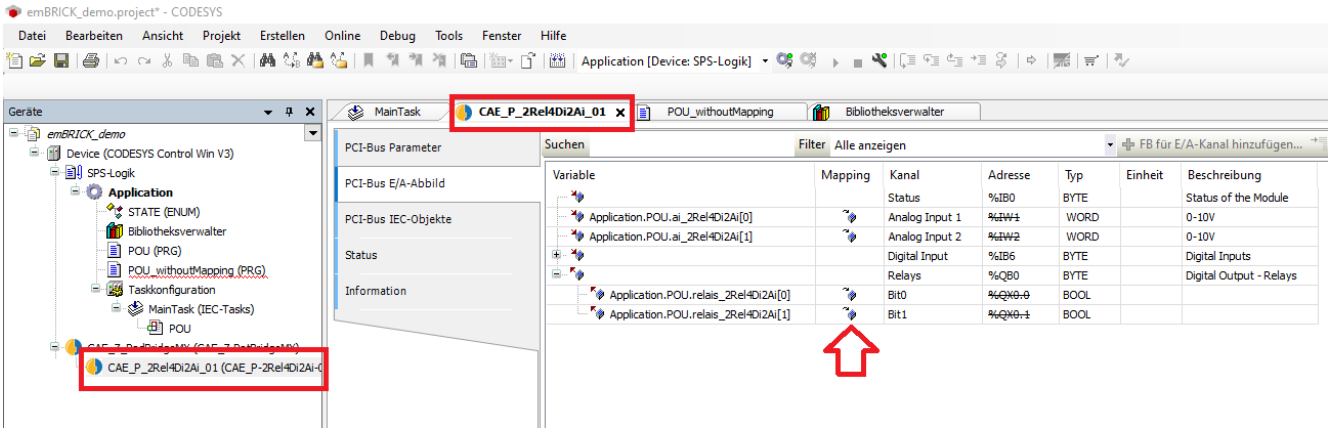
If Lan is selected:
IP-Adress <-- Which IP Adress has the Local Master?

Parameter	Type	Value	Default Value	Unit	Description
Manufacturer	STRING	'Please connect to Local-Master'	'Please connect to Local-Master'		Manufacturer of the device
ModelName	STRING	'Please connect to Local-Master'	'Please connect to Local-Master'		Model name of the device
Communication Type	UDINT	1	1		1-LAN, 2-RS485
Modbus Slave Number	UDINT	1	1		Modbus Slave Number
COM Port	UDINT	1	1		COM Port of Local-Master
COM Baudrate	UDINT	460800	460800		COM Baudrate of Local-Master
IP-Adress	STRING	'192.168.3.10'	'192.168.3.10'		IP-Adress of Local-Master
brickBUS_Size	BYTE	0	0		Number of connected emBRICK-Modules
LocalMaster_ID	WORD	0	0		ID of connected Local-Master
LocalMaster_Version	BYTE	0	0		Version of connected Local-Master
Protocol_Version	BYTE	0	0		Local-/Remote-Master Protocol Version
Manufacturer_ID	BYTE	0	0		ID of Manufacturer
Connected_Periphery	WORD	0	0		Periphery connected to Local-Master

5.5.3 Application POU(PRG) with Mapping and without Mapping

The Application code POU(PRG) and POU_withoutMapping both are wriiten in the Programming Language ST.

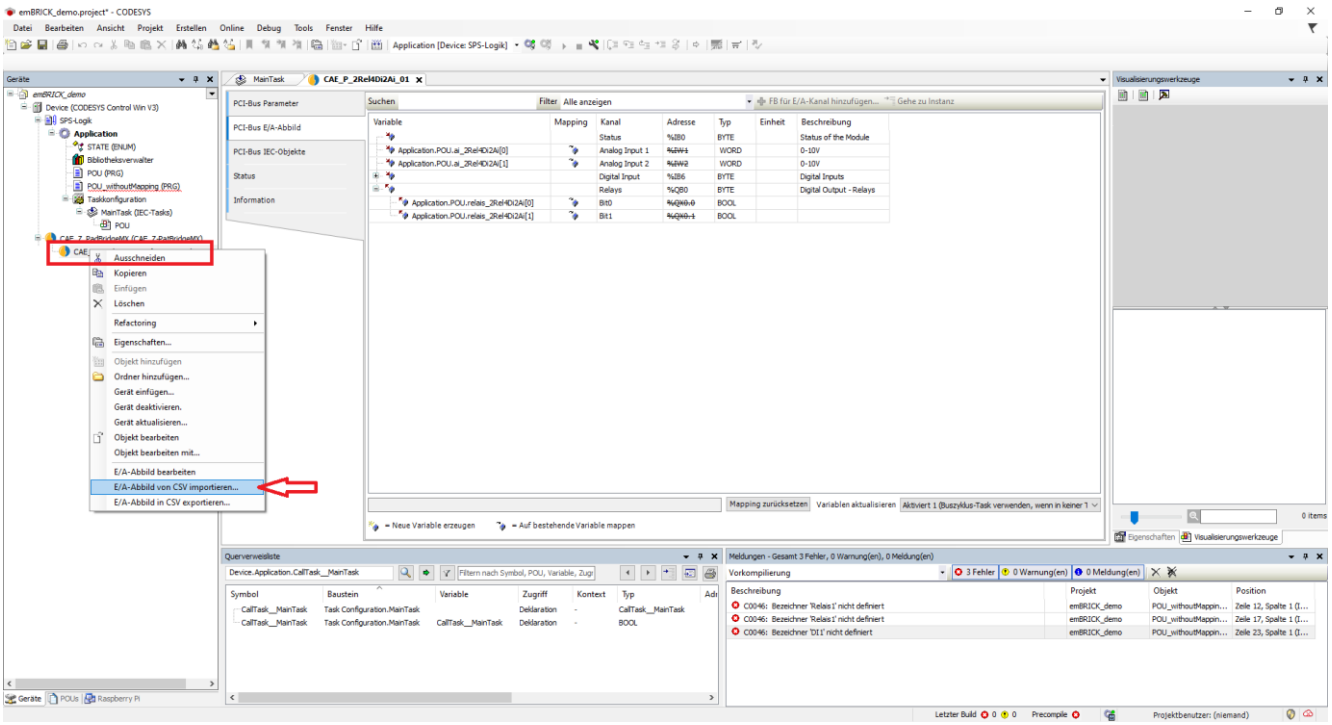
The Code “POU(PRG)” written with declared Variables (Mapping)



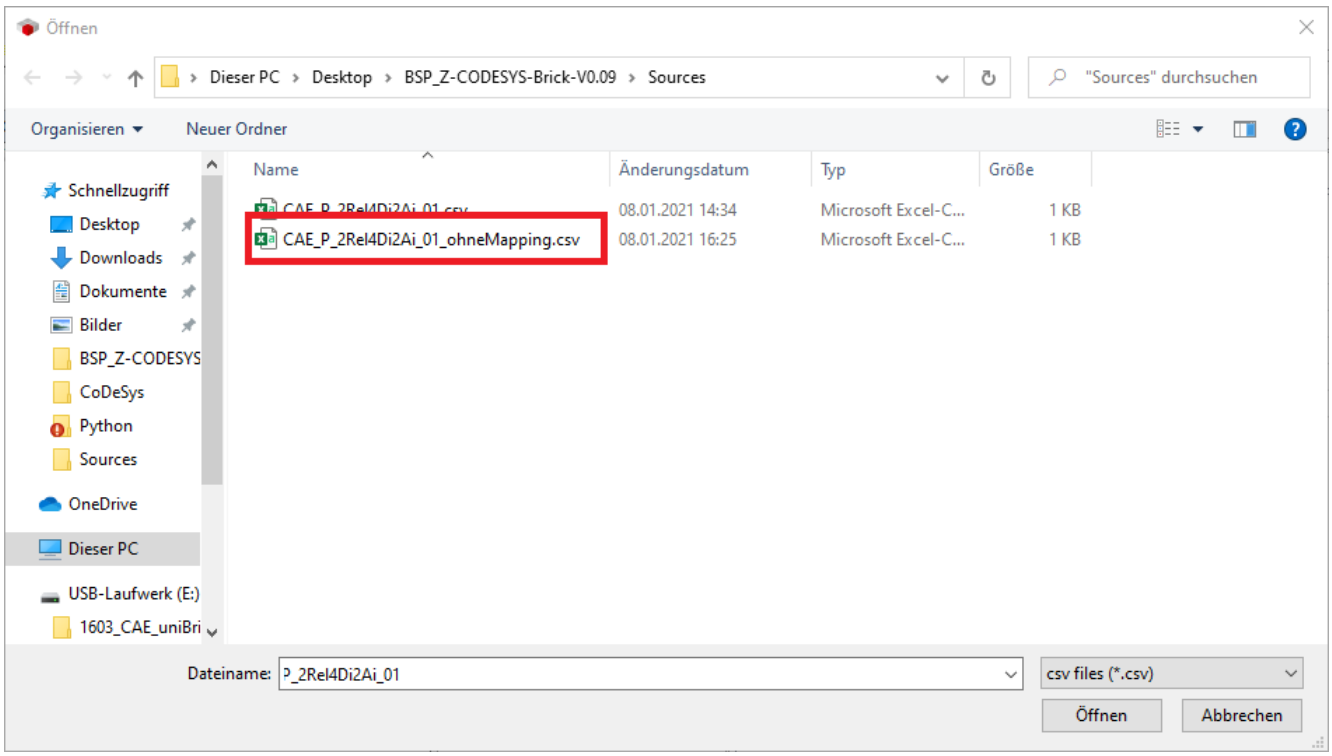
If you double click on the Mapping Symbol the Variable Name you give will be declared and you can’t use the Adress anymore.

The Code “POU_withoutMapping(PRG)” written without declared Variables (without Mapping).

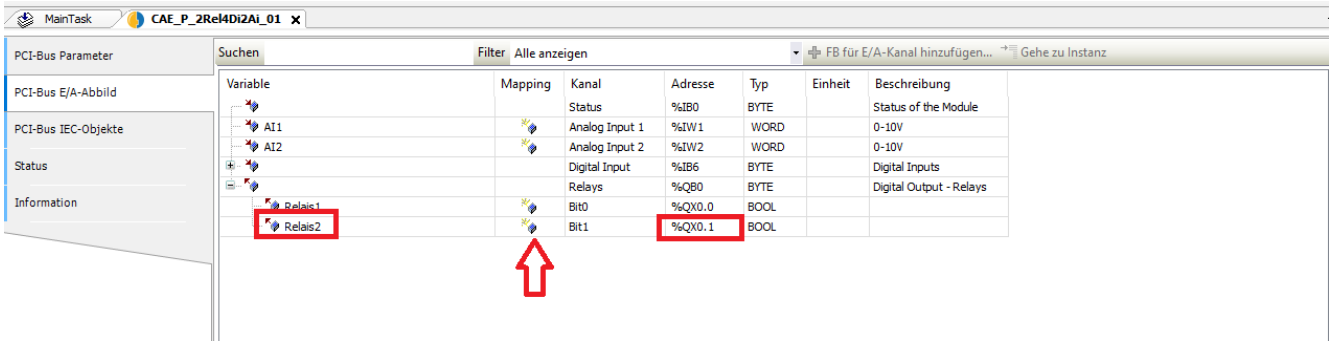
The Program is on start automatically setted on POU (PRG).



If you want to use the POU_withoutMapping you can change it by doing this click the right mouse click above “CAE_P_2REL4Di2Ai_01” and select “E/A-Abbild von CSV importieren”.

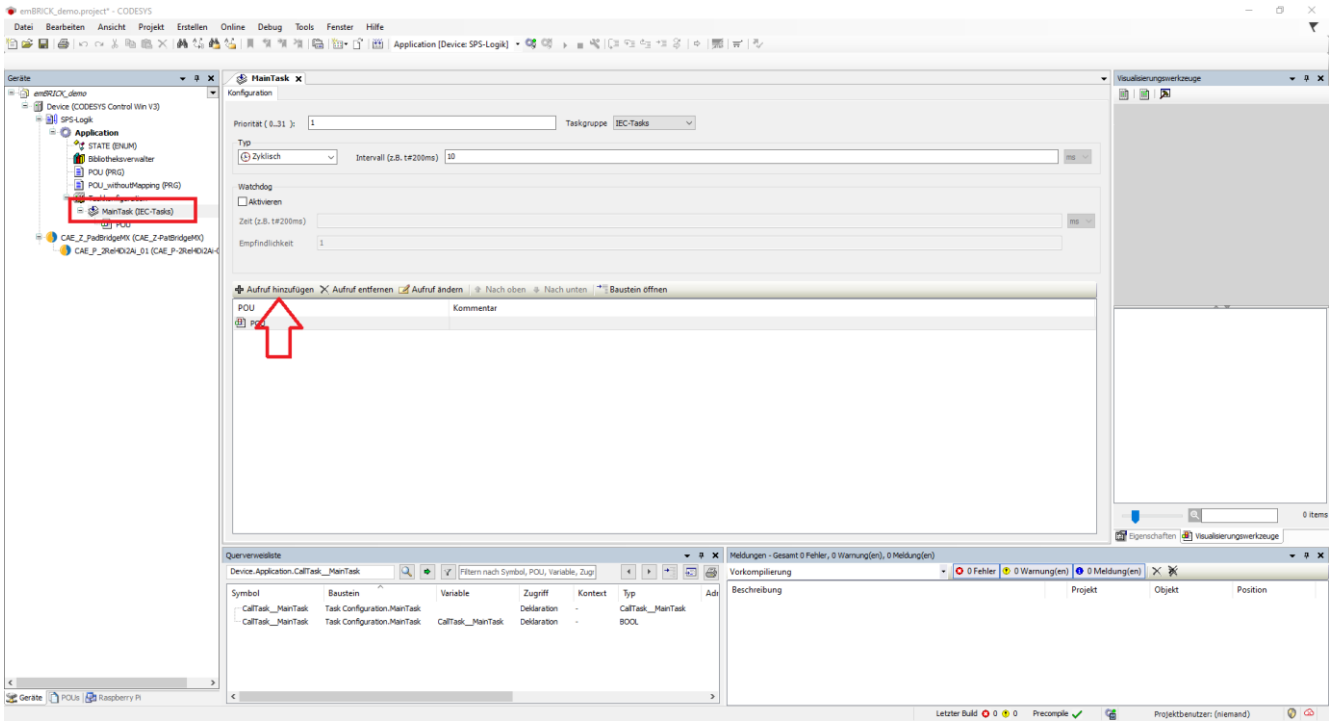


Then in the opened window you select “CAE_P_2Rel4Di2Ai_01_ohneMaping.csv.

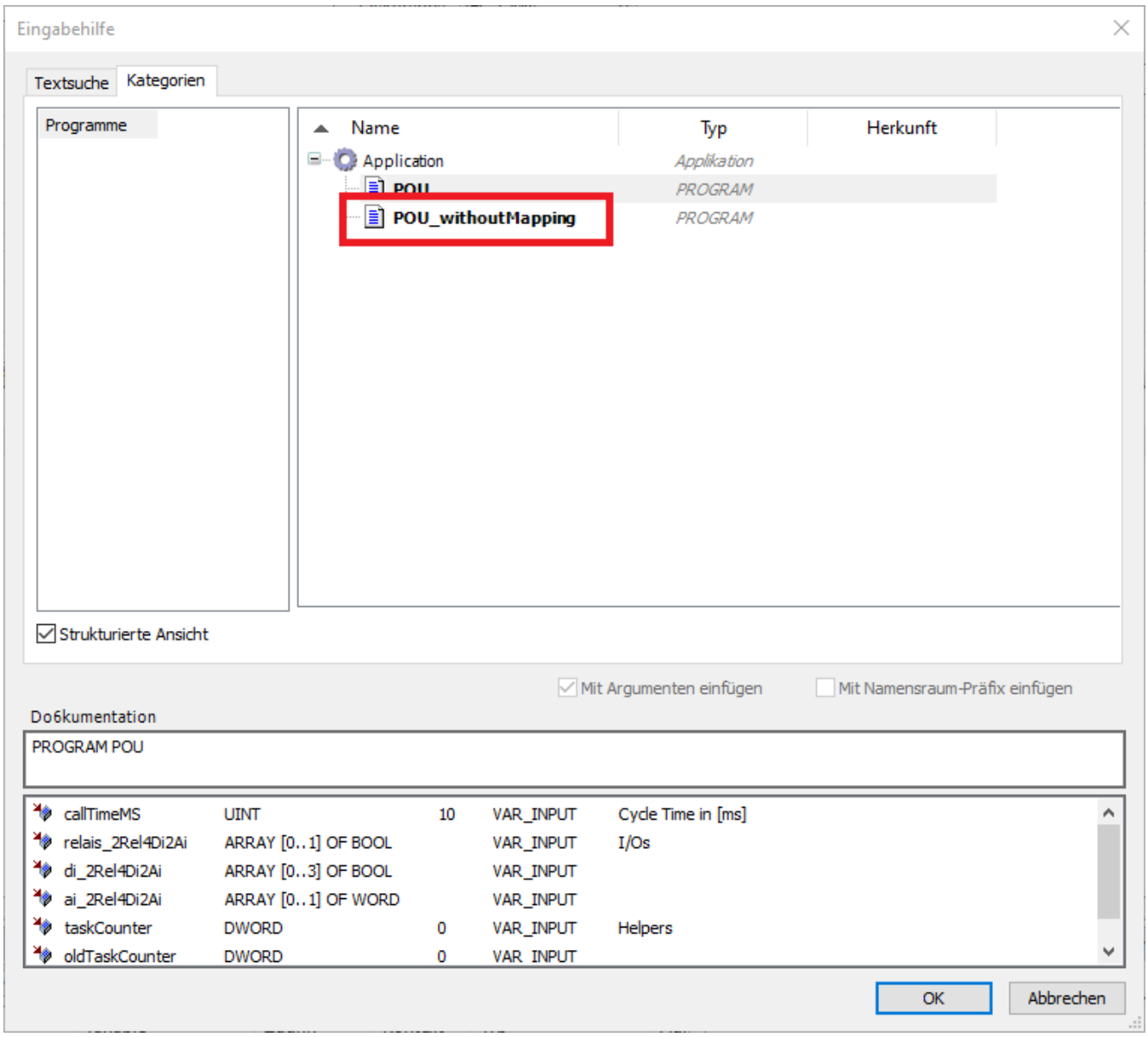


Because here is the Mapping is not active. We can code with the Variable Name and the with the Adress.

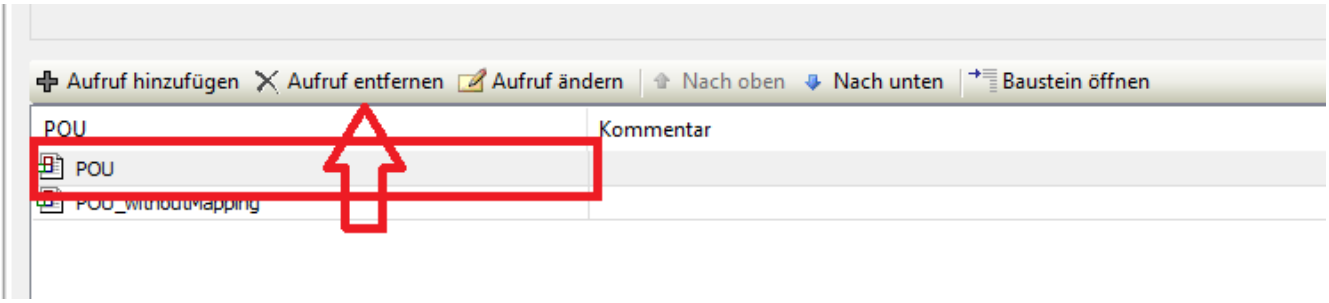
At least we change the main task.



To change the main task we doubleclick on the left sidebar on MainTask(IEC-Tasks) then we doubleclick on “Aufruf hinzufügen”.



Then select POU_withoutMapping and click on “OK”.



Next we delete “POU” while we click on “POU” and click “Aufruf entfernen”.

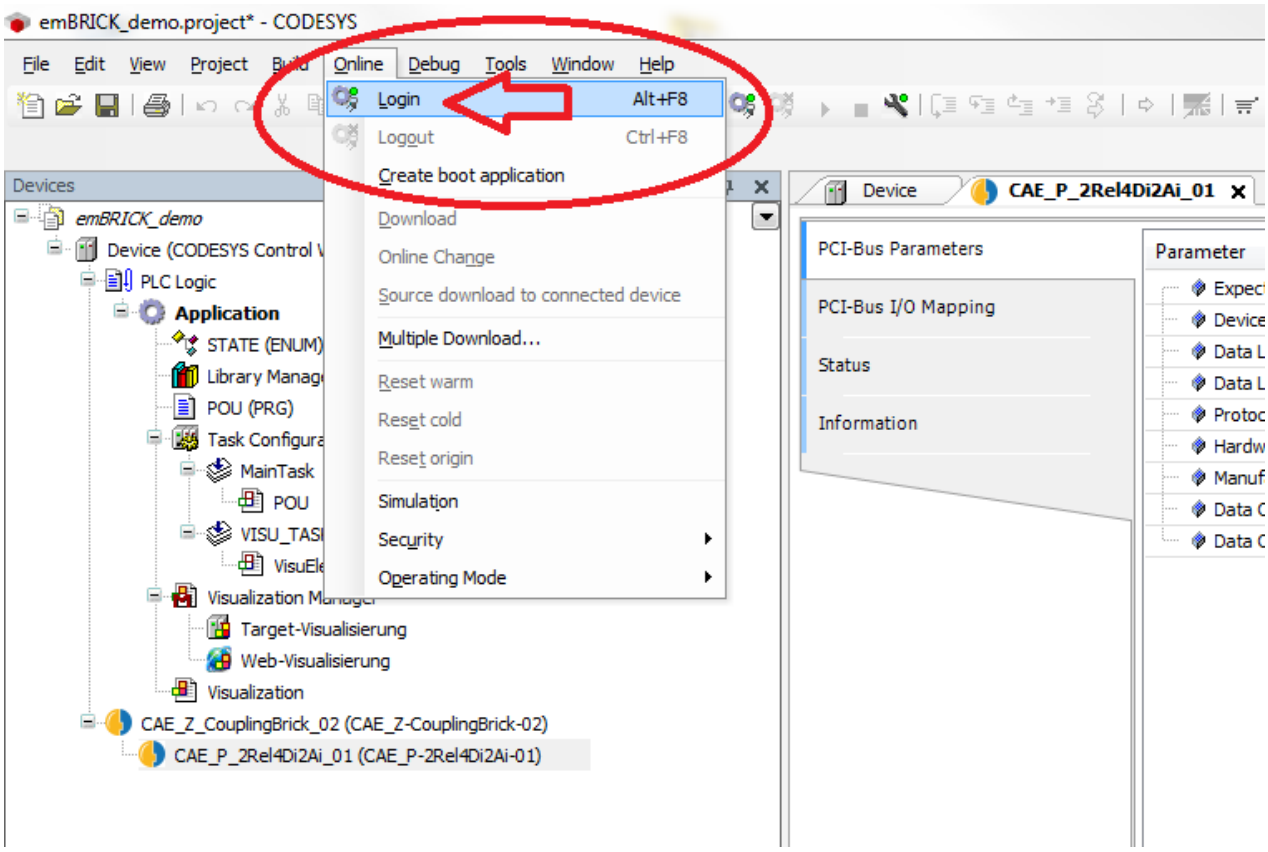
You can easily change in the opposite way if you want programming with Mapping.

5.5.4 Logging into the runtime

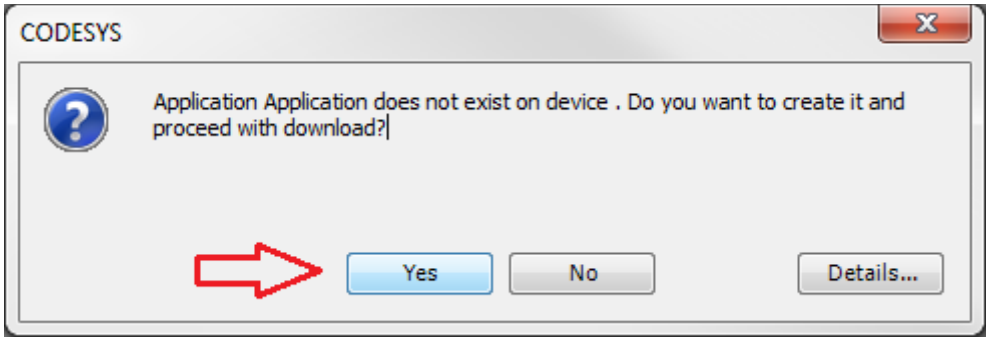
Make sure CODESYS Control Win V3 has started, then log into the runtime.

Your application will run indepently from your CODESYS IDE in the CODESYS runtime. You can make changes to your application while the last version of your application is running. To update (or create) the Application that was downloaded to the runtime, you will have to login into the runtime. In this process the Application will be compiled and downloaded to your runtime. After logging in, you will have access to the debugging methods of CODESYS.

First, log into the application as shown below.



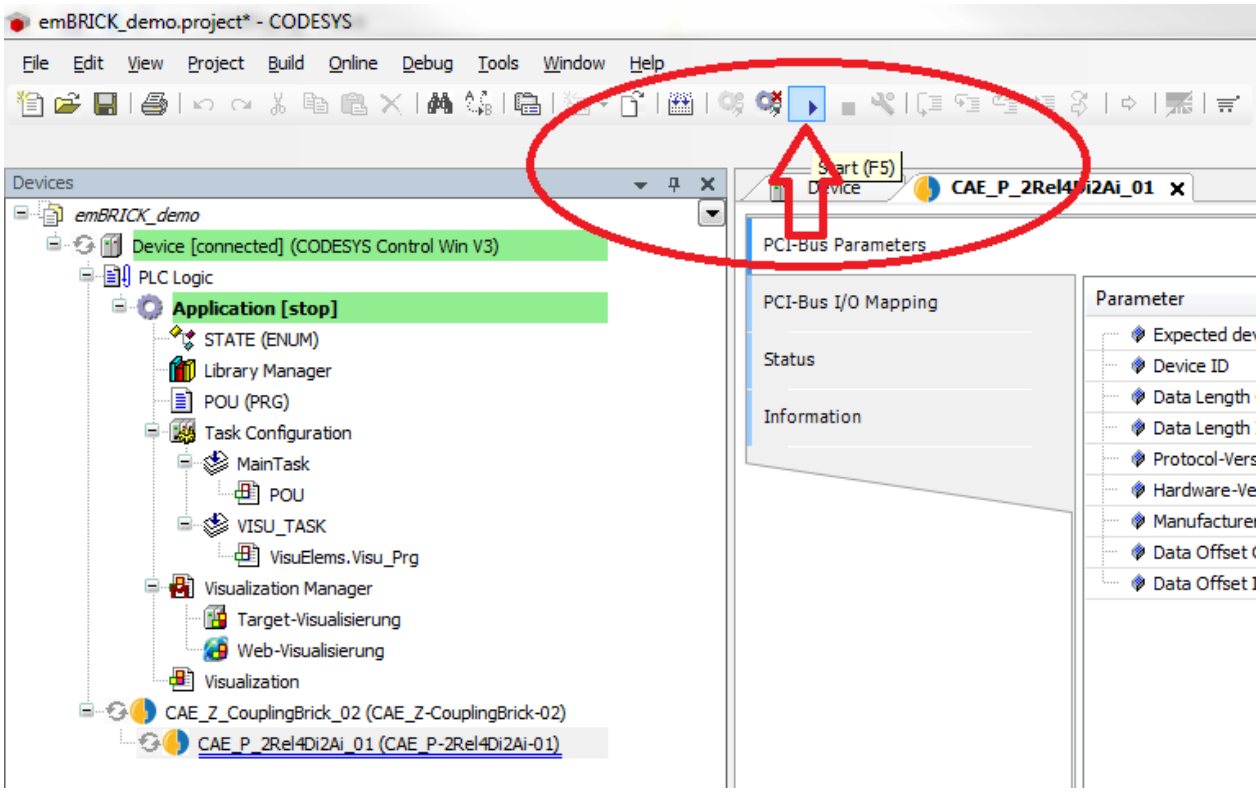
This window should appear when connecting to the runtime for the first time. Confirm it by clicking on “*Yes*”.



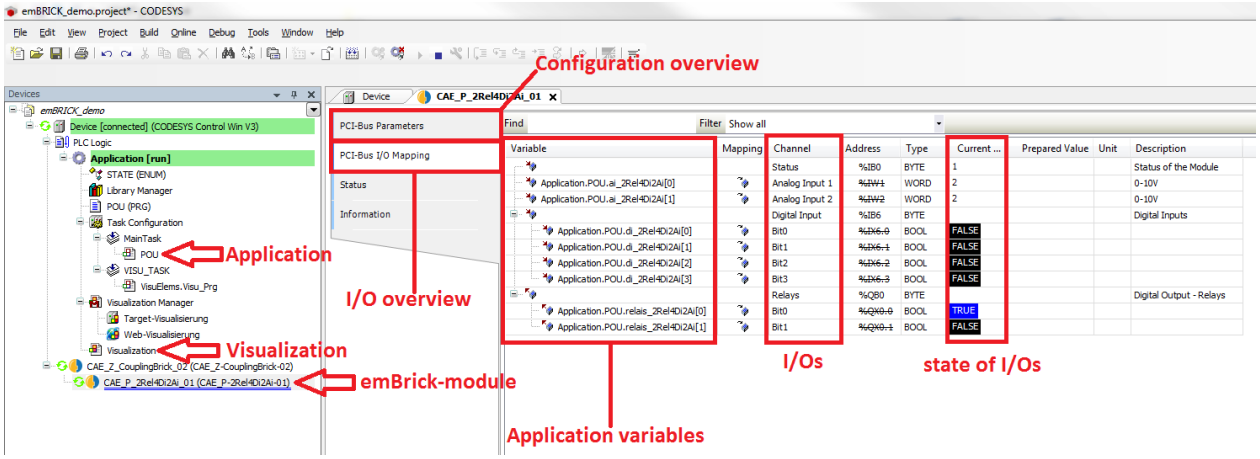
Note: in some cases, you will have to login twice. You are logged in when your IDE looks similar to the next picture.

Now you should be connected to the runtime, but the application has not started yet. Start the application by clicking on the "*Start*" button.

Result: The first relay ("Rel 1") should now toggle every second. The second relay (Rel 2) is activated by connecting Input 1 ("Button", or Pin 7) to ground (Pin 8).



The I/Os can also be watched inside the IDE. To do so, expand the *emBRICK*®_Localmaster, then double-click on the module P_2Rel4Di2Ai-01 and switch to the PCI-BUS I/O-Mapping.



This view lists all I/Os of the *emBRICK*®-modules. Typically for CODESYS you can easily connect variables of your application to the I/Os of the *emBRICK*®-module by simply doubleclicking into a variable field. In this Demo-Project the I/Os are also mapped to different elements in the visualization.

The application code itself can be found in "POU(PRG)" and is written in ST.

5.6 Starting the Demo Project on a Raspberry Pi

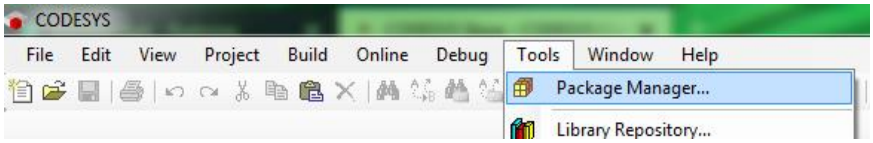
You need a Raspberry Pi 3B or 3B+ as a master device to control our bricks.

You can use a normal Raspberry Pi with its ethernet port or/and a USB to ethernet adapter to control the bricks with our CodeSys library over ethernet.

If you want to control the Bricks over RS485 with a Raspberry Pi then you need a USB to RS485 adapter or our **CAE_Z-RaspberryBrick-1#-RB** as a master with its build in RS485.

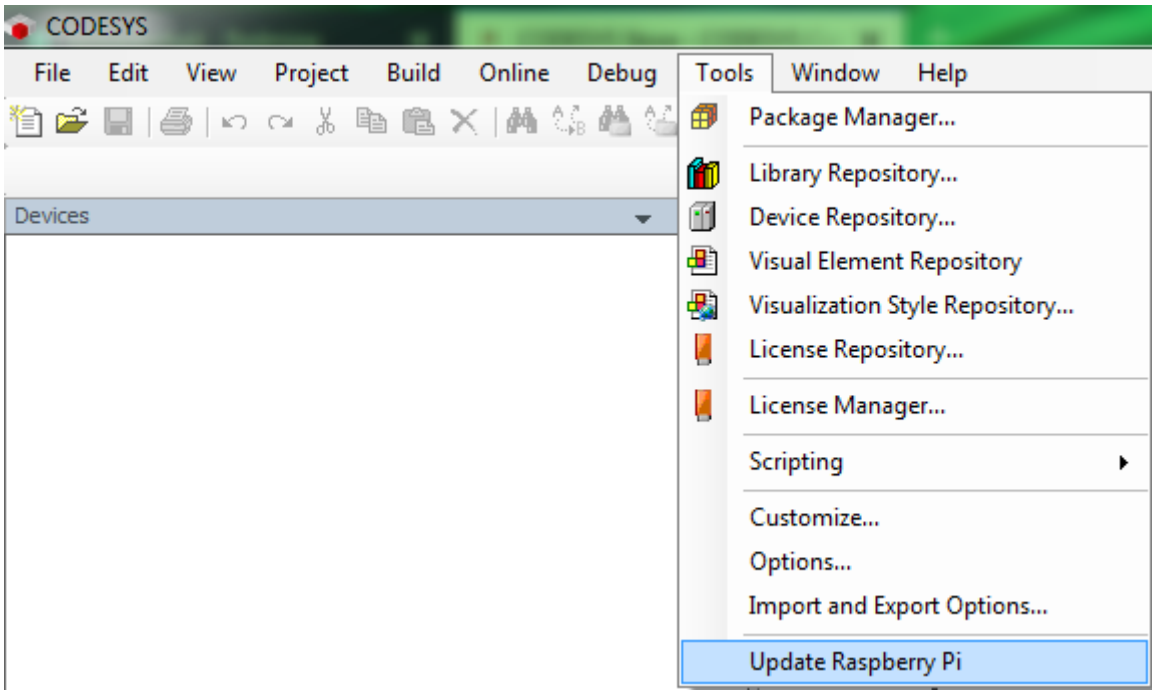
5.6.1 Installing CodeSys on the Raspberry Pi

Download CodeSys Control for Raspberry Pi SL from the CodeSys Store and install it through the Packet Manager.

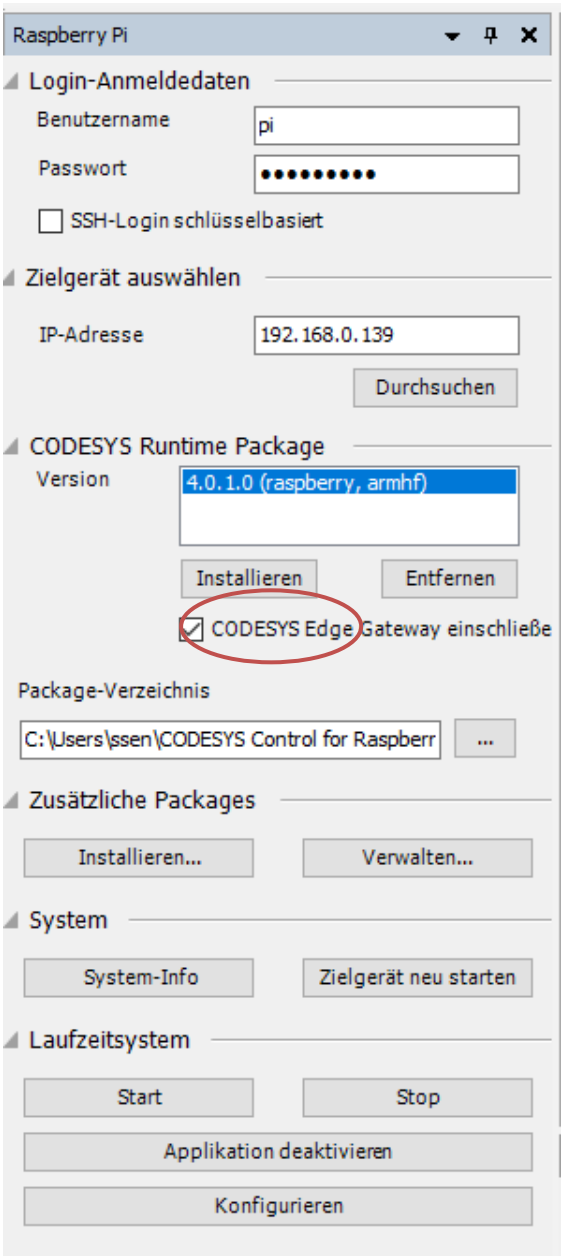


In the next window click on install and navigate to the downloaded CodeSys Control for Raspberry Pi SL packet and install it.

