Summary

This practical report documents all necessary steps to control the programmable logic controller of an industrial plant with a cloud computing platform via a single board computer. The programmable logic controller from B&R was selected because of the required interfaces. The industrial plant to be controlled is a storage station from Festo. Through the partnership with Microsoft, the "Azure" cloud computing platform will be used for this project. A "Raspberry Pi 3" with the operating system "Windows 10 IoT Core" is used for the single-board computer.

This practical report also provides insights into the activities and tasks carried out during the internship. It presents hardware and software used for programming, testing and execution.

After extensive research on the functionalities of the individual components, the hardware is set up and checked for function. This is easily possible with almost all components by default settings. Since many technical components have to communicate with each other, each individual component is equipped step by step with rudimentary functions and then checked whether communication with the following components is possible.

Inhalt

Einleitung

This paper deals with the connection and control of a supply chain to the Microsoft Cloud Germany. The following figure shows schematically how the individual elements of the project work together (see Figure 1 Structure of the project). The supply chain is simulated with a Festo "Storing Station" and controlled by a B&R PLC (programmable logic controller). The PLC is connected to the Raspberry Pi via an Ethernet interface. This Ethernet interface is used to communicate with the PLC via a router using the Modbus TCP protocol. The Raspberry Pi is also connected to the Internet via the router. This connection is used to communicate with Microsoft Azure. The Raspberry Pi serves as interpreter of the variables of Azure, which are stored in the cloud in the programming language Json. The C# code on the Raspberry Pi interprets these variables and forwards the required parameters to the PLC via Modbus TCP. The PLC communicates digitally or binary with the rotation and linear motor of the robot arm. In order to control the Storing Station, the variables are adapted in the web interface provided by Azure. The Raspberry Pi then interprets the commands and converts them into control commands for the PLC, which controls the Storing Station with its I/O signals.

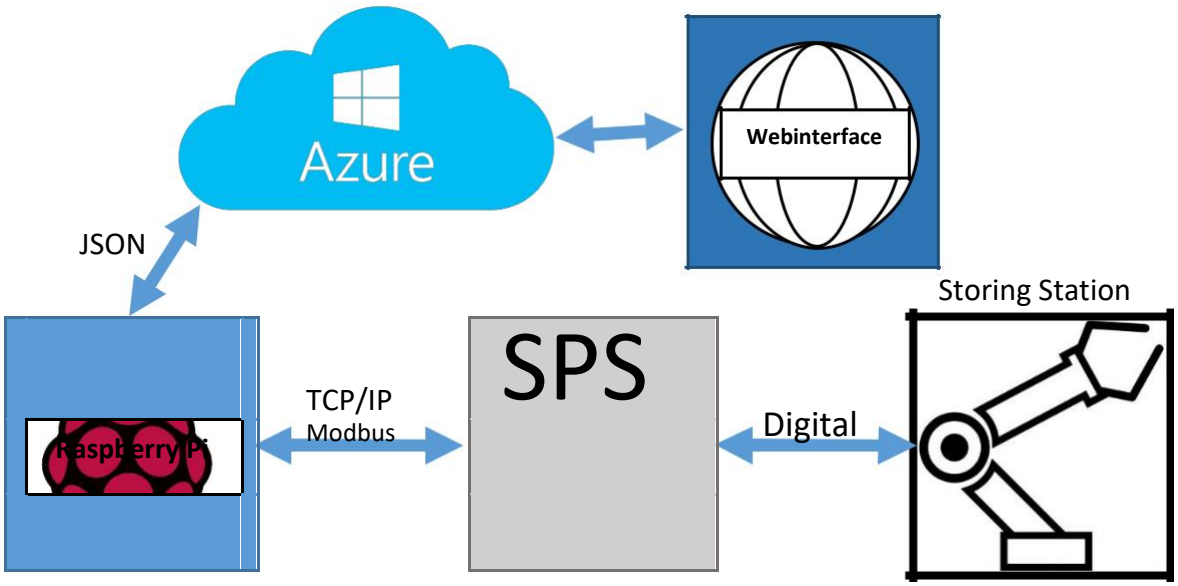


Abbildung 1 Aufbau des Projektes

Kommunikation

The following figure describes in more detail how the communication between Rasberry Pi and Storing Station is implemented (see Figure 2, Local Communication). Between Rasperry Pi and Storing Station is the PLC where a program is running. The PLC has input and output Modbus registers and digital input and output interfaces. These Modbus registers and digital interfaces are linked together in the PLC program. This process takes place cyclically. This means that each program section is executed at the same time interval. It should be noted that with a microprocessor, as used in the Raspberry Pi, no clocked program sequence is to be expected as in a PLC. This must be taken into account when creating the source code and can be illustrated using an example. If the Raspberry Pi changes variable values during program execution, this can happen faster than the PLC clock can implement this change. If now on the Raspberry Pi the code to open the gripper is executed and immediately afterwards the closing, then this can be done in a single PLC cycle. The PLC then makes the last change in the cycle and the gripper remains closed. In order to solve this problem, it is necessary to query the processing status of the PLC each time.

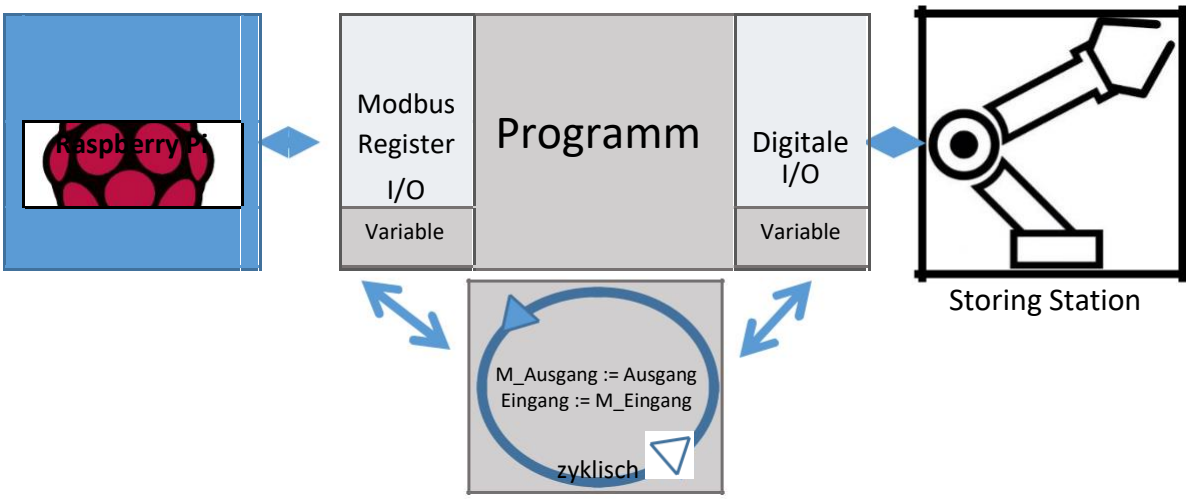


Abbildung 2 Darstellung der lokalen Kommunikation

Modbus TCP/IP

Modbus TCP/IP works and communicates via Ethernet. It is based on the master/slave or client/server architecture to enable communication with programmable logic controllers. Since Modbus is an open protocol, it has become a standard in industry. Modbus TCP has been part of the IEC 61158 standard since 2007.

Messages

Modbus offers various functions which are transmitted in the Modbus messages in order to be able to access registers. These functions cause either reading or writing of one or more registers. The Modbus Master can actively send and receive messages, while the Modbus Slave can only respond to them.

Read Funktionen des Modbus Masters aus dem Modbus Slave

Function 01 Read Coils contains the readout of one or more coils.

Function 02 Read Discrete Inputs contains the readout of one or more discrete inputs.

Function 03 Read Holding Registers contains the readout of one or more holding registers.

Function 04 Read Input Registers contains the reading of one or more input registers.

Write Funktionen des Modbus Master in den Modbus Slave

Function 05 Write Single Coil contains the writing of a coil.

Function 06 Write Single Register contains the writing of a holding register.

Function 15 Write Multiple Coils contains the writing of one or more coils.

Function 16 Write Multiple Register contains the writing of one or more holding registers.

Register

The Modbus registers (see Figure 3 Modbus registers) consist of four different register types. These have different access rights such as read and write for the respective Modbus Master and Modbus Slave. The Modbus registers are numbered in groups. Thus a maximum number of 65536 is possible.

The coil registers contain 1 bit information. They can be read and written by the Modbus master. The Modbus slave can only read this register.

The Discrete Inputs registers contain 1 bit information. They can only be read by the Modbus master. The Modbus Slave can read and write this register.

The coil registers contain 16 bit information. They can be read and written by the Modbus master. The Modbus slave can only read this register.

The Discrete Inputs registers contain 16 bit information. They can only be read by the Modbus master. The Modbus Slave can read and write this register.

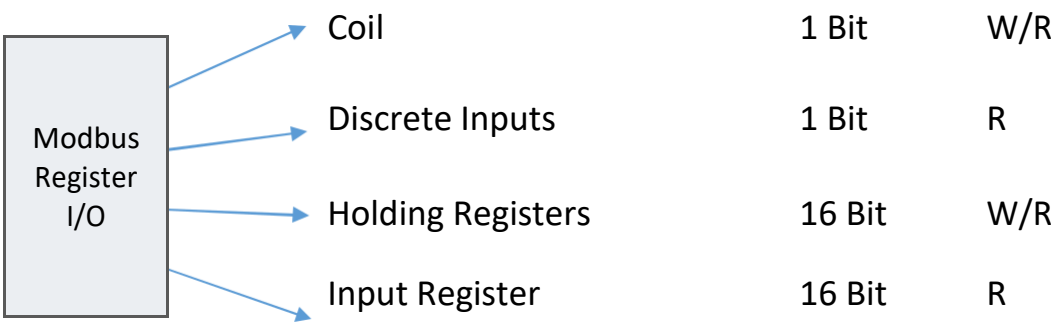


Abbildung 3 Modbus Register

SPS 1 Lager Station

SPS Hardware

The following figure shows a visualization of the components used in the "B&R Automation Studio" software (see Figure 4 PLC hardware view).

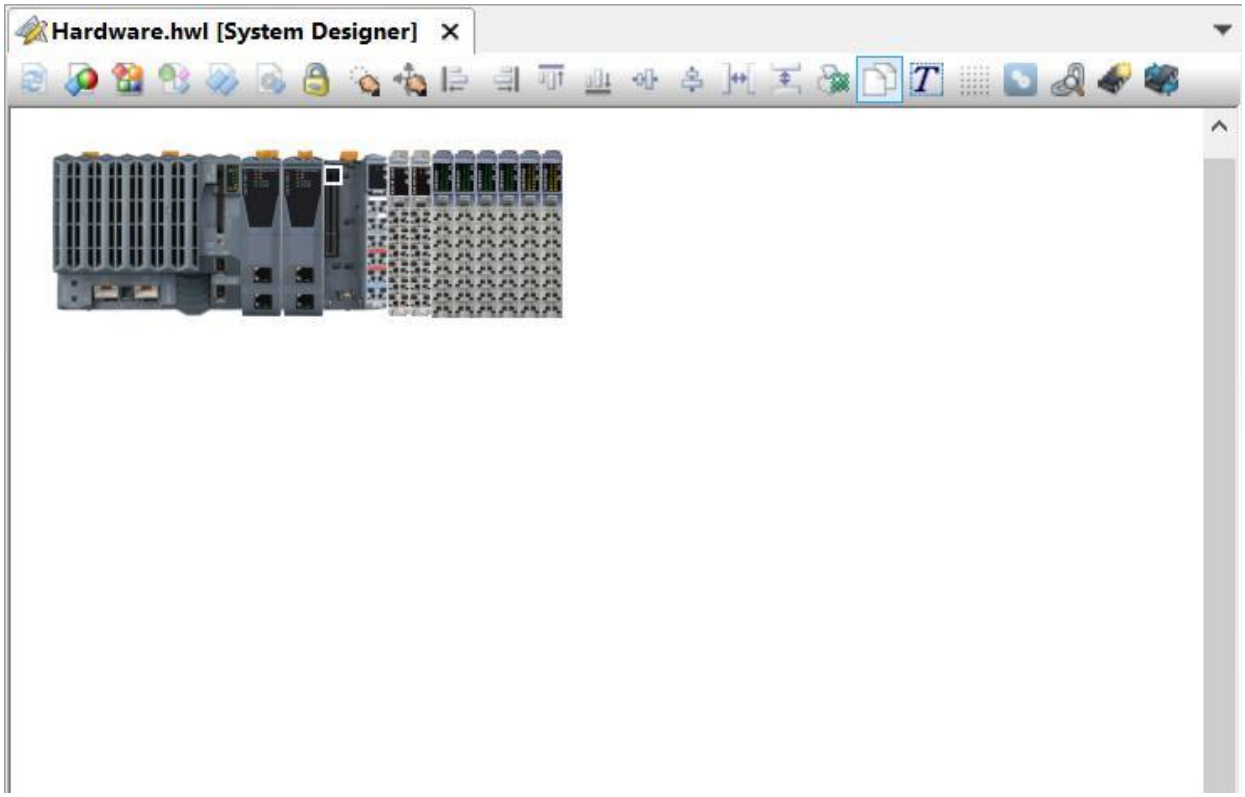


Abbildung 4 SPS Hardware Ansicht

The following figure lists the detailed structure in the "B&R Automation Studio" software (see Figure 5 PLC Hardware View Detail). The components that are installed from left to right in Figure 4 are listed from top to bottom in Figure 5.

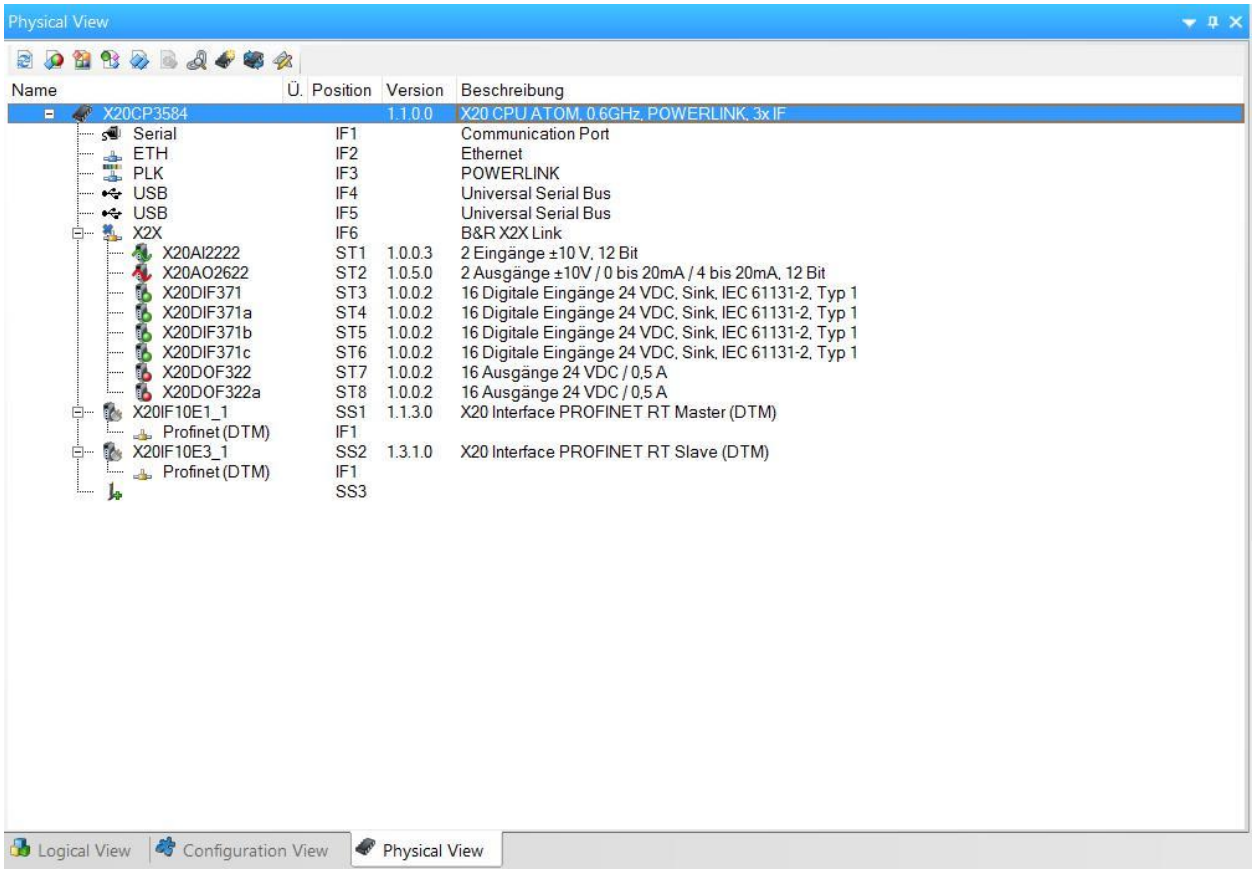
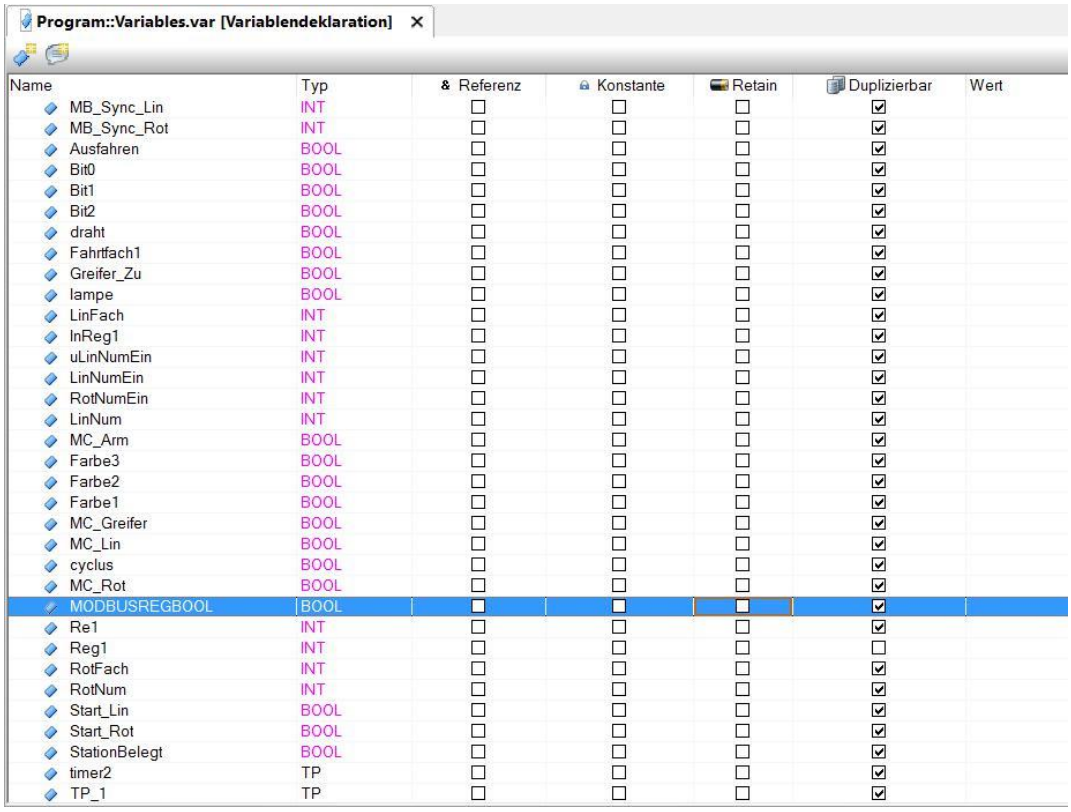


Abbildung 5 SPS Hardware Ansicht Detail

SPS I/O Zuordnung

I/O Speichertypen

In the variable declaration (see Figure 6 PLC variables with types 1, Figure 7 PLC variables with types 2, Figure 8 PLC variables with types 3) the variables for the inputs and outputs, Modbus and the program flow are declared.