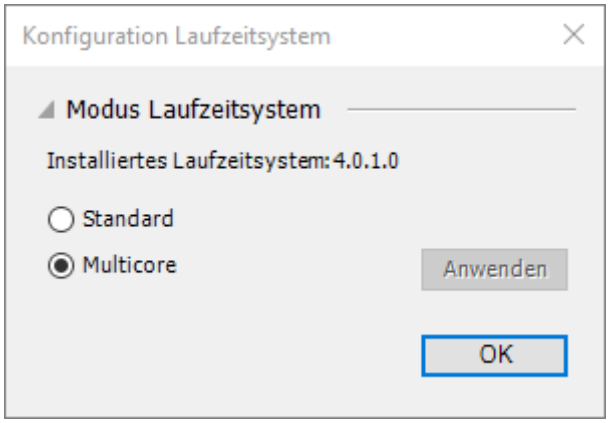
After sucssesfull installation and restart of CodeSys you should now finde the new entry “Update Raspberry Pi” in “Tools”.



After opening the new entry, you should see the following menu:



Enter your login credentials aswell as the IP of your Raspberry Pi and click on Install.



During the Installing will be pop up this Window select there Multicore and click “OK”.
Now the CodeSys runtime gets installed on the Raspberry Pi.

Our Test got performed with Versions between V3.5.13.20 to V4.0.1.0.

5.6.2 Installing and configure drivers

5.6.2.1 Ethernet

5.6.2.1.1 Installing the driver

You can use the build in ethernet port of the Raspberry Pi to controll our bricks, but then you can't use the port to connect to the internet.

Or you use the build in port for internet and use a USB to ethernet adapter for the communication with our bricks.

When you use the build in port you don't need to install a driver.

When you use a USB to ethernet adapter it might be possible that you need to install a driver.

Our USB to ethernet adapter didn't need a driver. We just plugged it in and checked the usb devices with "lsusb" for our adapter:

```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 005: ID 0b95:1790 ASIX Electronics Corp. AX88179 Gigabit Ethernet
Bus 001 Device 004: ID 0424:7800 Standard Microsystems Corp.
Bus 001 Device 003: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Device 005 is our adapter.

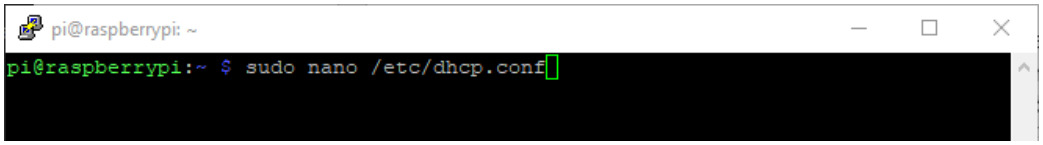
Now with "ifconfig" you can list all ethernet ports with their ip address.

```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.159  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::ec50:32bd:73f1:a128  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:df:78:d4  txqueuelen 1000  (Ethernet)
        RX packets 10189  bytes 1394381 (1.3 MiB)
        RX errors 0  dropped 435  overruns 0  frame 0
        TX packets 282  bytes 42113 (41.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 169.254.143.78  netmask 255.255.0.0  broadcast 169.254.255.255
        inet6 fe80::f9f:dbfc:3ee8:a240  prefixlen 64  scopeid 0x20<link>
        ether 50:3f:56:02:3c:c5  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33  bytes 5435 (5.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

5.6.2.1.2 Configuring ethernet ports

You need to set the IP of the ethernet adapter you would like to use to the ip range of "**192.168.3**.10" or any other IP you selectet with the dip switches or the over the VISU.

To do that login to your pi with putty and chang the file "dhcpcd.conf"



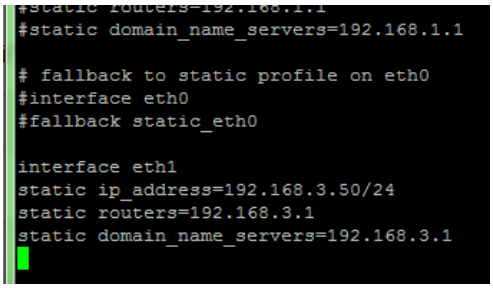

```
sudo nano /etc/dhcpd.conf
```

Now add the following lines after the last line from the "dhcpcd.conf" file

```
# Ethernetadapter for Bricks
interface eth1
# Do not choose the same ip as the Local Master
static ip_address=192.168.3.50/24
static routers=192.168.3.1
static domain_name_servers=192.168.3.1
```

eth0 is the build in ethernet port and eth1 is the USB to ethernet adapter.

If you want to use the build in port change all eth1 in the upper code to eth0.



Save the changes on the file with "CTRL+X" and then confirm with "Y" and then "Enter".

Now everything should be ready to adapt the Demo Project.

See chapter 5.6.3.

5.6.2.1 RS485

5.6.2.1.1 Installing the driver

When you use a USB to RS485 adapter it might be possible that you need to install a driver.

If you use our **CAE_Z-RaspberryBrick-1#-RB** as a master then you need to use our open-source driver. Just follow our tutorial on Github page to install them.

5.6.2.1.2 Configuring RS485

Now the settings for the COM Port must be made.

If you don't know where to set these settings then first read chapter 5.5.2.

The COM Port should always be 1 on the Raspberry Pi. That's because of the CodeSys config file "CODESYSControl.cfg". If you use the our drivers then this file gets installed automatically.

Now everything should be ready to start the Demo Project.

See chapter 5.6.3.

5.6.2.1.3 Configure CODESYSControl.cfg

Open File:

```
sudo nano /etc/CODESYSControl.cfg
```

The following lines must be added to the CODESYSControl.cfg:

```
[SysCom]
Linux.Devicefile=/dev/ttySC
portnum := COM.SysCom.SYS_COMPORT1;
```

Hit CTRL+X and confirm with Y to save the changes to the file.

With theses lines CodeSys can "see" the Serial Device.

Note:
If you update the installed CodeSys runtime on your pi you first need to deinstall the already existing runtime.
In that process CODESYSControl.cfg gets deleted and a new blank one gets installed.

That means after every Runtime change you need to edit this file!

5.6.3 Start the Demo Project for the Raspberry Pi

Note:

Depending on your operating system and CODESYS version, you might get additional dialogs that are not covered in this guide. As a rule of thumb, press 'Yes' or 'OK' on all other dialogs.

To ease starting the Demo-Project, it has been packed into a projectarchive which is prebuilt and contains all necessary files. To start, open the file "*emBRICK_demo_rpi_rs485.projectarchive*" you just unzipped.

CODESYS will start automatically. After CODESYS finished starting, a window will appear:

Click "Extract" to confirm the prompt.

After that you can follow the instruction to run the demo project in chapter 5.5.2.

5.7 Create your own project

The following instructions are aimed towards engineers already familiar with CODESYS. As such, it will not provide step-by-step instructions.

To create your own project, follow these steps.

- Create a new Standardproject. For PC, choose “CODESYS Control Win V3” as device.
- Open the Library Repository (Tools -> Library Repository) and install the library “emCoSys” (you might have to display all files).
- Open the Device Repository (Tools -> Device Repository) and install the devices „emBRICK®_LocalMaster“ and “CAE_P_2Rel4Di2Ai-01”
- Add the devices:
  Add Localmaster (Switch to “Devices” in the window on the left side if you haven’t already; Rightclick “Device (CODESYS Control Win V3)” -> ”Add Device” -> Miscellaneous -> emBRICK®_LocalMaster)
- Add emBRICK®-Module (Switch to “Devices” in the window on the left side if you haven’t already; Rightclick “emBRICK®_Localmaster”->” Add Device” -> Miscellaneous -> CAE_P_2Rel4Di2Ai-01)
- Expand the emBRICK®_Localmaster, then double-click on the module CAE_P_2Rel4Di2Ai-01 and switch to the PCI-BUS I/O-Mapping. Connect the variables in your Application with the I/Os of the module. Keep in mind that a simple DI or DO requires a single-bit-variable (like BOOL) while an analog Input requires a 2-Byte-variable (like WORD).

5.8 Create your own brick description

The devdesc.xml files are used to describe the Brick itself and its I/Os.

To create your own description, open the Programmers Manual and see chapter 8.7.1.

6 Hands on Software - with LabVIEW

6.1 Setup the LabView Development Environment

If you have not already Labview 2015 or higher installed, you can get an evaluation version for 1 Month at http://www.ni.com/download-labview/ and follow the installation instructions.

Result: Now you can create and edit applications and compile them. The software ist developed and tested with Labview 2016, previous versions might work.

6.2 Download and Install the Board Support Package

Download the Labview Starterkit from:

eB_LabVIEW Starterkit.zip

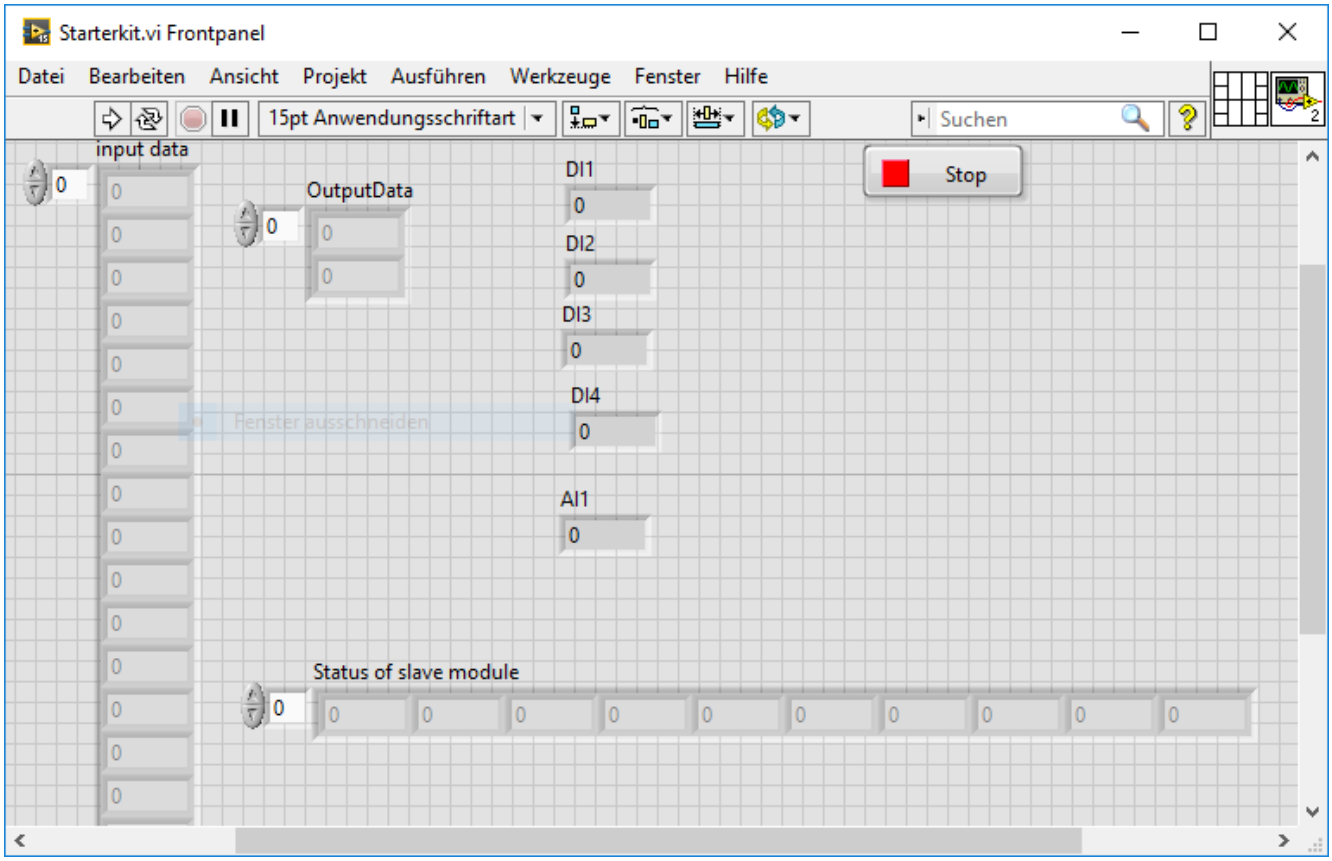
Extract the zip file to you're a into a new Folder of your choice

6.3 Check Hardware and LAN-Adapter Settings

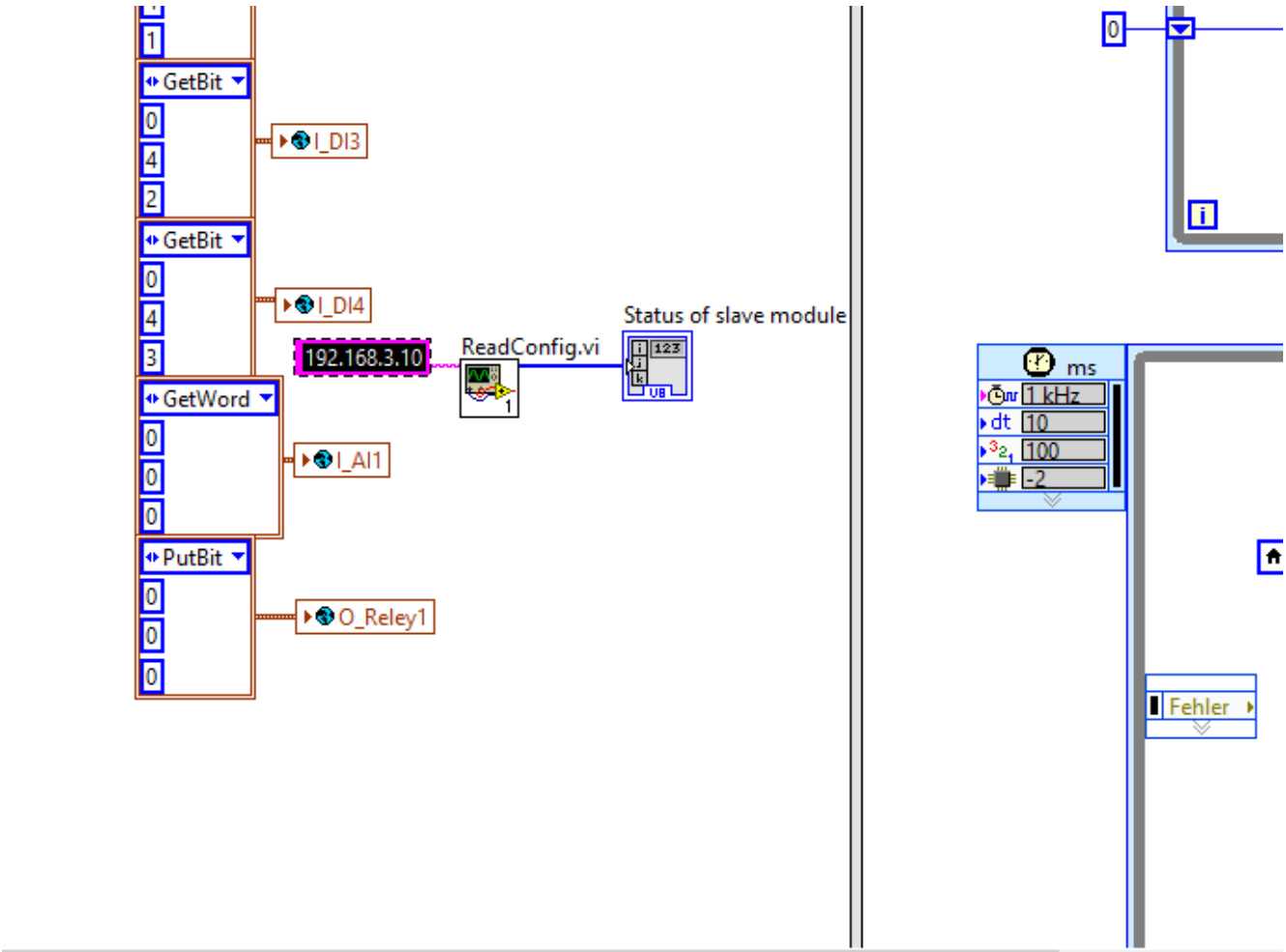
see 4.3

6.4 Load and start the Sample Application

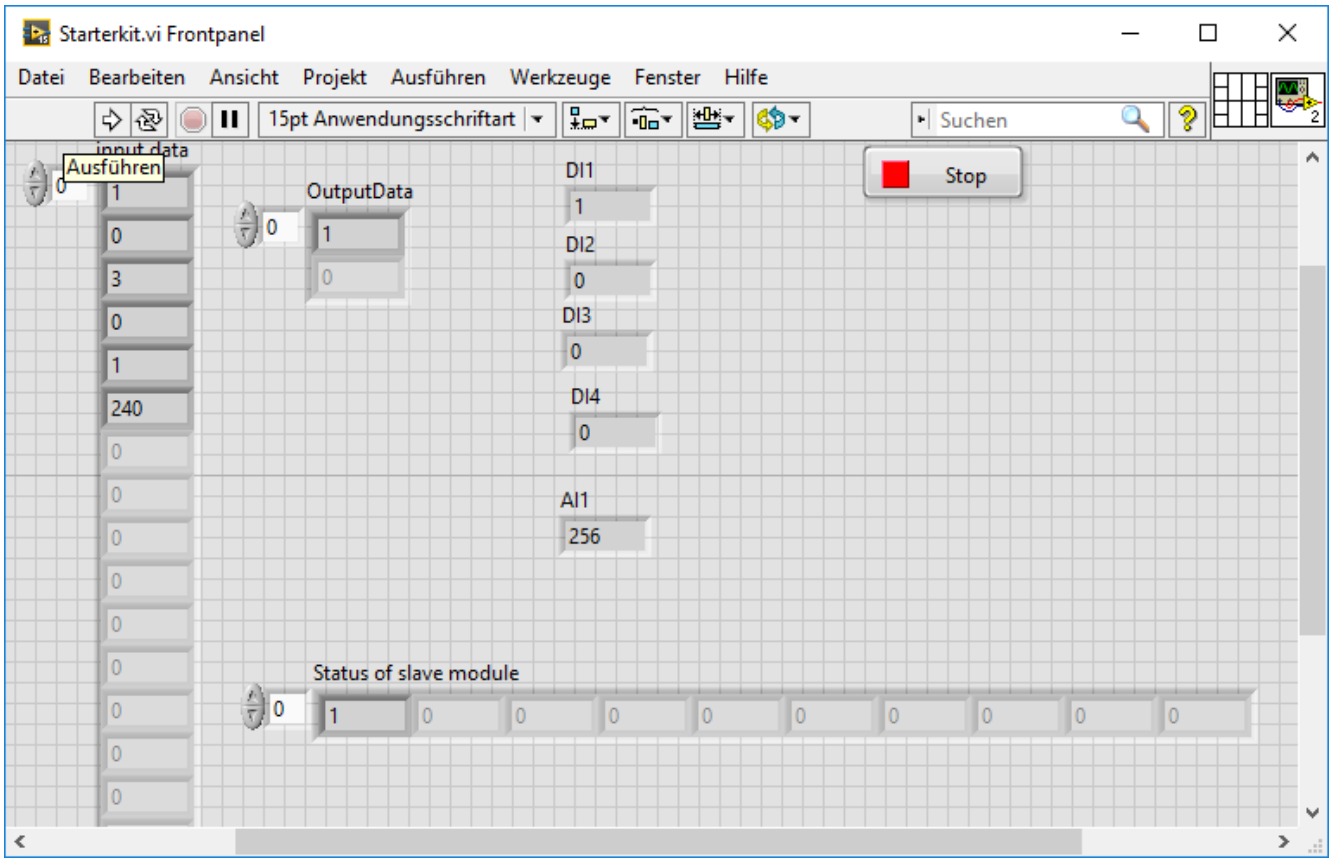
Doubleclick on Starterkit.vi. The application opens.



If you have configured a different IP-Address for the Starterkit than "192.168.3.10" open the block diagram with "CTRL E".



In the first frame of the sequenzdiagramm change the input string for the IP-Address for the input to ReadConfig.vi. Change back to the front panel with “CTRL E” and press the start button to start the application.

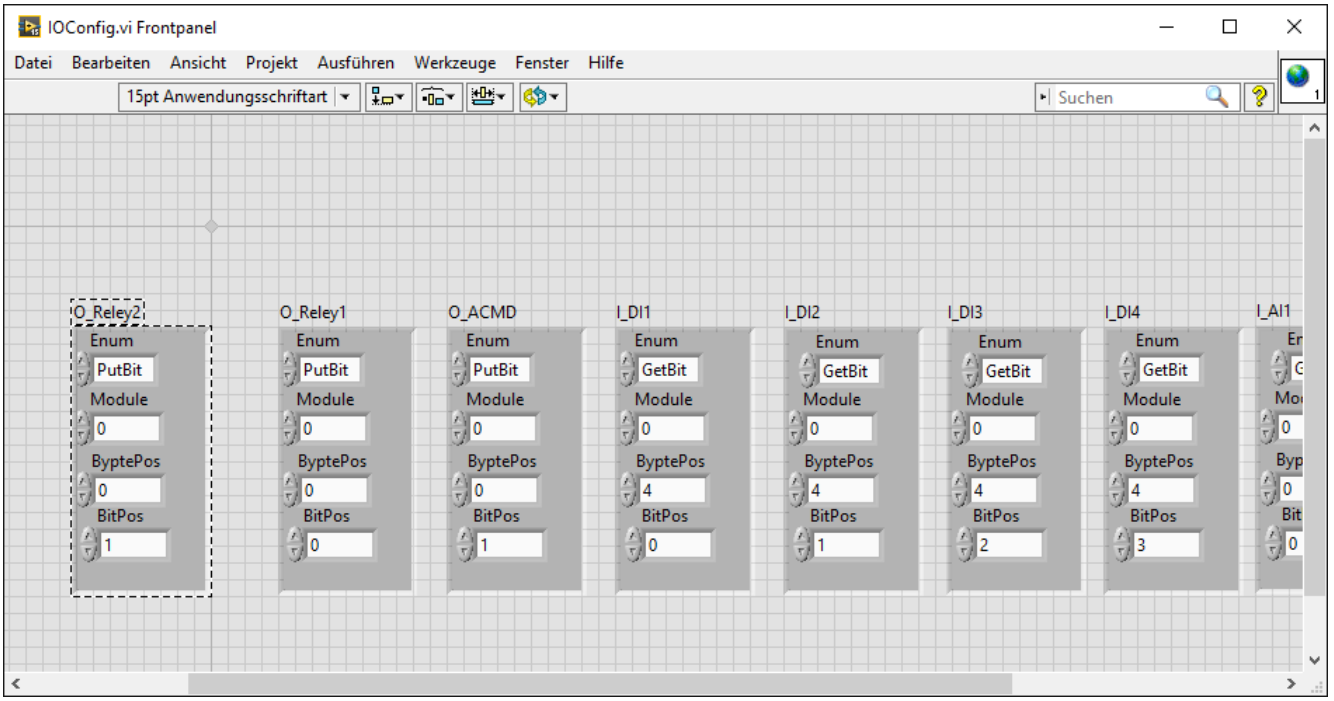


If the “Status of slave modes for the first module is 0, stop and start again. If it is still 0, check the wired connections and whether the IP-Address is correct.
Press the “Stop” button for more than one second to stop the application.

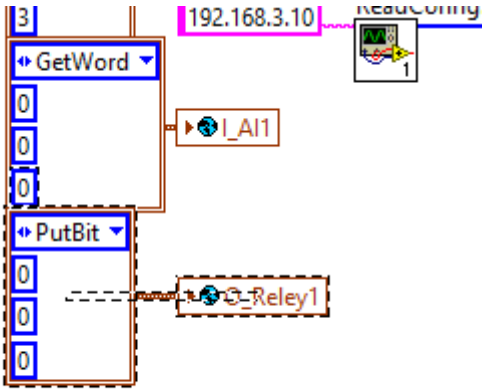
Result: The “Status of slave modes” is 1 for the first module. The application runs and one relay switches each second.

6.5 Create your own Application

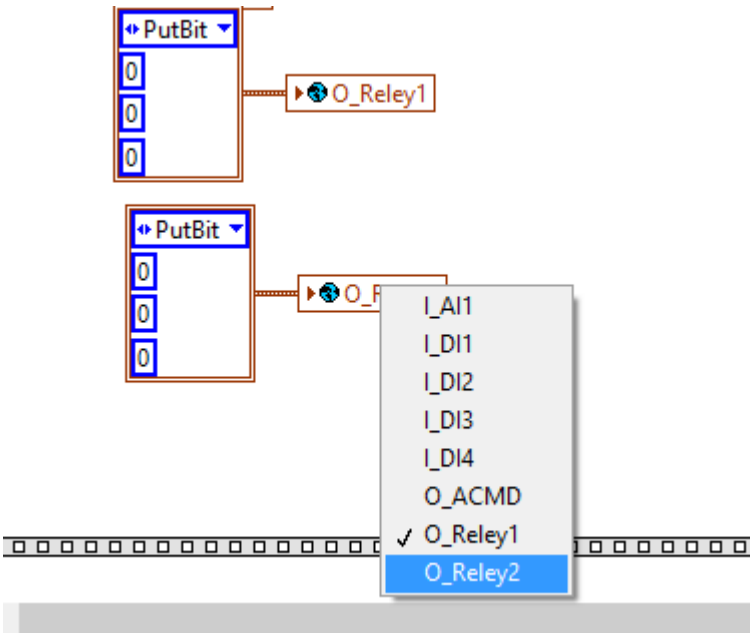
- Open IOConfig.vi and copy O_Reley1 and paste it
- Rename O_Relay1_2 to O_Relay2



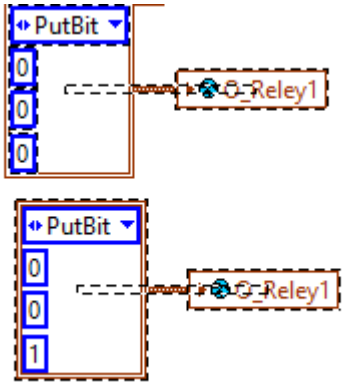
- Go Back to Starterkit.vi and open the block diagram
- Select the constant cluster connected to the O_Relay1 global variable and copy



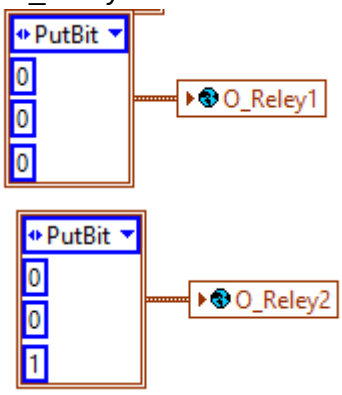
- Paste it under
- Select the O_Relay2 global variable



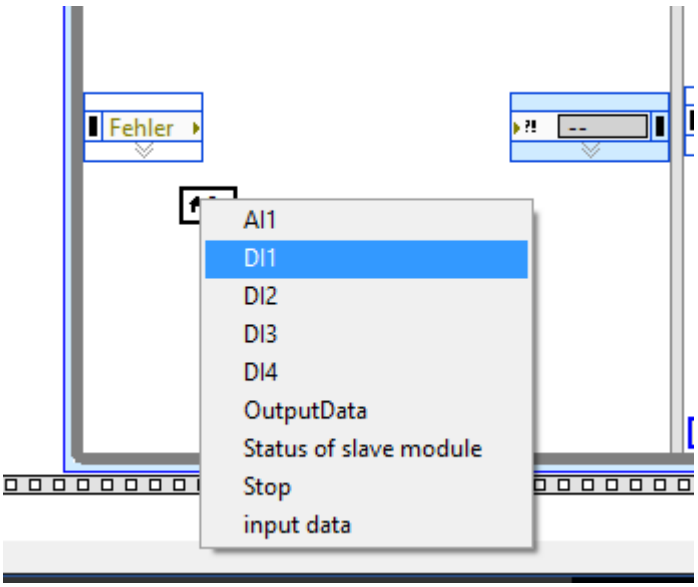
- Change the BitPos constant in the cluster to 1



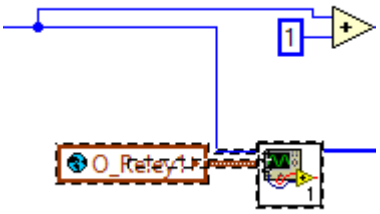
- Select O_Reley2 on the connected global variable



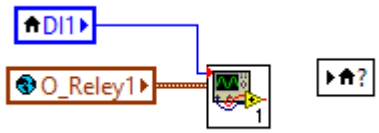
- In the second frame of the sequence, drop a local variable for reading into the timed loop where the “Embrick” vi is called.
- Select DI1



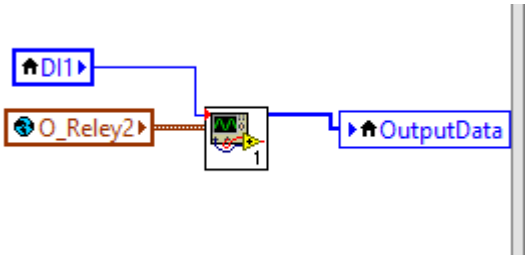
- Copy O_Relay1 with its vi connected to in the other timed loop



- Drop it under the new created local variable “DI1”
- Connect DI1 to the input port “Wert” of the vi
- Place another local variable for writing and select OutputData



- Connect the output port “OutputData” to the new createt local variable
- Select O_Relay2 from the global variable



- Save the application and run it

Result: While one Relay is switching every second, the other one is switching according to the digital input DI1 very fast. Hands on Software – with Labview

7 Hands on Software - with Gamma

7.1 Getting started

7.1.1 Setup the Development Environment

<in preparation>

8 Hands on Software - with Python

8.1 Python (Windows)

8.1.1 Setup the Development Environment

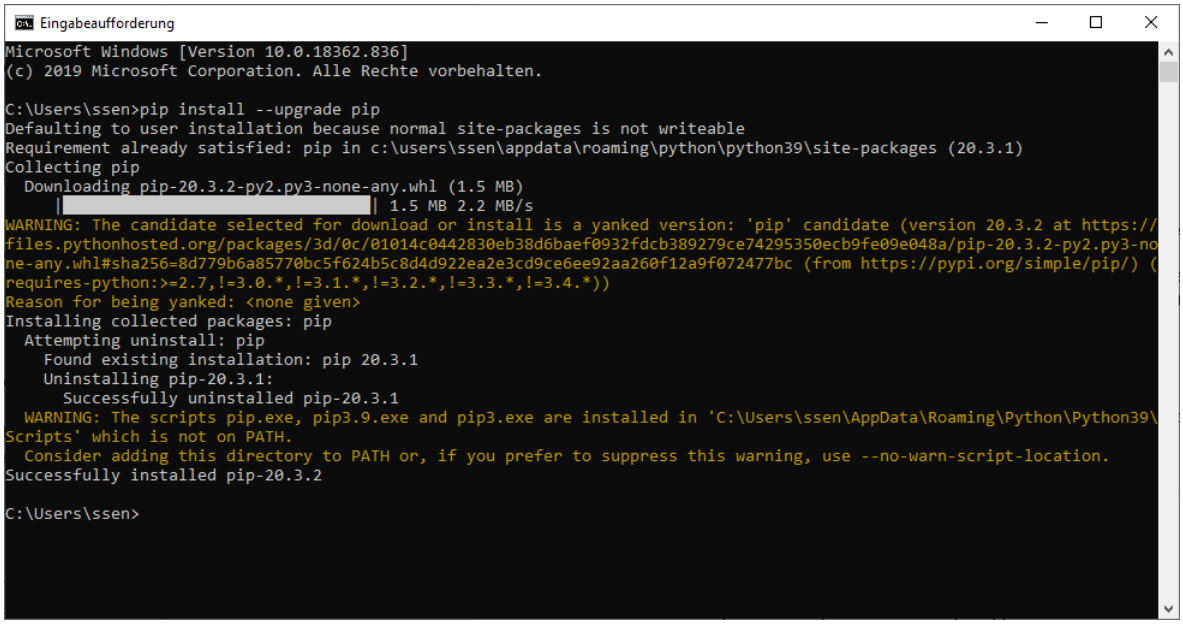
If you have not already Python 3.7 or higher installed, you can get a free version at http://www.python.org/downloads/ and follow the installation instructions.

Result: Now you can create and edit applications and compile them. The software is developed and tested with Python 3.7 and above, previous versions might work.

8.1.2 Installing of the Python Modules

First check if you have installed the newest pip installer. For that we press the “Windows Button + R” and type “cmd” in the opened Window and press “Enter”.

Then we type “pip install –upgrade pip to upgrade the pip installer.



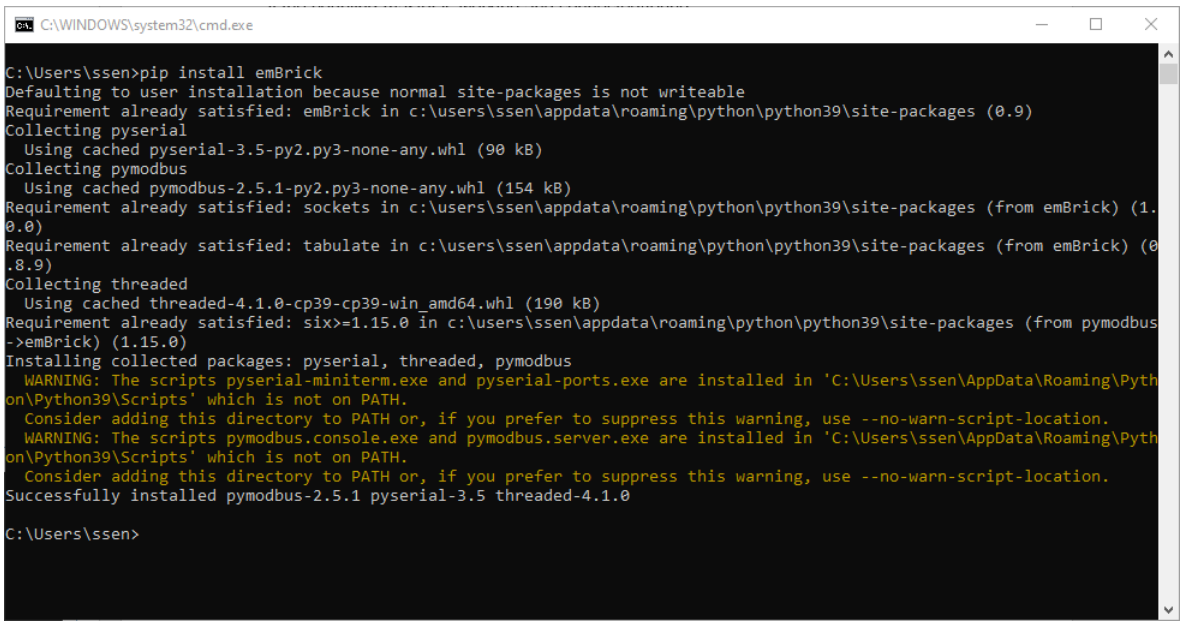
Installing pip

After that we install the needed Modules:

For that we type:

“pip install emBRICK”

and it will automaticly install all modules we needed included the emBRICK Driver for a Communication over Ethernet, Modbus RTU & Modbus TCP.



installing python modules

8.1.3 Download and Unzip the Board Support Package

Download the Python Starterkit from:
Python-starterkit.zip
Extract the zip file to you're a into a new Folder of your choice

The Zip Data contains the Tool netbrick.exe and the Example Applications.
In the Folder examples:

- # Example files for a communication over Ethernet
  - 1node_eth.py
  - 2nodes_threaded_eth.py
  - default_eth.py
  - default_threaded_eth.py
- # Example files for a communication over Modbus RTU (RS485)
  - 1node_rtu.py
  - 2nodes_threaded_rtu.py
  - default_eth.py
  - default_threaded_eth.py
- # Example files for a communication over Modbus TCP/IP
  - 1node_tcp.py
  - 2nodes_threaded_tcp.py
  - default_tcp.py
  - default_threaded_tcp.py

also, the tool *NetBRICK* to explore the LAN environment and search for connected *coupling-masters* (LWCS-Boards).

8.1.4 Check Hardware

Before starting with the software development, check the hardware by switch on the 24V power of LWCS board.

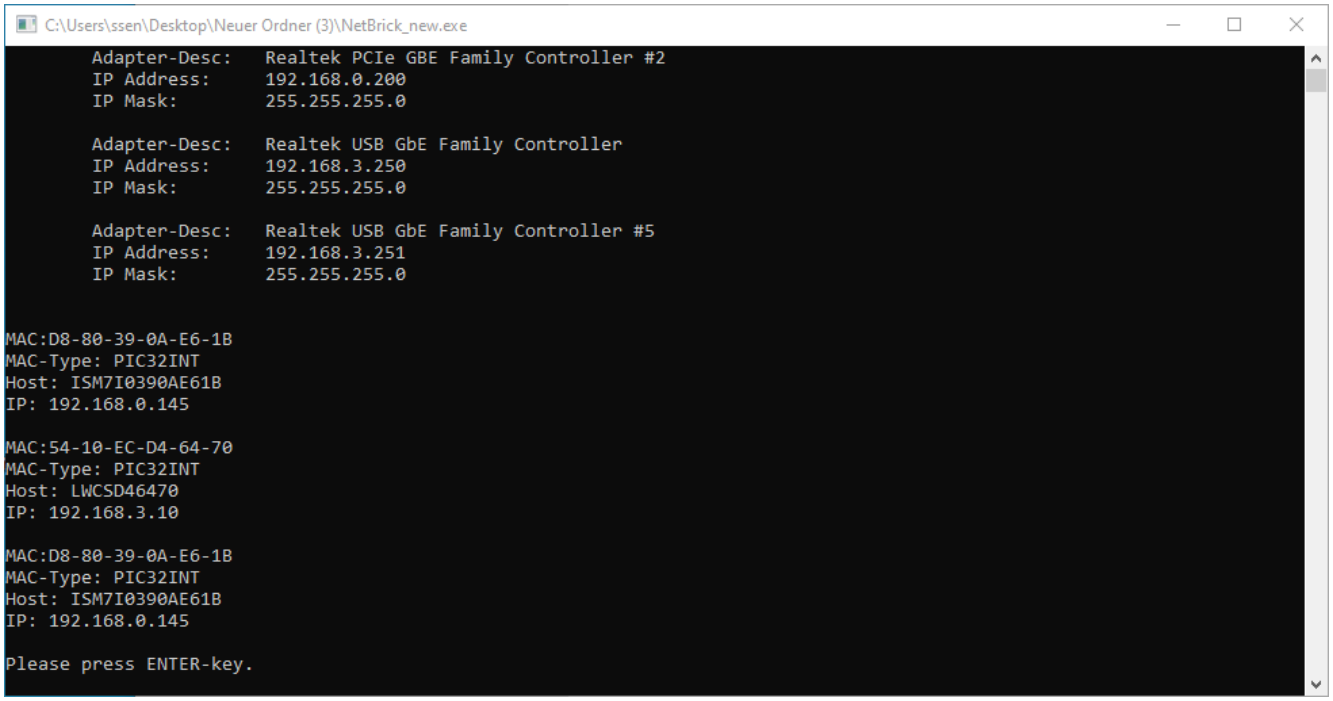
8.1.4.1 Connection over Lan-Adapter

Start the Application "NetBrick.exe".

NetBRICK is a useful tool that checks all network ports from your PC whether there is a *coupling-master* connected with its IP-address. All founded coupling-masters will be listed with their IP addresses. With *NetBRICK* you can simply check ...

- if your PC Ethernet-Adapters are correct configured
- if the *coupling-master* is working and connected/found
- the IP-addresses of the available *coupling-master*

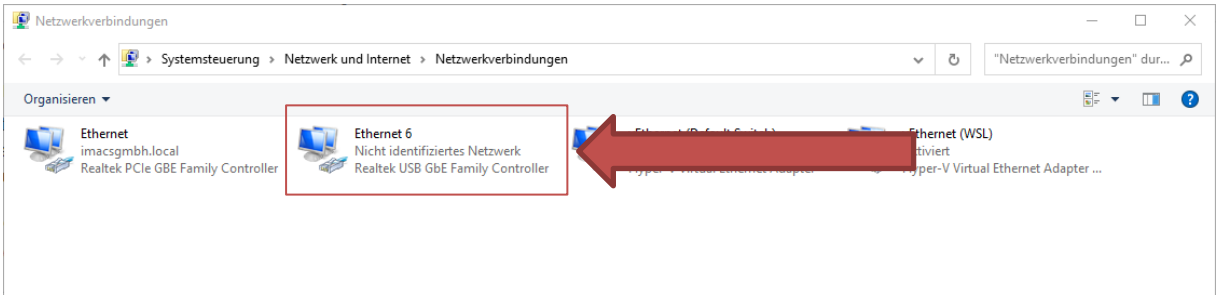
To start, double click on **NetBrick.exe**



starting netbrick.exe

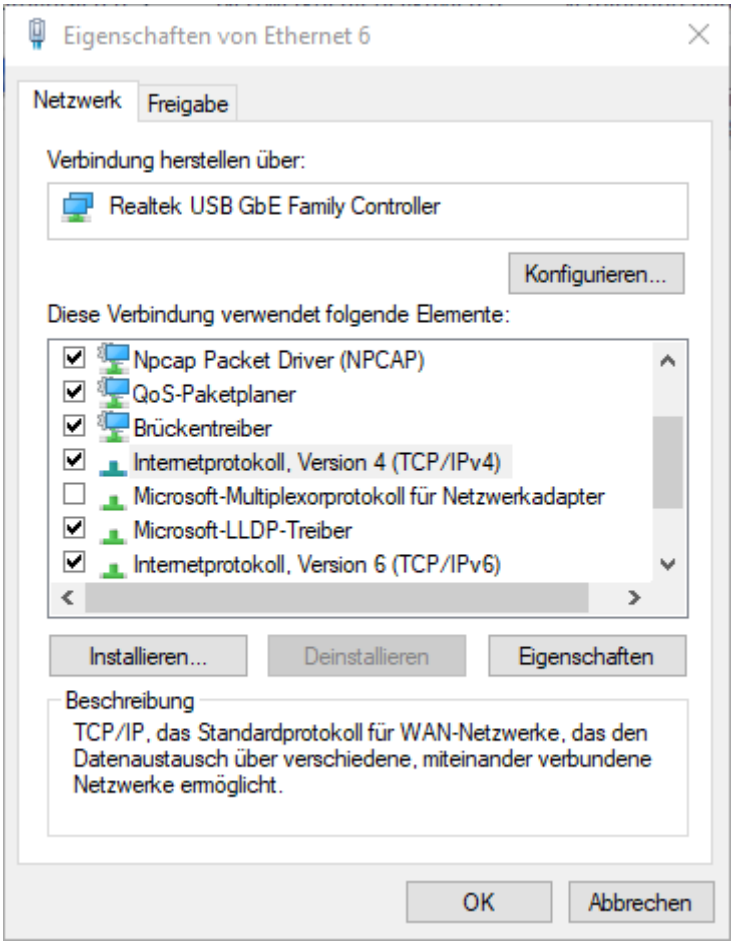
In this picture you can see the output of the *NetBrick* at first all your Ethernet-Adapters are listed and there after comes the detected *coupling-master* with the IP 192.168.3.10.

Is your PC Ethernet-Adapter not correct configured the Netbrick.exe will not detect a coupling-master, to configure the Ethernet-Adapter press “Windows Button + R” on your Keyboard and type “ncpa-cpl”. A window named “Netzwerkverbindungen” will opened.



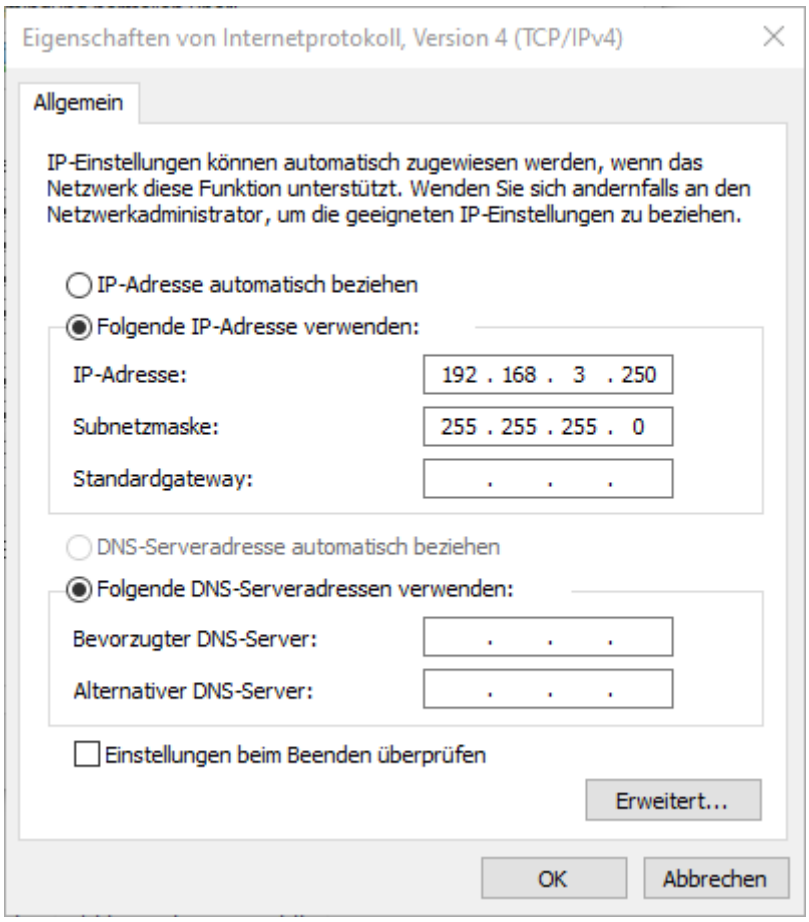
Netzwerkverbindungen

Here we right-click on the Ethernet Adapter and click on “Eigenschaften”.



Configuration of the Ethernet Adapter

In opened Window we Doppel click on “Internetprotokoll, Version 4(TCP/IPv4)” and configured it like in the Screenshot and press on “OK” and save the configuration. If we start the netbrick.exe now. The coupling-master will be shown.

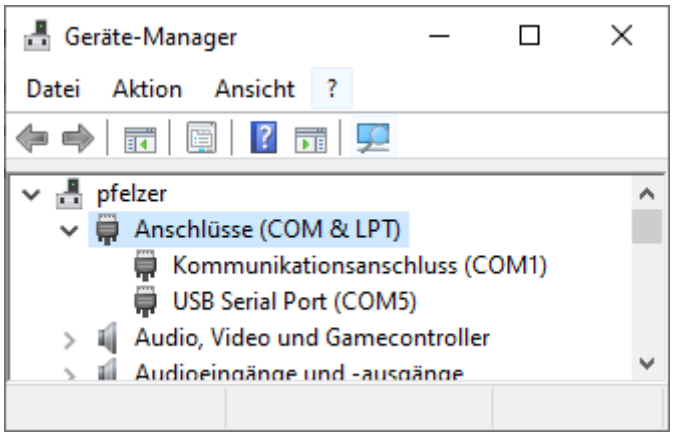


FiguIPV4 Configuration

Result: Now you can access the *coupling-master* and *slave-modules*.

8.1.4.2 Connection over Serial (RS458)

First check with which Com Port is your Serial Adapter connected. For that press “Windows Button + x” on your Keyboard and then on “g” to open the “Geräte-Manager”.



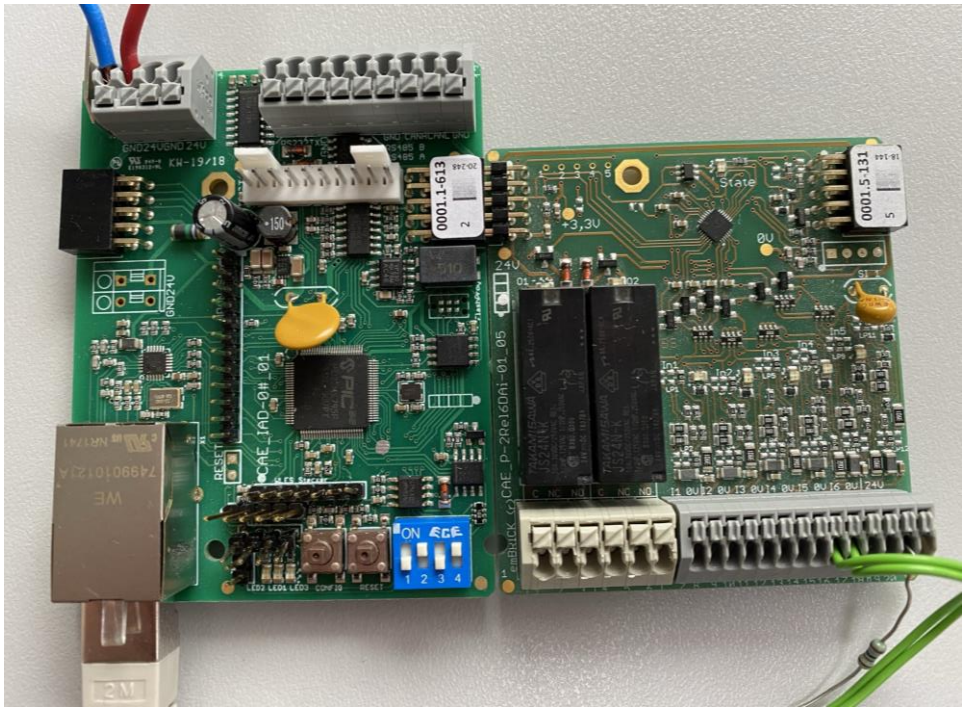
The USB Serial Port is connected on COM5

8.1.5 Load and run the Sample Application

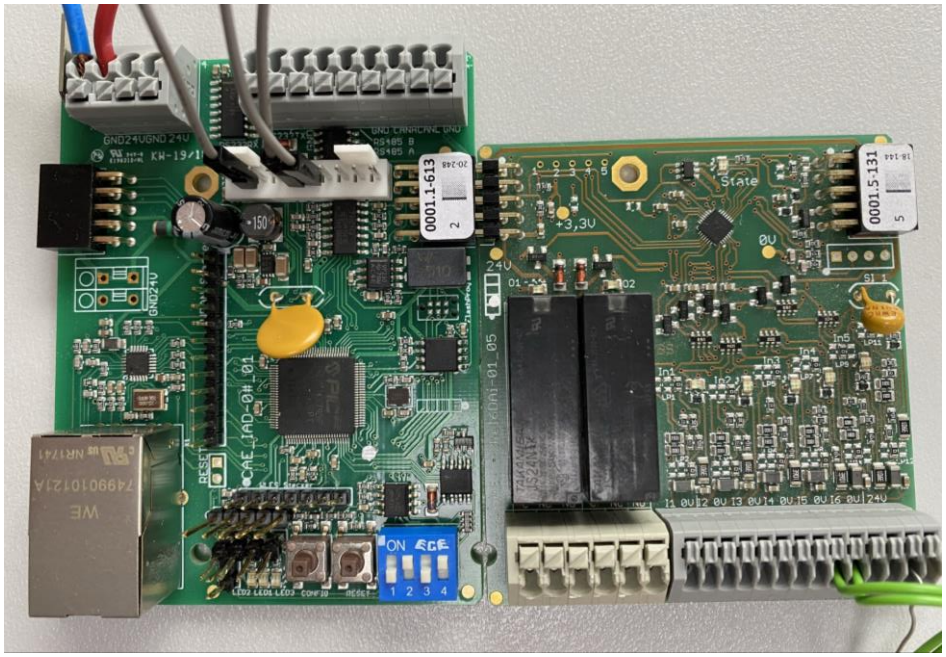
If you test all the Sample Application's you needed following Hardware:

For the Sample Appication's "1node.py" and "1node_threaded.py":

1x Remote Master with the Software Ver. 0.61
1x P-2Rel4Di2Ai-0# Module ID = 5-131



Remote Master with 2Rel4Di Ethernet

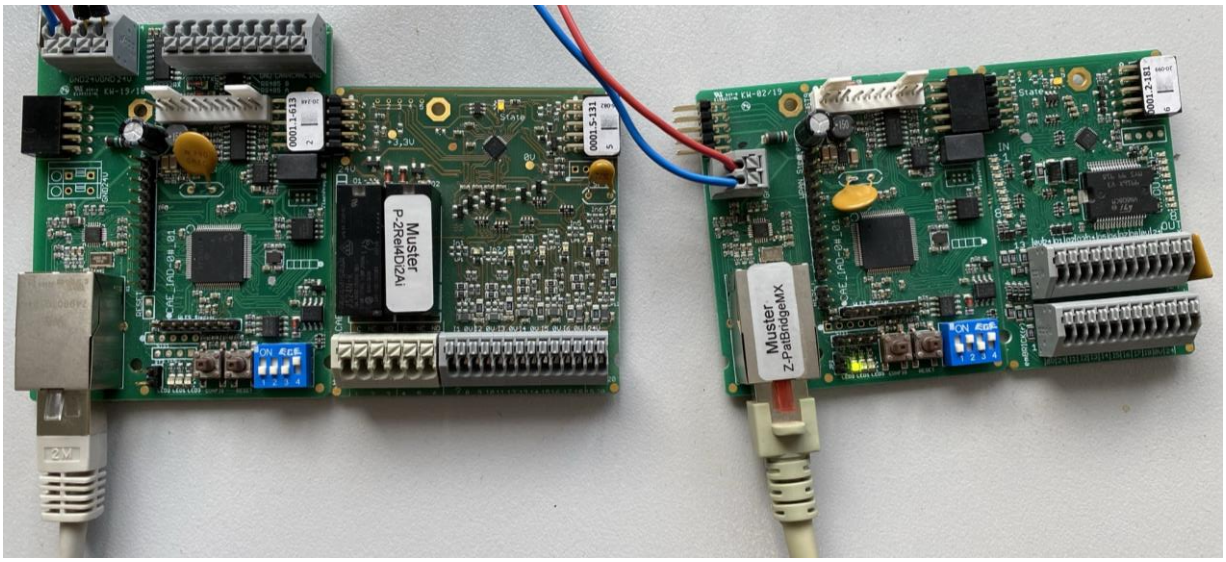


Remote Master with 2Rel4Di Serial

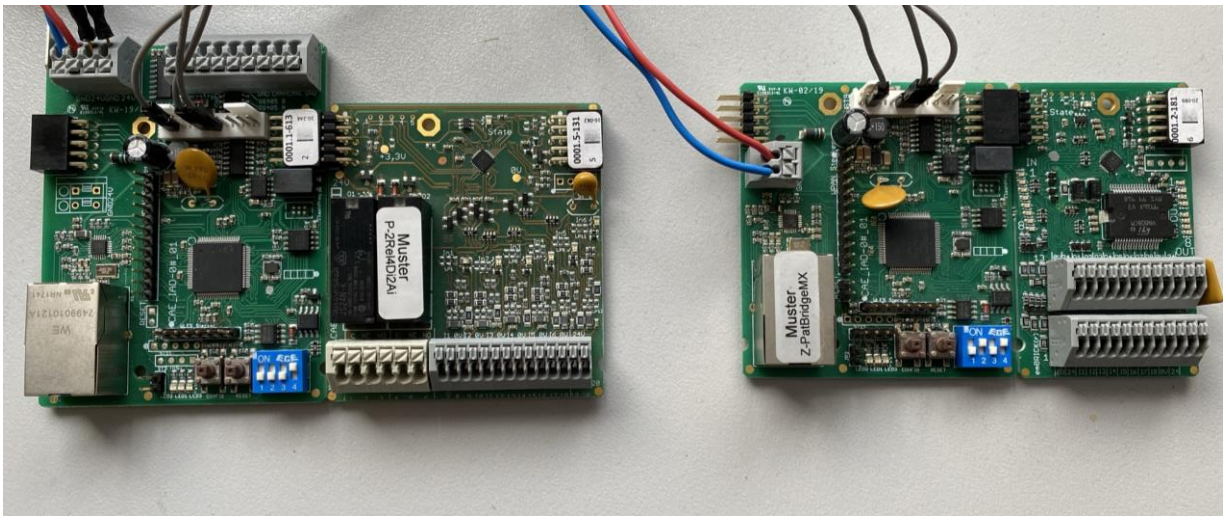
For the Sample Application's "2node.py" and "2node_threaded.py":

Node 1:
1x Remote Master with the Software Ver. 0.61
1x P-2Rel4Di2Ai-0# Module ID = 5-131

Node 2:
1x Remote Master with the Software Ver. 0.61
1x P-2Rel4Di2Ai-0# Module ID = 2-181

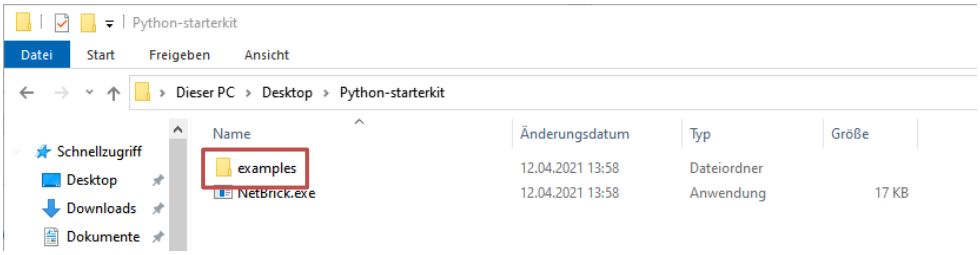


Node1 and Node2 per Ethernet



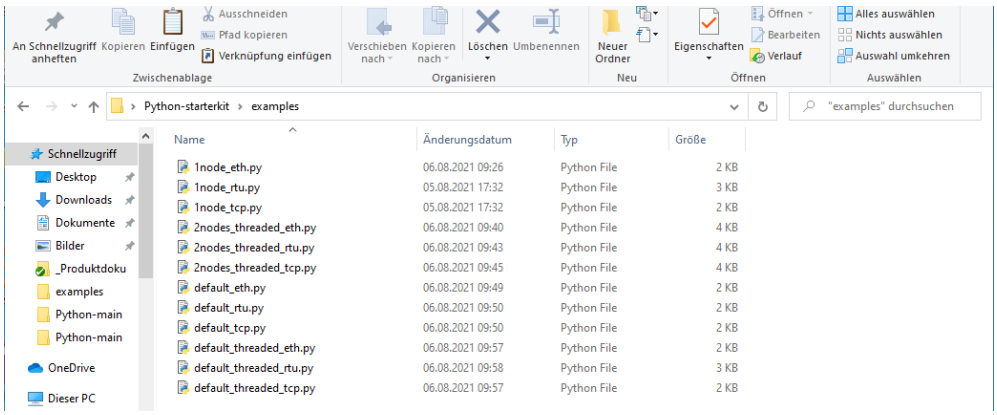
Node1 and Node2 per Serial

In the folder you have unzipped the “Python_starterkit” software package you will find a folder “examples”.



Python_starterkit unzipped

Here you can find all Sample Application’s.

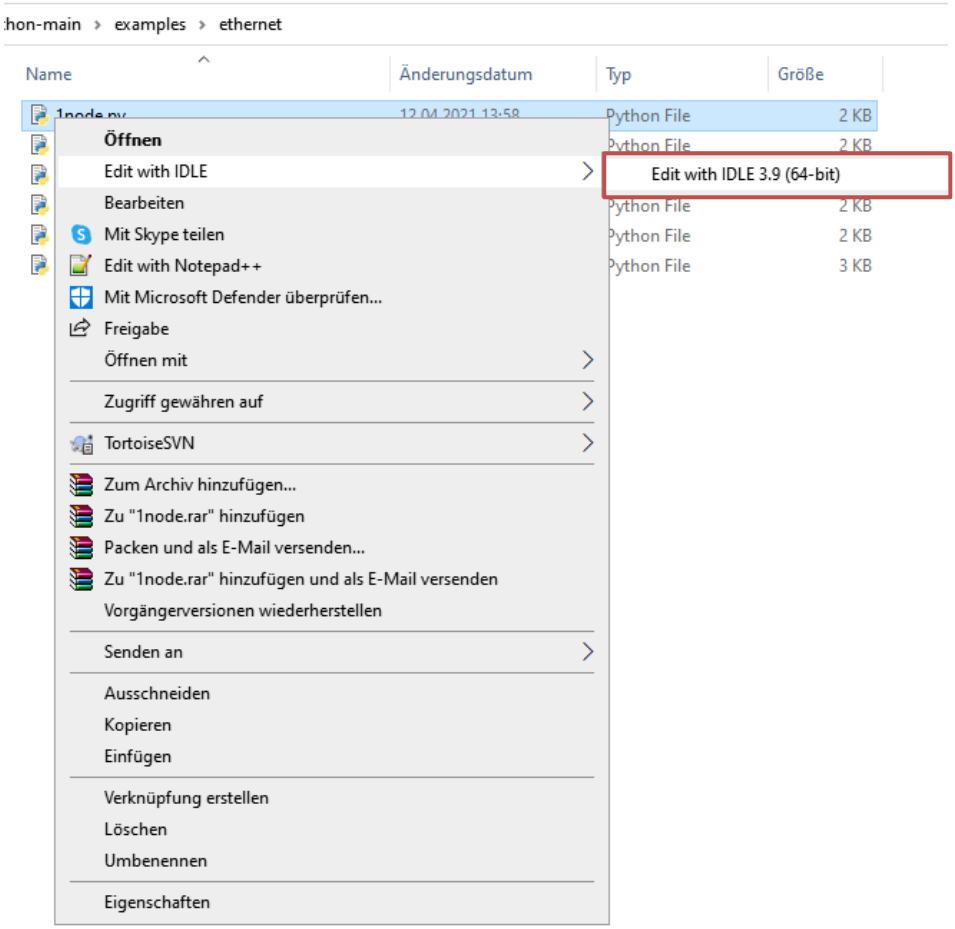


The Sample Application's

8.1.6 Start and explore the Sample Application's

Before starting the Sample Application's, we check and edit the configuration.

First open a Sample Application's with a right-click on it and then "Edit with IDLE" then click on "Edit with IDLE 3.9"

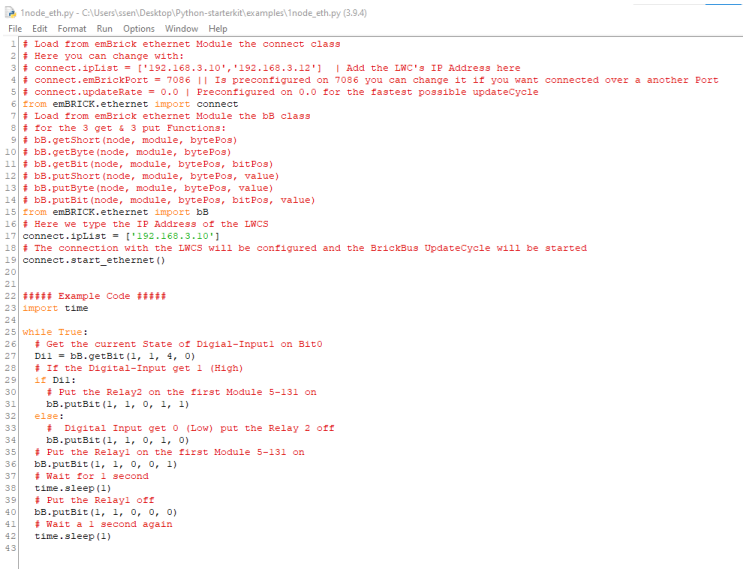


Edit Sample Application

8.1.6.1 Ethernet

8.1.6.1.1 1node_eth.py

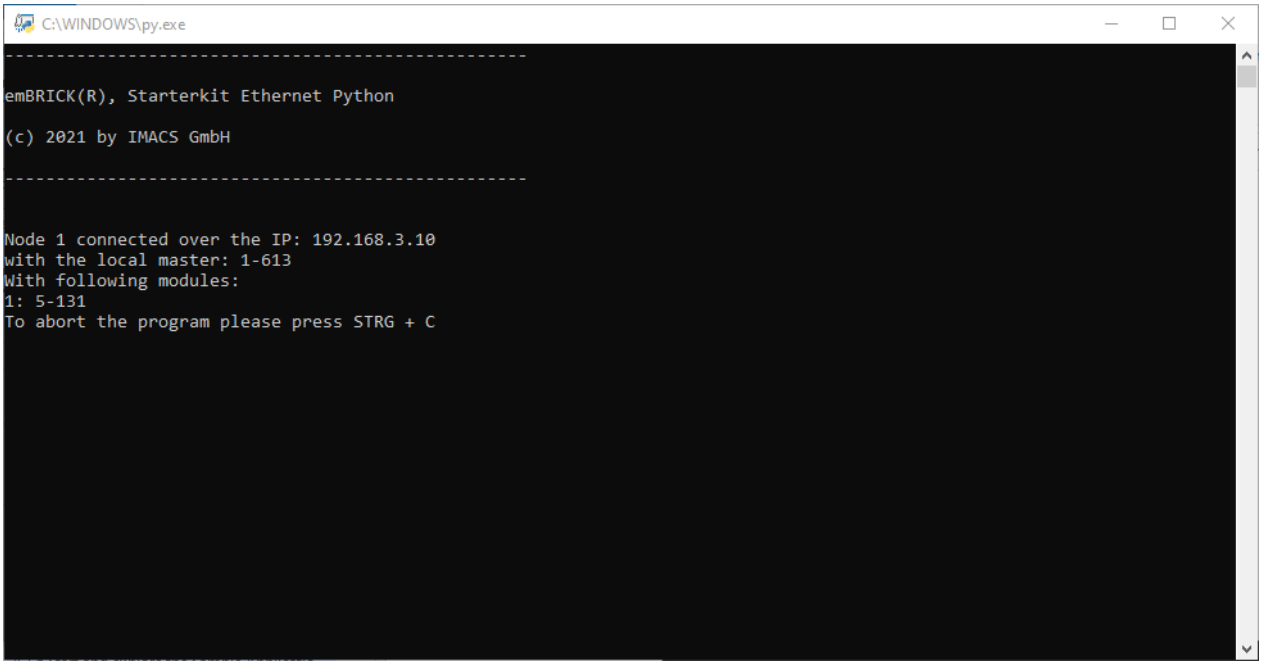
Here we change in the Line 17 the Ip Address if your Coupling Master have a another Ip. After that we save the changes with "Strg + s" and close the window.



```
1node_eth.py - C:\Users\ssen\Desktop\Python-starterkit\examples\1node_eth.py (3.9.4)
File  Edit  Format  Run  Options  Window  Help
# Load from emBrick ethernet Module the connect class
# Here you can change with:
# connect.ipList = ['192.168.3.10','192.168.3.12']  | Add the LWC's IP Address here
# connect.emBrickPort = 7086 || Is preconfigured on 7086 you can change it if you want connected over a another Port
# connect.updateRate = 0.0 | Preconfigured on 0.0 for the fastest possible updateCycle
from emBRICK.ethernet import connect
# Load from emBrick ethernet Module the bB class
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)
from emBRICK.ethernet import bB
# Here we type the IP Address of the LWCS
connect.ipList = ['192.168.3.10']
# The connection with the LWCS will be configured and the BrickBus UpdateCycle will be started
connect.start_ethernet()


##### Example Code #####
import time

while True:
  # Get the current State of Digial-Input1 on Bit0
  Di1 = bB.getBit(1, 1, 4, 0)
  # If the Digital-Input get 1 (High)
  if Di1:
    # Put the Relay2 on the first Module 5-131 on
    bB.putBit(1, 1, 0, 1, 1)
  else:
    #  Digital Input get 0 (Low) put the Relay 2 off
    bB.putBit(1, 1, 0, 1, 0)
  # Put the Relay1 on the first Module 5-131 on
  bB.putBit(1, 1, 0, 0, 1)
  # Wait for 1 second
  time.sleep(1)
  # Put the Relay1 off
  bB.putBit(1, 1, 0, 0, 0)
  # Wait a 1 second again
  time.sleep(1)
```

1node_eth.py Ethernet

After that we can double click on the .py file and the Sample Application will be started.



```
C:\WINDOWS\py.exe
--------------------------------------------------

emBRICK(R), Starterkit Ethernet Python

(c) 2021 by IMACS GmbH

--------------------------------------------------


Node 1 connected over the IP: 192.168.3.10
with the local master: 1-613
With following modules:
1: 5-131
To abort the program please press STRG + C
```

Running 1node.py Ethernet

Result:

Now the application is started and has read out the Local Master ID, module ID and it´s Ip Address of all connected Coupling Master's and modules.

In the Sample Application "1node_eth.py" the Relay1 will be goes on and off every second.

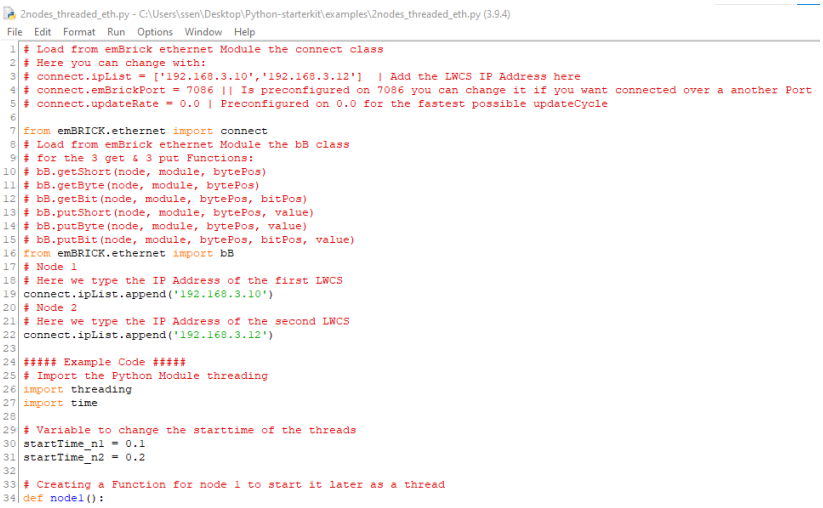
To exit, press [Ctrl+C] or you click on the x from the window.

For more details about the hardware, see *Product Calatogue* (search for "Z-CouplingBrick-02", "P-2Rel4Di2Ai-01", "CAE_G-8Di8Do-01").

8.1.6.1.2 2nodes_threaded_eth.py

Here we change in the Line 19 the Ip Address from node 1 if your Coupling Masters have a another Ip Address and the Line 22 for the node 2.