Name	Typ	Referenz	Konstante	Retain	Duplizierbar	Wert
MB_Sync_Lin	INT	☐	☐	☐	☑	
MB_Sync_Rot	INT	☐	☐	☐	☑	
Ausfahren	BOOL	☐	☐	☐	☑	
Bit0	BOOL	☐	☐	☐	☑	
Bit1	BOOL	☐	☐	☐	☑	
Bit2	BOOL	☐	☐	☐	☑	
draht	BOOL	☐	☐	☐	☑	
Fahrtfach1	BOOL	☐	☐	☐	☑	
Greifer_Zu	BOOL	☐	☐	☐	☑	
lampe	BOOL	☐	☐	☐	☑	
LinFach	INT	☐	☐	☐	☑	
lnReg1	INT	☐	☐	☐	☑	
uLinNumEin	INT	☐	☐	☐	☑	
LinNumEin	INT	☐	☐	☐	☑	
RotNumEin	INT	☐	☐	☐	☑	
LinNum	INT	☐	☐	☐	☑	
MC_Arm	BOOL	☐	☐	☐	☑	
Farbe3	BOOL	☐	☐	☐	☑	
Farbe2	BOOL	☐	☐	☐	☑	
Farbe1	BOOL	☐	☐	☐	☑	
MC_Greifer	BOOL	☐	☐	☐	☑	
MC_Lin	BOOL	☐	☐	☐	☑	
cyclus	BOOL	☐	☐	☐	☑	
MC_Rot	BOOL	☐	☐	☐	☑	
MODBUSREGBOOL	BOOL	☐	☐	☐	☑	
Re1	INT	☐	☐	☐	☑	
Reg1	INT	☐	☐	☐	☐	
RotFach	INT	☐	☐	☐	☑	
RotNum	INT	☐	☐	☐	☑	
Start_Lin	BOOL	☐	☐	☐	☑	
Start_Rot	BOOL	☐	☐	☐	☑	
StationBelegt	BOOL	☐	☐	☐	☑	
timer2	TP	☐	☐	☐	☑	
TP_1	TP	☐	☐	☐	☑	

Abbildung 6 SPS Variablen mit Typen 1

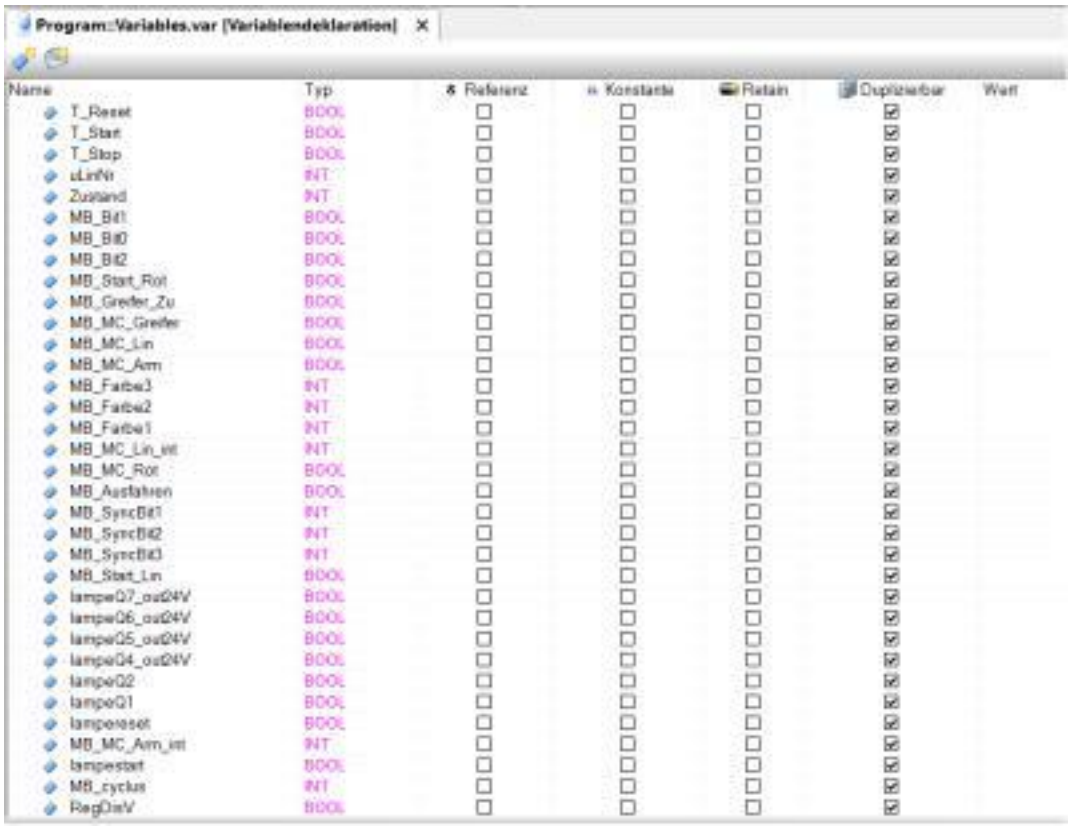


Abbildung 7 SPS Variablen mit Typen 2

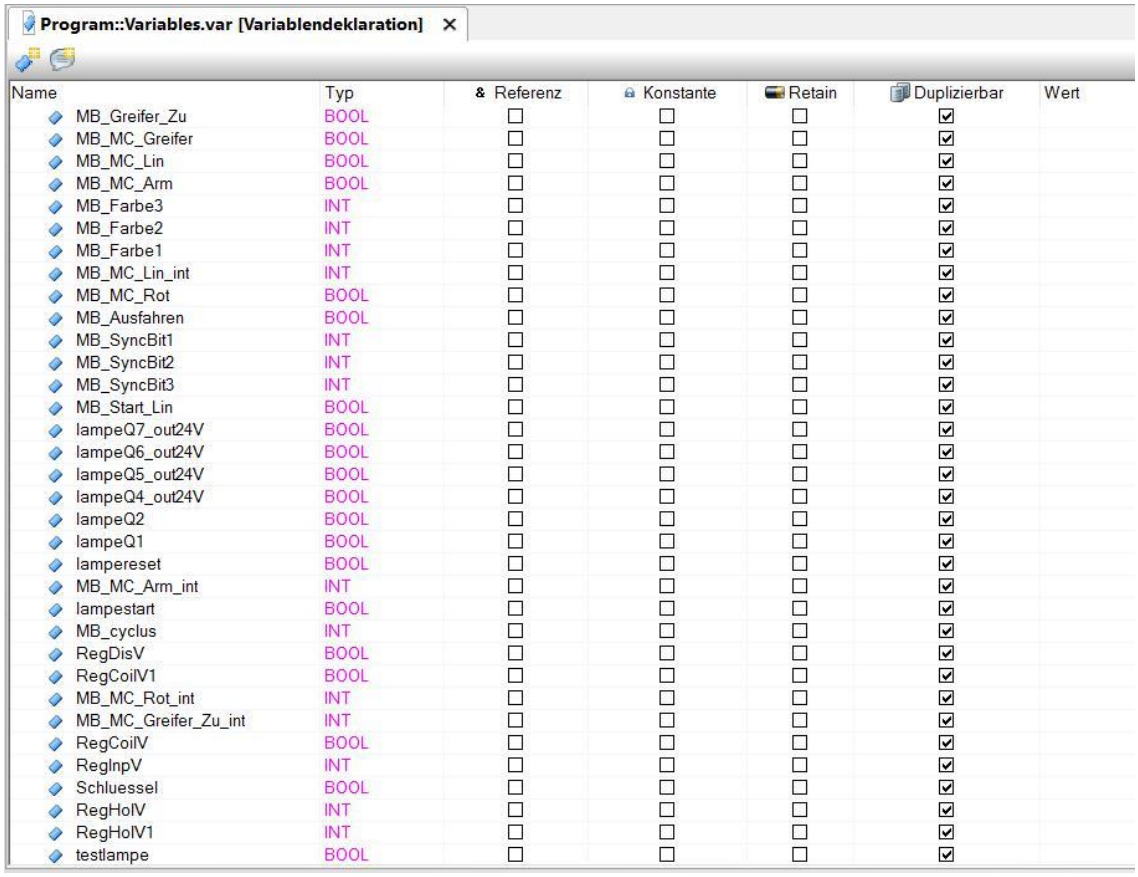
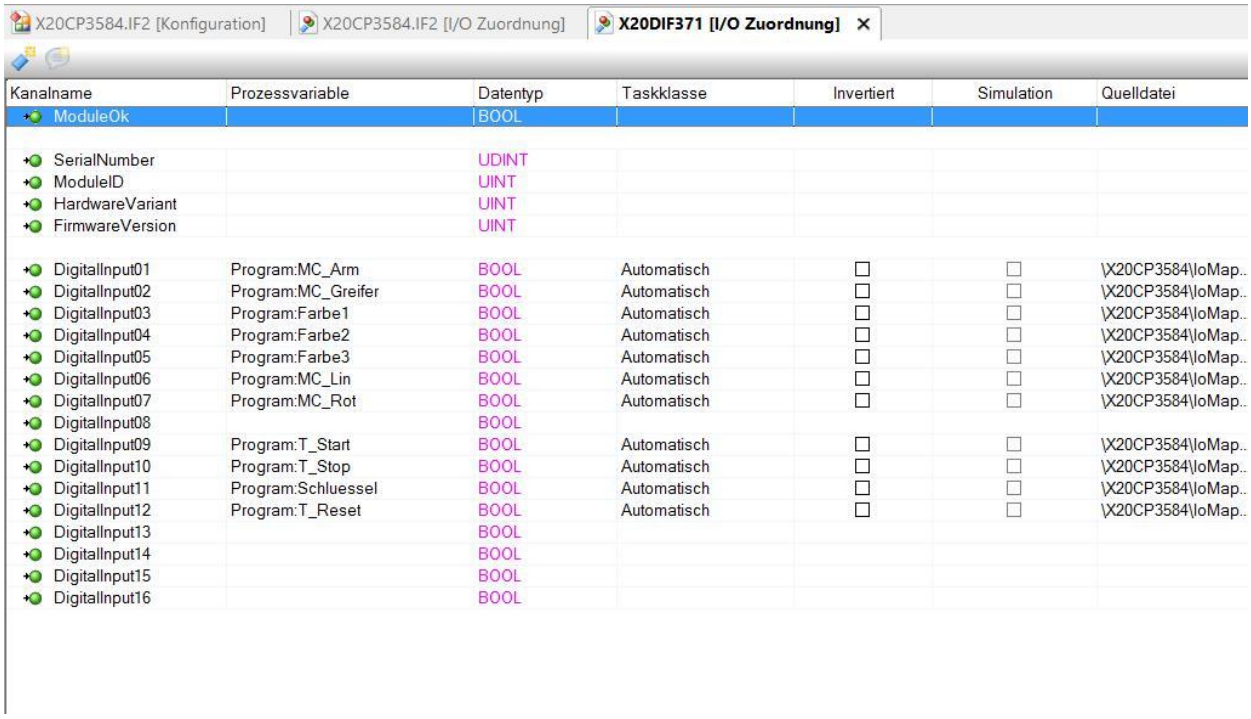


Abbildung 8 SPS Variablen mit Typen 3

I/O Hardwareeingang

In the input assignment of the PLC (see Figure 9 PLC Input Hardware), the interfaces DigitalInput01 to 16, the module X20DIF371, are assigned to the associated variables for the program code.



Kanalname	Prozessvariable	Datentyp	Taskklasse	Invertiert	Simulation	Quelldatei
ModuleOk		BOOL				
SerialNumber		UDINT				
ModuleID		UINT				
HardwareVariant		UINT				
FirmwareVersion		UINT				
DigitalInput01	Program:MC_Arm	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput02	Program:MC_Greifer	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput03	Program:Farbe1	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput04	Program:Farbe2	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput05	Program:Farbe3	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput06	Program:MC_Lin	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput07	Program:MC_Rot	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput08		BOOL				
DigitalInput09	Program:T_Start	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput10	Program:T_Stop	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput11	Program:Schluessel	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput12	Program:T_Reset	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap..
DigitalInput13		BOOL				
DigitalInput14		BOOL				
DigitalInput15		BOOL				
DigitalInput16		BOOL				

Abbildung 9 SPS Input Hardware

I/O Hardwareausgang

In the output assignment of the PLC (see Figure 10 PLC Output Hardware), the interfaces DigitalOutput01 to 16, the module X20DOF322, are assigned to the associated variables for the program code.

X20CP3584.IF2 [Konfiguration] | X20CP3584.IF2 [I/O Zuordnung] | X20DIF371 [I/O Zuordnung] | **X20DOF322 [I/O Zuordnung]**

Kanalname	Prozessvariable	Datentyp	Taskklasse	Invertiert	Simulation	Quelldatei
ModuleOk		BOOL				
SerialNumber		UDINT				
ModuleID		UINT				
HardwareVariant		UINT				
FirmwareVersion		UINT				
DigitalOutput01	Program:Ausfahren	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput02	Program:Greifer_Zu	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput03	Program:Bit0	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput04	Program:Bit1	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput05	Program:Bit2	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput06	Program:Start_Lin	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput07	Program:Start_Rot	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput08		BOOL				
DigitalOutput09	Program:lampestart	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput10	Program:lampereset	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput11	Program:lampeQ1	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput12	Program:lampeQ2	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput13	Program:lampeQ4_out24V	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput14	Program:lampeQ5_out24V	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput15	Program:lampeQ6_out24V	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
DigitalOutput16	Program:lampeQ7_out24V	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
StatusDigitalOutput01		BOOL				
StatusDigitalOutput02		BOOL				
StatusDigitalOutput03		BOOL				
StatusDigitalOutput04		BOOL				
StatusDigitalOutput05		BOOL				
StatusDigitalOutput06		BOOL				
StatusDigitalOutput07		BOOL				
StatusDigitalOutput08		BOOL				
StatusDigitalOutput09		BOOL				
StatusDigitalOutput10		BOOL				
StatusDigitalOutput11		BOOL				
StatusDigitalOutput12		BOOL				
StatusDigitalOutput13		BOOL				
StatusDigitalOutput14		BOOL				
StatusDigitalOutput15		BOOL				
StatusDigitalOutput16		BOOL				

Abbildung 10 SPS Output Hardware

I/O Modbus Eingänge und Ausgänge

In the Modbus register assignment of the PLC (see Figure 11 PLC Modbus I/O), the Modbus registers of the PLC are assigned to the associated variables for the program code.

X20CP3584.IF2 [Konfiguration] | **X20CP3584.IF2 [I/O Zuordnung]**

Kanalname	Prozessvariable	Datentyp	Taskklasse	Invertiert	Simulation	Quelldatei
NodeSwitch		USINT				
ActualNumberOfClien...		UINT				
PacketCount_1		DINT				
ErrorCount_1		DINT				
TimeSinceLastRequ...		DINT				
RegB1	Program:MB_Bit0	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
RegB2	Program:MB_Bit1	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
RegB3	Program:MB_Bit2	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
RegB4	Program:MB_Start_Lin	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
RegB5	Program:MB_Start_Rot	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
RegB6	Program:MB_Greifer_Zu	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
RegB7	Program:MB_Ausfahren	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
RegCoil	Program:RegCoilV	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
RegCoil1	Program:RegCoilV1	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
InRegB1	Program:MB_MC_Rot	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
InRegB2	Program:MB_MC_Lin	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
InRegB3	Program:MB_MC_Greifer	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
InRegB4	Program:MB_MC_Arm	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
InRegB5		BOOL				
InRegB6		BOOL				
InRegB7		BOOL				
InRegB8		BOOL				
InRegB9		BOOL				
RegDis	Program:RegDisV	BOOL	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg1	Program:MB_MC_Lin_int	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg2	Program:MB_MC_Rot_int	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg3	Program:MB_MC_Arm_int	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg4	Program:MB_MC_Greifer_Zu_int	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg5	Program:MB_Farbe1	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg6	Program:MB_Farbe2	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg7	Program:MB_Farbe3	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg8	Program:MB_cyclus	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg9	Program:MB_SyncBit1	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg10	Program:MB_SyncBit2	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg11	Program:MB_SyncBit3	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg12	Program:MB_Sync_Lin	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
InReg13	Program:MB_Sync_Rot	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
RegInp	Program:RegInpV	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
Reg1	Program:LinNum	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
Reg2	Program:Reg1	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
Reg3	Program:uLinNr	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...
Reg4	Program:RotNum	OCTET[2]	Automatisch	☐	☐	\X20CP3584\IoMap...