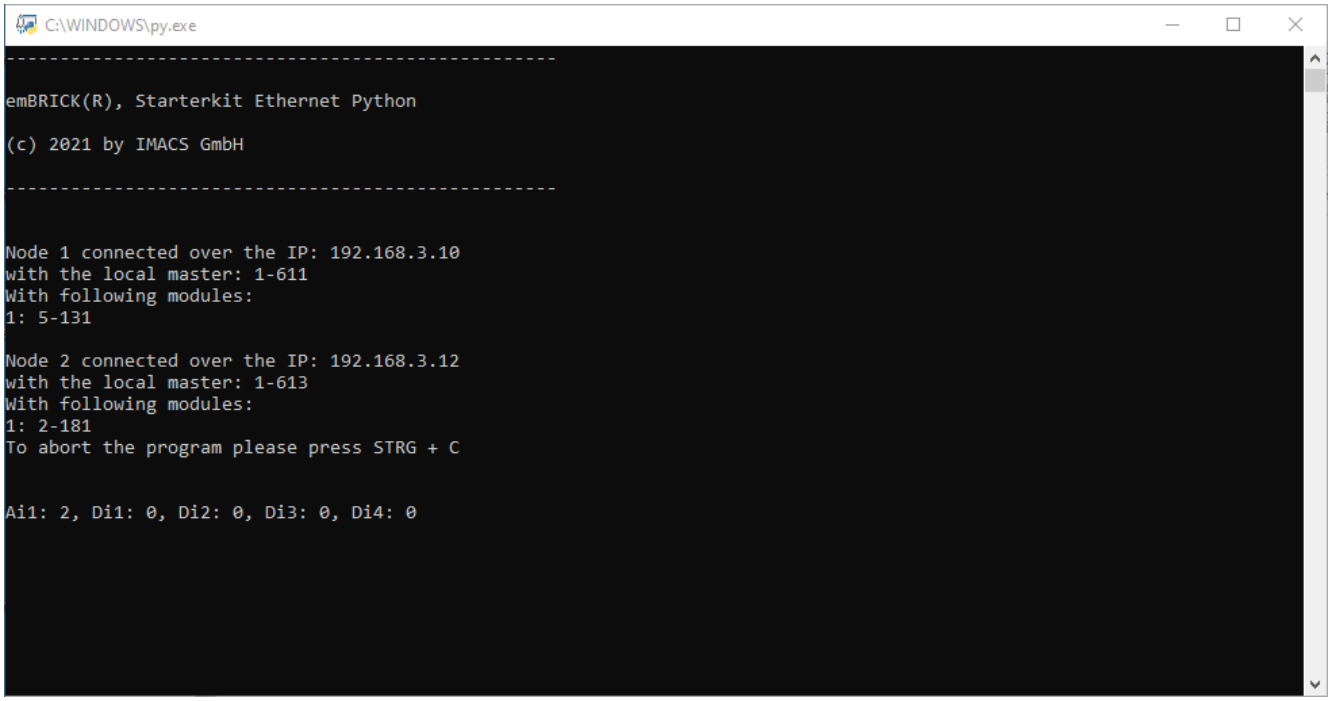
After that we save the changes with “Strg + s” and close the window.



```
2nodes_threaded_eth.py - C:\Users\ssen\Desktop\Python-starterkit\examples\2nodes_threaded_eth.py (3.9.4)
File  Edit  Format  Run  Options  Window  Help
# Load from emBrick ethernet Module the connect class
# Here you can change with:
# connect.ipList = ['192.168.3.10','192.168.3.12']  | Add the LWCS IP Address here
# connect.emBrickPort = 7086 || Is preconfigured on 7086 you can change it if you want connected over a another Port
# connect.updateRate = 0.0 | Preconfigured on 0.0 for the fastest possible updateCycle

from emBRICK.ethernet import connect
# Load from emBrick ethernet Module the bB class
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)
from emBRICK.ethernet import bB
# Node 1
# Here we type the IP Address of the first LWCS
connect.ipList.append('192.168.3.10')
# Node 2
# Here we type the IP Address of the second LWCS
connect.ipList.append('192.168.3.12')

##### Example Code #####
# Import the Python Module threading
import threading
import time

# Variable to change the starttime of the threads
startTime_n1 = 0.1
startTime_n2 = 0.2

# Creating a Function for node 1 to start it later as a thread
def node1():
```

2_nodes_threaded_eth.py

After that we can double click on the .py file and the Sample Application will be started.



```
C:\WINDOWS\py.exe
--------------------------------------------------

emBRICK(R), Starterkit Ethernet Python

(c) 2021 by IMACS GmbH

--------------------------------------------------


Node 1 connected over the IP: 192.168.3.10
with the local master: 1-611
With following modules:
1: 5-131

Node 2 connected over the IP: 192.168.3.12
with the local master: 1-613
With following modules:
1: 2-181
To abort the program please press STRG + C


Ai1: 2, Di1: 0, Di2: 0, Di3: 0, Di4: 0
```

Result:

Now the application is started and has read out the Local Master ID, module ID and it´s Ip Address of all connected Coupling Master’s and modules.

In “2nodes_threaded_eth.py” we connect the Remote Master with 2 Coupling Master and communicated with both simultaneously.

In first node, the one with module P-2Rel4Di2Ai-0#, we read the Analog Input 1 and the four Digital Inputs out and print them out. And if the Digital Input 1 goes high, the Relay 1 goes on for Seconds then off again. The same with Digital Input 2 but there goes the Relay 2 on and in 2 Seconds off again.

The second node, the one with module G-8Di8Do-01, there goes all DigitalOutputs high if the Digital Input 1 is high, else the DigitalOutputs are all low.

To exit, press [Ctrl+C] or you click on the x from the window.

For more details about the hardware, see *Product Calatogue* (search for "Z-CouplingBrick-02", “P-2Rel4Di2Ai-01”, “CAE_G-8Di8Do-01”).

8.1.6.2 Modbus RTU

8.1.6.2.1 1node_rtu

In Line 24, if you have another Port than “COM5” you muss change it in the Port you have. And in Line 32 you can edit the Modbus Address of your Coupling-Master.

```
1node_rtu.py - C:\Users\ssen\Desktop\Python-starterkit\examples\1node_rtu.py (3.9.4)
File  Edit  Format  Run  Options  Window  Help
# Load from emBrick serial Module the connect class
from emBRICK.modbus_rtu import connect
# Here you can change with:
# connect.port = "COM*" or "/dev/ttySC0" |your port
# connect.unit_id = [1,2] |your connected nodes
# connect.timeout = 0.1  | to change the Timeout for the Modbus Connection
# connect.baudrate = 460800 | to change the Baudrate
# connect.updateRate = 0.005 | for +5 ms that will be slow down the updateCycle between LWCS and RemoteMaster
# Load from emBrick serial Module the bB class
from emBRICK.modbus_rtu import bB
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)

# With os we can detect, which Operation system we use
import os
# nt is the Operation System "Windows"
if os.name == "nt":
# Windows: check please your COM port in "Geräte-Manger" and change the Port if you have a another Port
  connect.port = "COM5"
# posix is the Operation System "LINUX"
elif os.name == "posix":
# RaspberryPi: Here is the Port always the same if you use our PiBrick Driver
  connect.port = "/dev/ttySC0"
# The baudrate is preconfigured to 460800 if you want u can change it
#connect.baudrate = 460800
# Configure here with which Modbus Address/es the Programm should connect
connect.unit_id = [1]
# The connection with the LWCS will be configured and the BrickBus UpdateCycle will be started
connect.start()


##### Example Code #####
import time

while True:
  # Get the current State of Digial-Input1 on Bit0
  led1 = bB.getBit(1, 1, 4, 0)
  # If the Digital-Input get 1 (High)
  if led1:
    # Put the Relay2 on the first Module 5-131 on
    bB.putBit(1, 1, 0, 1, 1)
  else:
    #  Digital Input get 0 (Low) put the Relay 2 off
    bB.putBit(1, 1, 0, 1, 0)
  # Put the Relay1 on the first Module 5-131 on
  bB.putBit(1, 1, 0, 0, 1)
  # Wait for 1 second
  time.sleep(1)
  # Put the Relay1 off
  bB.putBit(1, 1, 0, 0, 0)
  # Wait a 1 second again
  time.sleep(1)
```

Figure 1 1node_rtu.py Serial

After that we can double click on the .py file and the Sample Application will be started.

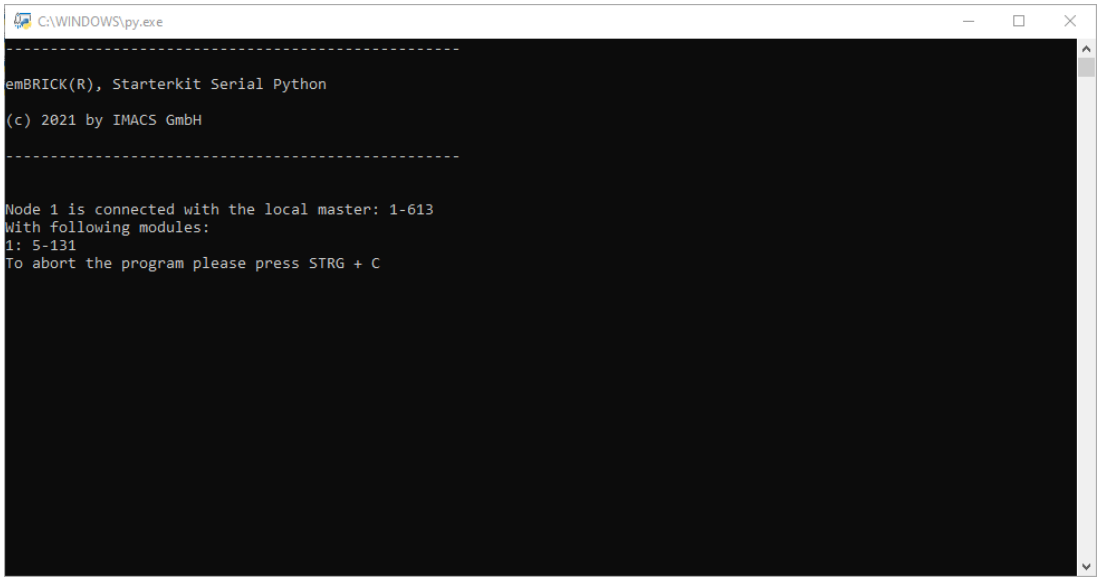


Figure 2 Running 1node_rtu.py Modbus RTU

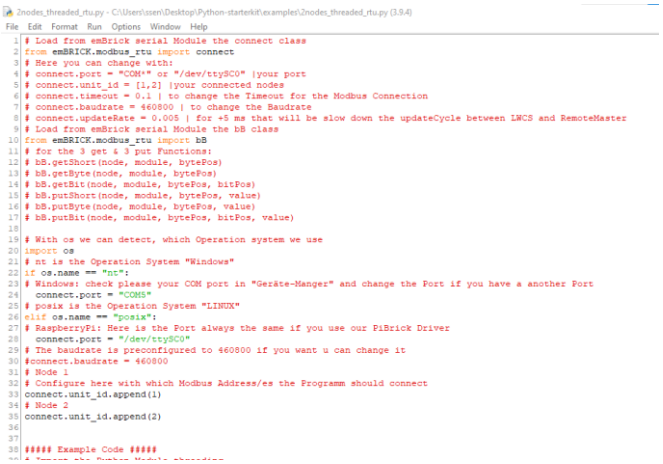
Result:
Now the application is started and has read out the Local Master ID, module ID and it´s Ip Address of all connected Local Master's and modules.

In the Sample Application "1node_rtu.py" the Relay1 will be goes on and off every second.

To exit, press [Ctrl+C] or you click on the x from the window.

For more details about the hardware, see *Product Calatogue* (search for "Z-CouplingBrick-02", "P-2Rel4Di2Ai-01", "CAE_G-8Di8Do-01").

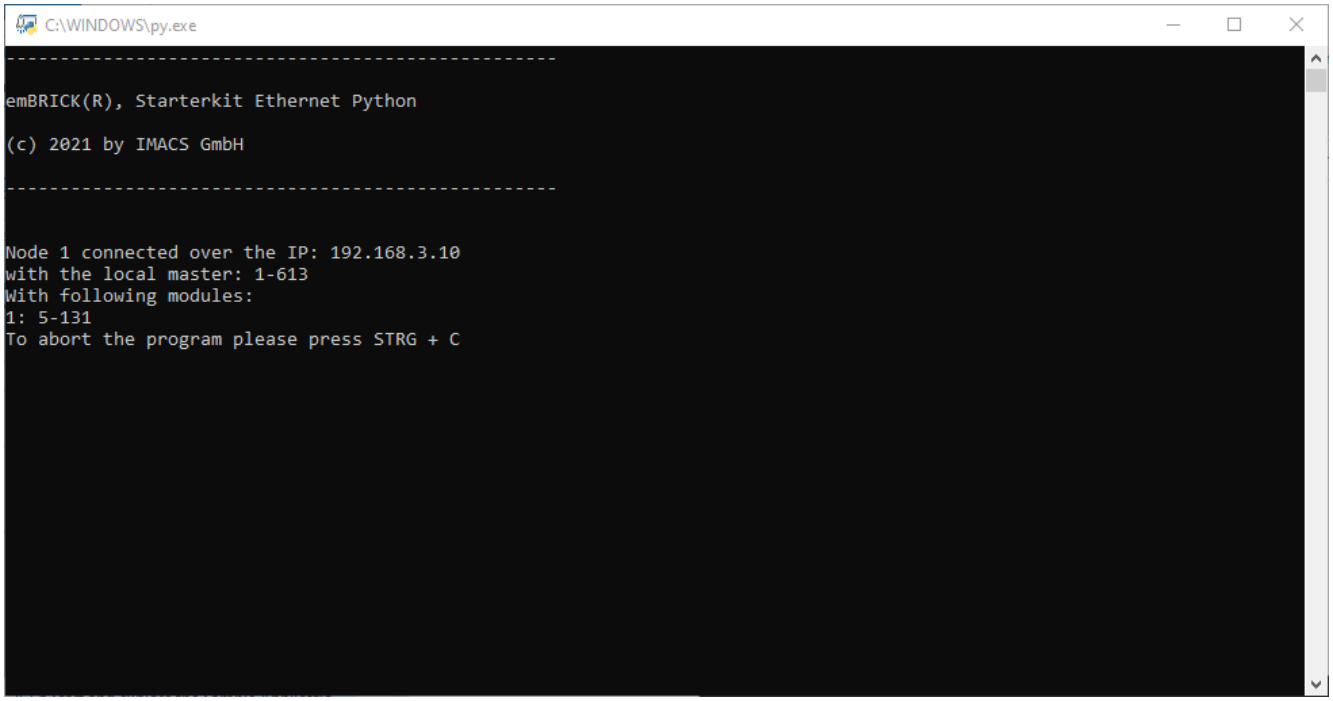
8.1.6.2.2 2nodes_threaded_rtu.py



```
# Load from emBrick serial Module the connect class#
from emBRICK.modbus_rtu import connect
# Here you can change with:
# connect.port = "COM*" or "/dev/ttySC0" |your port
# connect.unit_id = [1,2] |your connected nodes
# connect.timeout = 0.1 | to change the Timeout for the Modbus Connection
# connect.baudrate = 460800 | to change the Baudrate
# connect.updateRate = 0.005 | for +5 ms that will be slow down the updateCycle between LWCS and RemoteMaster
# Load from emBrick serial Module the bB class
from emBRICK.modbus_rtu import bB
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)

# With os we can detect, which Operation system we use
import os
# nt is the Operation System "Windows"
if os.name == "nt":
# Windows: check please your COM port in "Geräte-Manger" and change the Port if you have a another Port
  connect.port = "COM5"
# posix is the Operation System "LINUX"
elif os.name == "posix":
# RaspberryPi: Here is the Port always the same if you use our PiBrick Driver
  connect.port = "/dev/ttySC0"
# The baudrate is preconfigured to 460800 if you want u can change it
#connect.baudrate = 460800
# Node 1
# Configure here with which Modbus Address/es the Programm should connect
connect.unit_id.append(1)
# Node 2
connect.unit_id.append(2)


##### Example Code #####
```

Figure 3 2nodes_threaded_rtu.py

After that we can double click on the .py file and the Sample Application will be started.



Result:
Now the application is started and has read out the Local Master ID, module ID and it´s Ip Address of all connected Coupling Master's and modules.

In “2nodes_threaded_eth.py” we connect the Remote Master with 2 Coupling Master and communicated with both simultaneously.
In first node, the one with module P-2Rel4Di2Ai-0#, we read the Analog Input 1 and the four Digital Inputs out and print them out. And if the Digital Input 1 goes high, the Relay 1 goes on for Seconds then off again. The same with Digital Input 2 but there goes the Relay 2 on and in 2 Seconds off again.
The second node, the one with module G-8Di8Do-01, there goes all DigitalOutputs high if the Digital Input 1 is high, else the DigitalOutputs are all low.

To exit, press [Ctrl+C] or you click on the x from the window.

For more details about the hardware, see *Product Calatogue* (search for "Z-CouplingBrick-02", “P-2Rel4Di2Ai-01”, “CAE_G-8Di8Do-01”).

8.1.6.3 Mobus TCP

8.1.6.3.1 1node_rtu

In Line 22, if your Coupling-Master have a another Ip Address than “192.168.3.10” you should change it in the correct Ip Address you have. Save the modification with “Strg + s”.

1node_tcp.py - C:\Users\ssen\Desktop\Python-starterkit\examples\1node_tcp.py (3.9.4)

File Edit Format Run Options Window Help

```
# Load from emBrick serial Module the connect class
from emBRICK.modbus_tcp import connect
# Here you can change with:
# connect.ip_address = ['192.168.3.10','192.168.3.12'] | the Ip Addresses of the LWCS
# connect.ipPort = [502, 6965] | the Ip Port of the LWCS
# connect.timeout = 0.001 | to change the Timeout for the Modbus Connection
# connect.unit_id = 1 | is preconfigured to the default value 1, is the Modbus Address of the LWCS
# connect.updateRate = 0.0 | Preconfigured on 0.0 for the fastest possible updateCycle

# Load from emBrick serial Module the bB class
from emBRICK.modbus_tcp import bB
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)

#Config for Node
#LWCS IP Address
connect.ip_address = ["192.168.3.10"]
#Ip Port
connect.ipPort = [502]
# The connection with the LWCS will be configured and the BrickBus UpdateCycle will be started
connect.start()


##### Example Code #####
import time

while True:
  # Get the current State of Digial-Input1 on Bit0
  led1 = bB.getBit(1, 1, 4, 0)
  # If the Digital-Input get 1 (High)
  if led1:
    # Put the Relay2 on the first Module 5-131 on
    bB.putBit(1, 1, 0, 1, 1)
  else:
    #  Digital Input get 0 (Low) put the Relay 2 off
    bB.putBit(1, 1, 0, 1, 0)
  # Put the Relay1 on the first Module 5-131 on
  bB.putBit(1, 1, 0, 0, 1)
  # Wait for 1 second
  time.sleep(1)
  # Put the Relay1 off
  bB.putBit(1, 1, 0, 0, 0)
  # Wait a 1 second again
  time.sleep(1)

```

Figure 4 1node_tcp.py Serial

After that we can double click on the .py file and the Sample Application will be started.

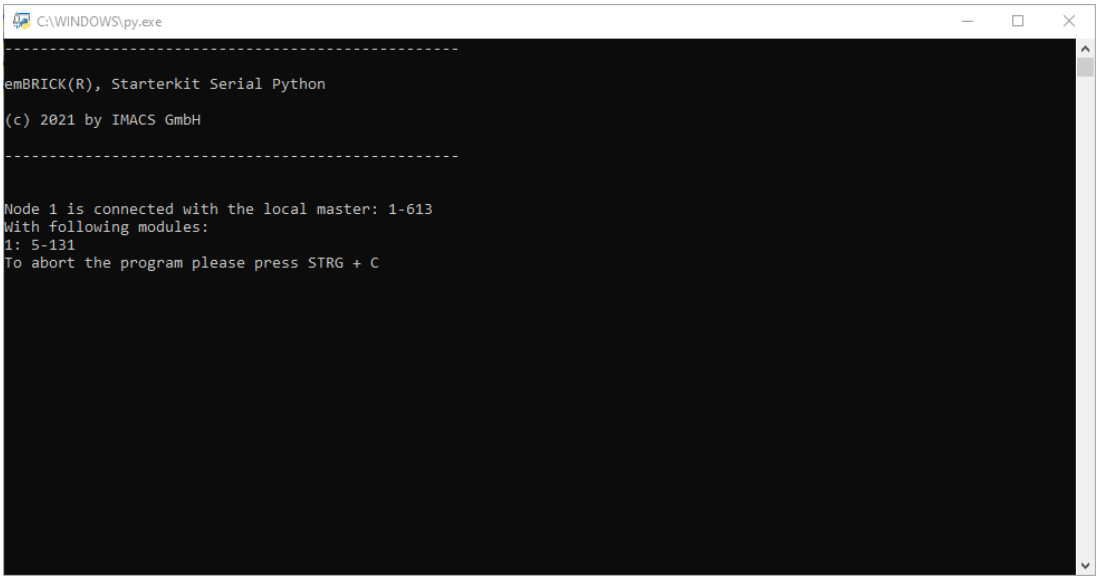


Figure 5 Running 1node_tcp.py Modbus TCP

Result:

Now the application is started and has read out the Local Master ID, module ID and it´s Ip Address of all connected Local Master's and modules.

In the Sample Application "1node_rtu.py" the Relay1 will be goes on and off every second.

To exit, press [Ctrl+C] or you click on the x from the window.

For more details about the hardware, see *Product Calatogue* (search for "Z-CouplingBrick-02", "P-2Rel4Di2Ai-01", "CAE_G-8Di8Do-01").

8.1.6.3.2 2nodes_threaded_rtu

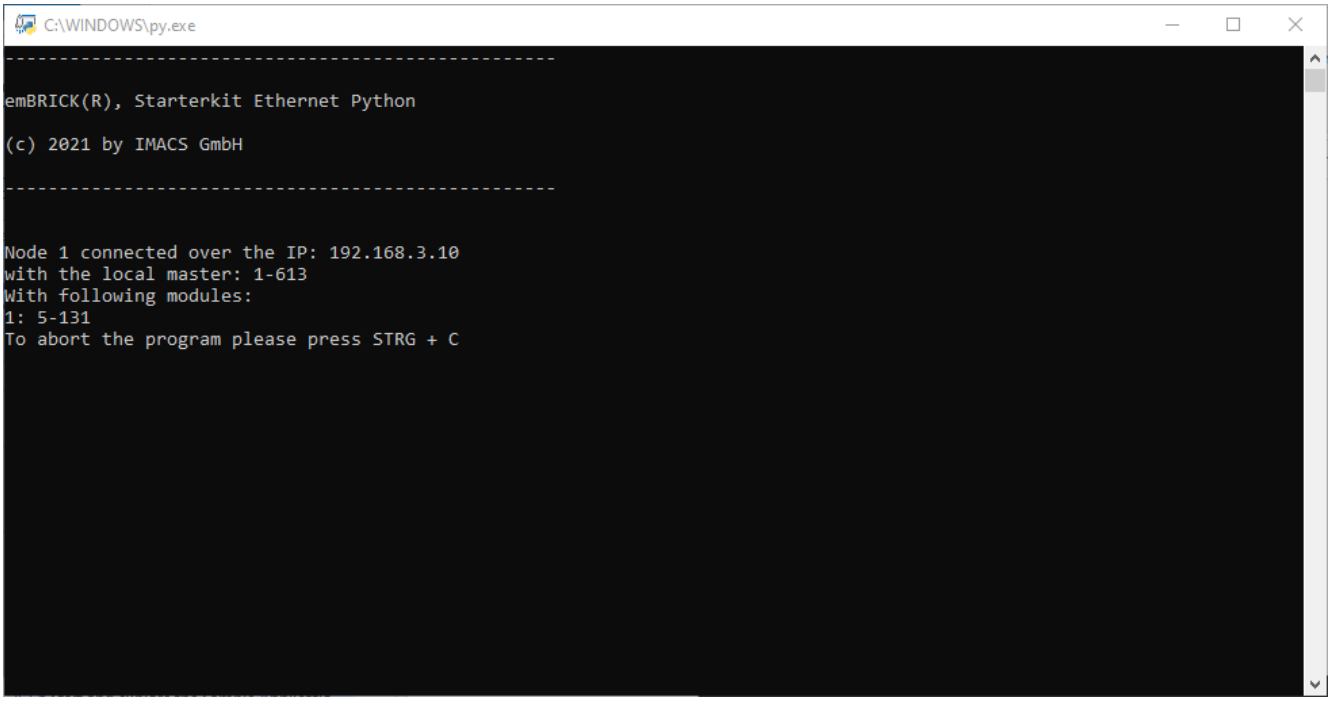
2nodes_threaded_rtu.py - C:\Users\ssen\Desktop\Python-starterkit\examples\2nodes_threaded_rtu.py (3.9.4)

File Edit Format Run Options Window Help

```
# Load from emBrick serial Module the connect class
from emBRICK.modbus_rtu import connect
# Here you can change with:
# connect.port = "COM*" or "/dev/ttySC0" |your port
# connect.unit_id = [1,2] |your connected nodes
# connect.timeout = 0.1 | to change the Timeout for the Modbus Connection
# connect.baudrate = 460800 | to change the Baudrate
# connect.updateRate = 0.005 | for +5 ms that will be slow down the updateCycle between LWCS and RemoteMaster
# Load from emBrick serial Module the bB class
from emBRICK.modbus_rtu import bB
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)

# With os we can detect, which Operation system we use
import os
# nt is the Operation System "Windows"
if os.name == "nt":
# Windows: check please your COM port in "Geräte-Manger" and change the Port if you have a another Port
  connect.port = "COM5"
# posix is the Operation System "LINUX"
elif os.name == "posix":
# RaspberryPi: Here is the Port always the same if you use our PiBrick Driver
  connect.port = "/dev/ttySC0"
# The baudrate is preconfigured to 460800 if you want u can change it
#connect.baudrate = 460800
# Node 1
# Configure here with which Modbus Address/es the Programm should connect
connect.unit_id.append(1)
# Node 2
connect.unit_id.append(2)


##### Example Code #####
```

Figure 6 2nodes_threaded_rtu

After that we can double click on the .py file and the Sample Application will be started.



Result:

In “2nodes_threaded_rtu.py” we connect the Remote Master with 2 Coupling Master and communicated with both simultaneously.
In first node, the one with module P-2Rel4Di2Ai-0#, we read the Analog Input 1 and the four Digital Inputs out and print them out. And if the Digital Input 1 goes high, the Relay 1 goes on for Seconds then off again. The same with Digital Input 2 but there goes the Relay 2 on and in 2 Seconds off again.
The second node, the one with module G-8Di8Do-01, there goes all DigitalOutputs high if the Digital Input 1 is high, else the DigitalOutputs are all low.

To exit, press [Ctrl+C] or you click on the x from the window.

For more details about the hardware, see *Product Calatogue* (search for "Z-CouplingBrick-02", “P-2Rel4Di2Ai-01”, “CAE_G-8Di8Do-01”).

8.1.7 Create your own application

This sub chapter describes how create your own application based on the “Sample Application” descripted above.

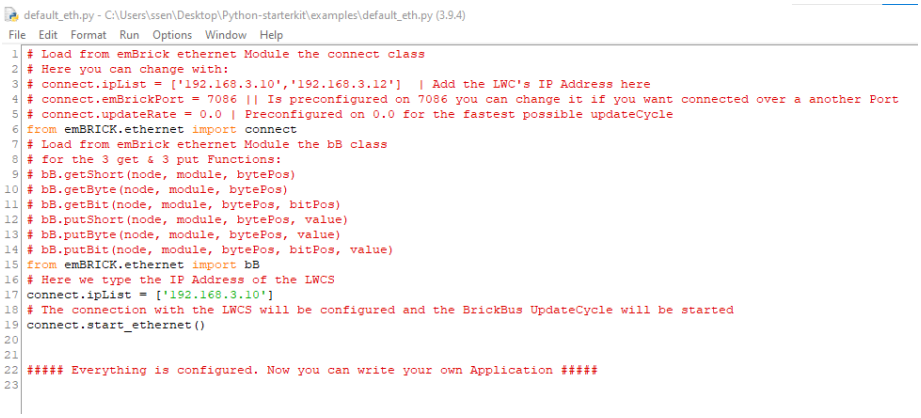
Preparation:

- Open the python file
  - “default_eth.py” or “default_threaded_eth.py” (for ethernet)
  - “default_rtu.py” or “default_threaded_rtu.py” (for Modbus RTU)
  - “default_tcp.py” pr “default_threaded_tcp.py” (for Modbus TCP)

  from example folder with the Python Idle with right click on the file and go on “Edit with Idle and click on “Edit with IDLE 3.9”.

  The “default_eth.py” & “default_rtu.py” & “default_tcp.py” works only really with one node, because we have only one thread for updates and a while loop for the communication and in the while loop the function order will be called in a row.

8.1.7.1 “default_eth.py”:



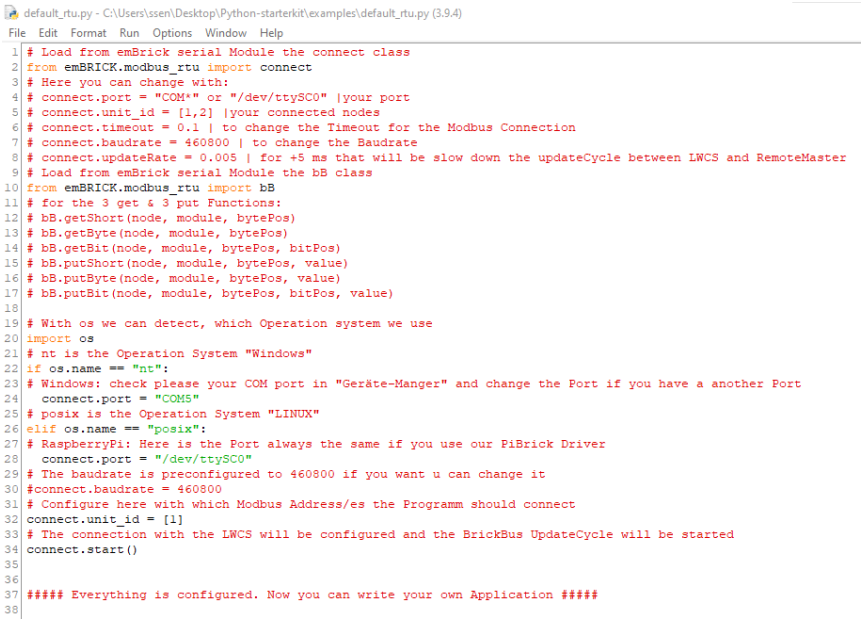
```
default_eth.py - C:\Users\ssen\Desktop\Python-starterkit\examples\default_eth.py (3.9.4)
File  Edit  Format  Run  Options  Window  Help
# Load from emBrick ethernet Module the connect class
# Here you can change with:
# connect.ipList = ['192.168.3.10','192.168.3.12']  | Add the LWC's IP Address here
# connect.emBrickPort = 7086 || Is preconfigured on 7086 you can change it if you want connected over a another Port
# connect.updateRate = 0.0 | Preconfigured on 0.0 for the fastest possible updateCycle
from emBRICK.ethernet import connect
# Load from emBrick ethernet Module the bB class
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)
from emBRICK.ethernet import bB
# Here we type the IP Address of the LWCS
connect.ipList = ['192.168.3.10']
# The connection with the LWCS will be configured and the BrickBus UpdateCycle will be started
connect.start_ethernet()


##### Everything is configured. Now you can write your own Application #####

```

Change in Line 17 the Ip Address if you give your Coupling Master another IP Address.
Add after Line 17 “connect.emBrickPort = ****”, if you want connect with another Port than 7086.
If you want modify the Updaterate, then add before “connect.start_ethernet()”, a Line with “connec.updateRate = 0.1 to change the UpdateCycle (0.1 stands for 100ms).

- Than write your own code after the comment (“Everything is configured. Now you can write your own Application”)

1. Write the wished Function example “bB.putBit(1, 1, 0, 1, 1)” The meaning of 1, 1, 0, 1, 1 is explained in Chapter 4.7.4.2 IO-Access Functions. The meaning of 1, 1, 0, 1, 1 is explained in Chapter 8.8.4.2 IO-Access Functions.
2. Save the change with “Strg + s”.
3. Now you can run your Application.

8.1.7.2 “default_rtu.py”:



```
default_rtu.py - C:\Users\ssen\Desktop\Python-starterkit\examples\default_rtu.py (3.9.4)
File  Edit  Format  Run  Options  Window  Help
# Load from emBrick serial Module the connect class
from emBRICK.modbus_rtu import connect
# Here you can change with:
# connect.port = "COM*" or "/dev/ttySC0" |your port
# connect.unit_id = [1,2] |your connected nodes
# connect.timeout = 0.1 | to change the Timeout for the Modbus Connection
# connect.baudrate = 460800 | to change the Baudrate
# connect.updateRate = 0.005 | for +5 ms that will be slow down the updateCycle between LWCS and RemoteMaster
# Load from emBrick serial Module the bB class
from emBRICK.modbus_rtu import bB
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)

# With os we can detect, which Operation system we use
import os
# nt is the Operation System "Windows"
if os.name == "nt":
# Windows: check please your COM port in "Geräte-Manger" and change the Port if you have a another Port
  connect.port = "COM5"
# posix is the Operation System "LINUX"
elif os.name == "posix":
# RaspberryPi: Here is the Port always the same if you use our PiBrick Driver
  connect.port = "/dev/ttySC0"
# The baudrate is preconfigured to 460800 if you want u can change it
#connect.baudrate = 460800
# Configure here with which Modbus Address/es the Programm should connect
connect.unit_id = [1]
# The connection with the LWCS will be configured and the BrickBus UpdateCycle will be started
connect.start()


##### Everything is configured. Now you can write your own Application #####

```

Change in Line 24 (Windows) or Line 28 (Linux) the COM Port number, if your Modbus Cable is plugged in another COM Port.
Change in Line 32 the Modbus Address if your Coupling Master have a another.
If you want use another baudrate then 460800 add a Line “connect.baudrate = 57600 (300 – 1000000) after Line 32.
If you want modify the Updaterate, then add before “connect.start ()”, a Line with “connec.updateRate = 0.1 to change the UpdateCycle (0.1 stands for 100ms).
If you add the Line with “connect.timeout = 0.1 (100ms)” before “connec.start(), it will abort the connection if in the configured time no communication occur between Remote Master and Coupling Master.

- Than write your own code after the comment (“Everything is configured. Now you can write your own Application”)

4. Write the wished Function example “bB.putBit(1, 1, 0, 1, 1)” The meaning of 1, 1, 0, 1, 1 is explained in Chapter 4.7.4.2 IO-Access Functions. The meaning of 1, 1, 0, 1, 1 is explained in Chapter 8.8.4.2 IO-Access Functions.
5. Save the change with “Strg + s”.
6. Now you can run your Application.

8.1.7.3 “default_tcp.py”:

default_tcp.py - C:\Users\ssen\Desktop\Python-starterkit\examples\default_tcp.py (3.9.4)

File Edit Format Run Options Window Help

```
# Load from emBrick serial Module the connect class
from emBRICK.modbus_tcp import connect
# Here you can change with:
# connect.ip_address = ['192.168.3.10','192.168.3.12'] | the Ip Addresses of the LWCS
# connect.ipPort = [502, 6965] | the Ip Port of the LWCS
# connect.timeout = 0.001 | to change the Timeout for the Modbus Connection
# connect.unit_id = 1 | is preconfigured to the default value 1, is the Modbus Address of the LWCS
# connect.updateRate = 0.0 | Preconfigured on 0.0 for the fastest possible updateCycle

# Load from emBrick serial Module the bB class
from emBRICK.modbus_tcp import bB
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)

#Config for Node
#LWCS IP Address
connect.ip_address = ["192.168.3.10"]
#Ip Port
connect.ipPort = [502]
# The connection with the LWCS will be configured and the BrickBus UpdateCycle will be started
connect.start()


##### Everything is configured. Now you can write your own Application #####

```

Change in Line 22 the Ip Address, if you give your Coupling Master another Ip Address
Change in Line 24 the Ip Port, if your Coupling Master configured to another Ip Port.
If you want modify the Updaterate, then add before “connect.start ()”, a Line with “connec.updateRate = 0.1 to change the UpdateCycle (0.1 stands for 100ms)
If you add the Line with “connect.timeout = 0.1 (100ms)” before “connec.start(), it will abort the connection if in the configured time no communication occur between Remote Master and Coupling Master

-
- Than write your own code after the comment (“Everything is configured. Now you can write your own Application”)

7. Write the wished Function example “bB.putBit(1, 1, 0, 1, 1)” The meaning of 1, 1, 0, 1, 1 is explained in Chapter 8.1.8 IO-Access Functions.
8. Save the change with “Strg + s”.
9. Now you can run your Application.

8.1.8 IO-Access Functions

The data of the I/O *slave-modules* are organized in a byte buffer for each node (a separate one of in- and out-data). To access this data, you need to define the ...

node number..... (here 1-32 node/s),

module number (1...)..... position of IO-module in the node (emBRICK-*string)*

offset_position(0...)..... relative position/offset of the data inside the module image. For details of each module refer to *Product Catalogue*, chapter 6.x.x., "process data image"

bit_position (0..7)..... **only** in case of a bit access, indicates the bit in the selected byte

The actual data access is performed by 6 simple functions and that differ in the direction (reading/writing) and the data width (bit, byte, word). Of course also own functions can be developed to do this.

data reading (from IO-modules to application):

bB.getBit(node, module, offset_pos, bit_pos)	**return the value of the bit**
bB.getByte(node, module, offset_pos)	**return the value of the byte**
bB.getShort (node, module, byte_pos)	**return the value of the short(word)**

writing (from the application to the IO-modules):

bB.putBit(node, module, offset_pos, bit_pos, value)	**set the bit to given value**
bB.putByte(node, module, offset_pos, value)	**set the byte to given value**
bB.putShort (node, module, offset_pos, value)	**set the short(word) to given value**

About their exact parameters and their return value, refer to the comments/description inside the files ***ethernet.py, modbus_rtu.py or modbus_tcp***, where they are defined and implemented.

Note: Access to the byte buffer is already buffered and secured by mutexes.

8.1.9 The Python Remote-Bus Driver

This driver provides a simple but efficient remote-bus access to the connected emBRICK strings (string = Coupling-Master + n x I/O-modules), further described as “node” via Ethernet (TCP/IP).

It is written in Python to allow an easy adaption/integration into own code/projects, although it is possible to create some of the supported code.

8.1.9.1 Features

The driver supports:

- Establishing the connection to the node(s) that is (are) connected to your PC or Raspberry Pi via Ethernet or via Modbus RTU(RS485) or via Modbus TCP.
- Read the configuration data of each node and its connected bricks
- Read and write I/O-data to each emBRICK node (and its bricks)

8.1.9.2 Mode of operation

Therefore, the actual native SPI update process (local emBRICK Bus) is controlled by the coupling master, the operation via this driver (remote emBRICK bus) contains only a few simply steps.

The actual data exchange is managed via a separate input and output buffer (shared memory). After the initialisation a permanent triggered process have to be called to execute the update function. Parallel to this, a set of simple access functions (in Python) allows a synchronized reading/writing access to the I/O-values.

During the initialization the node returns miscellaneous config-data to the driver that are used to locate the start of each I/O-module in the buffer.

8.1.9.3 Involved File

The Folder “emBRICK” contains the driver.

The driver is built from three files:

ethernet.py

In the ethernet.py all functions and modules are declared which parameters the functions need and they are given back for a communication over Ethernet.

Contains the functions are called from the application.

modbus_rtu.py

In the modbus_rtu.py all functions are declared which parameters the functions need an they are given back for a communication over Modbus RTU (RS485).

Contains the functions are called from the application.

modbus_tcp.py

In the modbus_rtu.py all functions are declared which parameters the functions need an they are given back for a communication over Modbus TCP.

Contains the functions are called from the application.

8.1.9.4 Basic implementation

In an own application the following steps have to be implemented:

8.1.9.4.1 Initializing and Starting

Initializing and starting the I/O-update has been split into two function groups

Initialize from driver:

- **from emBRICK.ethernet import connect** for Ethernet or
- **from emBRICK.modbus_rtu import connect** for Modbus RTU or
- **from emBRICK.modbus_tcp import connect** for Modbus TCP.

To import the class emBrickConnection from the installed emBRICK Module. With that we can configured the connection with the Coupling Master.

Initialize from driver:

- **from emBRICK.ethernet import bB** for Ethernet or
- **from emBRICK.modbus_rtu import bB** for Modbus RTU or
- **from emBRICK modbus_tcp import bB** for Modbus TCP

To import the class emBrickFunctions. In this are 6 functions for read and write the In- and Outputs from modules.

b) After initializing, the user can change **updateRate.** updateRate is for in which periodically you want to read and write the In- and Outputs.

8.1.9.4.2 IO-Access Functions

The data of the I/O *slave-modules* are organized in a byte buffer for each node (a separate one of in- and out-data). To access this data, you need to define the ...

node number..... (here the number of Local Master's),

module number (1...)..... position of IO-module in the node (emBRICK-*string)*

offset_position(0...)..... relative position/offset of the data inside the module image. For details of each module refer to *Product Catalogue*, chapter 6.x.x., "process data image"

bit_position (0..7)..... **only** in case of a bit access, indicates the bit in the selected byte

The actual data access is performed by 6 simple functions and that differ in the direction (reading/writing) and the data width (bit, byte, word). Of course, also own functions can be developed to do this.

data reading (from IO-modules to application):

bB.getBit(node, module, offset_pos, bit_pos)

bB.getByte(node, module, offset_pos)

bB.getShort (node, module, byte_pos)

writing (from the application to the IO-modules):

bB.putBit(node, module, offset_pos, bit_pos, value)

bB.putByte(node, module, offset_pos, value)

bB.putShort (node, module, offset_pos, value)

8.2 Python (on Raspberry Pi OS)

8.2.1 Setup the Development Environment

First check with the command **"python3 -V"** or **"python3 –version".**
If a compatible Python software is installed

```
pi@raspberrypi:~ $ python3 --version
Python 3.7.3
```

Figure 7 Python3 Version check

If no Python3.6 or above is installed.
You can install or update Python3 with the following command:
" sudo apt-get install python3" "

```
pi@raspberrypi:~ $ sudo apt-get install python3
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version (3.7.3-1).
python3 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 81 not upgraded.
```

Figure 8 Installing of Python3

8.2.2 Installing of the Python Modules

To install the embrick modules we need the tool pip.
To install or update pip enter the following command:
"sudo apt-get install python3-pip"

```
pi@raspberrypi:~ $ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-pip is already the newest version (18.1-5+rpt1).
0 upgraded, 0 newly installed, 0 to remove and 81 not upgraded.
```

Figure 9 Installing of PIP

After we have installed pip, we can install or update the required Python modules.
For that we type:
"pip3 install emBRICK"

8.2.3 Download and Unzip the Python Sample Application's

Download the Python Examples with:
"wget https://github.com/IMACS-GmbH/Python/raw/main/examples/examples.zip"

```
pi@raspberrypi: ~
pi@raspberrypi:~ $ wget https://github.com/IMACS-GmbH/Python/raw/main/examples/examples.zip
--2021-04-19 10:44:25--  https://github.com/IMACS-GmbH/Python/raw/main/examples/examples.zip
Resolving github.com (github.com)... 140.82.121.3
Connecting to github.com (github.com)|140.82.121.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/IMACS-GmbH/Python/main/examples/examples.zip [following]
--2021-04-19 10:44:26--  https://raw.githubusercontent.com/IMACS-GmbH/Python/main/examples/examples.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13558 (13K) [application/zip]
Saving to: 'examples.zip'

examples.zip                                        100%[=================================================================>]

2021-04-19 10:44:26 (3.15 MB/s) - 'examples.zip' saved [13558/13558]

pi@raspberrypi:~ $
```

Figure 10 Downloading the Sample Application's

Extract the zip file to you're a into a new Folder of your choice
"unzip examples.zip"

```
pi@raspberrypi: ~
pi@raspberrypi:~ $ unzip examples.zip
Archive:  examples.zip
   creating: ethernet/
  inflating: ethernet/1node.py
  inflating: ethernet/1node_threaded.py
  inflating: ethernet/2nodes.py
  inflating: ethernet/2nodes_threaded.py
  inflating: ethernet/default.py
  inflating: ethernet/default_threaded.py
   creating: serial/
  inflating: serial/1node.py
  inflating: serial/1node_threaded.py
  inflating: serial/2nodes.py
  inflating: serial/2nodes_threaded.py
  inflating: serial/default.py
  inflating: serial/default_threaded.py
  inflating: serial/energie.py
pi@raspberrypi:~ $
```

Figure 11 Unzip of examples.zip

The examples contain the the Example Application. See Chapter8.1.6

8.2.4 Check Hardware

Before starting with the software development, check the hardware by switch on the 24V power of LWCS board.

8.2.4.1 Connection over Lan-Adapter

You can use the build in ethernet port of the Raspberry Pi to control our bricks, but then you can't use the port to connect to the internet.

Or you use the build in port for internet and use a USB to ethernet adapter for the communication with our bricks.

When you use the build in port you don't need to install a driver.

When you use a USB to ethernet adapter it might be possible that you need to install a driver.

Our USB to ethernet adapter didn't need a driver. We just plugged it in and checked the usb devices with "lsusb" for our adapter:

```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 005: ID 0b95:1790 ASIX Electronics Corp. AX88179 Gigabit Ethernet
Bus 001 Device 004: ID 0424:7800 Standard Microsystems Corp.
Bus 001 Device 003: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figure 12 lsusb

Device 005 is our adapter.

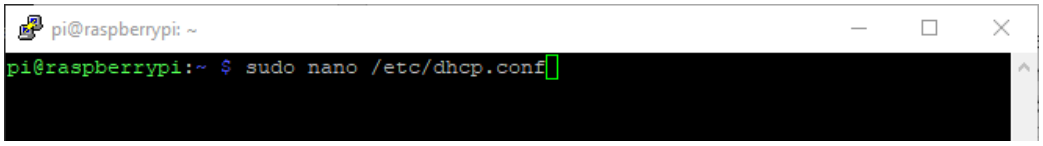
Now with "ip a" you can list all ethernet ports with their Ip address.

```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.159  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::ec50:32bd:73f1:a128  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:df:78:d4  txqueuelen 1000  (Ethernet)
        RX packets 10189  bytes 1394381 (1.3 MiB)
        RX errors 0  dropped 435  overruns 0  frame 0
        TX packets 282  bytes 42113 (41.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 169.254.143.78  netmask 255.255.0.0  broadcast 169.254.255.255
        inet6 fe80::f9f:dbfc:3ee8:a240  prefixlen 64  scopeid 0x20<link>
        ether 50:3f:56:02:3c:c5  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33  bytes 5435 (5.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 13 ifconfig

You need to set the IP of the ethernet adapter you would like to use to the ip range of "**192.168.3.10**" or any other IP you selected with the dip switches or the over the VISU.

To do that login to your pi with putty and change the file "dhcpcd.conf"



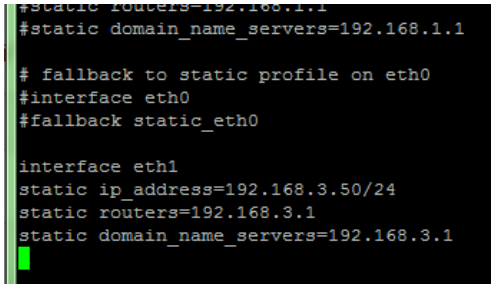
sudo nano /etc/dhcpd.conf

```
sudo nano /etc/dhcpd.conf
```

Now add the following lines after the last line from the "dhcpcd.conf" file

```
# Ethernetadapter for Bricks
interface eth1
# Do not choose the same ip as the Local Master
static ip_address=192.168.3.50/24
static routers=192.168.3.1
static domain_name_servers=192.168.3.1
```

eth0 is the build in ethernet port and eth1 is the USB to ethernet adapter.
If you want to use the build in port change all eth1 in the upper code to eth0.



Configure the Usb Lan Adapter

Save the changes on the file with “CTRL+X” and then confirm with “Y” and then “Enter”.

8.2.4.2 Connection over Modbus RTU (RS458)

When you use a USB to RS485 adapter it might be possible that you need to install a driver.
If you use our **CAE_Z-RaspberryBrick-1#-RB** as a master then you need to use the open source drivers from us. Just follow their tutorial on our Github page to install them.

Now the settings for the COM Port must be made.
If you don’t know where to set these settings then first read chapter 5.5.2.
The COM Port should be “/dev/ttySC0”.

8.2.5 Load and run the Sample Application

If you test all the Sample Application’s you needed following Hardware:

For the Sample Appication’s “1node.py” and “1node_threaded.py” with Ethernet:

- 1x Raspberry Pi
- 1x Remote Master with the Software Ver. 0.58
- 1x P-2Rel4Di2Ai-0# Module ID = 5-131

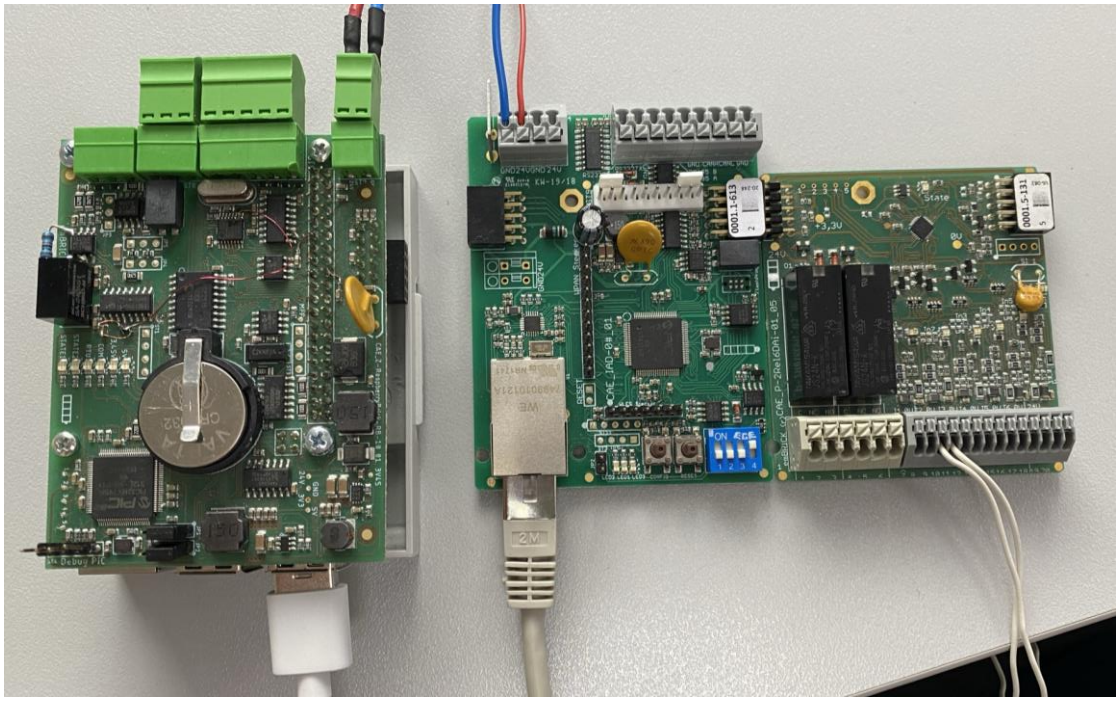


Figure 14 Raspberry PI connected with 1 Node over Ethernet

For the Sample Appication's "1node.py" and "1node_threaded.py" with Serial:

1x Raspberry Pi
1x RaspberryBrick-RB with the Software Ver. 0.58
1x P-2Rel4Di2Ai-0# Module ID = 5-131

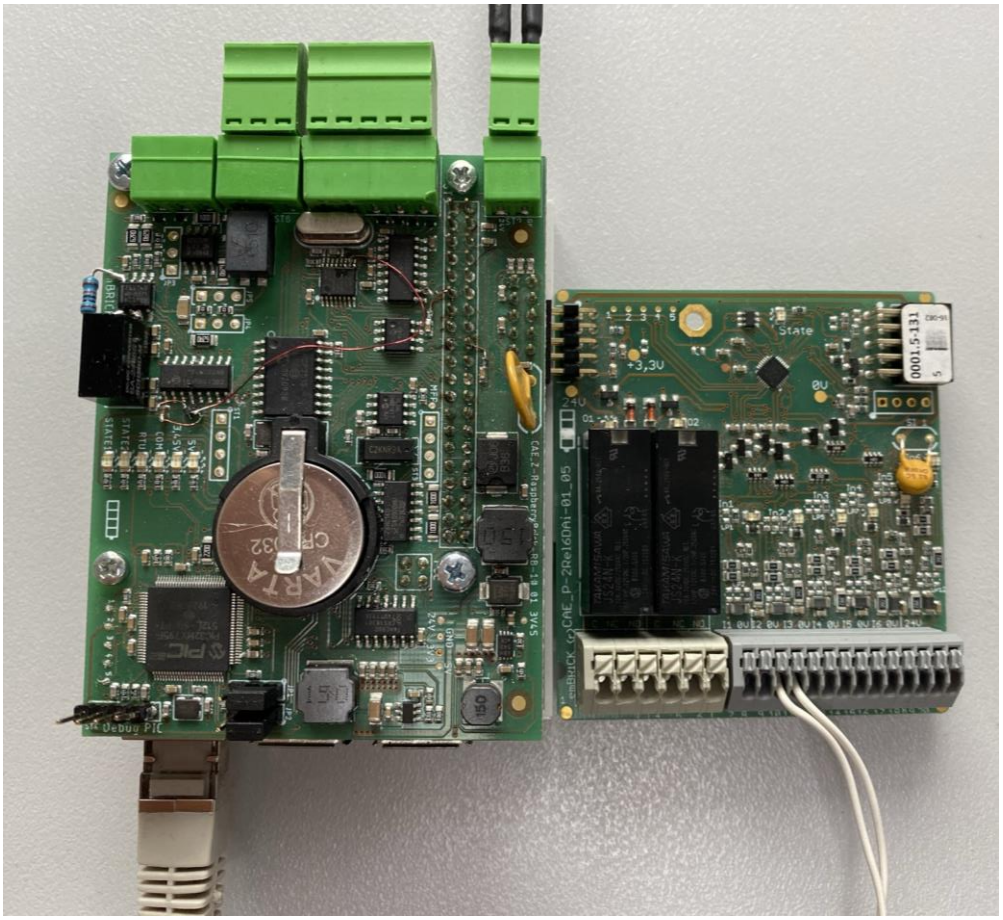


Figure 15 Raspberry Pi connected with 1 Node over Serial

For the Sample Application's "2node.py" and "2node_threaded.py" with Ethernet:

1x Raspberry Pi
Node 1:
1x Remote Master with the Software Ver. 0.58
1x P-2Rel4Di2Ai-0# Module ID = 5-131

Node 2:
1x Remote Master with the Software Ver. 0.58
1x P-2Rel4Di2Ai-0# Module ID = 5-131

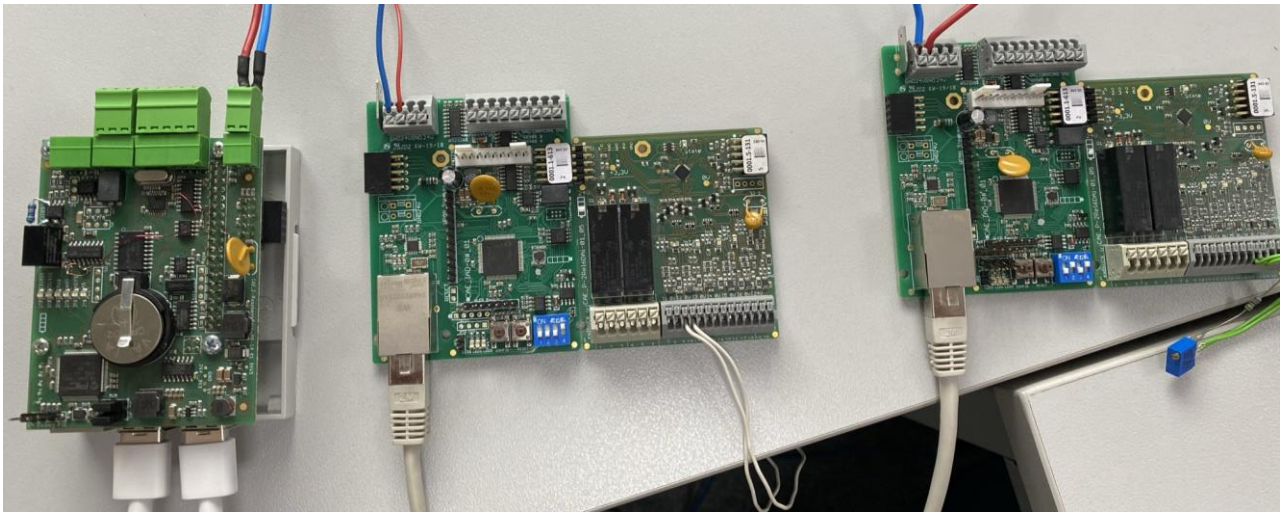


Figure 16 Raspberry Pi connected with 2 Nodes over Ethernet

For the Sample Application's "2node.py" and "2node_threaded.py" with Serial:

Node 1:
1x Raspberry Pi
1x RaspberryBrick-RB with the Software Ver. 0.58
1x P-2Rel4Di2Ai-0# Module ID = 5-131

Node 2:
1x Remote Master with the Software Ver. 0.58
1x P-2Rel4Di2Ai-0# Module ID = 5-131

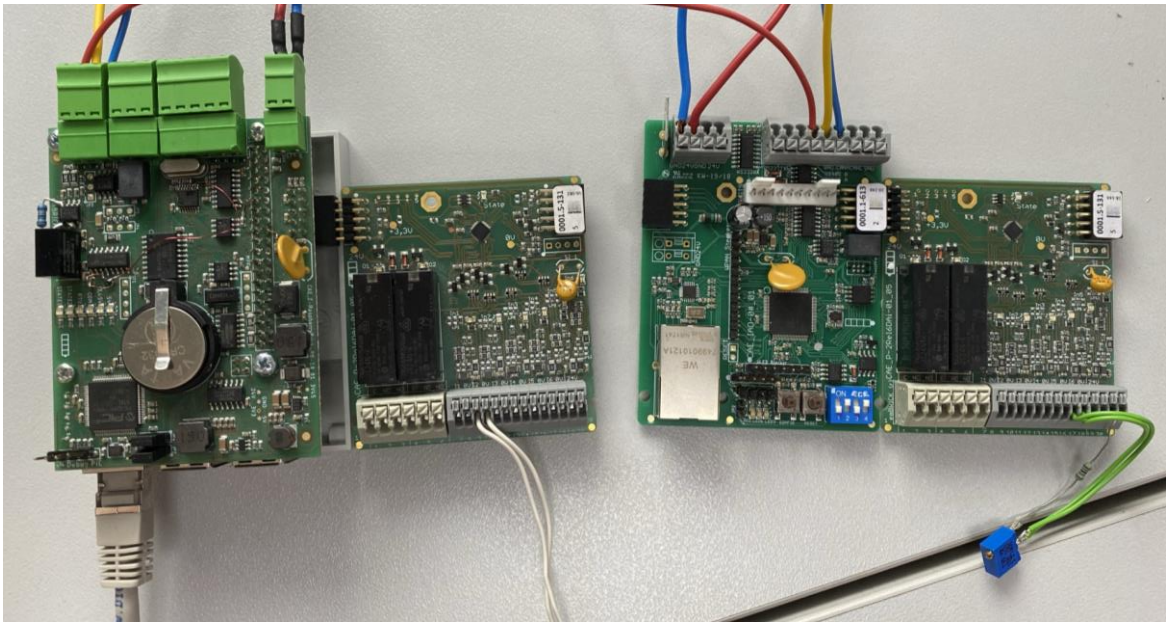


Figure 17 Raspberry Pi connected with 2 Node over Serial

For the Sample Application energie.py:

Node1:
1x Raspberry PI
1x RaspberryBrick-RB with the Software Ver. 0.58
1x B-3U3I-400-##-RB Module ID: 4-602, 4-603
2x G-2Mi2Ao-02 Module ID: 2-471

Node2:
1x Remote Master with the Software Ver. 0.58
1x B-3U3I-400-##-RB Module ID: 4-602, 4-603
2x G-2Mi2Ao-02 Module ID: 2-471

Figure 18 To measure of energy

8.2.6 Start and explore the Functionality of "Starter Kit"

Before starting the Sample Application's, we set the Port by Serial or the Ip Address by Ethernet

8.2.6.1 For Ethernet:

"cd ethernet" To jump in the ethernet folder

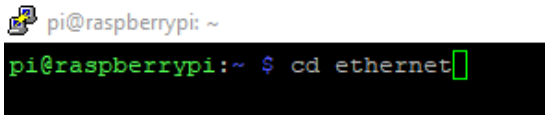


Figure 19

"sudo nano 1node.py" To edit the 1node.py, edit the Sample .py which you want to start

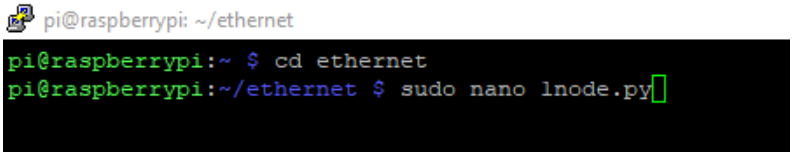


Figure 20

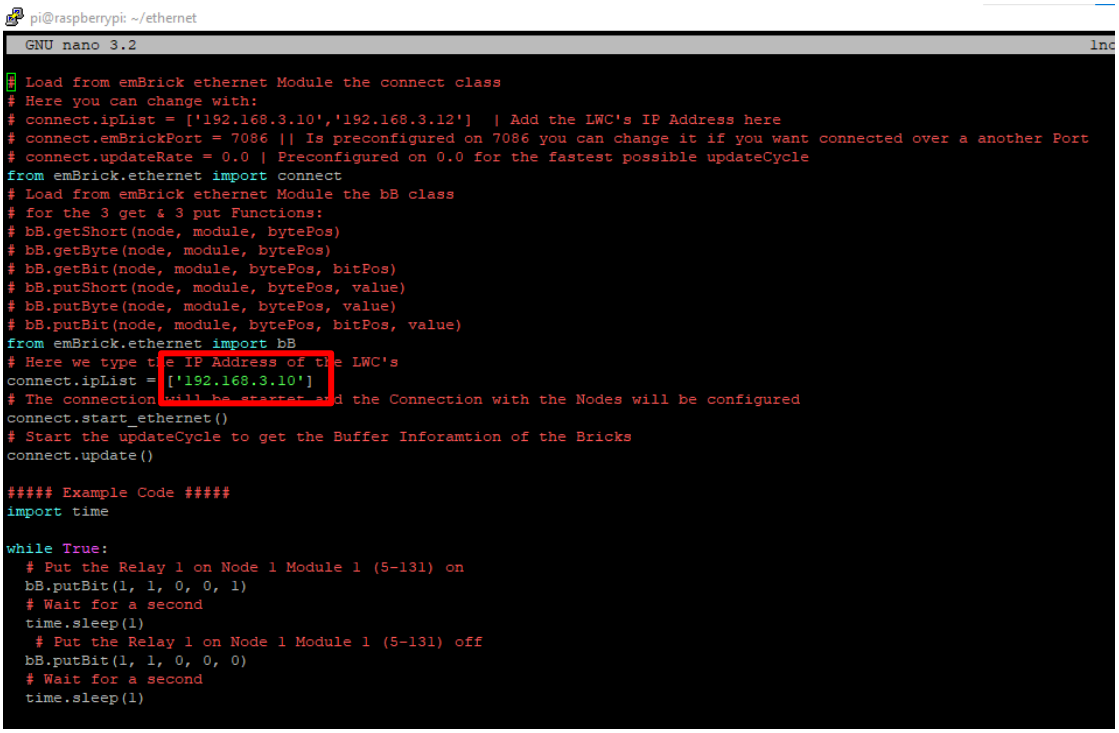


Figure 21 Edit 1node.py

Change the Ip-Adress if you have a another Ip or add a another Ip-Address if you want connect another Node additionaly. Press “Strg +x“ then “y“ to save the changes.

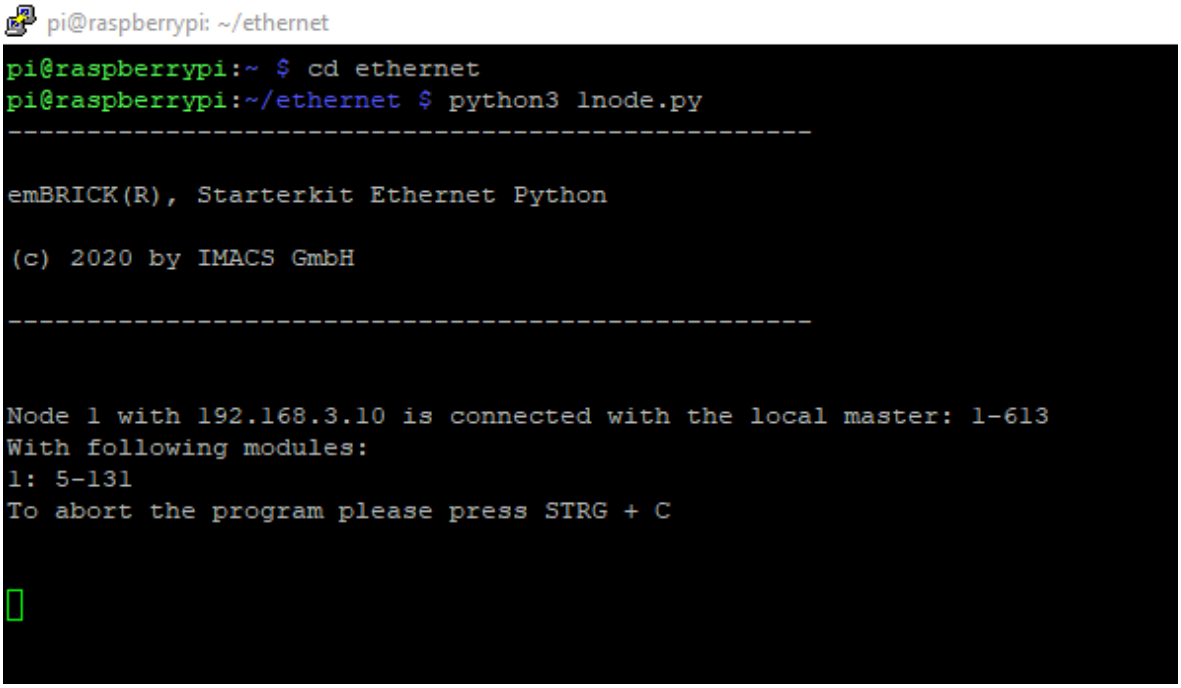


Figure 22 Running 1node.py

8.2.6.2 Serial

“cd serial” To jump in the serial folder

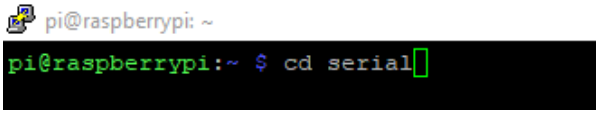


Figure 23

„sudo nano 1node.py“ To edit the 1node.py, edit the Sample .py which you want to start

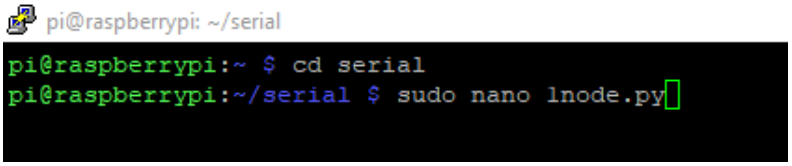


Figure 24

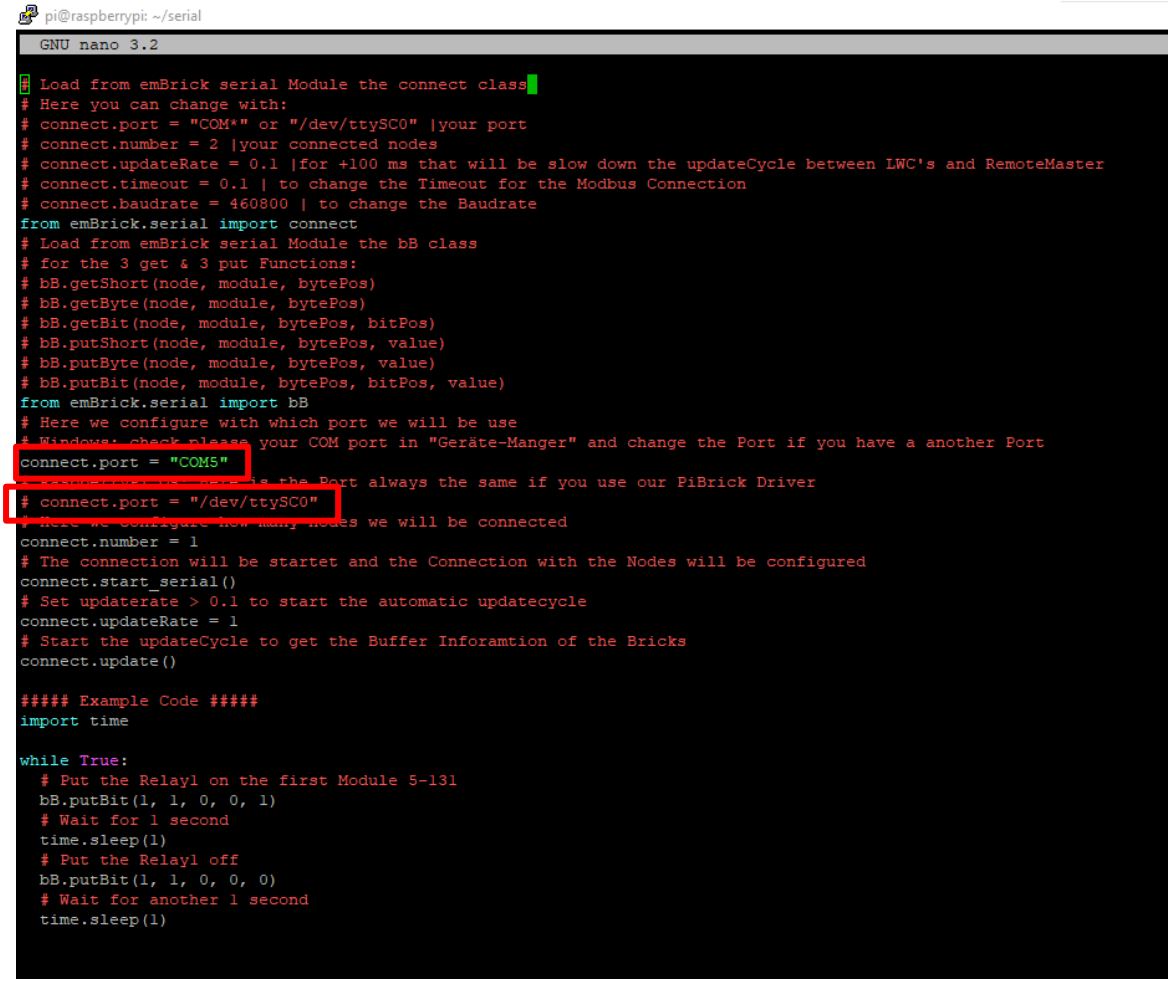


Figure 25 Edit 1node.py

Make the line “connect.port = “COM5” to a comment or delete and decomment the Line “# connect.port= “/dev/ttySC0”. Because the Port “/dev/ttySC0” is the preconfigured Port from RaspberryBrick. Press “Strg +x “ then “y“ to save the changes.