Abbildung 11 SPS Modbus I/O

SPS Modbus Modus, Register und Netzwerk Optionen

In the configuration of the network parameters of the PLC (see Figure 12 PLC Network Setting) the relevant network settings, Modbus register management (see Figure 13 PLC Mode Register Setting 1, Figure 14 PLC Mode Register Setting 2, Figure 15 PLC Mode Register Setting 3, Figure 16 PLC Mode Register Setting 4, Figure 17 PLC Mode Register Setting 5, Figure 18 PLC Mode Register Setting 6, Figure 19 PLC Mode Register Setting 7) and the general settings of Modbus are displayed.

(see Figure 20 PLC Mode Setting)

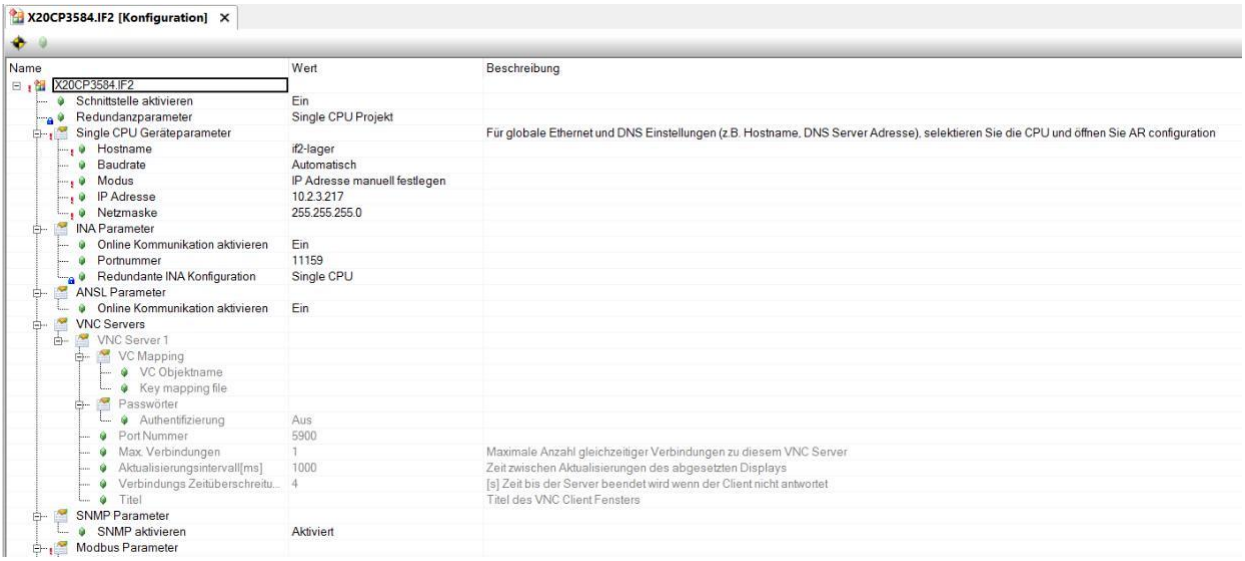


Abbildung 12 SPS Netzwerk Einstellung

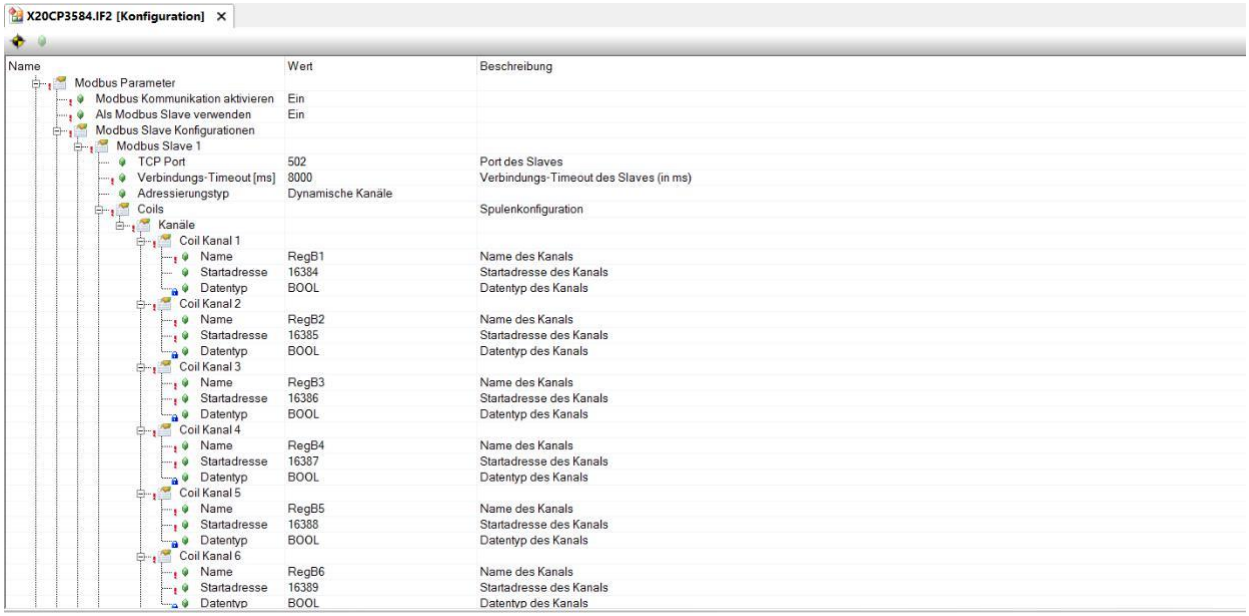


Abbildung 13 SPS Modus Register Einstellung 1

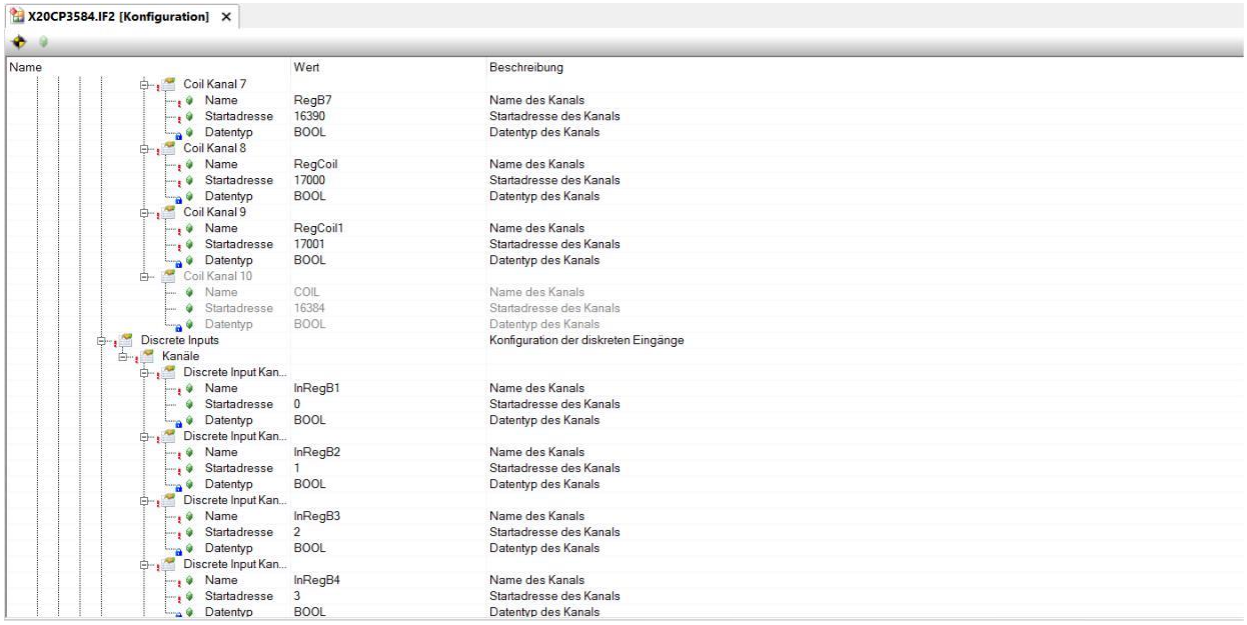


Abbildung 14 SPS Modus Register Einstellung 2

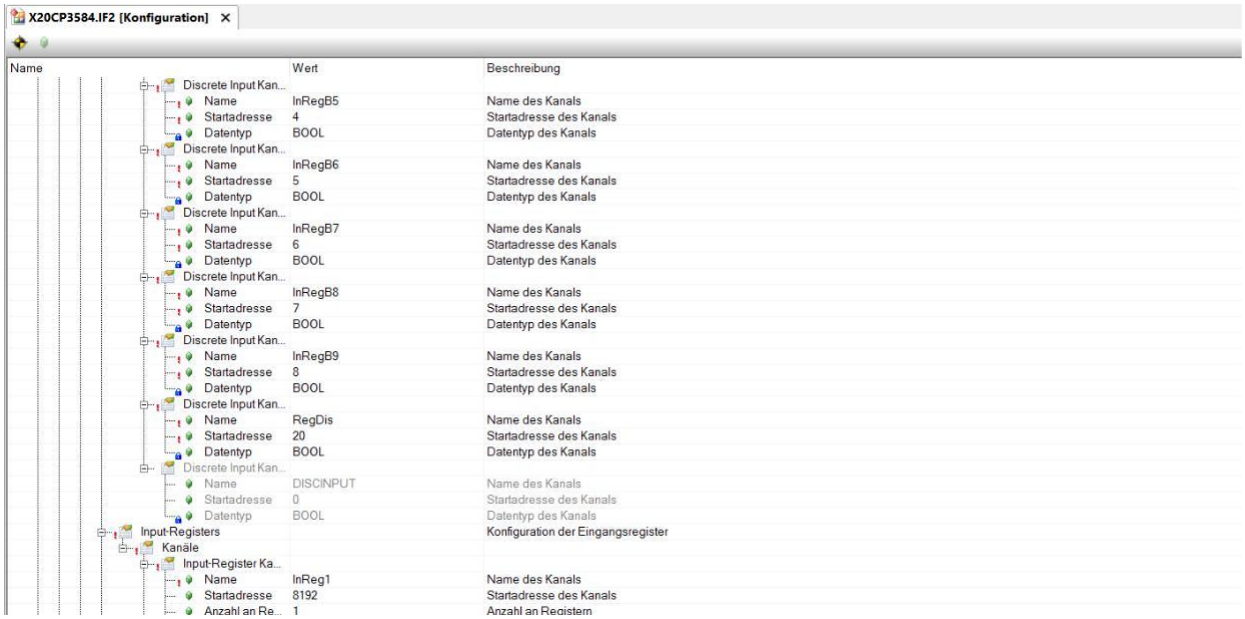


Abbildung 15 SPS Modus Register Einstellung 3

Abbildung 16 SPS Modus Register Einstellung 4

Abbildung 17 SPS Modus Register Einstellung 5

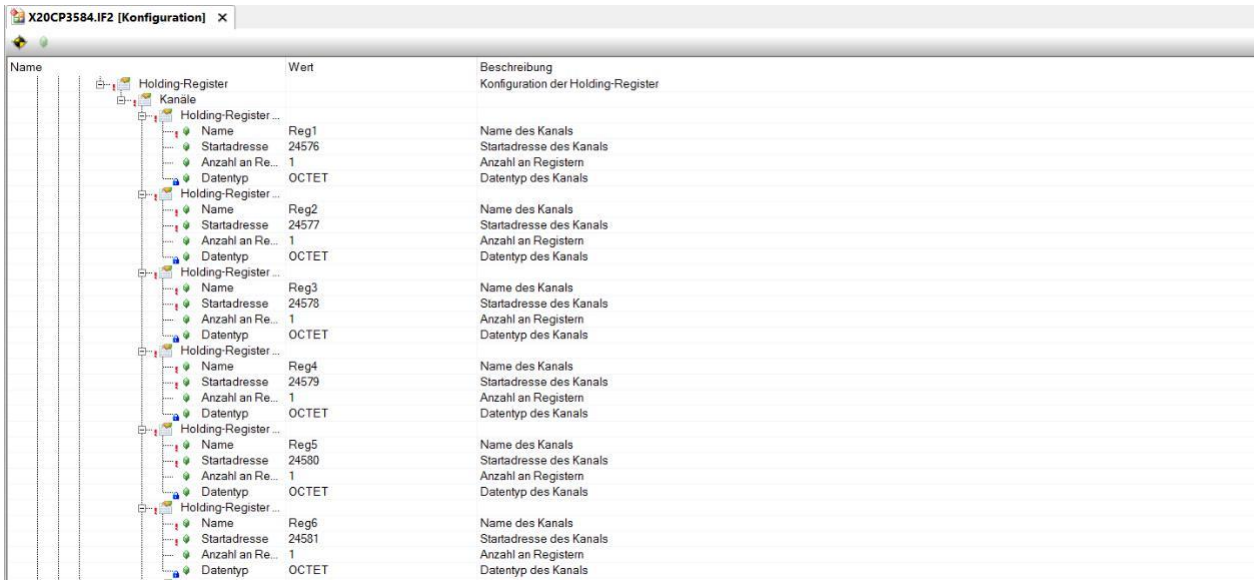


Abbildung 18 SPS Modus Register Einstellung 6

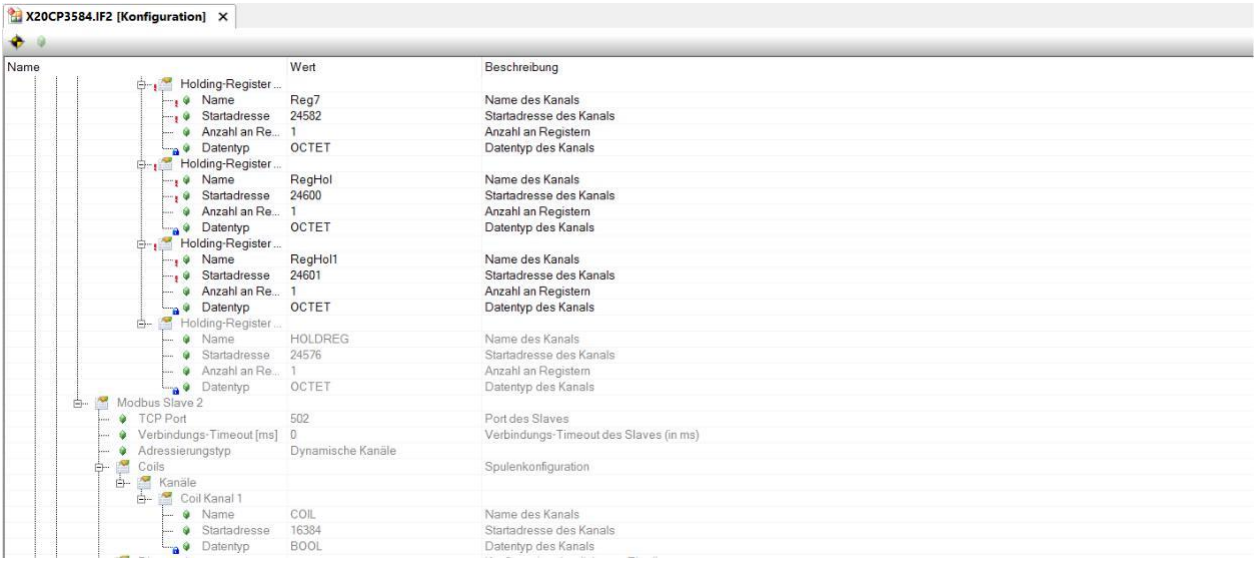


Abbildung 19 SPS Modus Register Einstellung 7

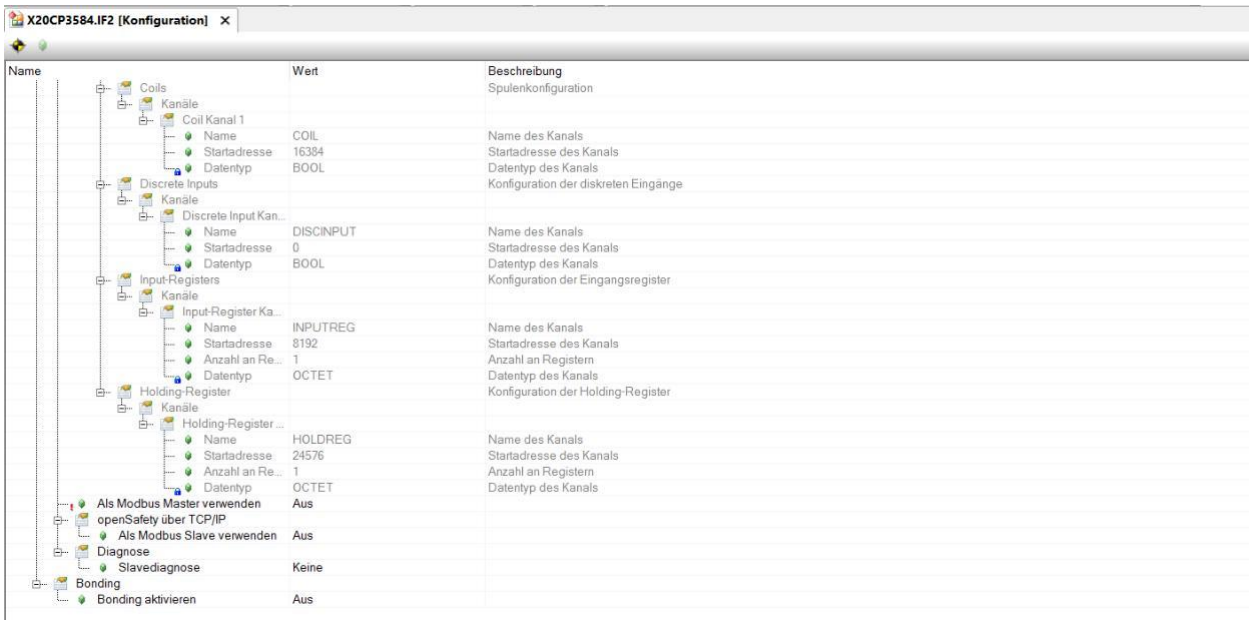


Abbildung 20 SPS Modus Einstellung

SPS Programmierung

In Case 0, the PLC program (see 1 PLC warehouse program code) has the function of waiting for the user to enter values at the machine and of outputting corresponding values at the display elements. Cases 51 to 54 are used to execute the reference run and to wait after completion until the command to change to Modbus mode is issued. The PLC program in Case 55 runs as a continuous loop and has only one query to interrupt it. The Case 55 fulfils the task to link the variables of the inputs and outputs of the PLC with the variables of the Modbus inputs and outputs and thus to forward the information.

```
1.  PROGRAM _CYCLIC
2.
3.      CASE Zustand OF                                  //Case defined as a state
4.          0:                                           //Case Condition 0
5.              lampeQ1:=TRUE;                           //Lamp Q1 turn on
6.              lampeQ2:=TRUE;                           //Lamp Q2 turn on
7.              Start_Lin := FALSE;                      //Linear motor stop
8.              Start_Rot := FALSE;                      //Rotation Motor stop
9.              lampereset:=TRUE;                        //Lamp reset turn on
10. IF Schluessel = FALSE AND T_Reset = TRUE THEN//IF Schluessel is FALSE and
    reset is pushed Then
11.                 Zustand:= 51;                        //jump in the Zustand 51
12.             END_IF                                   //IF close
13.             IF Schluessel = TRUE AND  T_Reset = TRUE THEN//IF Schluessel is
    TRUE Then
14.                 Zustand:= 61;                   //jump in case Zustand 61
15.             END_IF                                   //IF close
```

```
16.
17.         51:                                         //Case Zustand  51
18.             lampereset:=FALSE;                      //Lamp reset turn off
19.             lampeQ1:=FALSE;                         //Lamp Q1 turn off
20.             lampeQ2:=FALSE;                         //Lamp Q2 turn off
21.             Ausfahren:=FALSE;                       //retract arm
22.             Greifer_Zu:=FALSE;                      //closing gripper
23.             Start_Lin := TRUE;                      //motor linear start
24.             Bit0:=FALSE;     //Coordinate Bit 0 = 0 reference movementLinear
25.             Bit1:=FALSE;     //Coordinate Bit 1 = 0 reference movementLinear
26.             Bit2:=FALSE;     //Coordinate Bit 2 = 0 reference movementLinear
27.             Zustand:= 52;                          //jump in case Zustand 52
28.         52:                                         //Case Zustand  52
29.             Start_Lin := FALSE;//prevent the repeated linear movement
30.             Bit0:=TRUE;     //Coordinate Bit0 = 1 reference movement rotation
31.             Bit1:=TRUE;    //Coordinate Bit1 = 1  reference movement rotation
32.             Bit2:=TRUE;    //Coordinate Bit2 = 1  reference movement rotation
33.             Start_Rot := TRUE;                      //motor linear start
34.             Zustand:= 53;                         //jump in case Zustand 53
35.         53:                                         //Case Zustand 53
36.             Start_Rot := FALSE;  //prevent the repeated rotation movement
37.             Bit0:=FALSE;                            //Coordinate Bit0 = 0
38.             Bit1:=FALSE;                            //Coordinate Bit1 = 0
39.             Bit2:=FALSE;                            //Coordinate Bit2 = 0
40.             Zustand:=54;                       //jump in the Zustand 54
41.         54:                                         //Case Zustand 54
42.             lampeQ2:=TRUE;                          //Lamp Q2 turn on
43.             lampereset:=TRUE;                       //Lamp reset turn on
44.             lampestart:=TRUE;                       //Lamp start turn on
45.             IF T_Reset = TRUE THEN              //If Reset butten is pushed
46.                 lampereset:=FALSE;                  //Lamp reset turn off
47.                 lampestart:=FALSE;                  //Lamp start turn off
48.                 Zustand:= 51;                     //jump in case Zustand 51
49.             END_IF                                  //IF close
50.             IF T_Start = TRUE THEN              //If Reset butten is pushed
51.                 lampereset:=FALSE;                  //Lamp reset turn off
52.                 lampestart:=FALSE;                  //Lamp start
53.                 Zustand:= 55;                     //jump in case Zustand 55
54.             END_IF                                  //IF close
55.             IF Schluessel = TRUE THEN           //IF Schluessel TRUE Then
56.                 Zustand:= 0;                        //jump in the Zustand 0
57.                 lampereset:=FALSE;                  //Lamp reset turn off
58.                 lampestart:=FALSE;                  //Lamp start turn off
59.             END_IF                                  //IF close
60.         55:                   //Case Zustand 55 MODBUS variables handing over
61.             Bit0:=MB_Bit0;              //MODBUS Coordinate Bit0 in PLC Bit0
62.             Bit1:=MB_Bit1;              //MODBUS Coordinate Bit1 in PLC Bit1
63.             Bit2:=MB_Bit2;              //MODBUS Coordinate Bit2 in PLC Bit2
64.             Start_Lin:=MB_Start_Lin;     //MODBUS Start_Lin in PLC Start_Lin
65.             Start_Rot:=MB_Start_Rot;     //MODBUS Start_Rot in PLC Start_Rot
66.             Ausfahren:=MB_Ausfahren;     //MODBUS Ausfahren in PLC Ausfahren
67.             Greifer_Zu:=MB_Greifer_Zu;//MODBUS Greifer_Zu in PLC Greifer_Zu
68.             MB_MC_Lin_int:=MC_Lin;     //PLC MC_Lin in MODBUS MB_MC_Lin_int
69.             MB_MC_Rot_int:=MC_Rot;    //PLC MC_Rot in MODBUS MB_MC_Rot_int
70.             MB_MC_Arm_int:=MC_Arm;     //PLC MC_Arm in MODBUS MB_MC_Arm_int
71. MB_MC_Greifer_Zu_int:=MC_Greifer; //PLC MC_Greifer in MODBUS MB_MC
    _Greifer_Zu_int
72.             MB_cyclus:=cyclus;  //PLC cyclus in MODBUS MB_cyclus
73.             MB_Farbe1:=Farbe1;  //PLC Farbe1 in MODBUS MB_Farbe1
```

```
            MB_Farbe2:=Farbe2;  //PLC Farbe2 in MODBUS MB_Farbe2
            MB_Farbe3:=Farbe3;              //PLC Farbe3 in MODBUS MB_Farbe3

            MB_SyncBit1:=Bit0;       //PLC Bit0 in MODBUS MB_SyncBit1
            MB_SyncBit2:=Bit1;       //PLC Bit1 in MODBUS MB_SyncBit2
            MB_SyncBit3:=Bit2;       //PLC Bit2 in MODBUS MB_SyncBit3

            MB_Sync_Lin:= Start_Lin;  //PLC Start_Lin in MODBUS MB_Sync_Lin
            MB_Sync_Rot:= Start_Rot;    //PLC Start_Rot in MODBUS MB_Sync_Rot

            MB_MC_Arm:=MC_Arm;             //PLC MC_ARM in MODBUS MB_MC_Arm
            MB_MC_Greifer:=MC_Greifer;//PLC MC_Greifer in MODBUS MB_MC_Greifer

            MB_MC_Lin:=MC_Lin;            //PLC MC_Lin in MODBUS MB_MC_Lin
            MB_MC_Rot:=MC_Rot;            //PLC MC_Rot in MODBUS MB_MC_Rot

            IF T_Stop = FALSE THEN         //IF Reset button is pushed
                Zustand:= 54;                       //jump in the Zustand 54
            END_IF                                  //IF close

    END_CASE;


END_PROGRAM
```

1 Programmcode SPS Lager

SPS 2 Bearbeitungs Station

SPS I/O Zuordnung

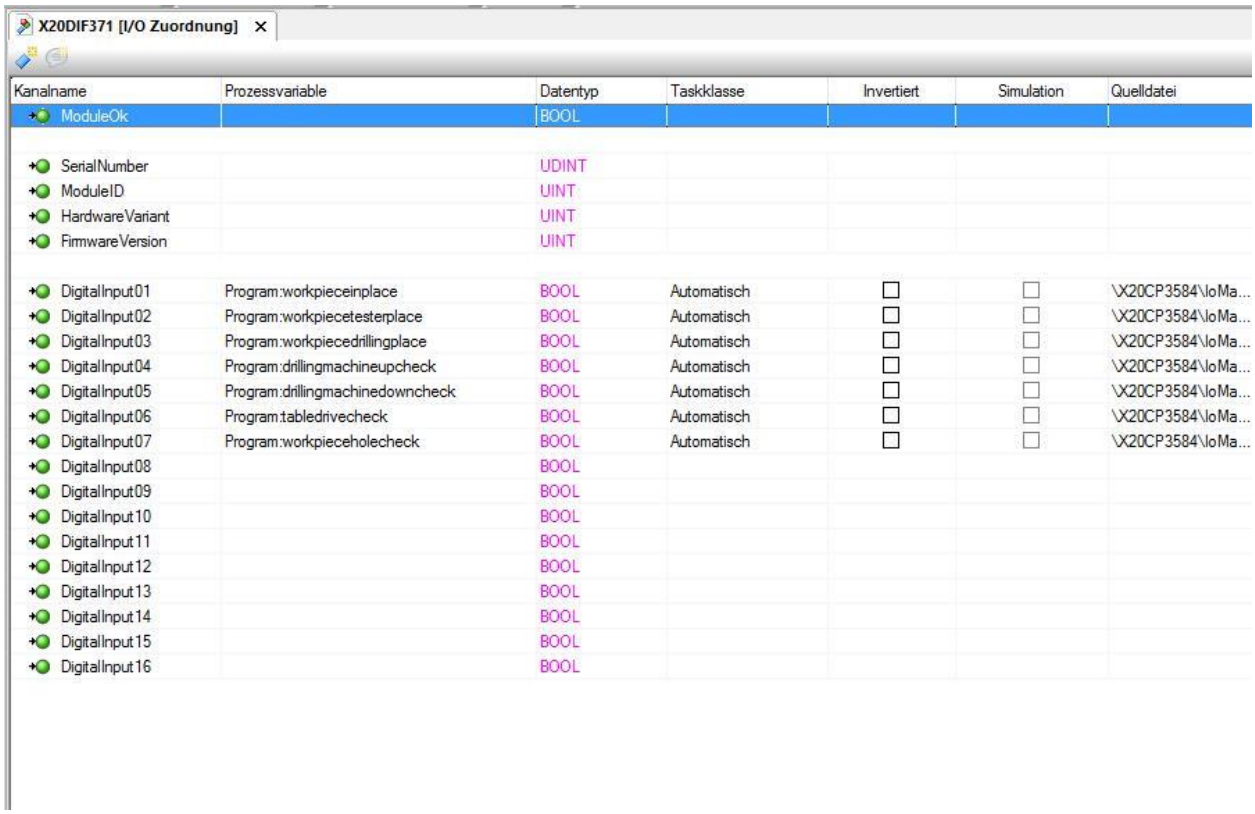
I/O Hardwareeingang

In the input assignment of the PLC (see Figure 21 PLC Input Hardware)

X20DIF371 [I/O Zuordnung]

Kanalname	Prozessvariable	Datentyp	Taskklasse	Invertiert	Simulation	Quelldatei
ModuleOk		BOOL				
SerialNumber		UDINT				
ModuleID		UINT				
HardwareVariant		UINT				
FirmwareVersion		UINT				
DigitalInput01	Program:workpieceinplace	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput02	Program:workpiecetesterplace	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput03	Program:workpiecedrillingplace	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput04	Program:drillingmachineupcheck	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput05	Program:drillingmachinedowncheck	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput06	Program:tabledrivecheck	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput07	Program:workpieceholecheck	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput08		BOOL				
DigitalInput09		BOOL				
DigitalInput10		BOOL				
DigitalInput11		BOOL				
DigitalInput12		BOOL				
DigitalInput13		BOOL				
DigitalInput14		BOOL				
DigitalInput15		BOOL				
DigitalInput16		BOOL				

the interfaces DigitalInput01 to 16, the module X20DIF371, are assigned to the corresponding variables for the program code.



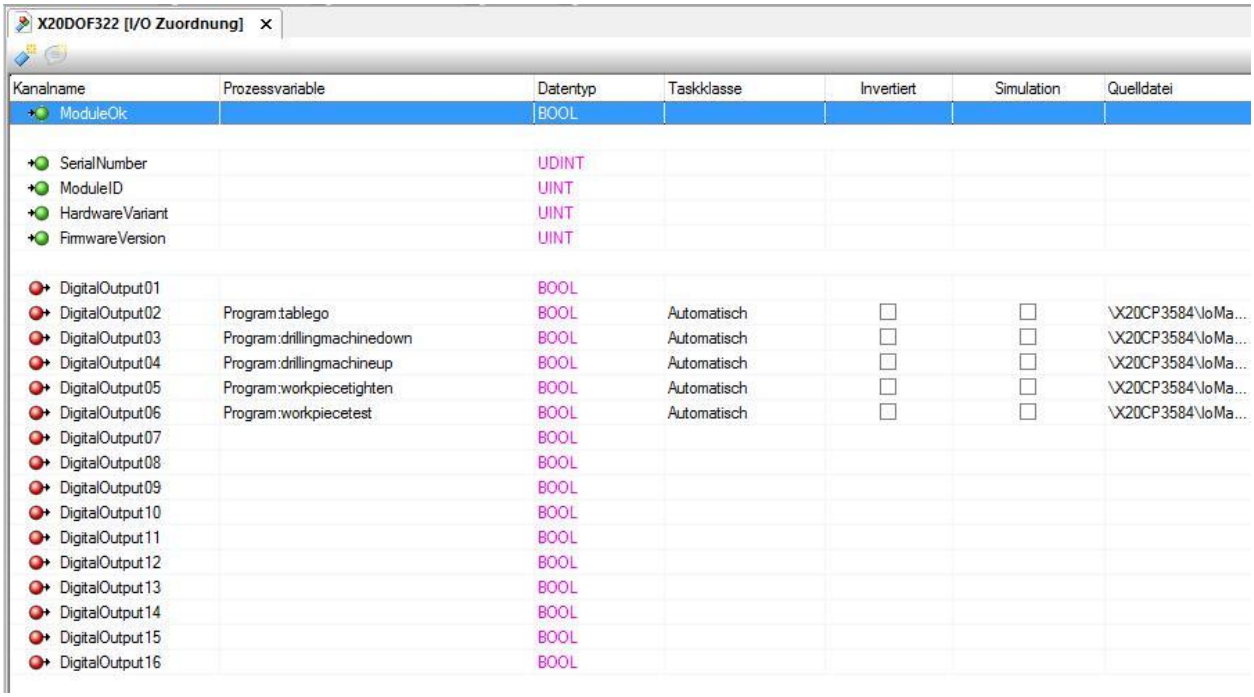
X20DIF371 [I/O Zuordnung]

Kanalname	Prozessvariable	Datentyp	Taskklasse	Invertiert	Simulation	Quelldatei
ModuleOk		BOOL				
SerialNumber		UDINT				
ModuleID		UINT				
HardwareVariant		UINT				
FirmwareVersion		UINT				
DigitalInput01	Program:workpieceinplace	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput02	Program:workpiecetesterplace	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput03	Program:workpiecedrillingplace	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput04	Program:drillingmachineupcheck	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput05	Program:drillingmachinedowncheck	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput06	Program:tabledrivecheck	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput07	Program:workpieceholecheck	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalInput08		BOOL				
DigitalInput09		BOOL				
DigitalInput10		BOOL				
DigitalInput11		BOOL				
DigitalInput12		BOOL				
DigitalInput13		BOOL				
DigitalInput14		BOOL				
DigitalInput15		BOOL				
DigitalInput16		BOOL				

Abbildung 21 SPS Input Hardware

I/O Hardwareausgang

In the output assignment of the PLC (see Figure 22 PLC Output Hardware), the interfaces DigitalOutput01 to 16, the module X20DOF322, are assigned to the associated variables for the program code.