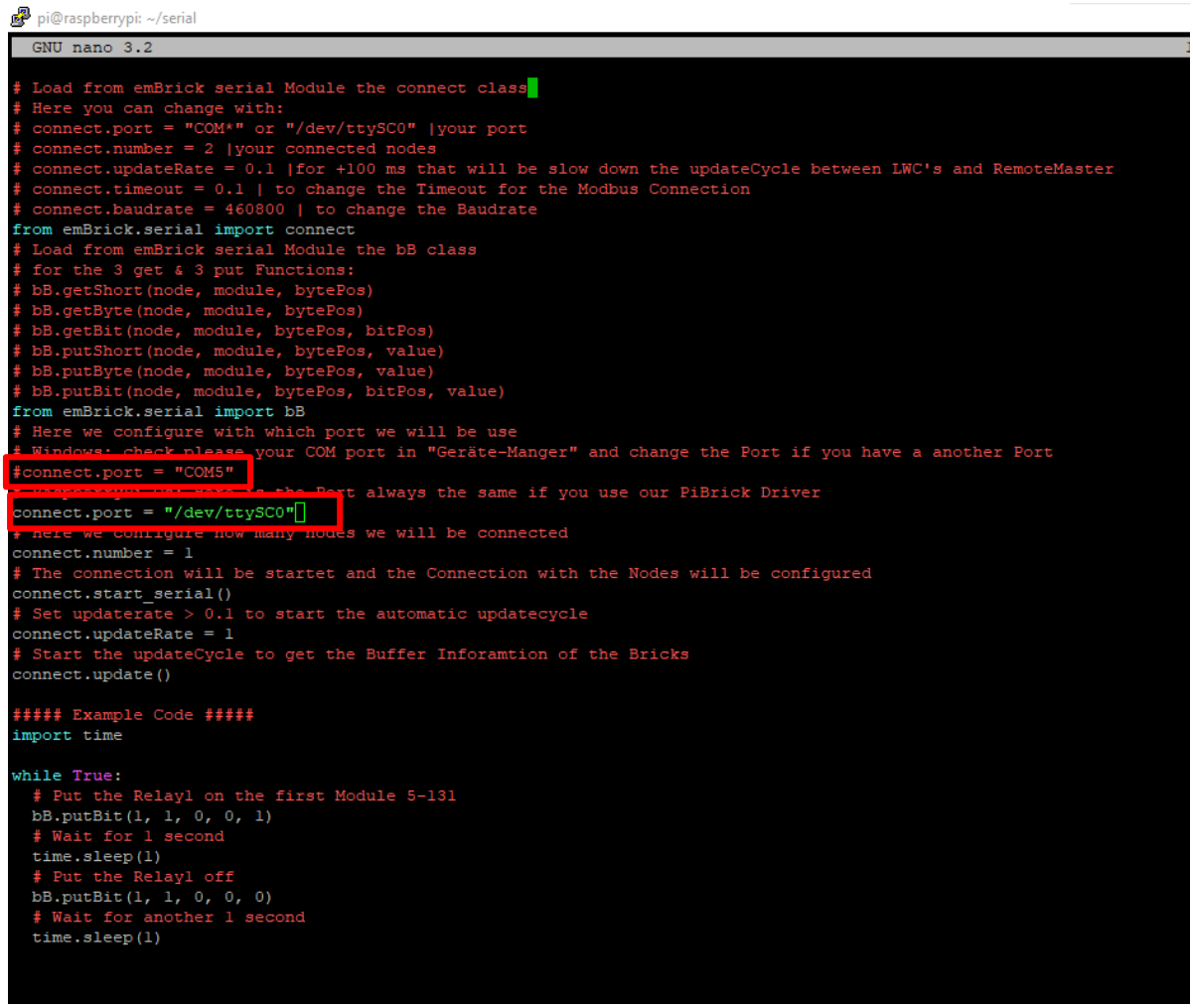


Figure 26 Configured 1node.py

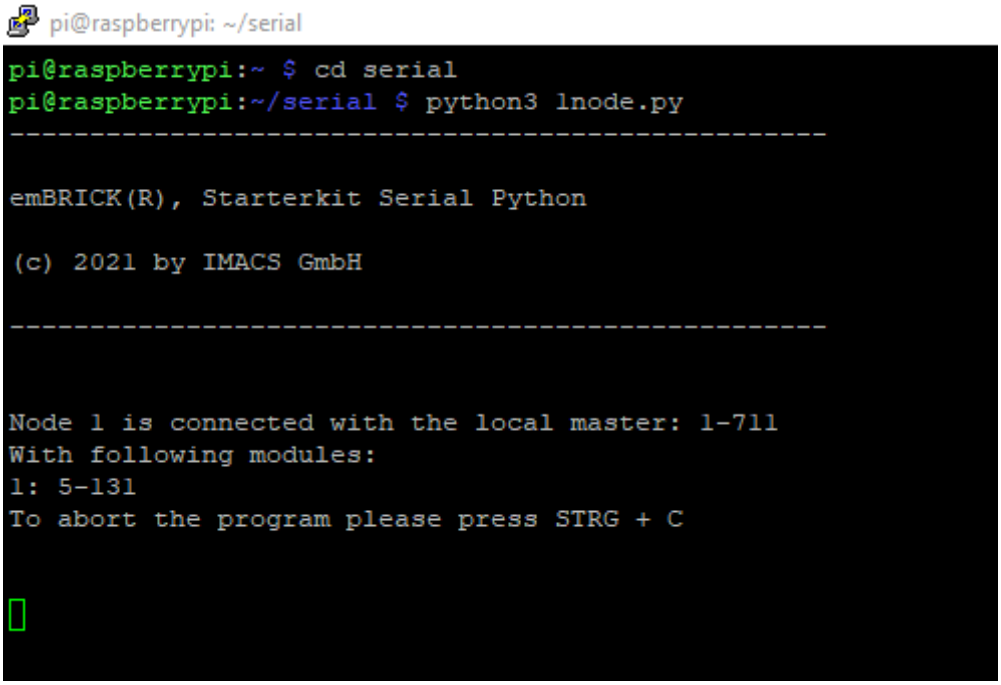


Figure 27 Running 1node.py

8.2.7 Create your own application

8.2.7.1 Serial

“cd ethernet” To jump in the ethernet folder

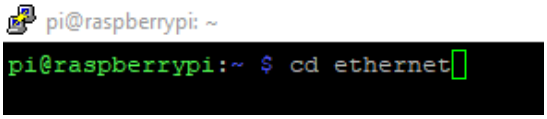


Figure 28

“sudo nano default.py” or without Thread

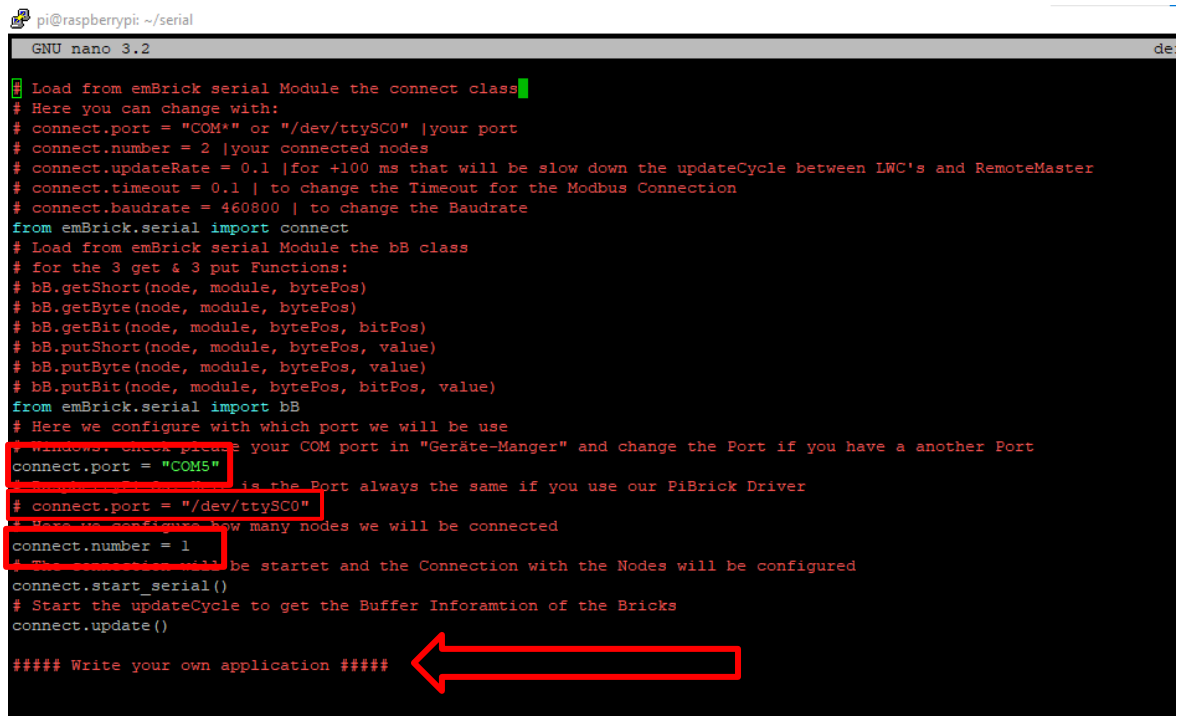


Figure 29 Edit default.py

Make the line “connect.port = “COM5” to a comment or delete and decomment the Line “# connect.port= “/dev/ttySC0”. Because the Port “/dev/ttySC0” is the preconfigured Port from RaspberryBrick.

“connect.number = 1” change it, if you have more nodes then 1.

And write your own code after the “#### Write your own application ####”

Press “Strg +x “ then “y“ to save the changes.

“sudo nano default_threaded.py” with Thread

```
pi@raspberrypi: ~/serial
  GNU nano 3.2                                                    default

# Load from emBrick serial Module the connect class
# Here you can change with:
# connect.port = "COM*" or "/dev/ttySC0" |your port
# connect.number = 2 |your connected nodes
# connect.updateRate = 0.1 |for +100 ms that will be slow down the updateCycle between LWC's and RemoteMaster
# connect.timeout = 0.1 | to change the Timeout for the Modbus Connection
# connect.baudrate = 460800 | to change the Baudrate
from emBrick.serial import connect
# Load from emBrick serial Module the bB class
# for the 3 get & 3 put Functions:
# bB.getShort(node, module, bytePos)
# bB.getByte(node, module, bytePos)
# bB.getBit(node, module, bytePos, bitPos)
# bB.putShort(node, module, bytePos, value)
# bB.putByte(node, module, bytePos, value)
# bB.putBit(node, module, bytePos, bitPos, value)
from emBrick.serial import bB
# Here we configure with which port we will be use
# Windows: check please your COM port in "Geräte-Manger" and change the Port if you have a another Port
connect.port = "COM5"
# Linux: Port is the Port always the same if you use our PiBrick Driver
# connect.port = "/dev/ttySC0"
# Here we configure how many nodes we will be connected
connect.number = 1
# Variable to change the update time of the thread
connect.updateRate = 0.1
# Import the Python Module threading
import threading

def userinterface():
  ### Write your own Code ###

  # In every UpdateCycle the counter raise by 1
  connect.counter += 1
  if not connect.thread:
    connect.update()
  # Start the Thread with given UpdateRate
  threading.Timer(connect.updateRate, userinterface).start()


if __name__ == "__main__":
  # The connection will be startet and the Connection with the Nodes will be configured
  connect.start_serial()
  # Start the updateCycle to get the Buffer Inforamtion of the Bricks
  connect.update()
  # Start the thread
  userinterface()
```

Figure 30 default_threaded.py

Make the line “connect.port = “COM5” to a comment or delete and decomment the Line “# connect.port= “/dev/ttySC0”. Because the Port “/dev/ttySC0” is the preconfigured Port from RaspberryBrick.

“connect.number = 1” change it, if you have more nodes then 1.

And write your own code after the “#### Write your own application ####”

Press “Strg +x “ then “y“ to save the changes.

8.2.8 The Python Remote-Bus Driver

See chapter 8.1.8.

9 Node-Red

9.1 Setup the Development Environment on Windows

9.1.1 What will be needed

- Node.js
- Node-Red
- A current Internet Browser like Google Chrome, Firefox, …
- Install additional Nodes

9.1.2 Installing Node.js

Download the latest 12.x LTS version of Node.js from the official Node.js home page. It will offer you the best version for your system.

Run the downloaded MSI file. Installing Node.js requires local administrator rights; if you are not a local administrator, you will be prompted for an administrator password on install. Accept the defaults when installing. After installation completes, close any open command prompts and re-open to ensure new environment variables are picked up.

Once installed, open a command prompt and run the following command to ensure Node.js and npm are installed correctly.

Using Powershell: `node --version; npm --version`

Using cmd: `node --version && npm --version`

You should receive back output that looks similar to:

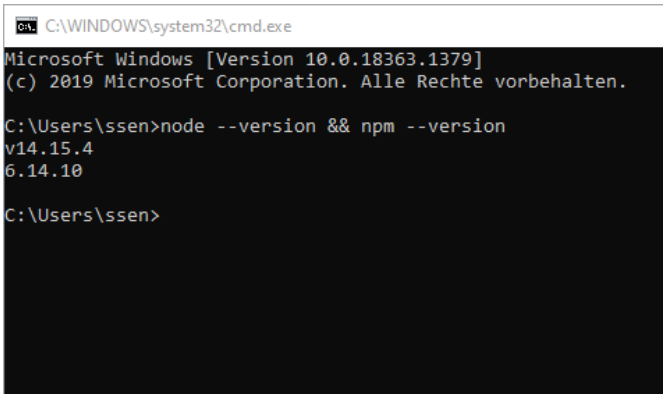


Figure 31 Node.js & Node-Red Version

9.1.3 Installing Node-Red

Installing Node-RED as a global module adds the command `node-red` to your system path. Execute the following at the command prompt:

npm install -g --unsafe-perm node-red

9.1.4 Run Node-Red

Once installed, you are ready to run Node-RED.

Press the Window Button + R to open the Run Command type in the Window cmd and press Enter.

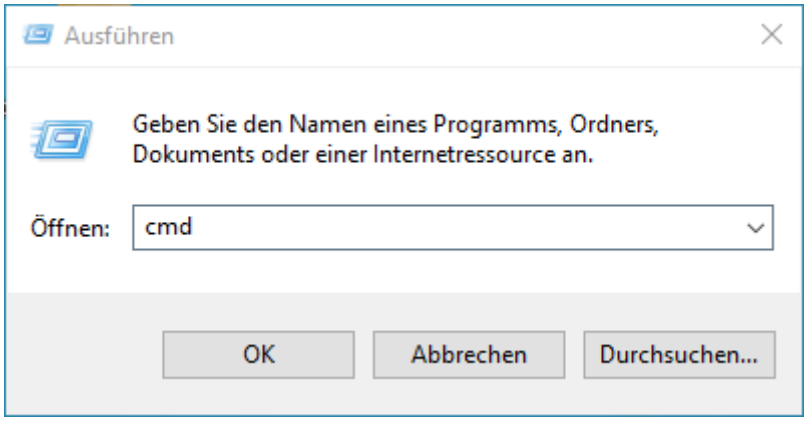


Figure 32 Start Command Window

In the new opening Window type node-red and press Enter dann will be Node-Red running.

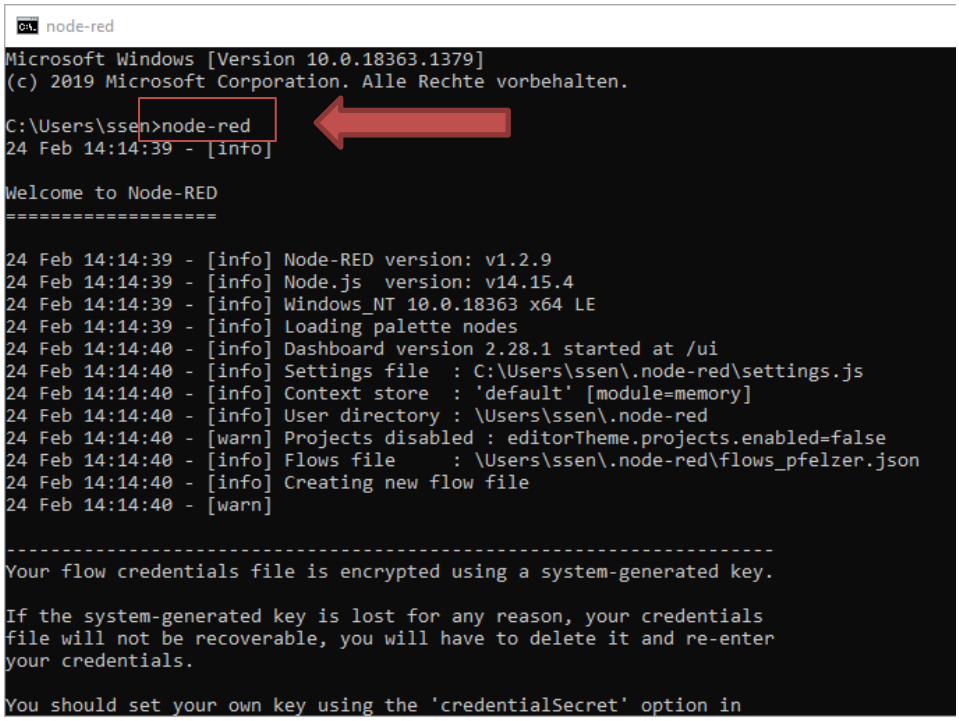


Figure 33 Starting Node-Red

Open your Internet Browser and type “localhost:1880” and press Enter. The node-red platform will be loaded.

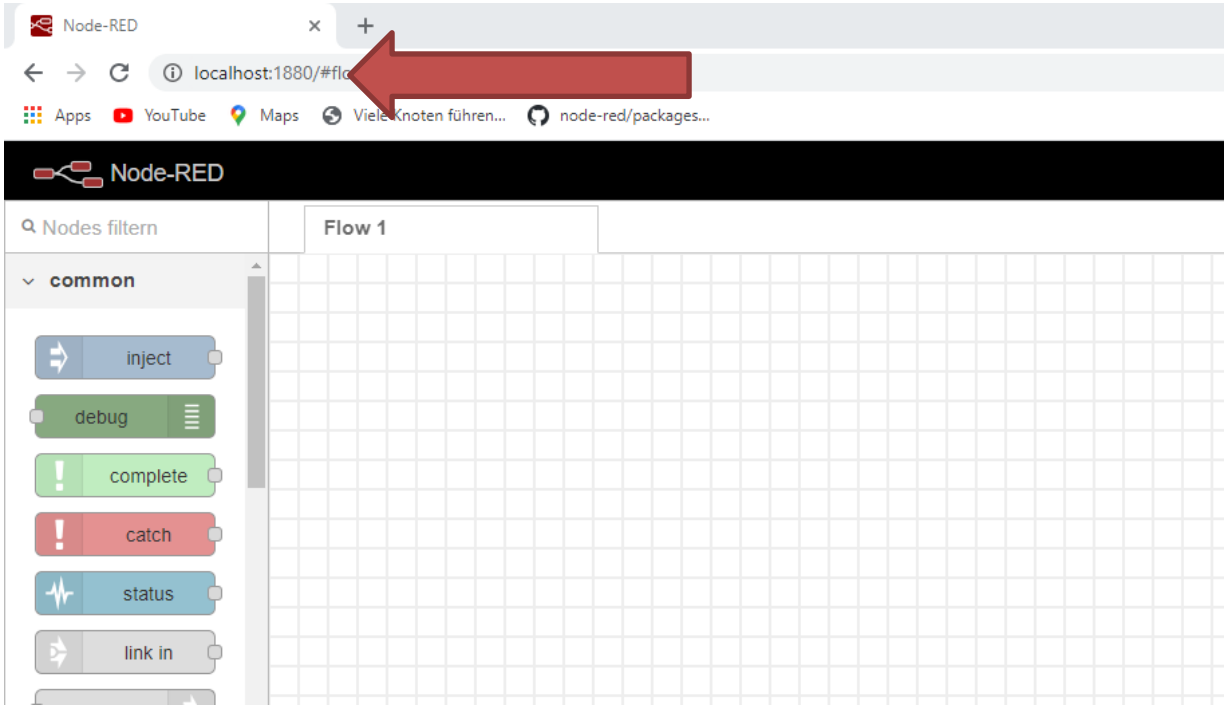


Figure 34 Open Node-Red in Internet Browser

9.2 Setup the Development Environment on Raspberry Pi

9.2.1 What will be needed

- Node.js
- Node-Red
- A current Internet Browser
- Install additionally Nodes

9.2.2 Installing Node.js, Node-Red & npm

To install follow the following Step on this Site:

https://nodered.org/docs/getting-started/raspberrypi

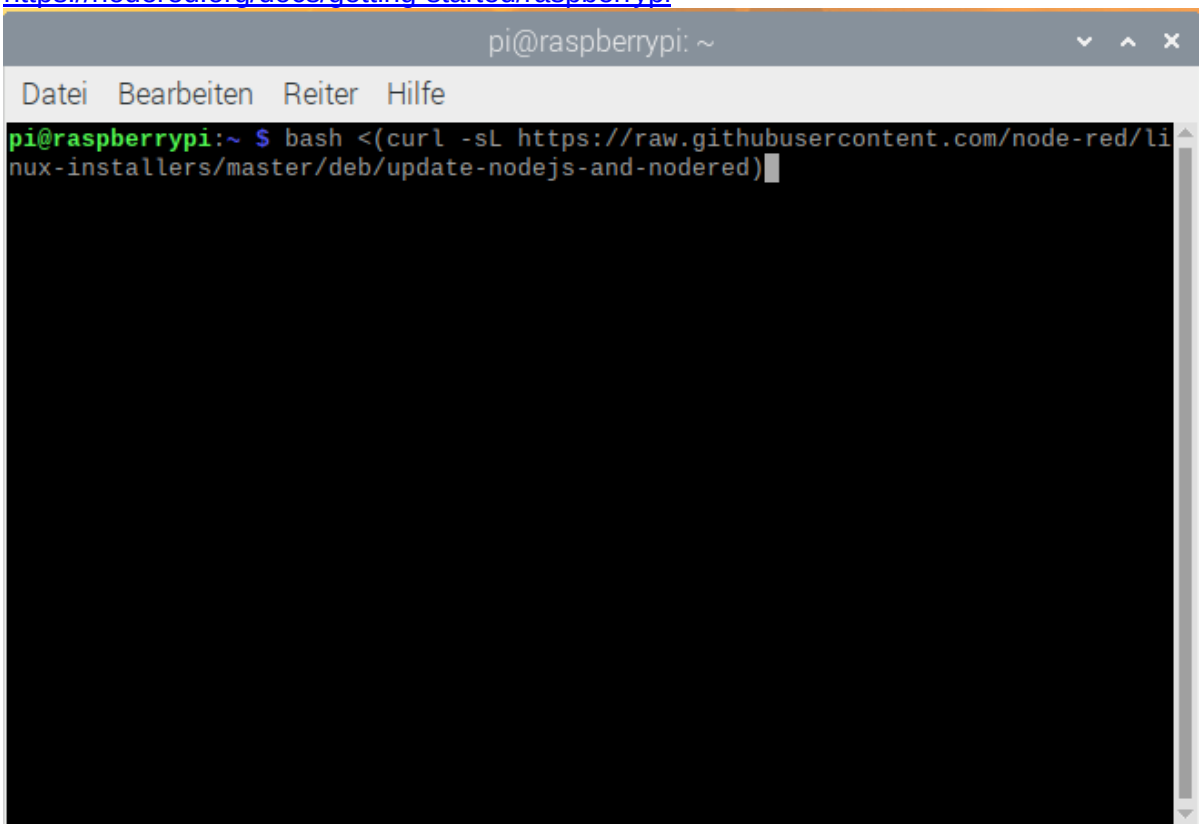


Figure 35 Starting Scipt to install Node.js, npm & Node-red

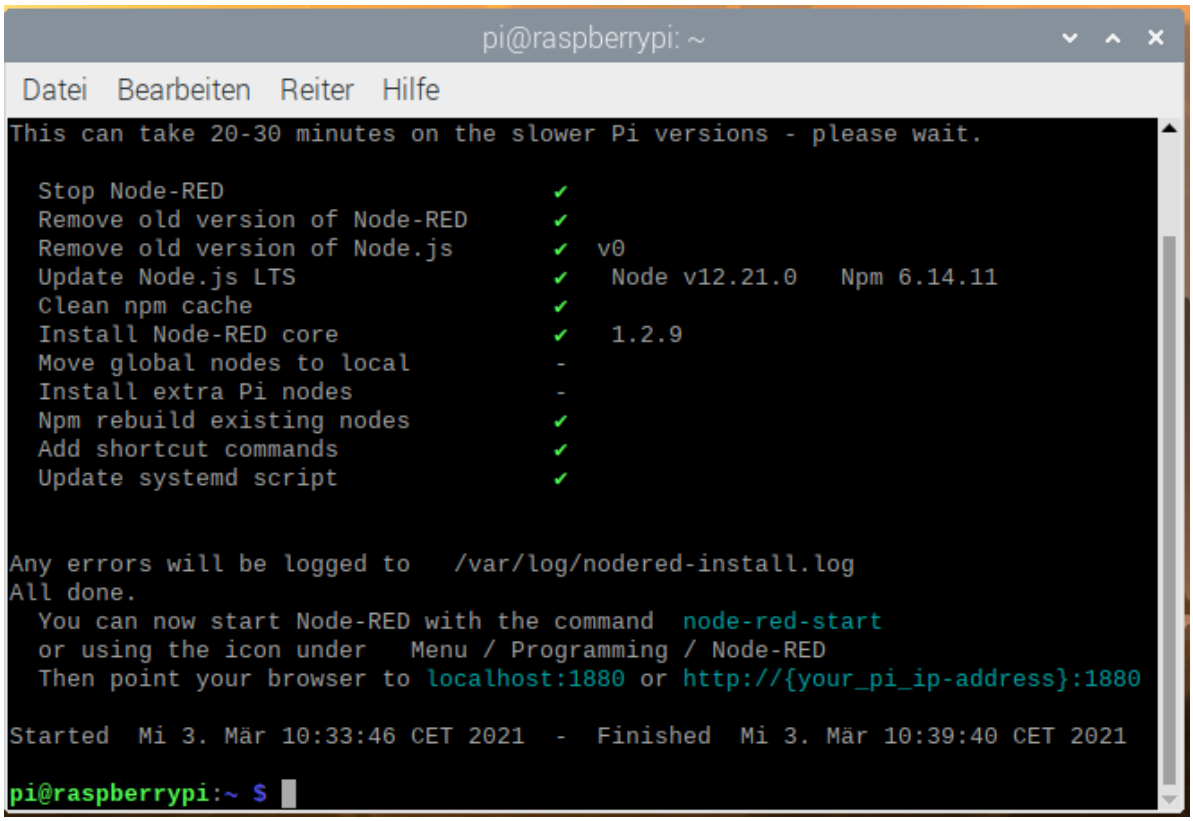


Figure 36 After successfull Installtion

9.2.3 Run Node-Red

The install script for the Pi also sets it up to run as a service. This means it can run in the background and be enabled to automatically start on boot.

The following commands are provided to work with the service:

`node-red-start` - this starts the Node-RED service and displays its log output. Pressing `Ctrl-C` or closing the window does *not* stop the service; it keeps running in the background

`node-red-stop` - this stops the Node-RED service

`node-red-restart` - this stops and restarts the Node-RED service

`node-red-log` - this displays the log output of the service

You can also start the Node-RED service on the Raspbian Desktop by selecting the `Menu -> Programming -> Node-RED` menu option.

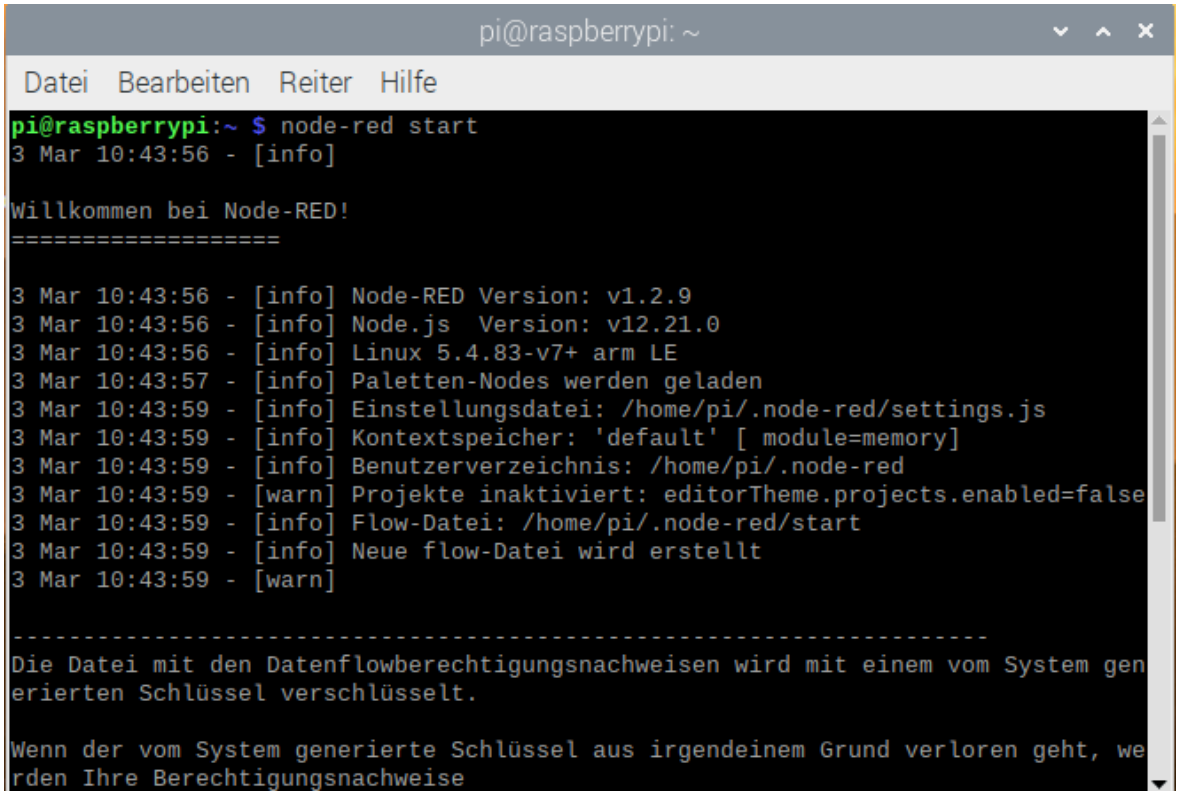


Figure 37 Starting Node-red

After start the Node-Red with the command node-red-start. We go on “localhost:1880” with Internet Browser to open Node-Red.

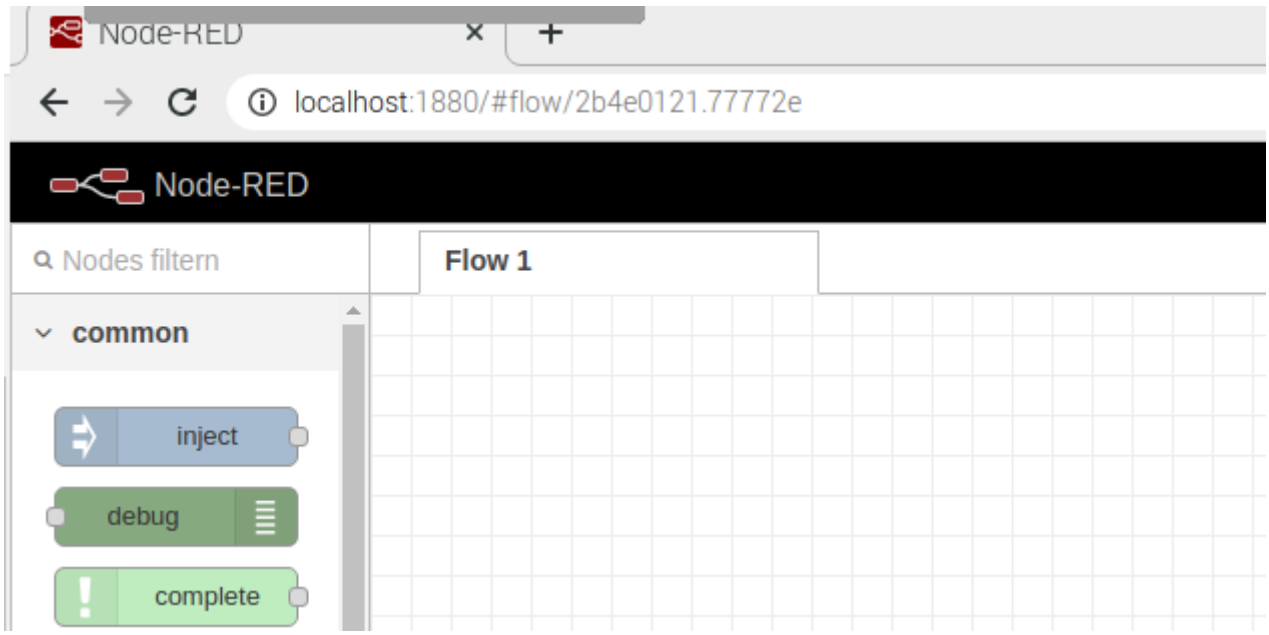


Figure 38 Open in Node-red

9.3 Installing of the additionally Nodes

To install the additionally Nodes first click on top right of the Menu icon. Then click on “Palette verwalten”.

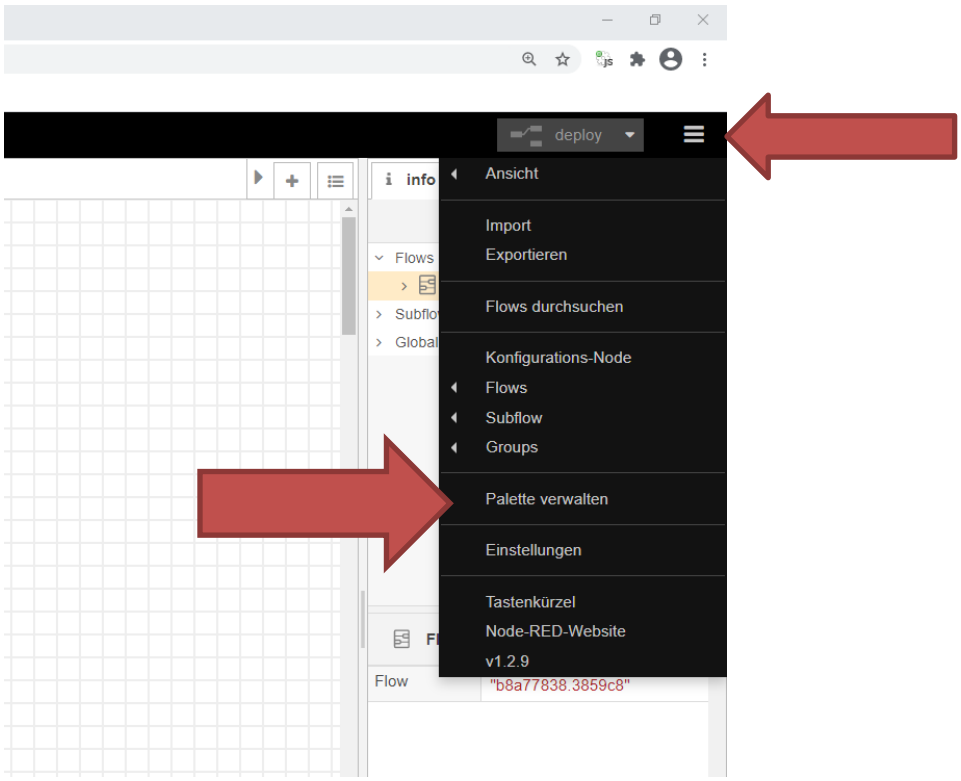


Figure 39 Add additional Nodes to Node-red

Then a right Sidebar will be open. On their click “Installieren” and so we can give in the Search. Bar the name of the additionally needed Nodes and install them.

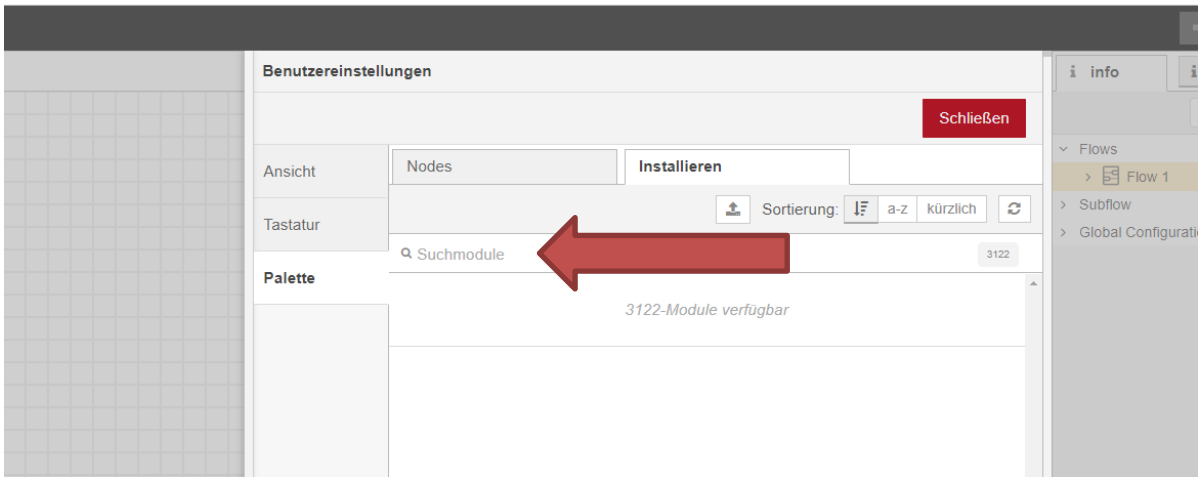


Figure 40 Search a Node

Search for node-red-dashboard and click “Installieren” to install the node. And that we make for all needed Nodes.

- Node-red-dashboard
- node-red-contrib-config
- red-contrib-embrick
- node-red-contrib-boolean-logic-ultimate
- optionally for connection with modbus: node-red-contrib-modbus

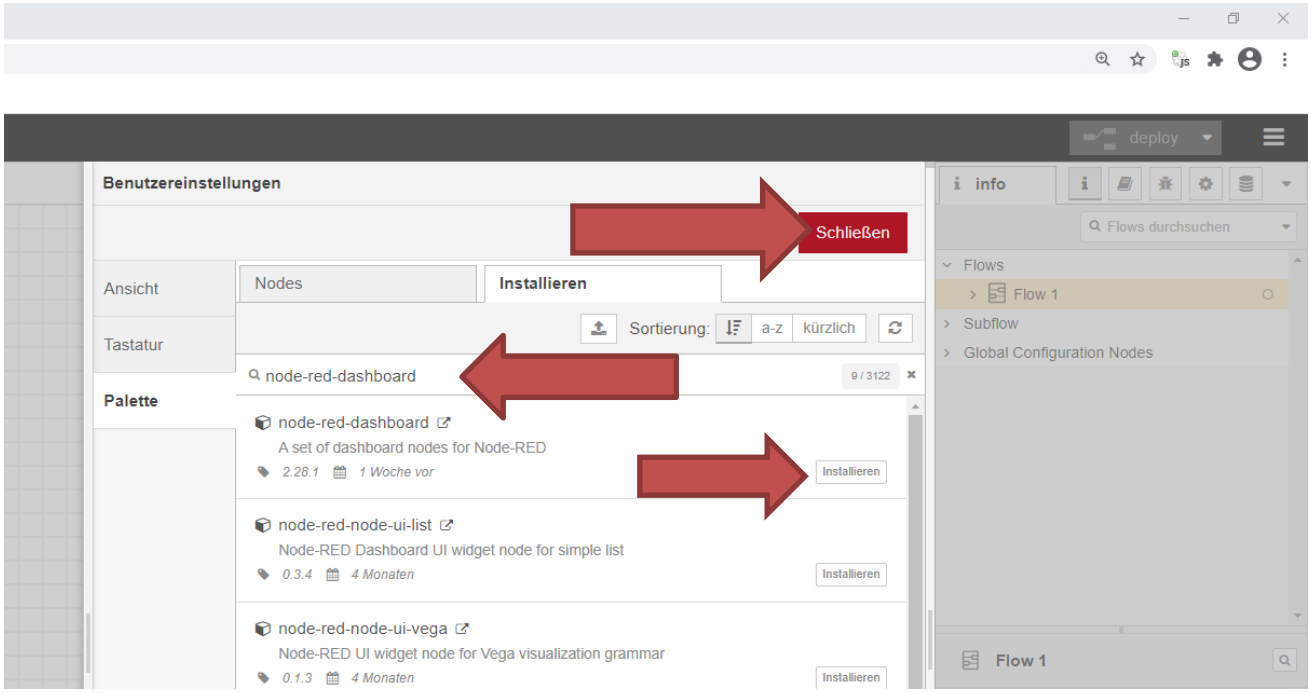


Figure 41 Install a Node

9.4 Check Hardware

Before starting with the software development, check the hardware by switch on the 24V power of LWCS board.