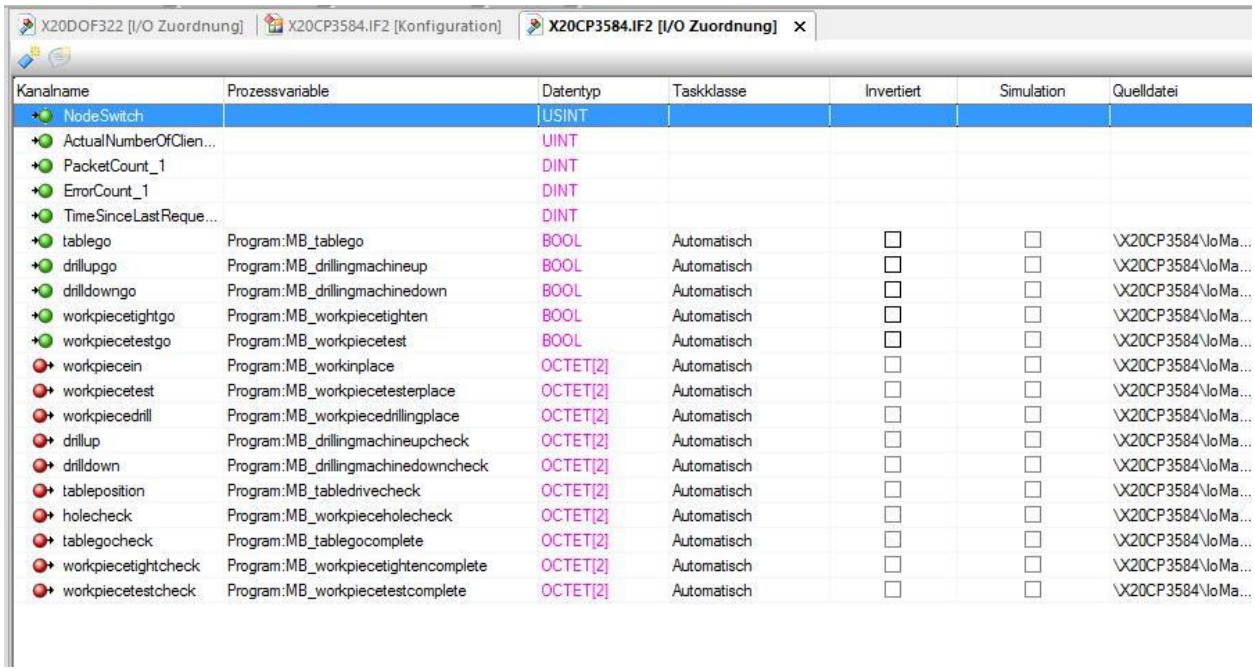


Kanalname	Prozessvariable	Datentyp	Taskklasse	Invertiert	Simulation	Quelldatei
ModuleOk		BOOL				
SerialNumber		UDINT				
ModuleID		UINT				
HardwareVariant		UINT				
FirmwareVersion		UINT				
DigitalOutput01		BOOL				
DigitalOutput02	Program:tablego	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalOutput03	Program:drillingmachinedown	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalOutput04	Program:drillingmachineup	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalOutput05	Program:workpiecetighten	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalOutput06	Program:workpiecetest	BOOL	Automatisch	☐	☐	\X20CP3584\IoMa...
DigitalOutput07		BOOL				
DigitalOutput08		BOOL				
DigitalOutput09		BOOL				
DigitalOutput10		BOOL				
DigitalOutput11		BOOL				
DigitalOutput12		BOOL				
DigitalOutput13		BOOL				
DigitalOutput14		BOOL				
DigitalOutput15		BOOL				
DigitalOutput16		BOOL				

Abbildung 22 SPS Output Hardware

I/O Modbus Eingänge und Ausgänge

In the Modbus register assignment of the PLC (see Figure 23 PLC Modbus I/O), the Modbus registers of the PLC are assigned to the associated variables for the program code.



Kanalname	Prozessvariable	Datentyp	Taskklasse	Invertiert	Simulation	Quelldatei
NodeSwitch		USINT				
ActualNumberOfClien...		UINT				
PacketCount_1		DINT				
ErrorCount_1		DINT				
TimeSinceLastReque...		DINT				
tablego	Program:MB_tablego	BOOL	Automatisch	☐	☐	\X20CP3584\loMa...
drillupgo	Program:MB_drillingmachineup	BOOL	Automatisch	☐	☐	\X20CP3584\loMa...
drilldowngo	Program:MB_drillingmachinedown	BOOL	Automatisch	☐	☐	\X20CP3584\loMa...
workpiecetightgo	Program:MB_workpiecetighten	BOOL	Automatisch	☐	☐	\X20CP3584\loMa...
workpiecetestgo	Program:MB_workpiecetest	BOOL	Automatisch	☐	☐	\X20CP3584\loMa...
workpiecein	Program:MB_workinplace	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
workpiecetest	Program:MB_workpiecetesterplace	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
workpiecedrill	Program:MB_workpiecedrillingplace	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
drillup	Program:MB_drillingmachineupcheck	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
drilldown	Program:MB_drillingmachinedowncheck	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
tableposition	Program:MB_tabledrivecheck	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
holecheck	Program:MB_workpieceholecheck	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
tablegocheck	Program:MB_tablegocomplete	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
workpiecetightcheck	Program:MB_workpiecetightencomplete	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...
workpiecetestcheck	Program:MB_workpiecetestcomplete	OCTET[2]	Automatisch	☐	☐	\X20CP3584\loMa...

Abbildung 23 SPS Modbus I/O

SPS Modbus Modus, Register und Netzwerk Optionen

The relevant network settings, Modbus register management (see Figure 25 PLC Mode Register Setting 1, Figure 14 PLC Mode Register Setting 2, Figure 27 PLC Mode Register Setting 3) and the general settings of Modbus (see Figure 28 PLC Mode Setting) are defined in the configuration of the PLC network parameters (see Figure 24 PLC Network Setting).

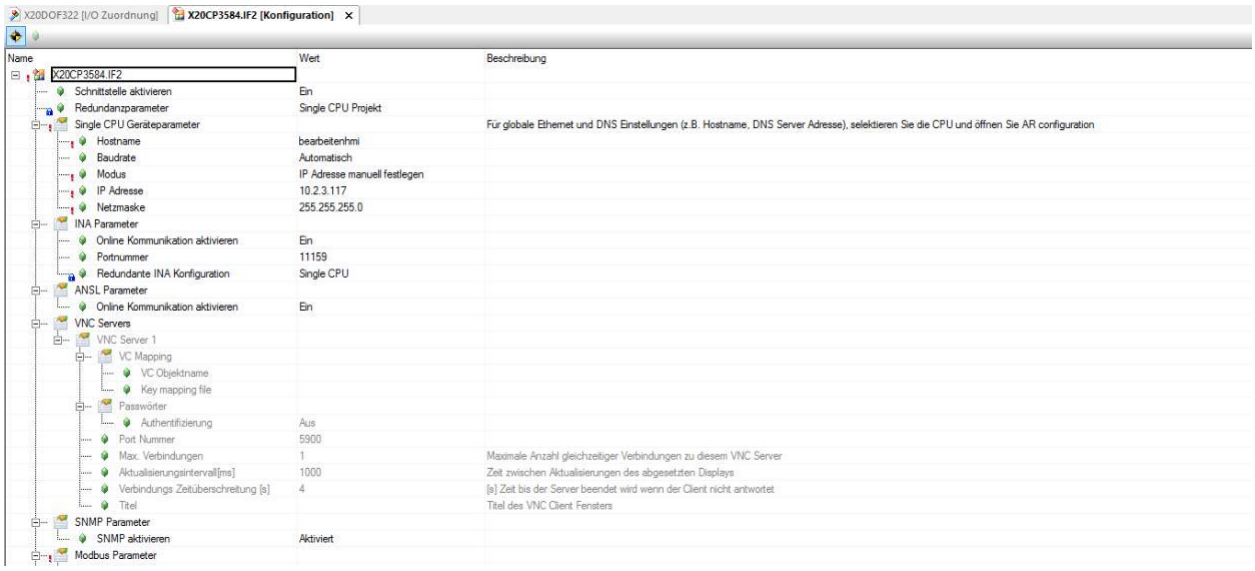


Abbildung 24 SPS Netzwerk Einstellung

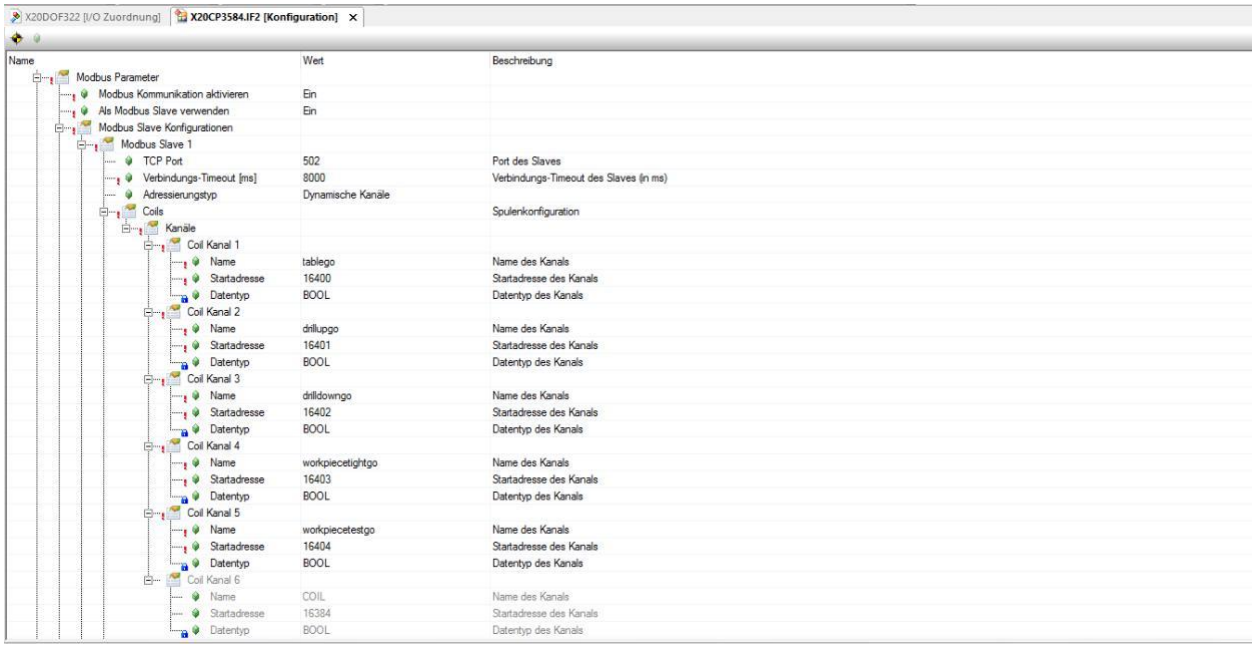


Abbildung 25 SPS Modus Register Einstellung 1

Abbildung 26 SPS Modus Register Einstellung 2

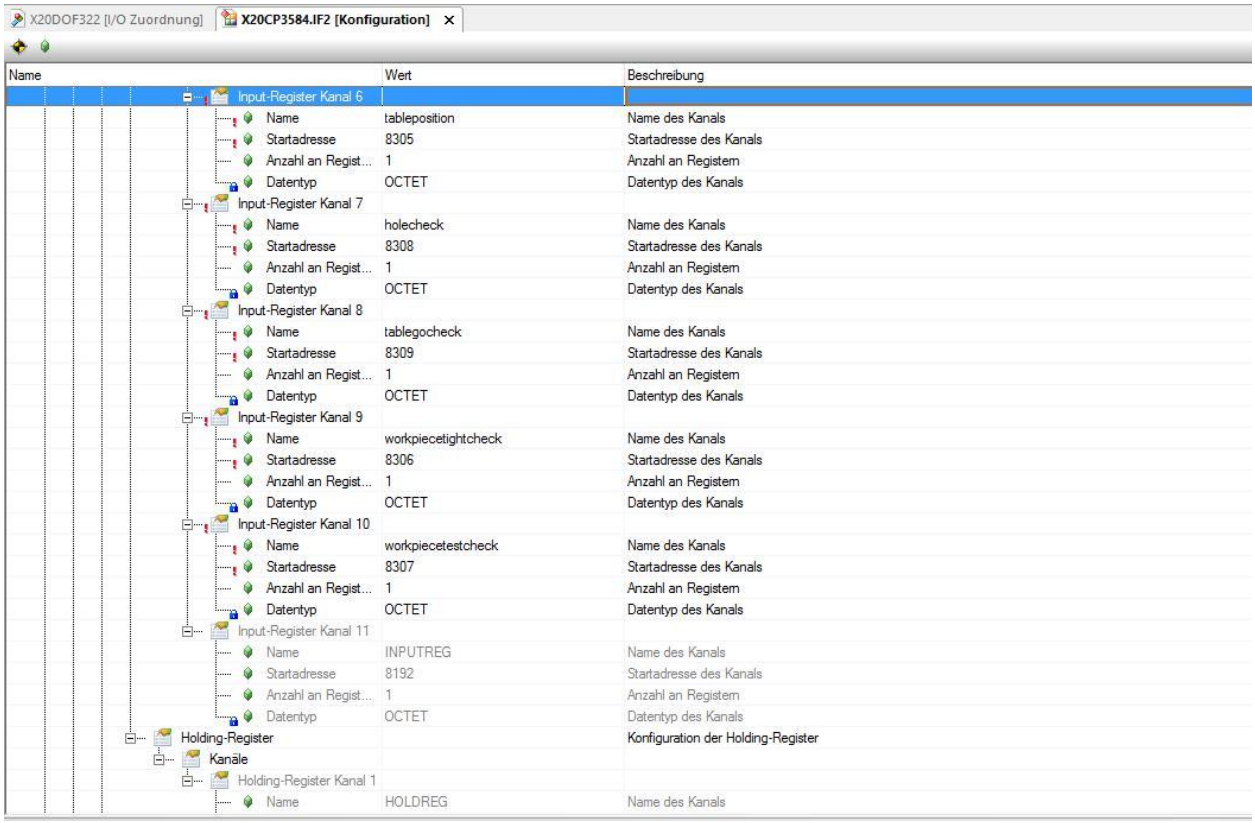


Abbildung 27 SPS Modus Register Einstellung 3

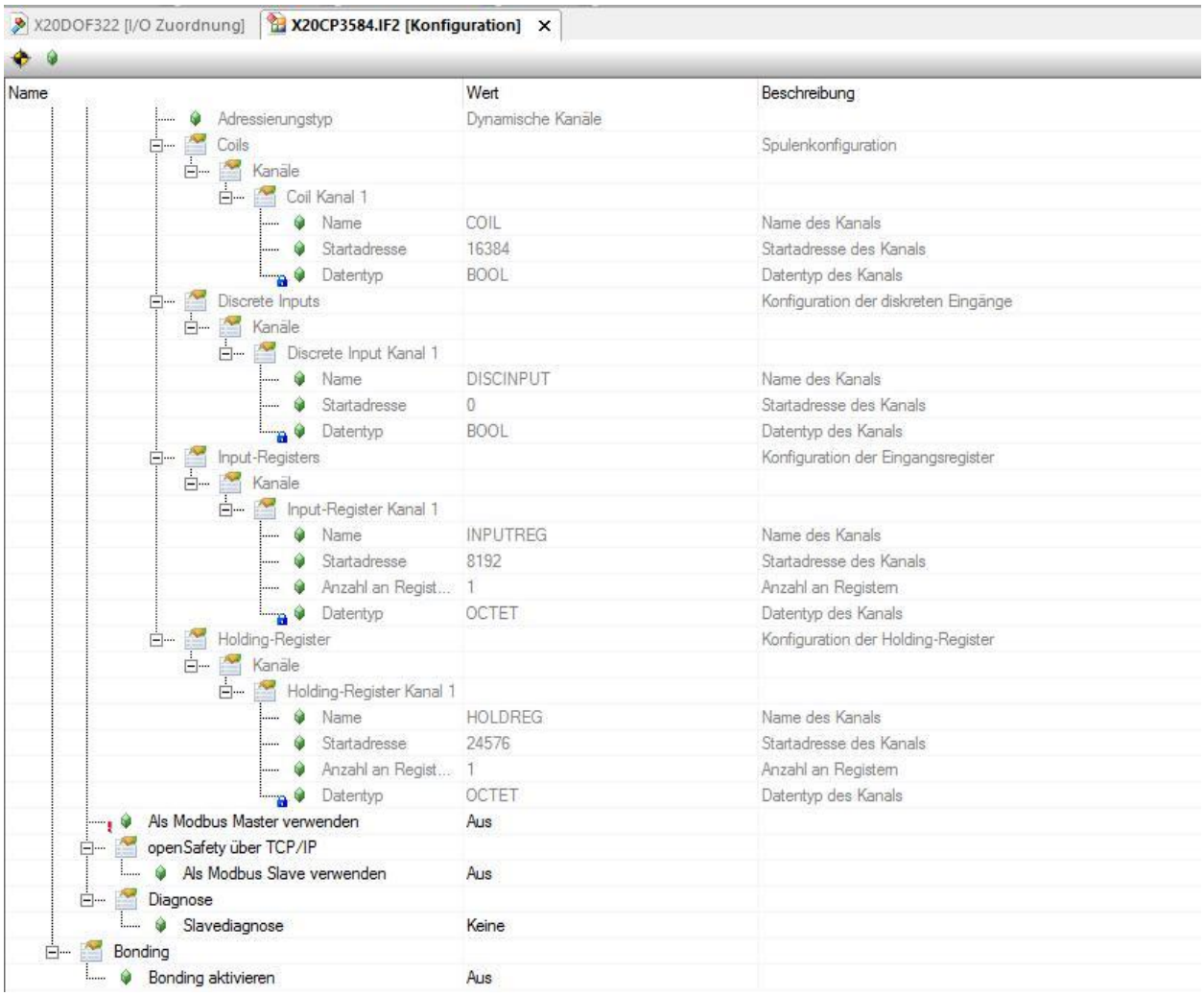


Abbildung 28 SPS Modus Einstellung

SPS Programmierung

The PLC program (see 2 PLC Edit program code) runs as a continuous loop and has no special requests. It performs the task of linking the variables of the inputs and outputs of the PLC with the variables of the Modbus inputs and outputs and thus forwarding the information.

```
1.  PROGRAM _CYCLIC
2.      tablego:=MB_tablego;
3.      drillingmachineup:=MB_drillingmachineup;
4.      drillingmachinedown:=MB_drillingmachinedown;
5.      workpiecetighten:=MB_workpiecetighten;
6.      workpiecetest:=MB_workpiecetest;
7.
8.
9.      MB_tabledrivecheck:=tabledrivecheck;
10.     drillingmachinedowncheck:=drillingmachinedown;
11.     MB_drillingmachinedowncheck:=drillingmachinedowncheck;
12.     drillingmachineupcheck:=drillingmachineup;
13.     MB_drillingmachineupcheck:=drillingmachineupcheck;
```

```
    MB_workpiecedrillingplace:=workpiecedrillingplace;
    MB_workpiecetesterplace:=workpiecetesterplace;
    MB_workinplace:=workpieceinplace;
    workpiecetightencomplete:=workpiecetighten;
    MB_workpiecetightencomplete:=workpiecetightencomplete;
    workpiecetestcomplete:=workpiecetest;
    MB_workpiecetestcomplete:=workpiecetestcomplete;
    MB_workpieceholecheck:=workpieceholecheck;
    tablegocomplete:=tablego;
    MB_tablegocomplete:=tablegocomplete;
END_PROGRAM
```

2 Programmcode SPS Bearbeiten

Gateway Programmierung

Bibliotheken und Variablen

In this program section (see 3 Libraries and Objects) the used libraries, variables and objects are stored.

```
using Daenet.Iot;
using Daenet.Iot.Entities;
using Daenet.Iot.FestoLagerSystem;
using Daenet.Iot.Interpreters;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Dynamic;
using System.IO;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices.WindowsRuntime;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?
   LinkId=402352&clcid=0x409

namespace Daenet.Iot
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.

    /// </summary>
```

```
34.     public sealed partial class MainPage : Page
35.     {
36.         internal static Settings Settings;
37.
38.         private CancellationTokenSource m_TokenSource;
39.
40.         private SmartAutomationGateway m_Gtw;
41.
42.         private List<ICommandInterpreter> m_CmdInterpreters = new List<ICommandIn
      terpreter>();
43.
44.         private bool m_IsProgramRunning = false;
45.
46.         public StoreStateModel SlotModel
47.         {
48.             get
49.             {
50.                 return this.m_Grid.DataContext as StoreStateModel;
51.             }
52.         }
53.
```

3 Bibliotheken und Objekte

Cloud Anbindung

With the MainPage method (see 4 Program Code Mainpage), the relevant cloud data is initialized and the runGateway method for connecting the cloud is started and the Main Page is initialized.

```
54.         public MainPage()
55.         {
56.             CallSite a;
57.             var bla = new { SlotId = 0, IsOnTop = false, IsOnPosition = true, IsE
      mpty = false };
58.             dynamic dyn = (dynamic)bla;
59.             dynamic obj = new ExpandoObject();
60.             obj.SlotId = 0;
61.
62.             //// Blackforest DEVGER - CENTRAL (UNI)
63.             Settings = new Settings()
64.             {
65.                 ConnStr = "HostName=IoTApiHub.azure-
      devices.de;SharedAccessKey=xjyDGIAUy22a56slWC/Imh85BNVKkfl/Jenk8mS0bbs=",
66.                 DeviceId = "FESTOLAGER01",
67.                 ServiceConnStr = ""
68.             };
69.
70.
71.
72.
73.             //Settings = new Settings()
74.             //{
75.             //    ConnStr = "HostName=SmartAutomationHub.azure-
      devices.de;SharedAccessKeyName=iothubowner;SharedAccessKey= =",
```

```
76.             //    //ConnStr = "HostName=SmartAutomationHub.azure-
      devices.de;SharedAccessKey= =",
77.             //    DeviceId = /*"FESTOBEARBEITEN01",//*/"FESTOLAGER01",
78.             //    ServiceConnStr = ""
79.             //};
80.
81.             ////
82.             // Settings = new Settings()
83.             // {
84.             //     ConnStr = "HostName=azureontour.azure-
      devices.net;SharedAccessKey= =",
85.             //     DeviceId = "FESTOLAGER01",
86.             //     ServiceConnStr = ""
87.             // };
88.
89.             this.InitializeComponent();
90.
91.             runGateway();
92.
93.             this.Loaded += MainPage_Loaded;
94.         }
95.
```

4 Programmcode Mainpage

Benutzeroberfläche laden

The MainPage_Loaded and initModel methods (see 5 Program Code MainPage Loaded) initialize the user interface of the program with the slots for the compartments.

```
96.         private void MainPage_Loaded(object sender, RoutedEventArgs e)
97.         {
98.             intiModel();
99.         }
100.
101.
102.           private void intiModel()
103.           {
104.               StoreStateModel model = new StoreStateModel();
105.   model.Slots = new System.Collections.ObjectModel.ObservableCollect
      ion<SlotState>();
106.
107.            model.Slots.Add(new SlotState("Start") { SlotId = 0 });
                //name start with SlotId 0
108.
109.              for (int i = 1; i < 19; i++)
110.              {
111.            model.Slots.Add(new SlotState(i.ToString()) { SlotId = i });
                //give slots unique SlotIds
112.              }
113.
114.              m_Grid.DataContext = this.DataContext = model;
115.          }
116.
```

5 Programmcode MainPage Loaded

Gateway

The method runGateway (see 6 Program code runGateway) is used to select the interpreter and start the gateway for the machine.

```
117.          private async Task runGateway()
118.          {
119.              m_CmdInterpreters = new List<ICommandInterpreter>();
120.              m_CmdInterpreters.Add(new FestoLagerInterpreter());
121.              //m_CmdInterpreters.Add(new StoreSimultorInterpreter());
122.
123.
124.              try
125.              {
126.  trace("Gateway started"); //system status

127.
128.                  m_TokenSource = new CancellationTokenSource();
129.
130.                  m_Gtw = new SmartAutomationGateway(Settings);
131.
132.                  await m_Gtw.Run(onNewCommand, onEventCallback, m_TokenSource);

133.
134.                  m_IsCommandReceiverActive = true;
135.              }
136.              catch (Exception ex)
137.              {
138.                  traceError(ex, "Gateway error");
139.              }
140.          }
141.
142.          private bool m_IsCommandReceiverActive = true;
143.
```

6 Programmcode runGateway

Kommando Verwaltung

The function of the onNewCommand method (see 7 Program code onNewCommand) is to interpret and execute a new command.

```
144.          private bool onNewCommand(DeviceCommandMessage deviceCommand)
145.          {
 146.             if (m_IsProgramRunning)
 147.             {
 148.                 trace("New commands cannot be executed as long the current pro
      gram is running.");
 149.                 return true;
 150.             }
 151.             if (m_IsCommandReceiverActive == false)
 152.             {
 153.                 return true;
 154.             }
 155.
```

```
156.                trace("New command received. {0}", deviceCommand.CommandType);
157.
158.                if (deviceCommand != null)
159.                {
160.                    bool isInterpreted = false;
161.
162.                    List<Task> interprTasks = new List<Task>();
163.
164.                    foreach (var interpreter in m_CmdInterpreters)
165.                    {
166.                        if (interpreter.CanInterprete(deviceCommand))
167.                        {
168.                            int cnt = interpreter.GetNumOfSequences(deviceCommand)
    ;
169.
170.                            // -1 means program start.
171.                            onEventCallback(interpreter, -
    1, DateTime.Now, cnt, null).Wait();
172.
173.                            isInterpreted = true;
174.
175.                            var initState = interpreter.GetInitialState(deviceComm
    and);
176.                            if (initState != null)
177.                                setInitialState(initState);
178.
179.                            interprTasks.Add(interpreter.Run(deviceCommand, traceC
    allback, onEventCallback, m_TokenSource).ContinueWith((tt) =>
180.                            {
181.                                Exception err = tt.Exception;
182.                                if (tt.Exception is AggregateException)
183.                                {
184.                                    err = ((AggregateException)tt.Exception).Inner
    Exceptions[0];
185.                                    trace(err.Message);
186.                                }
187.
188.                                // -2 means program end.
189.                                onEventCallback(interpreter, -
    2, DateTime.Now, cnt, err).Wait();
190.                                trace("--------- End of program -------");
191.                            }));
192.                        }
193.                    }
194.
195.                    Task.WaitAll(interprTasks.ToArray());
196.
197.                    if (isInterpreted == false)
198.                        trace("Unknown command received");
199.
200.                    string msgId = deviceCommand.MessageId;
201.
202.                    Debug.WriteLine(msgId);
203.                }
204.                else
205.                    Debug.WriteLine("No messages in command queue.");
206.
207.                return true;
208.            }
```

```
209.
```

7 Programmcode onNewCommand

Benutzeroberflächeninitiierung bei neuem Kommando

With the method setInitialState (see 8 Program code SetInitialState) the user interface is updated with a new command.

```
210.            private void setInitialState(object initState)
211.            {
212.                dynamic stateList = initState as dynamic;
213.
214.                foreach (var item in stateList.ToArray())
215.                {
216.                    dynamic obj = item as dynamic;
217.                    int slotNum = obj.SlotNumber;
218.                    bool isOccupied = obj.IsOccupied;
219.                    Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Nor
      mal, () =>
220.                    {
221.                        SlotState oldwriteslots = this.SlotModel.Slots[slotNum];
222.
223.                    var newwriteslots = new SlotState()
224.                    {
225.                        IsArmOut = oldwriteslots.IsArmOut,
226.                        IsClawClosed = oldwriteslots.IsClawClosed,
227.                        IsEmpty = isOccupied,
228.                        IsClawEmpty = oldwriteslots.IsClawEmpty,
229.                        IsOnPosition = oldwriteslots.IsOnPosition,
230.                        IsOnTopPosition = oldwriteslots.IsOnTopPosition,
231.                        SlotId = oldwriteslots.SlotId,
232.                        Name = oldwriteslots.Name,
233.                    };
234.
235.                this.SlotModel.Slots[slotNum] = newwriteslots;
236.                    }).AsTask().ContinueWith((t) =>
237.                    {
238.
239.                    });
240.                    //TODO..
241.                }
242.            }
243.
244.            private int m_CurrentStep;
245.            private int m_NumOfSteps;
246.
```

8 Programmcode SetInitialState

Ereignis Steuerung

The method onEventCallback (see 9 Program code onEventCallback) is used to update the user interface according to the executed command.

```
247.   private async Task onEventCallback(object source, int eventId, DateTim e
       timestamp, object obj, Exception ex)
248.           {
 249.              try
 250.              {
 251.                  if (eventId == -1)
 252.                  {
 253.                      m_IsProgramRunning = true;
 254.                      m_CurrentStep = 0;
 255.                      m_NumOfSteps = (int)obj;
 256.                  }
 257.
 258.
 259.                  if (source.GetType() == typeof(FestoLagerSystem.FestoLagerInte
       rpreter) || source.GetType() == typeof(Daenet.Iot.Interpreters.StoreSimultorIn
       terpreter))
 260.                  {
 261.                      await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPr
       iority.Normal, () =>
 262.                      {
 263.                          try
 264.                          {
 265.                              var slot = this.SlotModel.Slots.FirstOrDefault(s =
       > s.SlotId == this.SlotModel.CurrentPosition);
 266.                              if (slot != null)
 267.                              {
 268.                                 // throw new Exception("ERROR (:");
 269.                                  if (this.SlotModel.CurrentPosition == slot.Slo
       tId)
 270.                                  {
 271.                                      SlotState newPosState = this.SlotModel.Slo
       ts[this.SlotModel.CurrentPosition];
 272.                                      int oldStatePos = -1;
 273.
 274.                                      switch (eventId)
 275.                                      {
 276.                                          // Arm
 277.                                          case 1:
 278.
 279.                                              bool isArmOut = (bool)obj;
 280.                                              newPosState.IsArmOut = isArmOut;
 281.                                              break;
 282.
 283.                                          // Claw
 284.                                          case 2:
 285.                                              bool isClawClosed = (bool)obj;
 286.                                              newPosState.IsClawClosed = isClawC
       losed;
 287.                                              if (newPosState.IsEmpty == false &
       & newPosState.IsOnPosition == true && newPosState.IsClawClosed == true && newP
       osState.IsClawEmpty == true && newPosState.IsArmOut == true)
 288.                                              {
```

```
289.                                                    newPosState.IsClawEmpty = fals
      e;
290.                                                newPosState.IsEmpty = true;
291.                                             }
292.                                             if (newPosState.IsEmpty == true &&
       newPosState.IsOnPosition == true && newPosState.IsClawClosed == false && newP
      osState.IsClawEmpty == false && newPosState.IsArmOut == true)
293.                                             {
294.                                                 newPosState.IsClawEmpty = true
      ;
295.                                                newPosState.IsEmpty = false;
296.                                             }
297.                                             if (newPosState.IsEmpty == false &
      & newPosState.IsOnPosition == true && newPosState.IsClawClosed == false && new
      PosState.IsClawEmpty == false && newPosState.IsArmOut == true)
298.                                            {
299.                                                newPosState.IsClawEmpty = true
      ;
300.                                                newPosState.IsEmpty = true;
301.                                            }
302.                                            break;
303.
304.                                        // Move to Overposition
305.                                        case 3:
306.
307.                                            Int32 onOverPos = (Int16)obj;//fes
      to
308.                                            newPosState = this.SlotModel.Slots
      [onOverPos];
309.                                            oldStatePos = this.SlotModel.Curre
      ntPosition;
310.
311.                                            var oldstate = this.SlotModel.Slot
      s[oldStatePos];
312.                                            oldstate.IsOnPosition = false;
313.                                            oldstate.IsOnTopPosition = false;

314.
315.                                            var writeoldState = new SlotState(
      )
316.                                            {
317.                                                IsArmOut = oldstate.IsArmOut,

318.                                                IsClawClosed = oldstate.IsClaw
      Closed,
319.                                                IsEmpty = oldstate.IsEmpty,
320.                                                IsOnPosition = oldstate.IsOnPo
      sition,
321.                                                IsOnTopPosition = oldstate.IsO
      nTopPosition,
322.                                                SlotId = oldstate.SlotId,
323.                                                Name = oldstate.Name,
324.                                            };
325.
326.                                            this.SlotModel.Slots[oldStatePos]
      = writeoldState;
327.
328.                                            newPosState.IsClawClosed = oldstat
      e.IsClawClosed;
```

```
329.                                                newPosState.IsArmOut = oldstate.Is
      ArmOut;
330.                                                newPosState.IsClawEmpty = oldstate
      .IsClawEmpty;
331.                                                newPosState.IsOnTopPosition = true
      ;
332.                                                newPosState.IsOnPosition = false;

333.                                                this.SlotModel.CurrentPosition = o
      nOverPos;
334.                                                newPosState.SlotId = onOverPos;
335.
336.                                                break;
337.
338.                                            // Move to position
339.                                            case 4:
340.
341.                                                Int32 onPos = (Int16)obj;//festo
342.                                                if (onPos != 19)
343.                                                {
344.                                                    newPosState = this.SlotModel.S
      lots[onPos];
345.                                                    newPosState.IsOnPosition = tru
      e;
346.                                                    newPosState.IsOnTopPosition =
      false;
347.
348.                                                    this.SlotModel.CurrentPosition
       = onPos;
349.                                                    newPosState.SlotId = onPos;
350.                                                }
351.                                                if (onPos == 0)
352.                                                {
353.                                                    newPosState.IsEmpty = false;
354.                                                }
355.                                                if (onPos == 0 && newPosState.IsCl
      awEmpty == true && newPosState.IsClawClosed == true)
356.                                                {
357.                                                    newPosState.IsEmpty = true;
358.                                                }
359.                                                break;
360.                                        }
361.
362.                                        var writenewState = new SlotState()
363.                                        {
364.                                            IsArmOut = newPosState.IsArmOut,
365.                                            IsClawClosed = newPosState.IsClawClose
      d,
366.                                            IsEmpty = newPosState.IsEmpty,
367.                                            IsClawEmpty = newPosState.IsClawEmpty,

368.                                            IsOnPosition = newPosState.IsOnPositio
      n,
369.                                            IsOnTopPosition = newPosState.IsOnTopP
      osition,
370.                                            SlotId = newPosState.SlotId,
371.                                            Name = newPosState.Name,
372.                                        };
373.
```

```
374.                                         this.SlotModel.Slots[this.SlotModel.Curren
      tPosition] = writenewState;
375.
376.                                     }
377.                                 }
378.                             }
379.                             catch (Exception err)
380.                             {
381.                                 sendErrorToCloudService(err).Wait();
382.                                 return;
383.                             }
384.                         });
385.
386.                                     DeviceCommandMessage msg;
387.
388.                                     if (ex == null)
389.                                     {
390.                                         msg = new DeviceCommandMessage()
391.                                         {
392.                                             CommandType = DeviceCommandMessage.Cmd
      TelemetryEvent,
393.                                             DeviceId = Settings.DeviceId,
394.                                             MessageId = Guid.NewGuid().ToString(),

395.                                             Data = new { EventId = eventId, Curren
      tStep = m_CurrentStep++, NumOfSteps = m_NumOfSteps },
396.                                         };
397.
398.                             if (eventId == -2)
399.                                 Debug.WriteLine("Sending -2 *********");
400.
401.                             await m_Gtw.SendEventToCloud(msg);
402.                                     }
403.                                     else
404.                                     {
405.                             await sendErrorToCloudService(ex);
406.                                     }
407.
408.                         if (eventId == -2)
409.                             {
410.                             m_IsProgramRunning = false;
411.                             m_TokenSource.Cancel();
412.
413.                             Task.Delay(2000).Wait();
414.
415.                             runGateway();
416.                             }
417.                     }
418.
419.                 }
420.                 catch (Exception err)
421.                 {
422.                     traceError(err, "Error in processing of 'onEventCallback'");
423.
424.                     sendErrorToCloudService(ex);
425.                 }
426.                 }
427.
```

```
428.
```