9.4.1 Connection over Lan-Adapter

Start the Application "NetBrick.exe".

NetBRICK is a useful tool that checks all network ports from your PC whether there is a *coupling-master* connected with its IP-address. All founded coupling-masters will be listed with their IP addresses. With *NetBRICK* you can simply check ...

- if your PC Ethernet-Adapters are correct configured
- if the *coupling-master* is working and connected/found
- the IP-addresses of the available *coupling-master*

To start, double click on **NetBrick.exe**

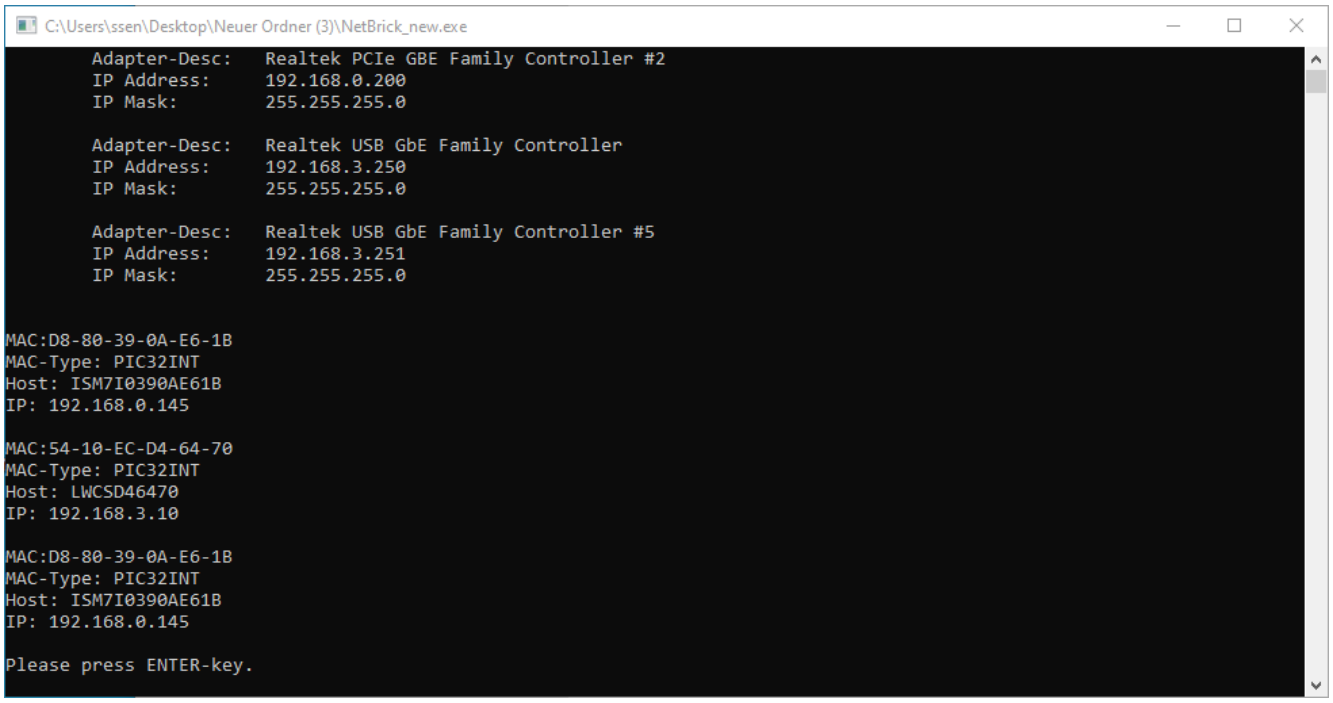


Figure 42 show's IP Address from Remote Master

In this picture you can see the output of the *NetBrick* at first all your Ethernet-Adapters are listed and there after comes the detected *coupling-master* with the IP 192.168.3.10.

Result: Now you can access the *coupling-master* and *slave-modules*.

9.4.1.1 Change Ip-Address

If you want use a another Ip-Address you can change easily with a double click on the config Node “change updaterate(in ms), ip-address or port.

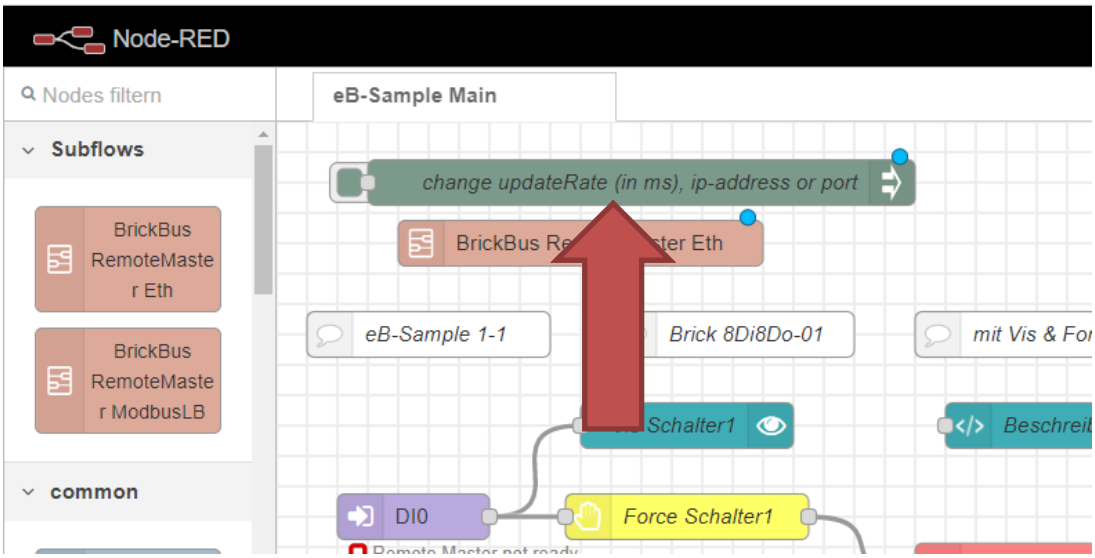


Figure 43 edit the Config Node „change updateRate, ip-address or port

Then will be open a right Side Bar with the name “config Node bearbeiten”. Here we can by flow.host our Ip-Address and after that click on red “Fertig” Button and then on the “deploy” Button. Now the ip-Address are changed.

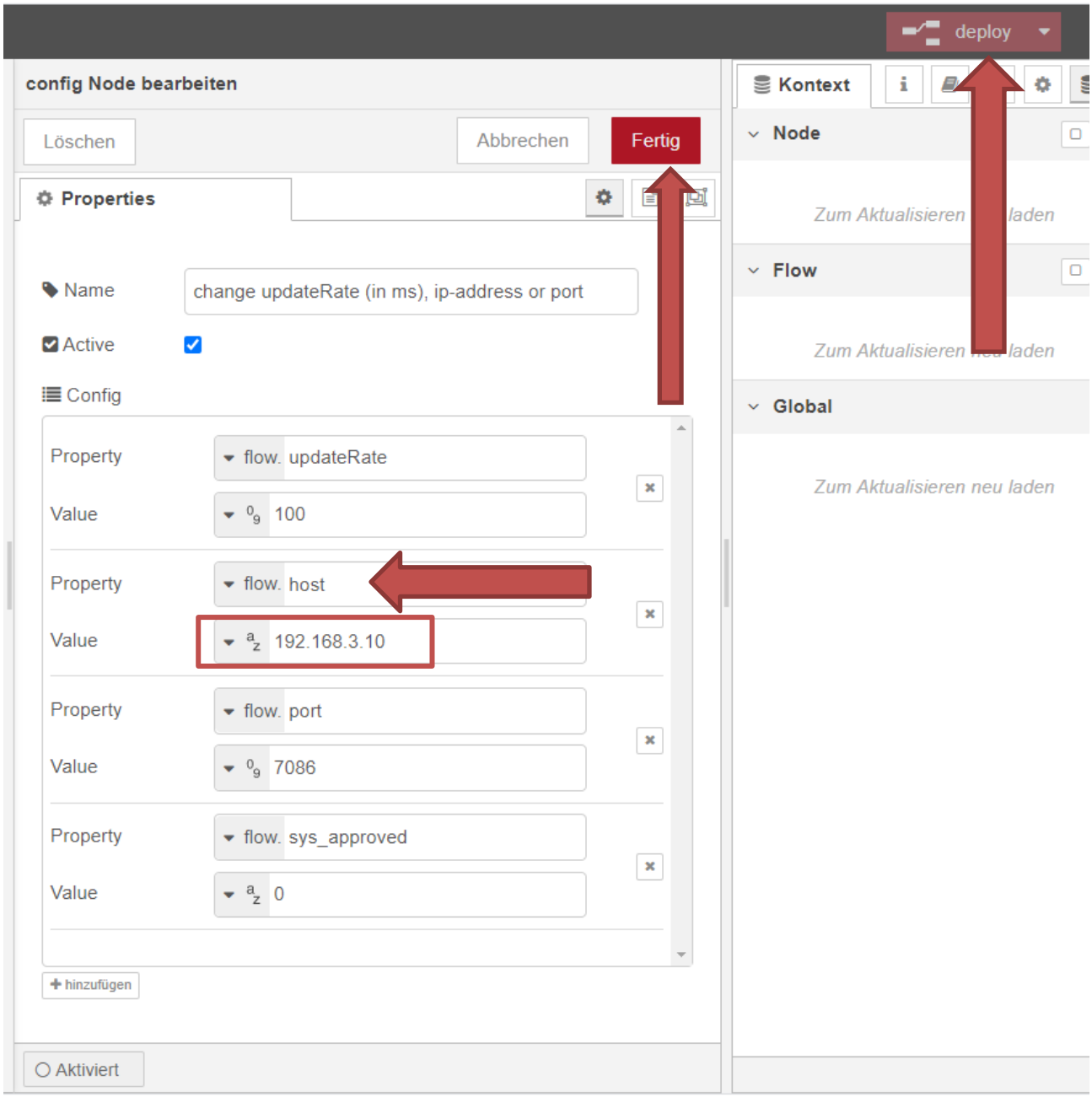


Figure 44 change ip-Address

9.4.2 Add Multiple Remote Master to Project

The Remote Master are writing for a single Flow. If you want to add additional Node, we need to add a new Flow. For that we click to “Add” Button on the right to add a new Flow.

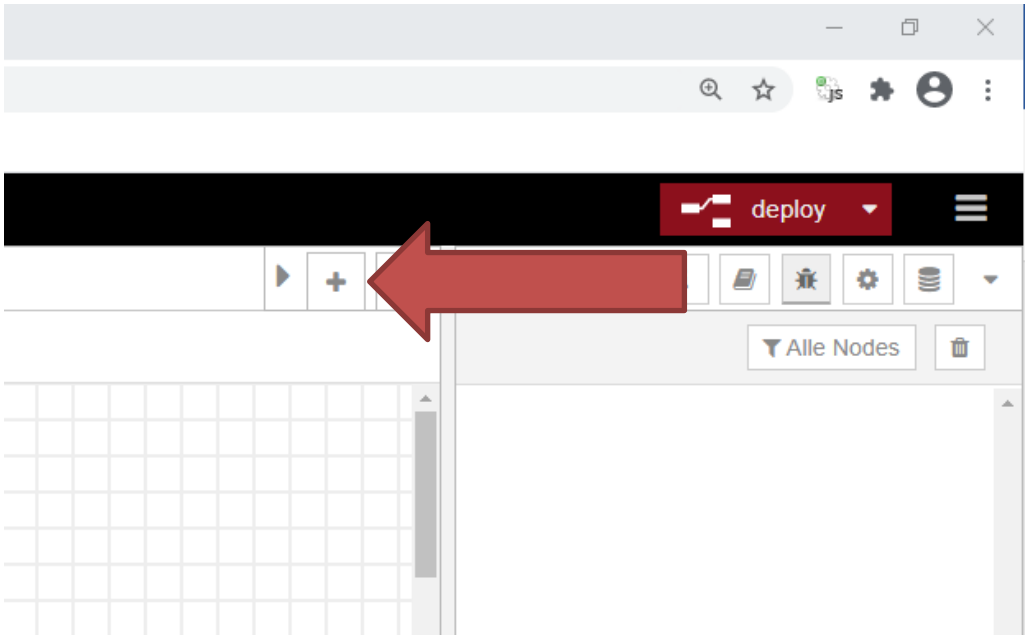


Figure 45 Add a new Flow

After that we copy “change updateRate (in ms), ip-address or port” & “BrickBus RemoteMaster Eth” Node and add them to the new Flow. For that mark the Nodes and press “Strg+c”.

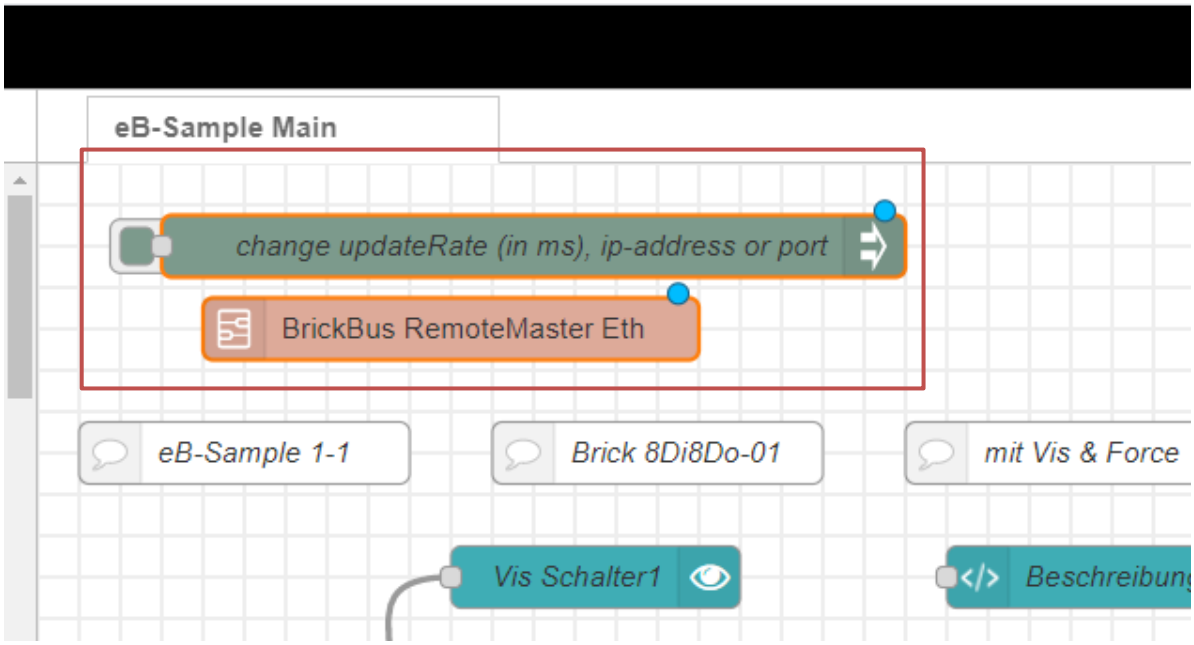


Figure 46 Copy "BrickBus RemoteMaster Eth" & the Config Node

And paste it with “Strg+v” in the new Flow.

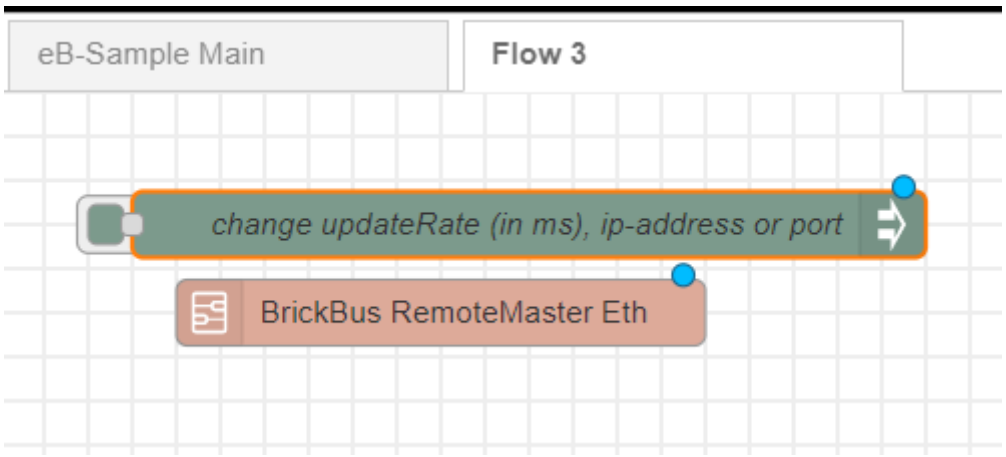


Figure 47 Copy "BrickBus RemoteMaster Eth" & the Config Node

There we only muss change the IP-Address (like in 2.4.3.1.1).

9.4.3 Connection over Serial (RS458)

First check with which Com Port is your Serial Adapter connected.

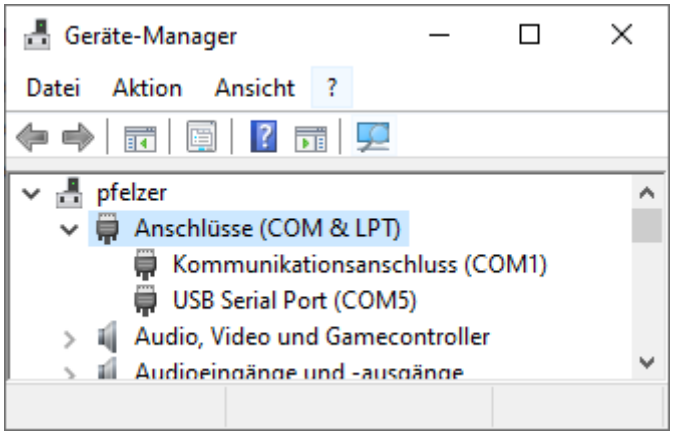


Figure 48 Geräte-Manager

The USB Serial Port is connected on COM5

9.4.3.1 Change Modbus Port Number

For that we click on the Configuration Button on the right Sidebar. After that double click on the “modbus-serial@COM5”.

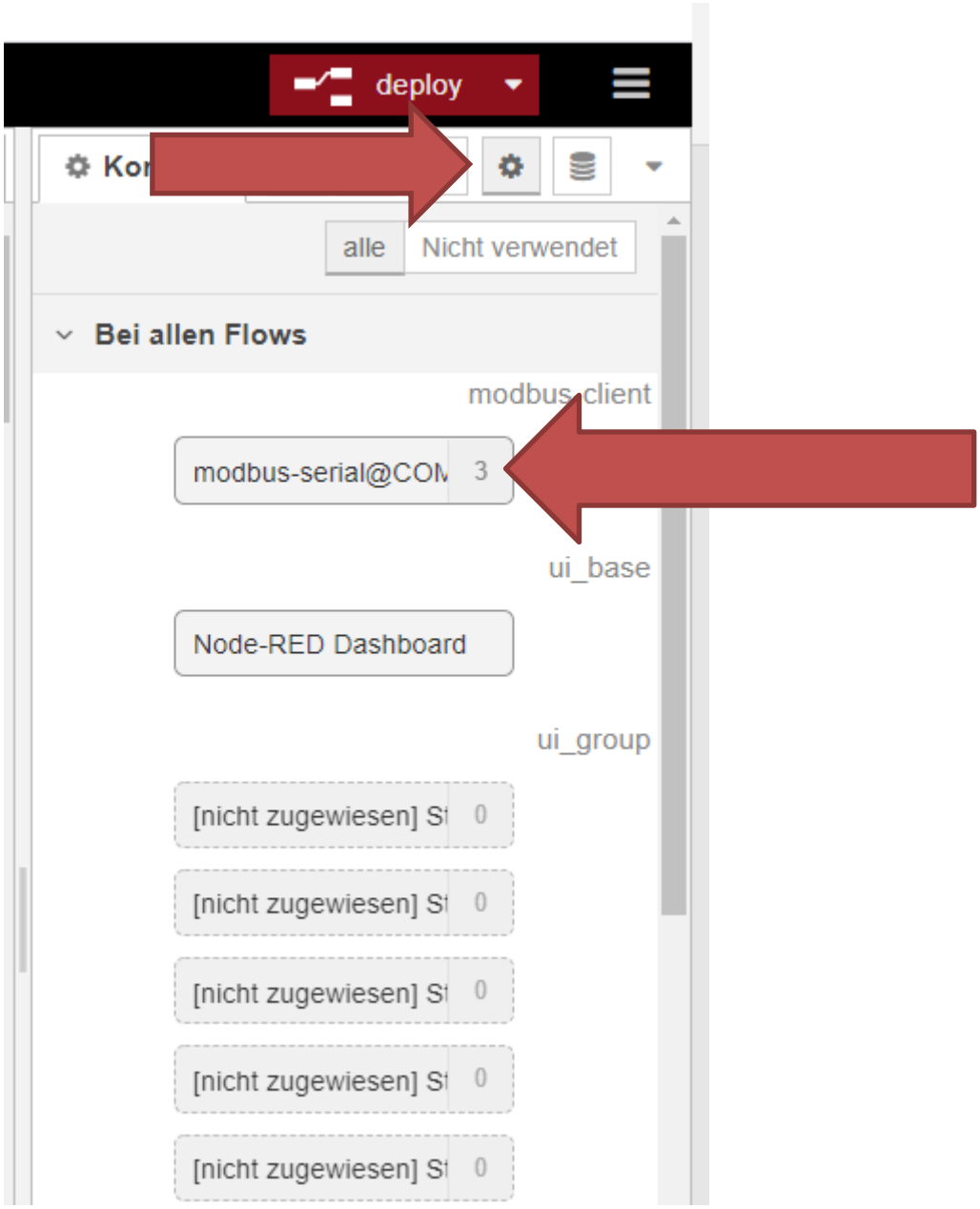


Figure 49 Konfigurationsmenü

There we can manually change the by Com-Port our Port or we click on search Button then will popped up the possible Port's. There we can choice the right one and save it with a click on "aktualisieren" then "deploy".

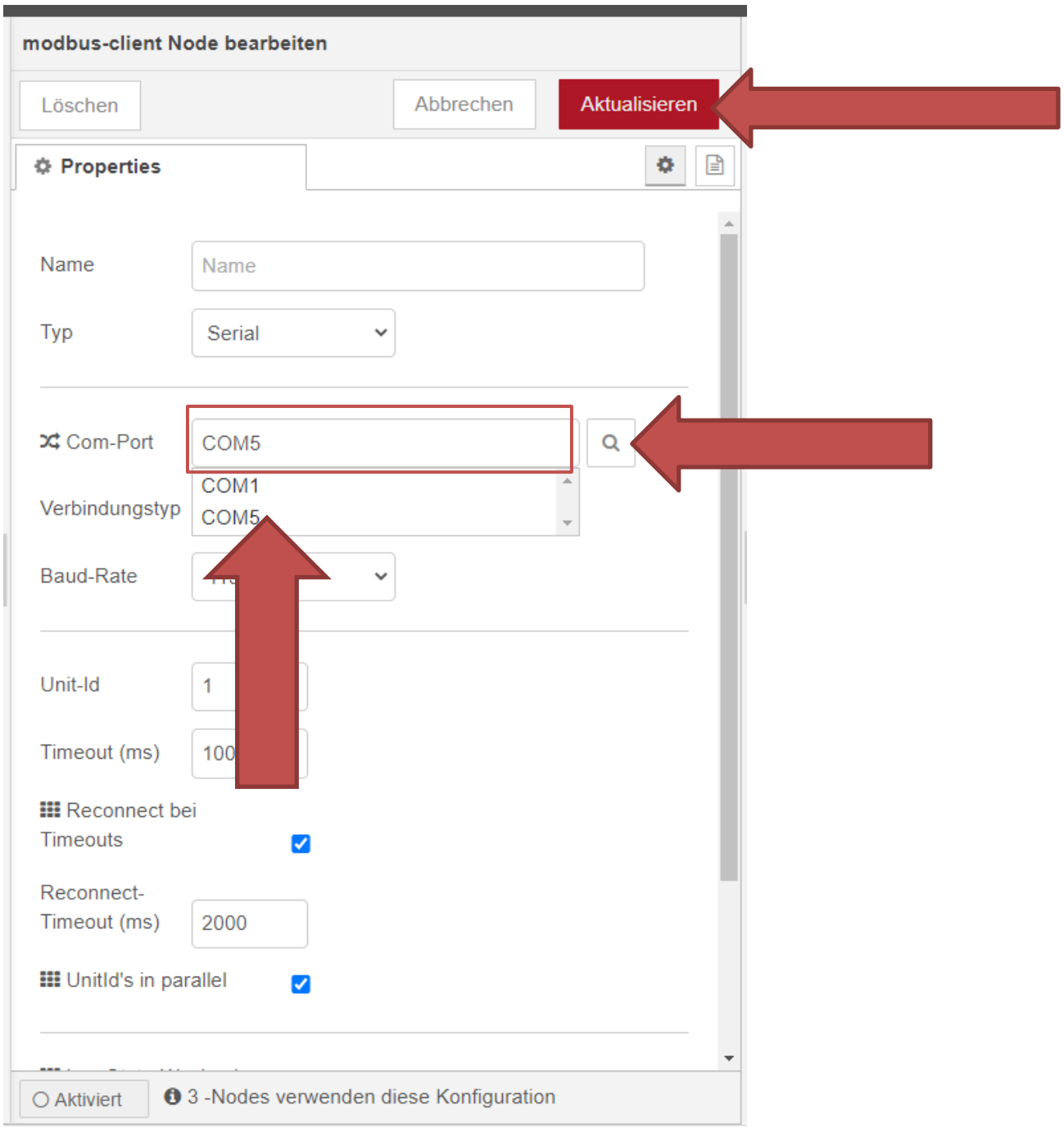


Figure 50 change Modbus Com-Port

9.5 Load and run the Sample Applications

For the Sample Application you needed following Hardware:

1. Remote Master with the Software Ver. 0.55

The Sample Applications can be loaded separately when you don't have one of these Modules

2. G-8Di8Do-01 Module ID = 2-181
3. P-2Rel4Di2Ai-01 Module ID = 5-131

4. G-2Mi2Ao-02 Module ID = 2-472

To load the Examples, we click in Node Red at the Menu Button in the Top Right then on Import.

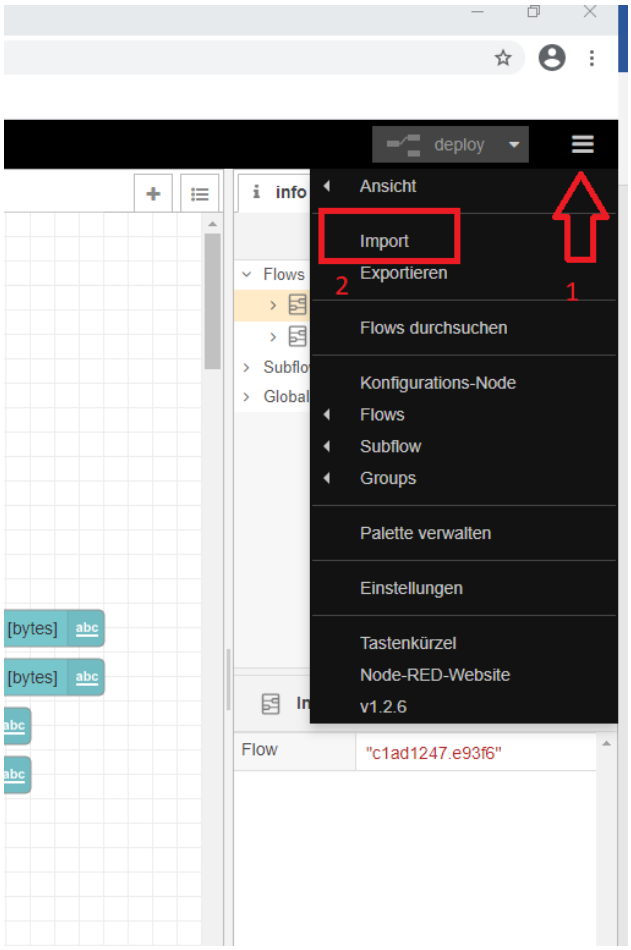


Figure 51 Open Import Window

It will open a new window there we click on the left on “Beispiele” then on file red-contrib-embrick. Then you can load the Example you want.

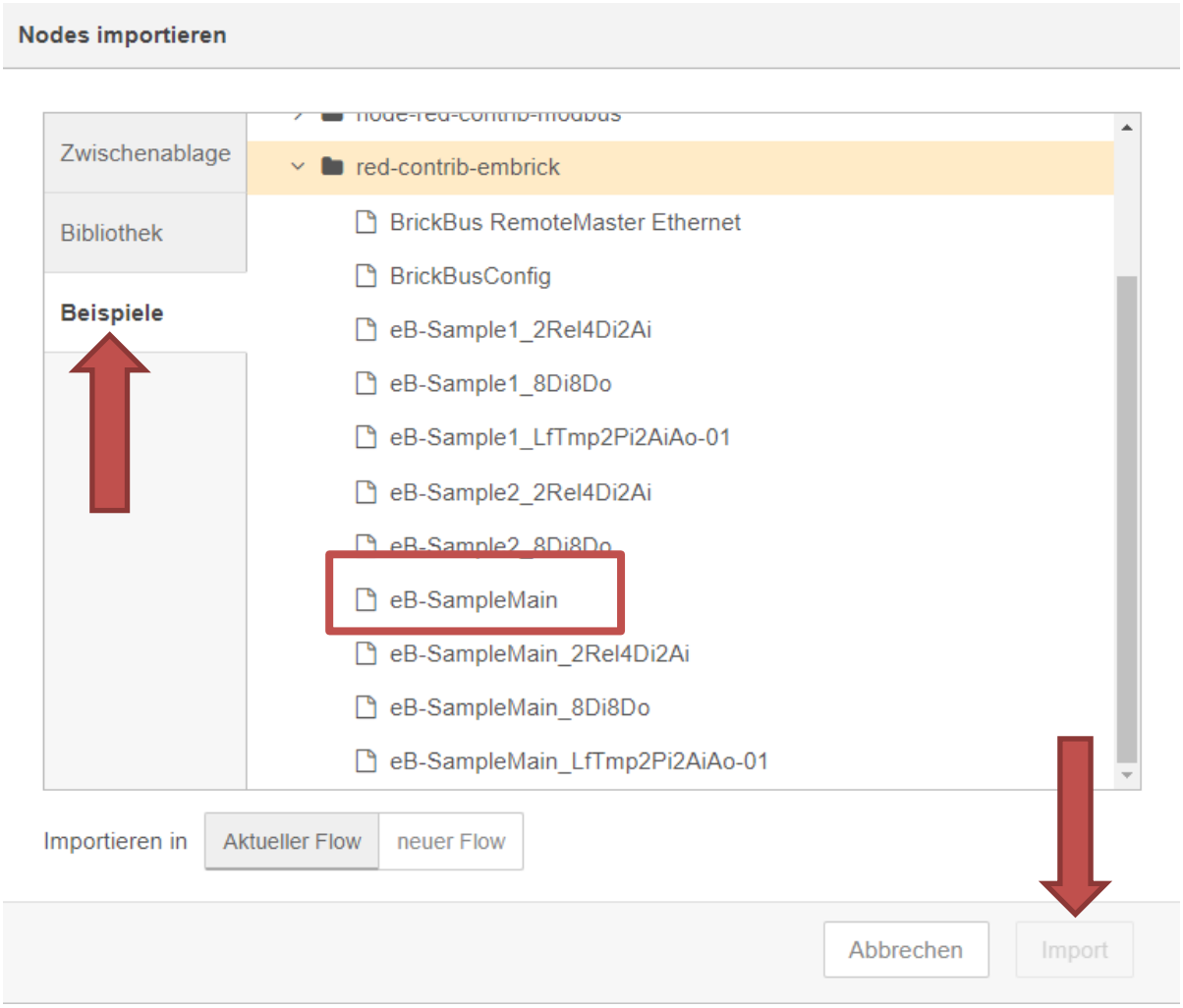


Figure 52 Add a Sample Application

If you have all three Bricks for the example you can take eB-SampleMain to load the full Sample Application and click then on Import.

When you have only one of the Brick Modules you can easily take the one you have to load a Sample Application of the Module.

9.6 Start and explore the Functionality of emBrick Nodes

You can find our embrick Nodes after Installation in Left Sidebar. You can easily search in Search Bar for the exactly node you want or scroll down to embrick there you can find our all nodes.

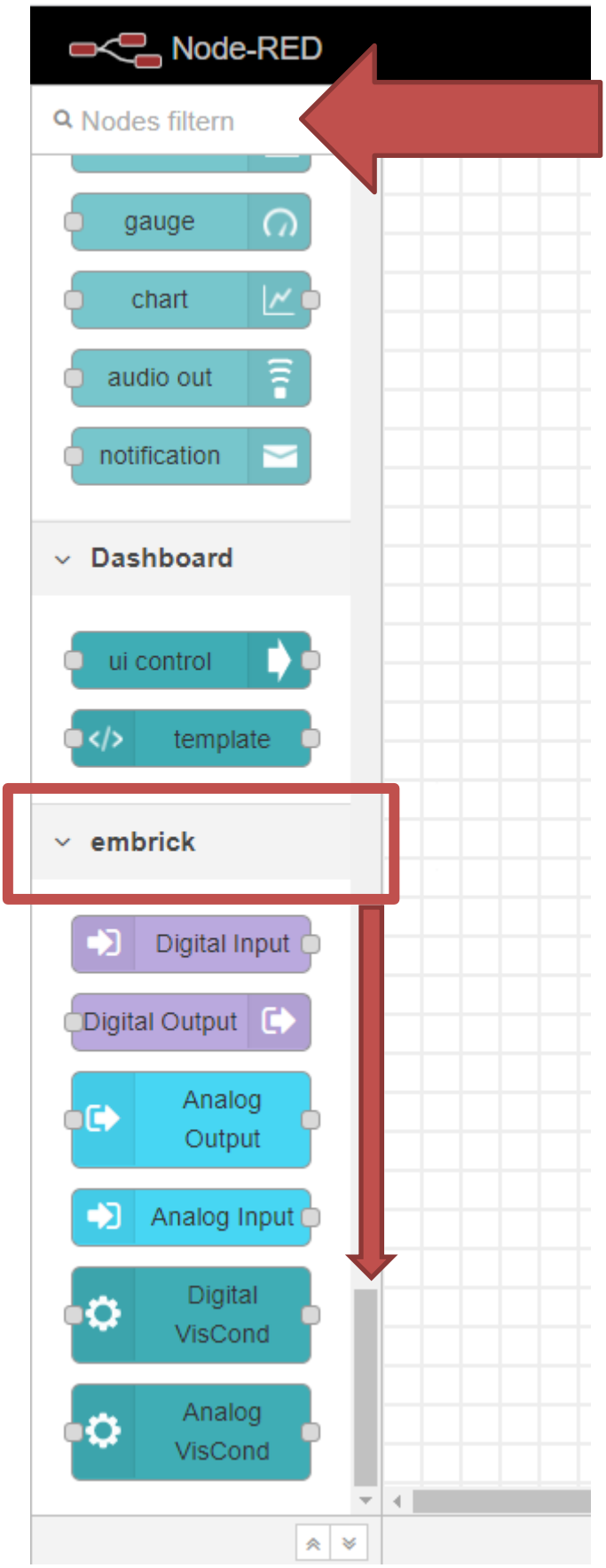


Figure 53 Find our Nodes

9.6.1 Digital Input

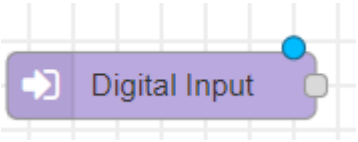


Figure 54 Digital Input

With the Digital Input Node, you can show the State of a Digital Input from your Brick.
To configure the Node, we DoubleClick on the Node and the Node Edit Window will be open.