9 Programmcode onEventCallback

Error Meldung an die Cloud

With the method sendErrorToCloudService (see 10 Program code sendError) an error message can be transmitted to the connected cloud.

```
429.          private async Task sendErrorToCloudService(Exception ex)
430.          {
 431.              var errMsg = new DeviceCommandMessage()
 432.              {
 433.                  CommandType = DeviceCommandMessage.CmdProgFailedEvent,
 434.                  DeviceId = Settings.DeviceId,
 435.                  MessageId = Guid.NewGuid().ToString(),
 436.                  Data = ex.Message,
 437.              };
 438.
 439.              await m_Gtw.SendEventToCloud(errMsg);
440.          }
441.
```

10 Programmcode sendError

Error Ausgabe

With the traceError method (see 11 Program code Error), events are output to the user in the form of errors.

```
442.          private void traceError(Exception ex, string message, params object[]
      args)
443.          {
 444.
 445.              string txt;
 446.
 447.              if (args != null)
 448.                  txt = String.Format(message, args);                    //
      txt <- status and task
 449.              else
 450.                  txt = message;                                         //
      txt <- status
 451.
 452.              StringBuilder sb = new StringBuilder(txt);                 //
      sb = build txt in string
 453.
 454.              sb.Append("\n Exception type: " + ex.GetType().Name + " "); //
      output Type of Exception
 455.              sb.Append(ex.Message);                                     //
      output describes Exception
 456.
 457.
 458.              Exception currEx = ex;
 459.
 460.              while (currEx.InnerException != null)
```

```
461.             {
462.                 currEx = currEx.InnerException;                          //
      currEx is the current Exception
463.                  sb.Append("\n Exception type: " + currEx.GetType().Name + " ")
      ;//output Type of Exception
464.                 sb.Append(currEx.Message);                               //
      output describes current Exception
465.             }
466.             traceCallback(sb.ToString(), args.Length == 0 ? null : args);
467.         }
```

11 Programmcode Error

Event Ausgabe

The trace method (see 12 Program Code Trace) outputs events to the user.

```
468.         private void trace(string message, params object[] args)
469.         {
470.             traceCallback(message, args);
                 //status and task
471.         }
472.
473.         int tracecount = 0;
474.
```

12 Programmcode Trace

Event Text ausgabe

The traceCallback method (see 13 Program code TraceCallback) is used to output events that are returned when the machine is running..

```
475.         private async void traceCallback(string message, object[]
                 args) //output in textbox
476.         {
477.             string txt;
478.
479.             if (args != null)
480.                 txt = String.Format(message, args);
                   //txt <- status and task
481.             else
482.                 txt = message;
                   //txt <- status
483.
484.             Debug.WriteLine(txt);
485.             await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.N
      ormal, () =>
486.             {
487.                 tracecount++;
488.                 if (tracecount == 20)
489.                 {
490.                     m_TxtTrace.Text = String.Empty;
491.                     tracecount = 0;
492.                 }
```

```
493.   m_TxtTrace.Text += DateTime.Now.ToString("h:mm:ss:fff") + " " + txt + "\n";
       //text output (date time txt)
494.               });
495.           }
496.
```

13 Programmcode TraceCallback

Gateway starten

The function of the Start_Click method (see 14 Program Code Start Button) is to start the gateway and communicate with the interface in the cloud in this way.

```
497.           private async void Start_Click(object sender, RoutedEventArgs e)
498.           {
499.               await runGateway();
500.           }
501.
```

14 Programmcode Start Button

Programm anhalten Button

The method Stop_Click (see 15 Program code Stop Button) interrupts the program for the machine when a button is pressed.

```
502.           private void Stop_Click(object sender, RoutedEventArgs e)
503.           {
504.               m_IsProgramRunning = false;
505.               m_TokenSource.Cancel();
506.           }
507.
508.           private bool m_StopProgramExecution;
509.
```

15 Programmcode Stop Button

Programm anhalten

The Stop_Program method (see 16 Stop Program program code) interrupts the operation of the machine.

```
510.           private void Stop_Program(object sender, RoutedEventArgs e)
511.           {
512.               m_IsProgramRunning = false;
513.               m_TokenSource.Cancel();
514.           }
515.
```

16 Programmcode Stop Programm

Befehle löschen

By calling the method Clear_Commands (see 17 Program code Clear Commands), the Json commands are deleted and reception is also switched off. A new call of the method activates the reception again.

```
516.           private void Clear_Commands(object sender, RoutedEventArgs e)
517.           {
518.               if (m_IsCommandReceiverActive == false)
519.               {
520.                   m_IsCommandReceiverActive = true;
521.   trace("Commands will be executed. Command Receiver is Activate d");

522.               }
523.               else
524.               {
525.                   m_IsCommandReceiverActive = false;
526.   trace("Commands will be cleared. Command Receiver is Deactivat ed");

527.               }
528.           }
529.
```

17 Programmcode Clear Commands

Nachrichten löschen

Durch den Aufruf der Methode Clear_Messages (siehe 18 Programmcode Clear Messages) werden die Event Messages im Log gelöscht.

```
530.           private void Clear_Messages(object sender, RoutedEventArgs e)
531.           {
532.               m_TxtTrace.Text = String.Empty;
533.           }
534.       }
535.   }
```

18 Programmcode Clear Messages

Interpreter Programmierung

Bibliotheken und Variablen

When you call the method Clear_Messages (see 18 Program Code Clear Messages), the event messages are deleted from the log.

```
1. using FestoLagerApi;
2. using Modbus.Device;
3. using Newtonsoft.Json;
4. using System;
5. using System.Collections.Generic;
6. using System.Dynamic;
7. using System.Linq;
8. using System.Text;
9. using System.Threading;
10.    using System.Threading.Tasks;
11.
12.    namespace Daenet.Iot.FestoLagerSystem
13.    {
14.        public class FestoLagerInterpreter : ICommandInterpreter
15.        {
16.    const ushort CoilMoveCompleteLinear = 8192;//Read Register for Move Co mplete
       of Linear Motor
17.    const ushort CoilMoveCompleteRotation = 8193;//Read Register for Move
       Complete of Rotation Motor
18.            const ushort CoilMoveCompleteArmIn = 8194;//Read Register for Arm In
19.            const ushort CoilMoveCompleteArmOut = 8195;//Register for Arm Out
20.            const ushort ColorOne = 8196;//Read Register for Color One grau
21.            const ushort ColorTwo = 8197;//Read Register for Color Two rot
22.            const ushort ColorThree = 8198;//Read Register for Color Three schwarz

23.            const ushort controlbit0 = 8200;//Read Register for Bit0
24.            const ushort controlbit1 = 8201;//Read Register for Bit1
25.            const ushort controlbit2 = 8202;//Read Register for Bit2
26.            const ushort controllin = 8203;//Read Register for Linear Motor Start

27.            const ushort controlrot = 8204;//Read Register for Rotation Motor Star
       t
28.            const ushort armregister = 16390;//Write Register for Arm
29.            const ushort clawregister = 16389;//Write Register for Claw
30.            const ushort bit0register = 16384;//Write Register for Bit0
31.            const ushort bit1register = 16385;//Write Register for Bit1
32.            const ushort bit2register = 16386;//Write Register for Bit2
33.            const ushort linearmotorregister = 16387;//Write Register for Linear M
       otor
34.    const ushort rotationmotorregister = 16388;//Write Register for Rotati on
       Motor
35.
36.            //bearbeiten
37.            const ushort workpieceinplace = 8300;
38.            const ushort workpiecetesterplace = 8301;
39.            const ushort workpiecedrillingplace = 8302;
40.            const ushort drillingmachineupcheck = 8303;
41.            const ushort drillingmachinedowncheck = 8304;
42.            const ushort tabledrivecheck = 8305;
43.            const ushort workpiecetightencomplete = 8306;
```

```
44.             const ushort workpiecetestcomplete = 8307;
45.             const ushort workpieceholecheck = 8308;
46.             const ushort tablegocomplete = 8309;
47.             const ushort tablego = 16400;
48.             const ushort drillingmachineup = 16401;
49.             const ushort drillingmachinedown = 16402;
50.             const ushort workpiecetighten = 16403;
51.             const ushort workpiecetest = 16404;
52.             ushort holestatus;
53.             //bearbeiten
54.
55.             // const ushort cyclic = 8199;
56.             /// <summary>
57.             /// Callback function used for tracing.
58.             /// </summary>
59.             private  Action<string, object[]> m_TraceEventService;
60.
61.     private Func<object, int, DateTime, object, Exception, Task> m_OnEvent
        Callback;
62.
```

19 Programmcode Bibliotheken und Variablen

Aufgaben Abarbeitung

With the Run method (see 20 Program Code Tasks) the Json commands are processed by a switch case query depending on the command. In addition, events for e.g. wrong commands are also output.

```
63.             public Task Run(DeviceCommandMessage deviceCommand,
64.                 Action<string, object[]> traceEventService,
65.     Func<object, int, DateTime, object, Exception, Task> onEventCallba ck = null,

66.                 CancellationTokenSource cancelationTokenSource = null)
67.             {
 68.                 PlcProgram program = JsonConvert.DeserializeObject<PlcProgram>(dev
        iceCommand.Data.ToString());
 69.                 FestoLagerInterpreter interpreter = new FestoLagerInterpreter();
 70.
 71.                 var t = new Task(() =>
 72.                 {
 73.                     bool nocommand = true;
 74.
 75.                     m_TraceEventService = traceEventService;
 76.
 77.                     m_OnEventCallback = onEventCallback;
 78.
 79.                     ModbusIpMaster master = null;
 80.
 81.                     m_TraceEventService(TextResources.PrgStart, new object[] { pro
        gram.Name });//output when event start
 82.
 83.
 84.
 85.                     foreach (var command in program.ProgramSequence)
 86.                     {
```

```
87.                         nocommand = false;
88.
89.
90.                         switch (command.CommandName)
91.                         {
92.                             case Commands.ConnectToSlave:
93.                                 master = execConnectToSlave(command);
94.                                 break;
95.
96.                             case Commands.MoveArm:
97.
98.                                 execMoveArm(master, command);
99.                                 break;
100.
101.                            case Commands.MoveClaw:
102.
103.                                execMoveClaw(master, command);
104.                                break;
105.
106.                            case Commands.MoveToOverPosition:
107.
108.                                execMoveToOverPosition(master, command);
109.                                break;
110.
111.                            case Commands.MoveToPosition:
112.                                execMoveToPosition(master, command);
113.                                break;
114.
115.                            case Commands.Demo:
116.                                execDemo(master,command);
117.                                break;
118.                            //bearbeiten
119.                            case Commands.MoveTest:
120.                                execMoveTest(master, command);
121.                                break;
122.
123.                            case Commands.MoveDrill:
124.                                execMoveDrill(master, command);
125.                                break;
126.                            case Commands.MoveTight:
127.                                execMoveTight(master, command);
128.                                break;
129.                            case Commands.MoveTable:
130.                                execMoveTable(master, command);
131.                                break;
132.                            //bearbeiten
133.                            default:
134.                                m_TraceEventService(TextResources.ErrCmd, new obje
        ct[] { "Wrong command" });//wrong command
135.                                break;
136.                        }
137.
138.
```

```
139.
140.                     }
141.                     if (nocommand == true)
142.                     {
143.                         m_TraceEventService(TextResources.NoCmd, new object[] { "N
      o command" });//wrong command
144.                     }
145.
146.
147.
148.                     m_TraceEventService(TextResources.PrgFinish, new object[] { pr
      ogram.Name });//output when event finish
149.                 }, cancelationTokenSource.Token);
150.
151.                 t.Start();
152.
153.                 return t;
154.             }
155.
```

20 Programmcode Aufgaben Abarbeitung

Kommando: Verbindung mit Slave, Prüfung und Event

The ModbusIpMaster method (see 21 Program code command: Connection with slave with check and event) is used to establish a connection between the program and the PLC. Events are also created depending on the event.

```
156.             private  ModbusIpMaster execConnectToSlave(PlcCommand cmd)
157.             {
158.   m_TraceEventService(TextResources.CmdConnectSlave, new object[] {
       cmd.Args["Address"], cmd.Args["Port"] });//output when event start
159.   m_OnEventCallback(this, 5, DateTime.Now, "connected", null); //Tod o

160.   return       ModbusIpMaster.CreateIp((string)cmd.Args["Address"],       Conver
       t.ToInt16(cmd.Args["Port"]));
161.   //TODO//m_TraceEventService(TextResources.CmdConnectSlaveFinish, n ew object[]
       { cmd.Args["Address"], cmd.Args["Port"] });//output when event fin ish
162.             }
163.
```

21 Programmcode Kommando: Verbindung mit Slave mit Prüfung und Event

Kommando: Demo, Prüfung und Event

The method execDemo (see 22 Program code command: Demo with check and event) is used to run the machine in random mode during a presentation.

```
164.             private void execDemo(ModbusIpMaster master, PlcCommand cmd)
165.             {
 166.                 Random rnd = new Random();
 167.                 int InshelfNr = 0;
 168.                 int OutshelfNr=0;
```

```
169.                int rndcyl = 0;
170.                int cyl = 0;
171.                int cylsave2 = 0;
172.                int cylsave1 = 0;
173.                int cylsave0 = 0;
174.                int oldrndcyl0 = 0;
175.                int oldrndcyl1 = 0;
176.                int oldrndcyl2 = 0;
177.                for (int x = 0; x != Convert.ToInt16(cmd.Args["Count"]); x++)//tes
      t
178.                {
179.                    if (cyl == 2)
180.                    {
181.                        while (oldrndcyl2 == rndcyl)
182.                        {
183.                            rndcyl = rnd.Next(0, 3);
184.                        }
185.                        oldrndcyl2 = rndcyl;
186.                    }
187.                        if (cyl == 1) {
188.                    while (oldrndcyl1 == rndcyl)
189.                    {
190.                        rndcyl = rnd.Next(0, 3);
191.                    }
192.                    oldrndcyl1 = rndcyl;
193.                }
194.                    if (cyl == 0)
195.                    {
196.                        while (oldrndcyl0 == rndcyl)
197.                        {
198.                            rndcyl = rnd.Next(0, 3);
199.                        }
200.                        oldrndcyl0 = rndcyl;
201.                    }
202.                        if (cyl == 2)
203.                        {
204.                            if (rndcyl == 3)
205.                            {
206.                                OutshelfNr = 9;
207.                            }
208.                            if (rndcyl == 2)
209.                            {
210.                                OutshelfNr = 18;
211.                            }
212.                            if (rndcyl == 1)
213.                            {
214.                                OutshelfNr = 1;
215.                            }
216.                            if (rndcyl == 0)
217.                            {
218.                                OutshelfNr = 8;
219.                            }
220.                        }
221.
222.                        if (cyl == 1)
223.                    {
224.                        if (rndcyl == 3)
225.                        {
226.                            OutshelfNr = 13;
```

```
227.                     }
228.                     if (rndcyl == 2)
229.                     {
230.                         OutshelfNr = 5;
231.                     }
232.                     if (rndcyl == 1)
233.                     {
234.                         OutshelfNr = 16;
235.                     }
236.                     if (rndcyl == 0)
237.                     {
238.                         OutshelfNr = 15;
239.                     }
240.                 }
241.
242.                 if (cyl == 0)
243.                 {
244.
245.                     if (rndcyl == 3)
246.                     {
247.                         OutshelfNr = 11;
248.                     }
249.                     if (rndcyl == 2)
250.                     {
251.                         OutshelfNr = 10;
252.                     }
253.                     if (rndcyl == 1)
254.                     {
255.                         OutshelfNr = 3;
256.                     }
257.                     if (rndcyl == 0)
258.                     {
259.                         OutshelfNr = 2;
260.                     }
261.                 }
262.
263.
264.                 m_TraceEventService(TextResources.CmdMoveArm, new object[] { f
    alse });//output when event start
265.                 movearm(master, false);
266.                 m_TraceEventService(TextResources.CmdMoveArmFinish, new object
    [] { false });//output when event finish
267.
268.                 m_OnEventCallback(this, 1, DateTime.Now, false, null);
269.
270.
271.
272.                 m_TraceEventService(TextResources.CmdMoveClaw, new object[] {
    false });//output when event start
273.                 master.WriteSingleCoil(clawregister, false);
274.                 m_TraceEventService(TextResources.CmdMoveClawFinish, new objec
    t[] { false });//output when event finish
275.
276.                 m_OnEventCallback(this, 2, DateTime.Now, false, null);
277.
278.
279.
280.                 var nr = Convert.ToInt16(InshelfNr);
```

```
281.                     m_TraceEventService(TextResources.CmdMoveToOverPosition, new o
      bject[] { nr });//output when event start
282.                     movetooverposition(master, nr);
283.                     m_TraceEventService(TextResources.CmdMoveToOverPositionFinish,
       new object[] { nr });//output when event finish
284.
285.                     m_OnEventCallback(this, 3, DateTime.Now, nr, null);
286.
287.
288.                     m_TraceEventService(TextResources.CmdMoveArm, new object[] { t
      rue });//output when event start
289.                     movearm(master, true);
290.                     m_TraceEventService(TextResources.CmdMoveArmFinish, new object
      [] { true });//output when event finish
291.
292.                     m_OnEventCallback(this, 1, DateTime.Now, true, null);
293.
294.
295.                     m_TraceEventService(TextResources.CmdMoveToPosition, new objec
      t[] { nr });//output when event start
296.                     movetoposition(master, nr);
297.                     m_TraceEventService(TextResources.CmdMoveToPositionFinish, new
       object[] { nr });//output when event finish
298.
299.                     m_OnEventCallback(this, 4, DateTime.Now, nr, null);
300.
301.                     m_TraceEventService(TextResources.CmdMoveClaw, new object[] {
      true });//output when event start
302.                     master.WriteSingleCoil(clawregister, true);
303.                     m_TraceEventService(TextResources.CmdMoveClawFinish, new objec
      t[] { true });//output when event finish
304.
305.                     m_OnEventCallback(this, 2, DateTime.Now, true, null);
306.
307.                     m_TraceEventService(TextResources.CmdMoveToOverPosition, new o
      bject[] { nr });//output when event start
308.                     movetooverposition(master, nr);
309.                     m_TraceEventService(TextResources.CmdMoveToOverPositionFinish,
       new object[] { nr });//output when event finish
310.
311.                     m_OnEventCallback(this, 3, DateTime.Now, nr, null);
312.
313.
314.                     m_TraceEventService(TextResources.CmdMoveArm, new object[] { f
      alse });//output when event start
315.                     movearm(master, false);
316.                     m_TraceEventService(TextResources.CmdMoveArmFinish, new object
      [] { false });//output when event finish
317.
318.                     m_OnEventCallback(this, 1, DateTime.Now, false, null);
319.
320.                     var outnr = Convert.ToInt16(OutshelfNr);
321.                     m_TraceEventService(TextResources.CmdMoveToOverPosition, new o
      bject[] { outnr });//output when event start
322.                     movetooverposition(master, outnr);
323.                     m_TraceEventService(TextResources.CmdMoveToOverPositionFinish,
       new object[] { outnr });//output when event finish
324.
325.                     m_OnEventCallback(this, 3, DateTime.Now, outnr, null);
```

```
326.
327.
328.                     m_TraceEventService(TextResources.CmdMoveArm, new object[] { t
    rue });//output when event start
329.                     movearm(master, true);
330.                     m_TraceEventService(TextResources.CmdMoveArmFinish, new object
    [] { true });//output when event finish
331.
332.                     m_OnEventCallback(this, 1, DateTime.Now, true, null);
333.
334.
335.                     m_TraceEventService(TextResources.CmdMoveToPosition, new objec
    t[] { outnr });//output when event start
336.                     movetoposition(master, outnr);
337.                     m_TraceEventService(TextResources.CmdMoveToPositionFinish,  new
     object[] { outnr });//output when event finish
338.
339.                     m_OnEventCallback(this, 4, DateTime.Now, outnr, null);
340.
341.
342.                     m_TraceEventService(TextResources.CmdMoveClaw, new object[] {
    false });//output when event start
343.                     master.WriteSingleCoil(clawregister, false);
344.                     m_TraceEventService(TextResources.CmdMoveClawFinish, new objec
    t[] { false });//output when event finish
345.
346.                     m_OnEventCallback(this, 2, DateTime.Now, false, null);
347.
348.                     m_TraceEventService(TextResources.CmdMoveToOverPosition, new o
    bject[] { outnr });//output when event start
349.                     movetooverposition(master, outnr);
350.                     m_TraceEventService(TextResources.CmdMoveToOverPositionFinish,
     new object[] { outnr });//output when event finish
351.
352.                     m_OnEventCallback(this, 3, DateTime.Now, outnr, null);
353.
354.                     m_TraceEventService(TextResources.CmdMoveArm, new object[] { f
    alse });//output when event start
355.                     movearm(master, false);
356.                     m_TraceEventService(TextResources.CmdMoveArmFinish, new object
    [] { false });//output when event finish
357.
358.                     m_OnEventCallback(this, 1, DateTime.Now, false, null);
359.
360.                     if (cyl == 1)
361.                     {
362.                         cyl = 0;
363.                         cylsave1 = OutshelfNr;
364.                         InshelfNr = cylsave0;
365.                     }
366.                     else if (cyl == 0)
367.                     {
368.                         cyl = 2;
369.                         cylsave0 = OutshelfNr;
370.                         InshelfNr = cylsave2;
371.                     }
372.                     else
373.                     {
374.                         cyl = 1;
```

```
375.                        cylsave2 = OutshelfNr;
376.                        InshelfNr = cylsave1;
377.                    }
378.                }
379.            }
380.
```