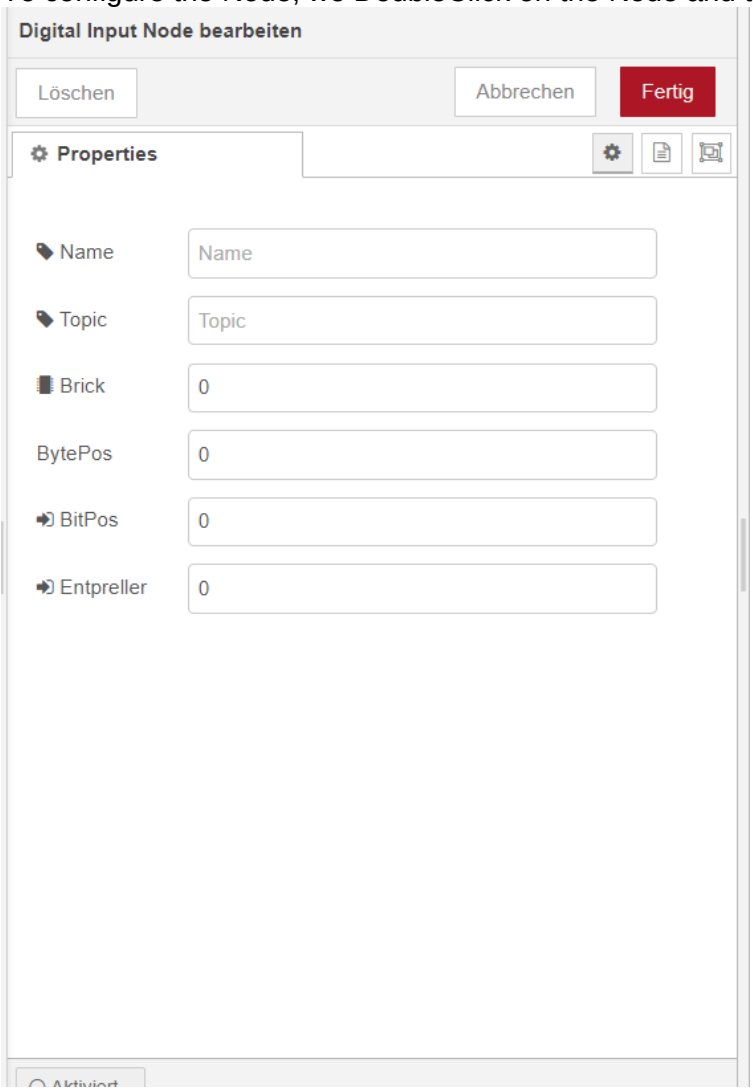


Figure 55 Digital Input Edit

1. Name: Here you can change the Name of the Node to have a better Overview on your Project.
2. Topic: This can let be empty is irrelevant for us.
3. Brick: Here type the Brick Nr. from which you want read the Digital Input. The Brick Number begins from 0 also the first Brick is 0.
4. BytePos: Here type the Byte Position in which the Digital Input is placed. The first Byte Position is 1, because on the 0 is the Status of the Brick.
5. BitPos: Here type which Digital Input you want to read. Digital Input goes from 0 to 7. The first Digital Input is on 0.
6. Entpreller: Is the Debounce function. The Debounce can be placed from 0 to 10000 milli-seconds. Entpreller means when the Input is for the placed milliseconds 1, then it gives a 1 back. Otherwise, it gives a 0 back.

9.6.2 Digital Output

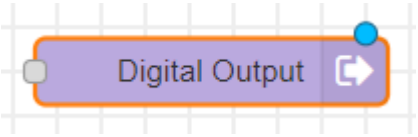


Figure 56 Digital Output

With the Digital Output Node, you can place the State of a Digital Output from your Brick to 1 or 0.
To configure the Node, we DoubleClick on the Node and the Node Edit Window will be open.

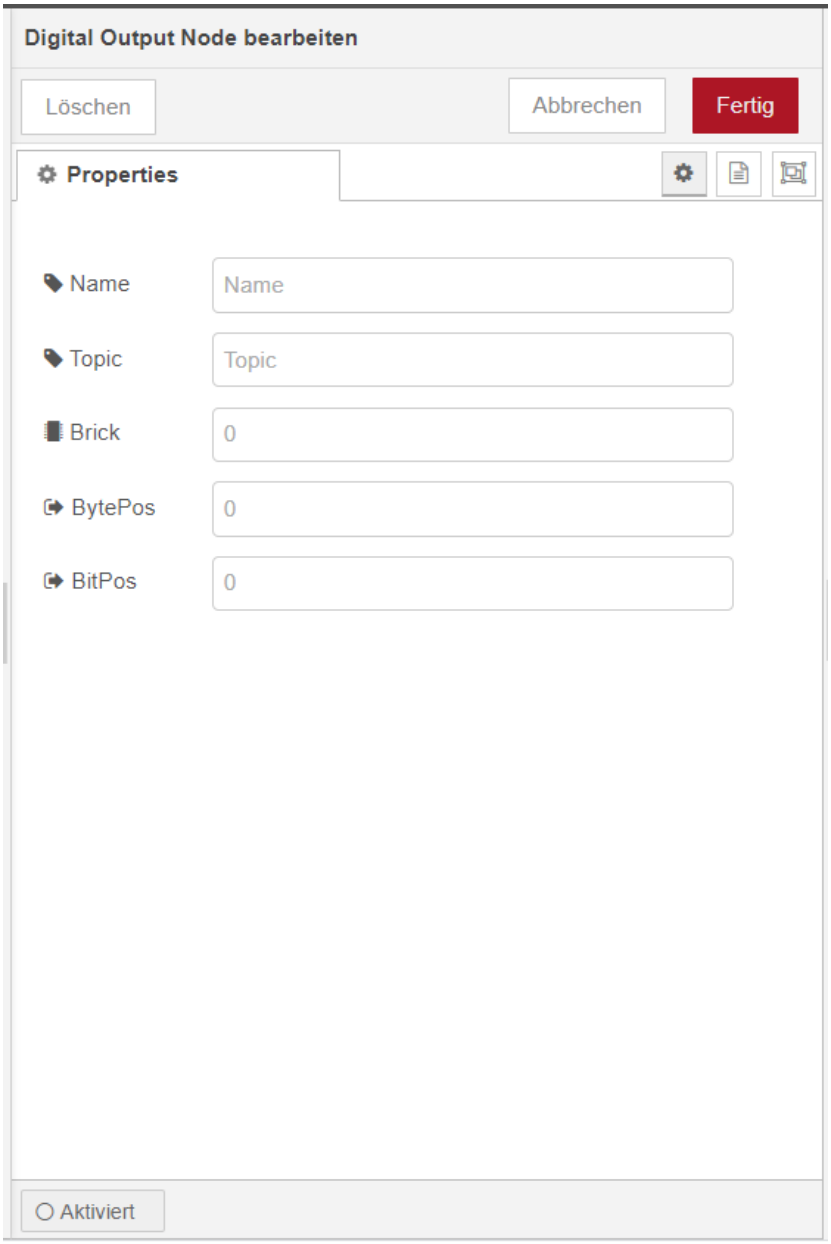


Figure 57 Digital Output Edit

1. Name: Here you can change the Name of the Node to have a better Overview on your Project.
2. Topic: This can let be empty is irrelevant for us.
3. Brick: Here type the Brick Nr. from which you want placed the Digital Output. The Brick Number begins from 0 also the first Brick is 0.
4. BytePos: Here type the Byte Position in which the Digital Output is placed. The first Byte Position is 0.

5. BitPos: Here type which Digital Output you want to place the state. Digital Output goes from 0 to 7. The first Digital Output is on 0.

9.6.3 Analog Input

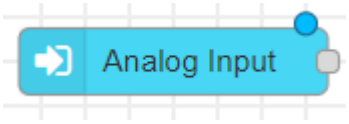


Figure 58 Analog Input

With the Analog Input Node, you can show the current State of the Analog Input from your Brick. To configure the Node, we DoubleClick on the Node and the Node Edit Window will be open.

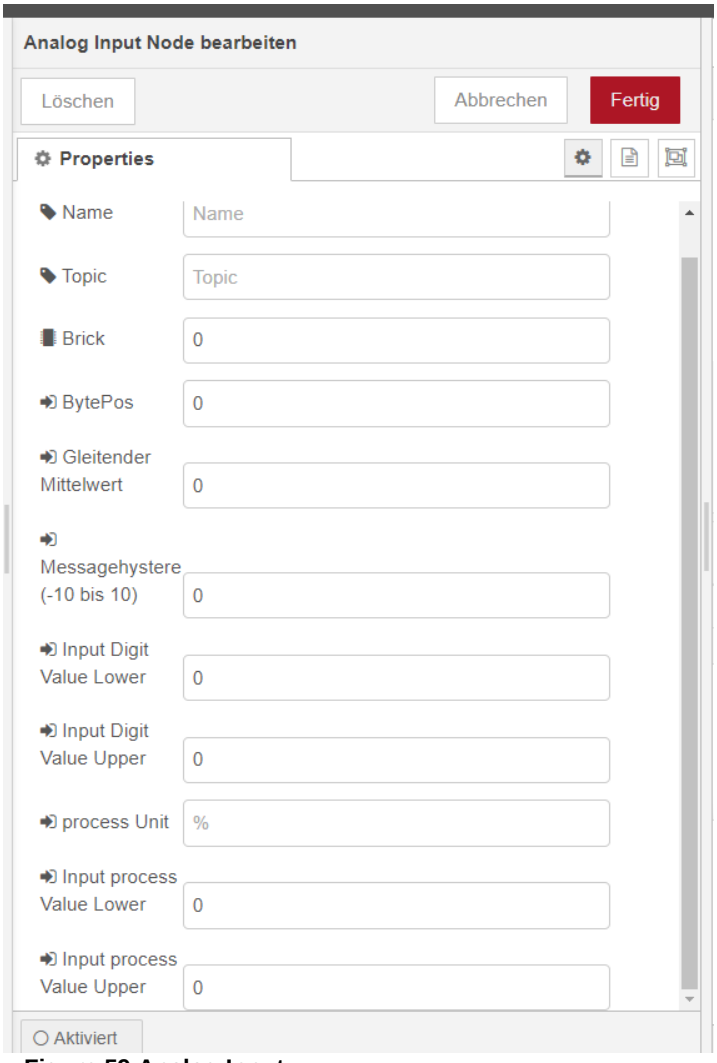


Figure 59 Analog Input

1. Name: Here you can change the Name of the Node to have a better Overview on your Project.
2. Topic: This can let be empty is irrelevant for us.
3. Brick: Here type the Brick Nr. from which you want read the Anaglo Input. The Brick Number begins from 0 also the first Brick is 0
4. BytePos: Here type the Byte Position in which the Analog Input is placed. The first Byte Position is 1, because on the 0 is the Status of the Brick

5. Gleitender Mittelwert: Calculate the Mean of the Input Values. You can type a number between 0 and 100.
6. Messagehystere: Change the Input Value only when it is bigger or smaller than the given value. Value can be placed from 0 to 10.
7. Input Digit Value Lower: Type here lower Digit Value which can be different for every Analog Input.
8. Input Digit Value Upper: Type here upper Digit Value which can be different for every Analog Input.
9. Process Unit: Here you can type the Process Unit of the Input, like mA, V or %.
10. Input Process Value Lower: Type here the lower Process Value.
11. Input Process Value Upper: Type here the upper Process Value.

9.6.4 Analog Output

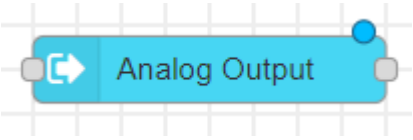


Figure 60 Analog Output

With the Digital Output Node, you can placed the State of a Digital Output from your Brick to 1 or 0.
To configure the Node, we DoubleClick on the Node and the Node Edit Window will be open.

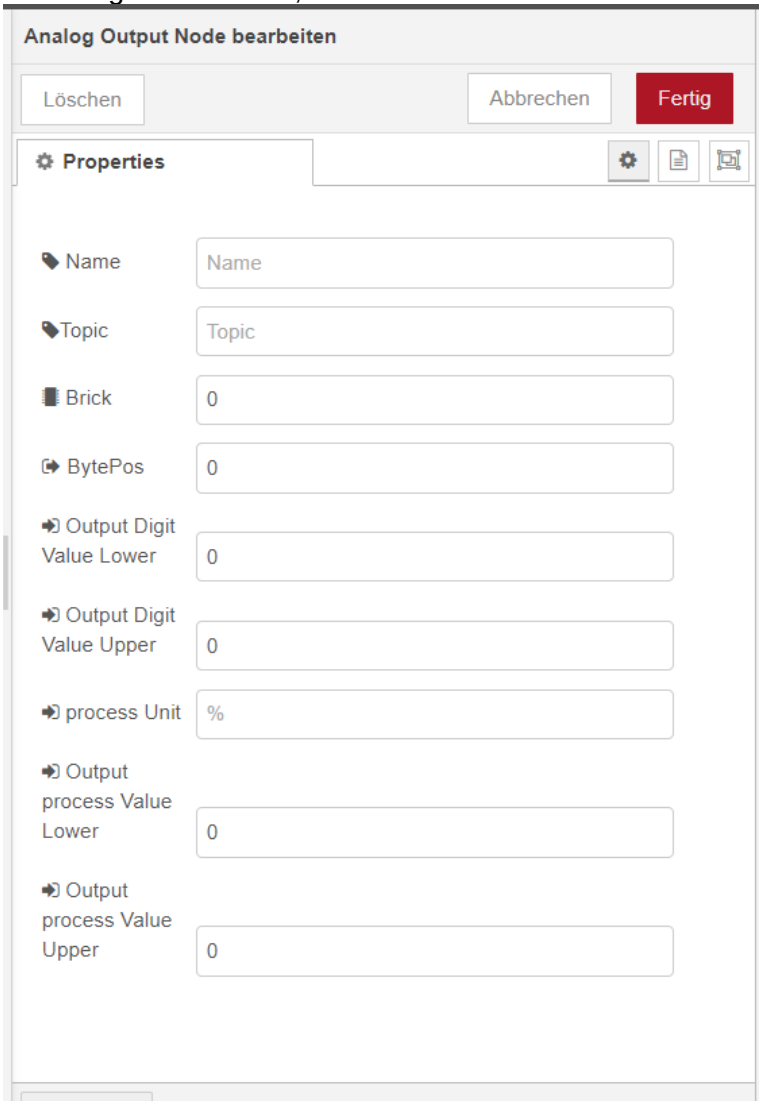


Figure 61 Analog Output Edit

1. Name: Here you can change the Name of the Node to have a better Overview on your Project.
2. Topic: This can let be empty is irrelevant for us.
3. Brick: Here type the Brick Nr. from which you want read the Anaglo Input. The Brick Number begins from 0 also the first Brick is 0
4. BytePos: Here type the Byte Position in which the Analog Output will write. The first Byte Position is 0.
5. Output Digit Value Lower: Type here lower Digit Value which can be different for every Analog Output.
6. Output Digit Value Upper: Type here upper Digit Value which can be different for every Analog Output.
7. Process Unit: Here you can type the Process Unit of the Input, like mA, V or %.
8. Output Process Value Lower: Type here the lower Process Value.
9. Output Process Value Upper: Type here the upper Process Value.

9.6.5 Digital Vis

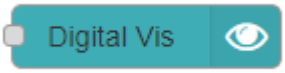


Figure 62 Digital Vis

Show the Digital Input or Output in a Dashboard with the name you can modify in the Configuration and the current state (yellow circle for 1 & black circle for 0).

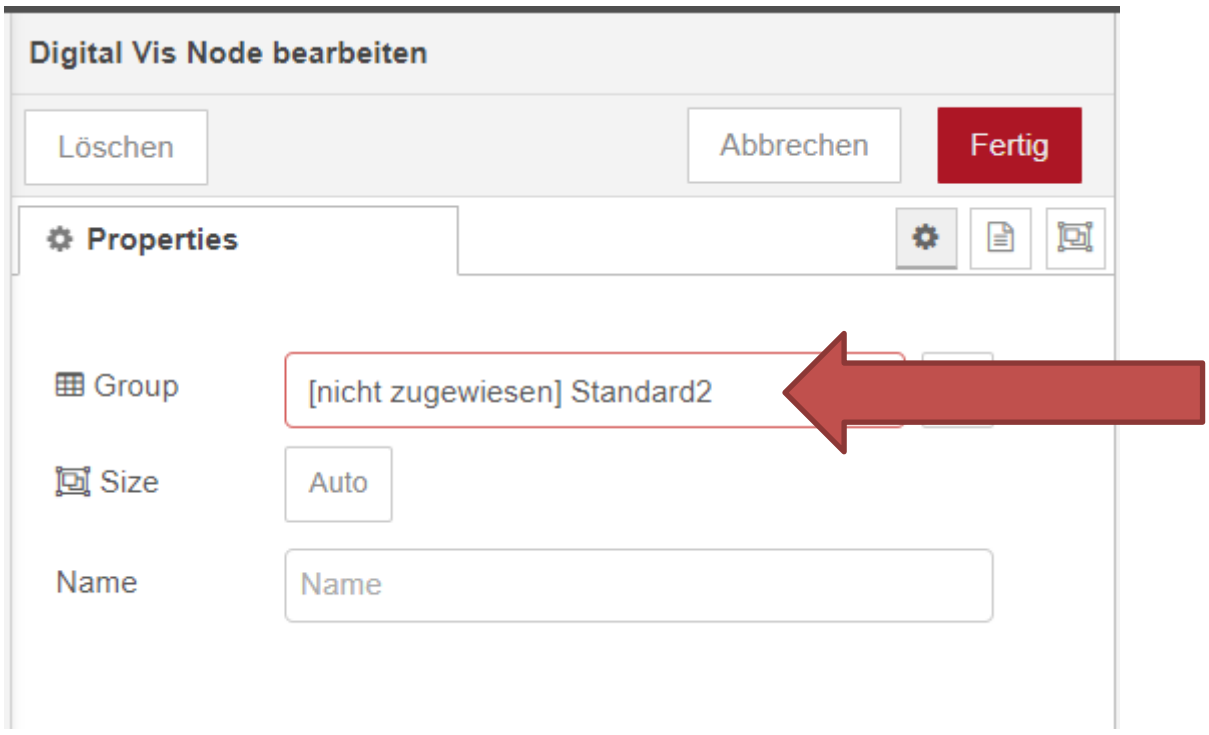


Figure 63 Digital Vis Edit

Group: This Field is required. Here you select or create the Site on the Dashboard. The Dashboard you can reach when you type “localhost:1880/ui” in the Internet Browser.

Size: configured the Size of the Node on the Dashboard.

Name: Here you can change the Name of the Node. The name you will give will show in the Node-Red Site and on Dashboard.

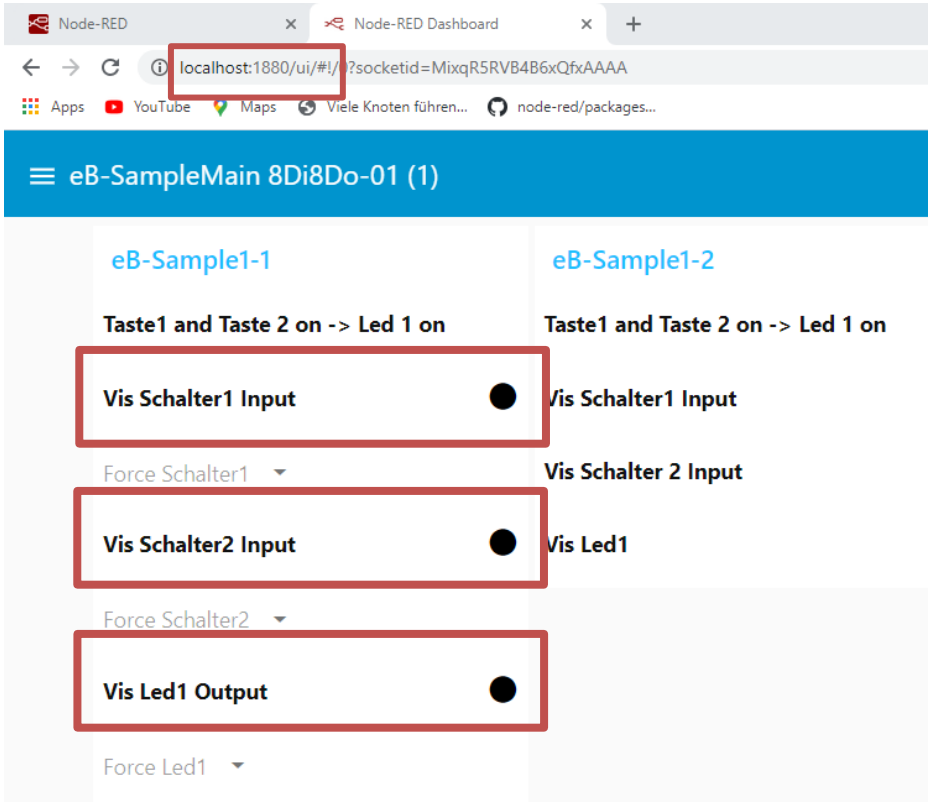


Figure 64 Digital Vis on Dashboard

This is the Main Dashboard Page of the Example Application. Like you see in the Browser, we type localhost:1880/ui and press Enter

In the eB-Sample1-1 shown the Digital Vis in Dashboard. First stand the Name of the Node then the current State. They are current black (0).

9.6.6 Analog Vis

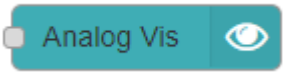


Figure 65 Analog Vis

Show the Analog Input or Output in a Dashboard with the name you can modify in the Configuration and the current state if you give in the Analog Input or Output the Process unit and the lower upper Process values. It will show the current Value in the given Process Unit, if not it will give the current value in Digits.

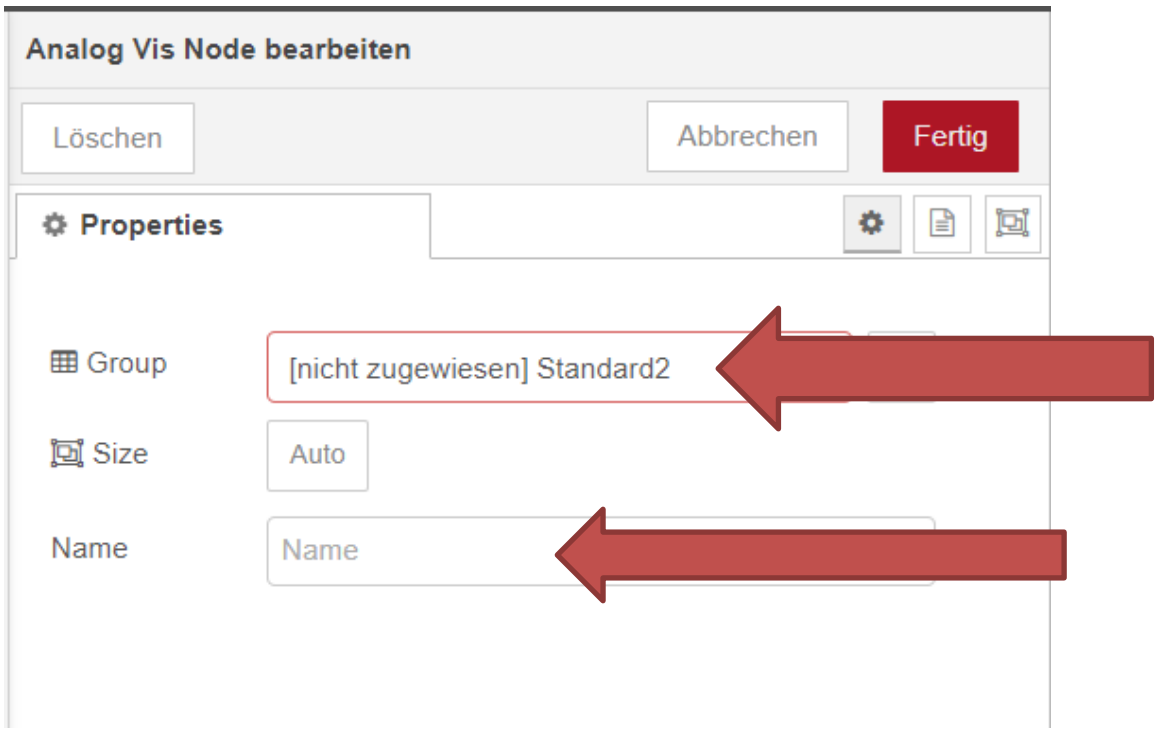


Figure 66 Analog Vis Edit

Group: This Field is required. Here you select or create the Site on the Dashboard. The Dashboard you can reach when you type “localhost:1880/ui” in the Internet Browser.

Size: configured the Size of the Node on the Dashboard.

Name: Here you can change the Name of the Node. The name you will given will shown in the Node-Red Site and on Dashboard.

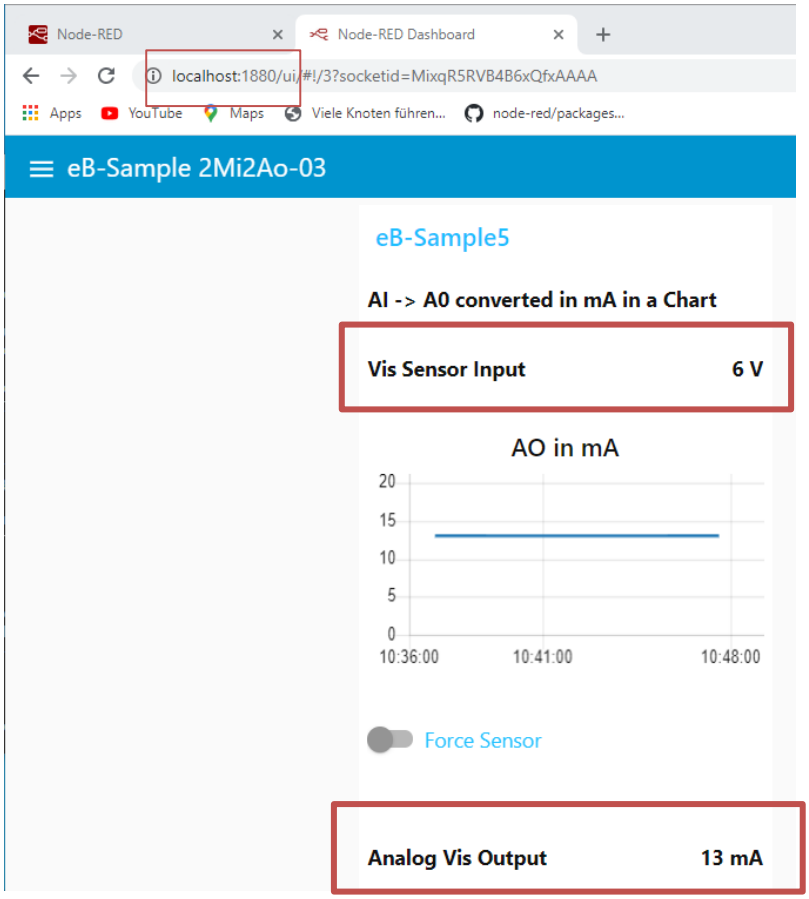


Figure 67 Analog Vis on Dashboard

9.6.7 Digital Force

Figure 68 Digital Force

The Digital Force Node give you the opportunity to force the Input or Output to 1 or 0. It's a big help by Debug or Test something.

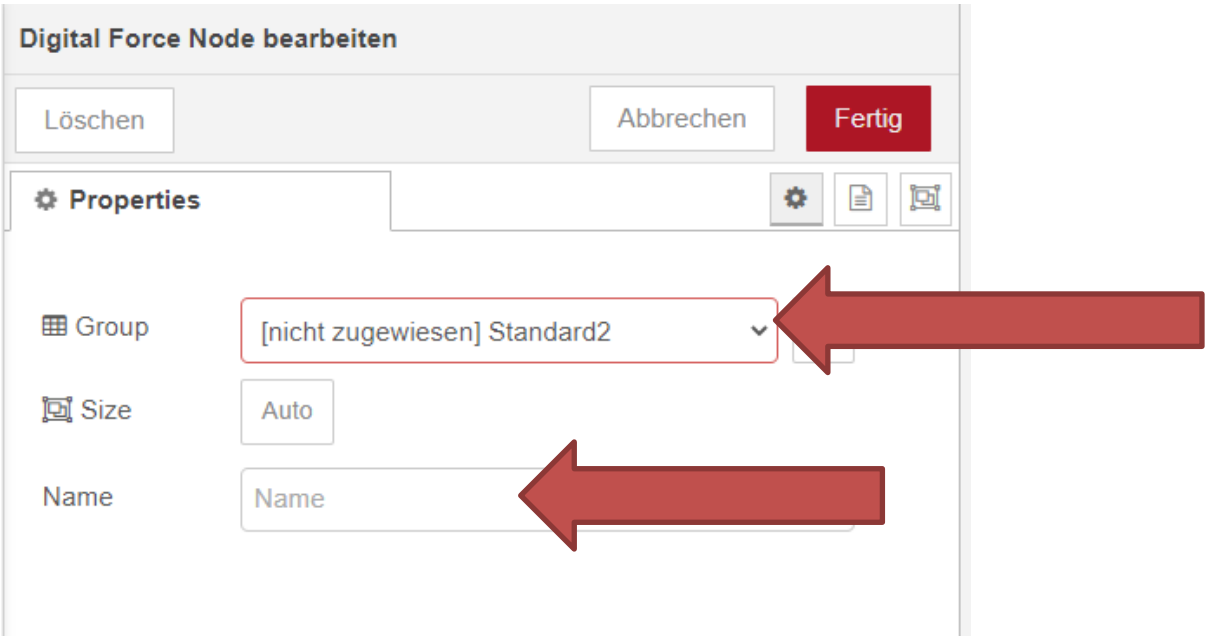


Figure 69 Digital Force Edit

Group: This Field is required. Here you select or create the Site on the Dashboard. The Dashboard you can reach when you type "localhost:1880/ui" in the Internet Browser.

Size: configured the Size of the Node on the Dashboard.

Name: Here you can change the Name of the Node. The name you will give will show in the Node-Red Site and on Dashboard.

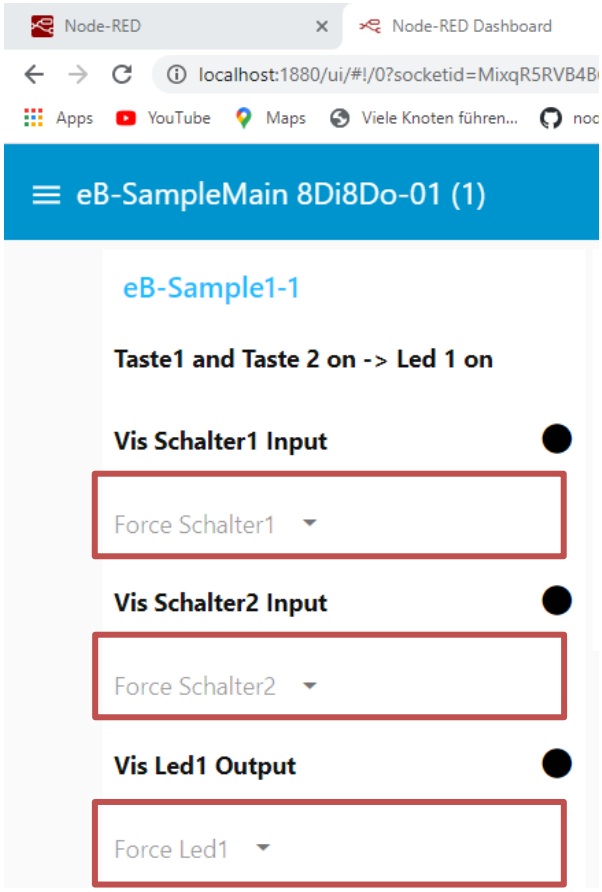


Figure 70 Digital Vis on Dashboard

9.6.8 Analog Force

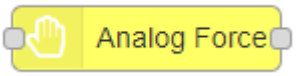


Figure 71 Analog Force

The Analog Force Node give you the opportunity to force the Input or Output to the input value. It's a big help by Debug or Test something.

Figure 72 Analog Force Edit

Group: This Field is required. Here you select or create the Site on the Dashboard. The Dashboard you can reach when you type “localhost:1880/ui” in the Internet Browser.

Size: configured the Size of the Node on the Dashboard.

Name: Here you can change the Name of the Node. The name you will give will show in the Node-Red Site and on Dashboard.

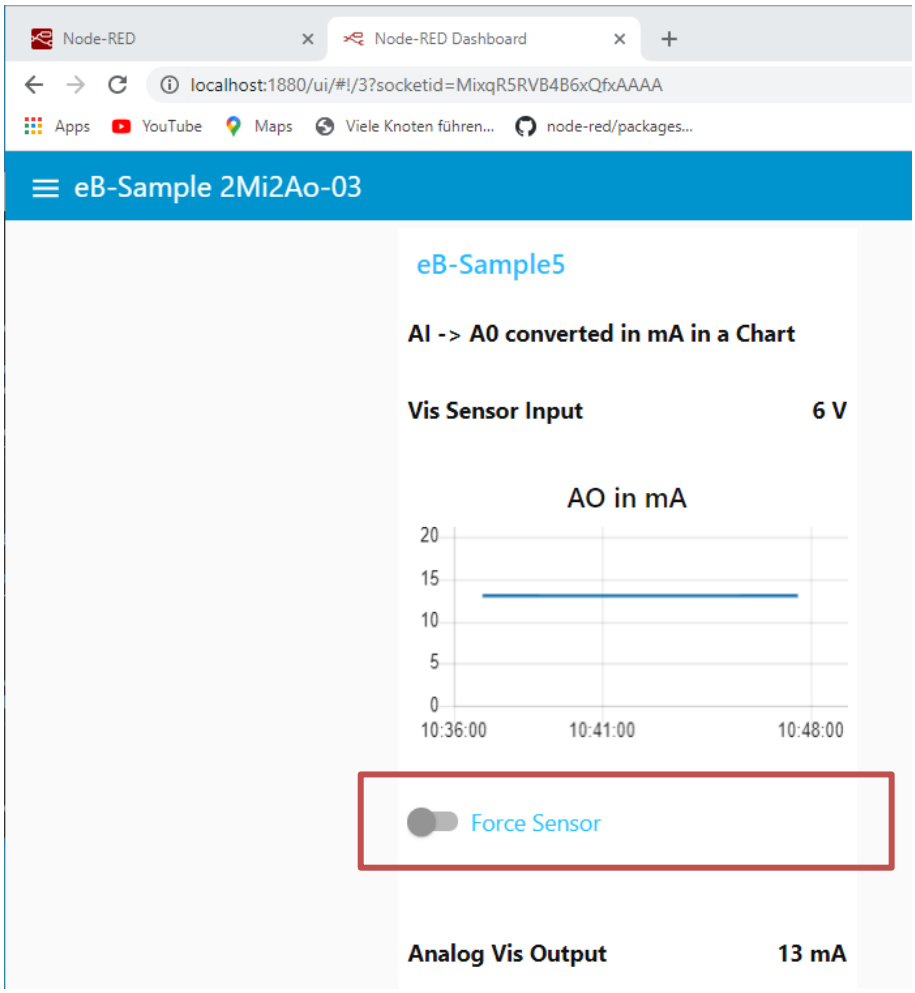


Figure 73 Analog Force on Dashboard

9.7 Create your own application

To create your own Application, start the Node-Red and delete the Sample Application. Then import the “BrickBus RemoteMaster Ethernet” or “BrickBus RemoteMaster Modbus” which you prefer to connect with the Remote Master from Example Folder.

After that configure the “Config Node” with the Remote Master Ip-Address or Modbus Port.

Now we are ready to create your own application.
You can drag drop our Embrick Nodes to read a Digital Input or write an Analog Output.