22 Programmcode Kommando: Demo mit Prüfung und Event

Kommando: Arm Aus- und Einfahren, Prüfung und Event

With the method execMoveArm (see 23 Program code command: Arm extension and retraction with test and event) the Json command for the movement of the arm is processed and returned with corresponding events.

```
381.            private  void execMoveArm(ModbusIpMaster master, PlcCommand cmd)
382.            {
383.                if (cmd.Args.ContainsKey("Direction") == false)
384.    throw new ArgumentException("Command MoveArm must contain argu ment
        'Direction' of boolean type");
385.
386.    m_TraceEventService(TextResources.CmdMoveArm, new object[] { cmd.A
        rgs["Direction"] });//output when event start
387.                movearm(master, (bool)cmd.Args["Direction"]);
388.     m_TraceEventService(TextResources.CmdMoveArmFinish, new object[] {
         cmd.Args["Direction"] });//output when event finish
389.
390.                m_OnEventCallback(this, 1, DateTime.Now, (bool)cmd.Args["Direction"
        ], null);
391.            }
392.
```

23 Programmcode Kommando: Arm Aus- und Einfahren mit Prüfung und Event

Kommando Klaue Bewegen, Prüfung und Event

With the method execMoveClaw (see 24 Program Code Command: Claw Move with Check and Event) the Json command of the claw movement is processed and returned with corresponding events.

```
393.            private  void execMoveClaw(ModbusIpMaster master, PlcCommand cmd)
394.            {
395.
396.                if (cmd.Args.ContainsKey("Condition") == false)
397.    throw new ArgumentException("Command MoveClaw must contain arg ument
        'Condition' of boolean type");
398.    m_TraceEventService(TextResources.CmdMoveClaw, new object[] { cmd.
        Args["Condition"] });//output when event start
399.                master.WriteSingleCoil(clawregister, (bool)cmd.Args["Condition"]);

400.    m_TraceEventService(TextResources.CmdMoveClawFinish, new object[] {
        cmd.Args["Condition"] });//output when event finish
401.
```

```
402.                 m_OnEventCallback(this, 2, DateTime.Now, (bool)cmd.Args["Condition
      "], null);
403.             }
```

24 Programmcode Kommando: Klaue Bewegen mit Prüfung und Event

Kommando: Über Fach fahren, Prüfung und Event

The method execMoveToOverPosition (see 25 Program Code Command: Traversing via Tray with Check and Event) processes the Json command of the movement via Tray and returns it with corresponding events..

```
404.             private  void execMoveToOverPosition(ModbusIpMaster master, PlcCommand
       cmd)
405.             {
406.                 if (cmd.Args.ContainsKey("shelfNr") == false)
407.   throw new ArgumentException("Command MoveToOverPosition must c ontain argument
      'shelfNr' of integer type");
408.
409.                 var nr = Convert.ToInt16(cmd.Args["shelfNr"]);
410.   m_TraceEventService(TextResources.CmdMoveToOverPosition, new objec t[] { nr
      });//output when event start
411.                 movetooverposition(master, nr);
412.    m_TraceEventService(TextResources.CmdMoveToOverPositionFinish, new object[] {
       nr });//output when event finish
413.
414.                 m_OnEventCallback(this, 3, DateTime.Now, nr, null);
415.             }
416.
```

25 Programmcode Kommando: Über Fach fahren mit Prüfung und Event

Kommando: Fach einlagern, Prüfung und Event

With the method execMoveToPosition (see 26 Program Code Command: Store Partition with Check and Event), the Json command of the movement is processed in the Store Partition and returned with the corresponding events.

```
417.             private  void execMoveToPosition(ModbusIpMaster master, PlcCommand cmd
      )
418.             {
419.                 if (cmd.Args.ContainsKey("shelfNr") == false)
420.   throw new ArgumentException("Command MoveToOverPosition must c ontain argument
      'shelfNr' of integer type");
421.                 var nr = Convert.ToInt16(cmd.Args["shelfNr"]);
422.   m_TraceEventService(TextResources.CmdMoveToPosition, new object[] { nr
      });//output when event start
423.                 movetoposition(master, nr);
424.   m_TraceEventService(TextResources.CmdMoveToPositionFinish, new obj ect[] { nr
      });//output when event finish
425.
426.                 m_OnEventCallback(this, 4, DateTime.Now, nr, null);//TODO..
427.             }
```

```
428.
```

26 Programmcode Kommando: Fach einlagern mit Prüfung und Event

Überprüfung: Warte bis Bewegung abgeschlossen

With the method waitToComplete (see 27 Program code check: Wait until movement is completed) you wait until a corresponding register "coil" has assumed a corresponding value "waitState".

```
429.    private static void waitToComplete(ModbusIpMaster master, ushort coil, ushort
        waitState)
430.            {
431.                ushort[] state = new ushort[1];
432.                do
433.                {
434.                    state = master.ReadInputRegisters(coil, 1);
435.                } while (state[0] == waitState);
436.            }
437.
```

27 Programmcode Überprüfung: Warte bis Bewegung abgeschlossen

Bewegung Arm: ausfahren und einfahren

The method movearm (see 28 Program code Movement arm: extend and retract) controls the process of the robot arm movement. For this purpose, the state that the arm is to assume (true = extend arm; false = retract arm) is sent via Modbus. Thereupon it is queried via Modbus whether the arm has reached this state.

```
438.            private static void movearm(ModbusIpMaster master, bool state)
439.            {
 440.               master.WriteSingleCoil(armregister, state); //move arm out
 441.               if (state == true)
 442.               {
 443.                   waitToComplete(master, CoilMoveCompleteArmOut, 0);//check if a
       rm out
 444.               }
 445.               else
 446.               {
 447.                   waitToComplete(master, CoilMoveCompleteArmIn, 0);//check if ar
       m in
 448.               }
 449.
450.            }
451.
```

28 Programmcode Bewegung Arm: ausfahren und einfahren

Berechnung: Linear Fach Nummer

With the method linshelfbool (see 29 Program Code Calculation: Linear Partition Number30 Program Code Calculation: Rotation Partition Number) the correct bit sequence for the linear motor is output depending on the selected shelf "shelfbool". A switch case query first saves the bit combination as an integer "LinNr" and then converts it to a boolean register "linbool" where it is stored and returned.

```
452.            public static bool[] linshelfbool(int shelfbool)
453.            {
454.                bool[] linbool = new bool[8];
455.                int LinNr = 0;
456.
457.                switch (shelfbool)
458.                {
459.                    case 0:
460.                        LinNr = 7;//111 input Bay
461.                        break;
462.                    case 1:
463.                    case 2:
464.                    case 3:
465.                    case 4:
466.                    case 5:
467.                    case 6:
468.                        LinNr = 2;//010 top Bay
469.                        break;
470.                    case 7:
471.                    case 8:
472.                    case 9:
473.                    case 10:
474.                    case 11:
475.                    case 12:
476.                        LinNr = 4;//100 middle Bay
477.                        break;
478.                    case 13:
479.                    case 14:
480.                    case 15:
481.                    case 16:
482.                    case 17:
483.                    case 18:
484.                        LinNr = 5;//101 lowest Bay
485.                        break;
486.                    case 19:
487.                        LinNr = 0;//000 reference drive
488.                        break;
489.                    default:
490.                        //exception for other listings
491.                        break;
492.                }
493.                ///Convert to bool[]
494.                string Lins = Convert.ToString(LinNr, 2); //Convert to binary in a
        string
495.
496.                int[] Linbits = Lins.PadLeft(8, '0') // Add 0's from left
```

```
497.                             .Select(c => int.Parse(c.ToString())) // convert each
       char to int
498.                             .ToArray(); // Convert IEnumerable from select to Arr
      ay
499.
500.                 if (Linbits[7] == 1)
501.                 {
502.                     linbool[7] = true;
503.                 }
504.                 else
505.                 {
506.                     linbool[7] = false;
507.                 }
508.                 if (Linbits[6] == 1)
509.                 {
510.                     linbool[6] = true;
511.                 }
512.                 else
513.                 {
514.                     linbool[6] = false;
515.                 }
516.                 if (Linbits[5] == 1)
517.                 {
518.                     linbool[5] = true;
519.                 }
520.                 else
521.                 {
522.                     linbool[5] = false;
523.                 }
524.                 return linbool;
525.             }//Linear Shelf Bit Selector
526.
```

29 Programmcode Berechnung: Linear Fach Nummer

Berechnung: Rotations Fach Nummer

In the method rotshelfbool (see 30 Program code Calculation: Rotation partition number) the correct bit sequence for the rotation motor is output depending on the selected partition "shelfbool". A switch case query first saves the bit combination as an integer "RotNr" and then converts it to a boolean register "rotbool" where it is stored and returned.

```
527.             public static bool[] rotshelfbool(int shelfbool)
528.             {
 529.                 bool[] rotbool = new bool[8];
 530.                 int RotNr = 0;
 531.                 switch (shelfbool)
 532.                 {
 533.                     case 0:
 534.                         RotNr = 14;//110 input line
 535.                         break;
 536.                     case 1:
 537.                     case 7:
 538.                     case 13:
```

```
539.                        RotNr = 13;//101 first line
540.                        break;
541.                    case 2:
542.                    case 8:
543.                    case 14:
544.                        RotNr = 12;// 100 second line
545.                        break;
546.                    case 3:
547.                    case 9:
548.                    case 15:
549.                        RotNr = 11;//011 third line
550.                        break;
551.                    case 4:
552.                    case 10:
553.                    case 16:
554.                        RotNr = 10;//010 fourth line
555.                        break;
556.                    case 5:
557.                    case 17:
558.                    case 11:
559.                        RotNr = 9;//001 fifth line
560.                        break;
561.                    case 6:
562.                    case 12:
563.                    case 18:
564.                        RotNr = 8;//000 sixt line
565.                        break;
566.                    case 19:
567.                        RotNr = 15;//111 Reference drive
568.                        break;
569.                    default:
570.                        //exception for other listings
571.                        break;
572.
573.                }
574.                ///Convert to bool[]
575.                string Rots = Convert.ToString(RotNr, 2); //Convert to binary in a
       string
576.
577.                int[] Rotbits = Rots.PadLeft(8, '0') // Add 0's from left
578.                                    .Select(c => int.Parse(c.ToString())) // convert each
       char to int
579.                                    .ToArray(); // Convert IEnumerable from select to Arr
      ay
580.                if (Rotbits[7] == 1)
581.                {
582.                    rotbool[7] = true;
583.                }
584.                else
585.                {
586.                    rotbool[7] = false;
587.                }
588.                if (Rotbits[6] == 1)
589.                {
590.                    rotbool[6] = true;
591.                }
592.                else
593.                {
594.                    rotbool[6] = false;
```

```
595.                }
596.                if (Rotbits[5] == 1)
597.                {
598.                    rotbool[5] = true;
599.                }
600.                else
601.                {
602.                    rotbool[5] = false;
603.                }
604.                return rotbool;
605.            }//Rotation Shelf Bit Selector
606.
```

30 Programmcode Berechnung: Rotations Fach Nummer

Berechnung: Lineare Ablege Postion

With the method ulinshelfbool (see 31 Program code calculation: Linear deposit and 30 Program code calculation: Rotation tray number), the correct bit sequence "shelfbool" is output for the linear motor, depending on the selected tray. This position is required to deposit objects. A switch case query first saves the bit combination as an integer "uLinNr" and then converts it to a Boolean register "ulinbool" where it is stored and returned.

```
607.            public static bool[] ulinshelfbool(int shelfbool)
608.            {
 609.               bool[] ulinbool = new bool[8];
 610.               int uLinNr = 0;
 611.
 612.               switch (shelfbool)
 613.               {
 614.                   case 0:
 615.                       uLinNr = 6; //110 over lowest Bay
 616.                       break;
 617.                   case 1:
 618.                   case 2:
 619.                   case 3:
 620.                   case 4:
 621.                   case 5:
 622.                   case 6:
 623.                       uLinNr = 1; //001 over top Bay
 624.                       break;
 625.                   case 7:
 626.                   case 8:
 627.                   case 9:
 628.                   case 10:
 629.                   case 11:
 630.                   case 12:
 631.                       uLinNr = 3;//011 over middle Bay
 632.                       break;
 633.                   case 13:
 634.                   case 14:
 635.                   case 15:
 636.                   case 16:
```

```
637.                     case 17:
638.                     case 18:
639.                         uLinNr = 6; //110 over lowest Bay
640.                         break;
641.                     case 19:
642.                         uLinNr = 0; //000 reference drive
643.                         break;
644.                     default:
645.                         //exception for other listings
646.                         break;
647.                 }
648.                 ///Convert to bool[]
649.                 string Lins = Convert.ToString(uLinNr, 2); //Convert to binary in
      a string
650.
651.                 int[] uLinbits = Lins.PadLeft(8, '0') // Add 0's from left
652.                                 .Select(c => int.Parse(c.ToString())) // convert each
       char to int
653.                                 .ToArray(); // Convert IEnumerable from select to Arr
      ay
654.
655.                 if (uLinbits[7] == 1)
656.                 {
657.                     ulinbool[7] = true;
658.                 }
659.                 else
660.                 {
661.                     ulinbool[7] = false;
662.                 }
663.                 if (uLinbits[6] == 1)
664.                 {
665.                     ulinbool[6] = true;
666.                 }
667.                 else
668.                 {
669.                     ulinbool[6] = false;
670.                 }
671.                 if (uLinbits[5] == 1)
672.                 {
673.                     ulinbool[5] = true;
674.                 }
675.                 else
676.                 {
677.                     ulinbool[5] = false;
678.                 }
679.                 return ulinbool;
680.             }//Linear Over Shelf Bit Selector
681.
```

31 Programmcode Berechnung: Lineare Ablege Position

Bewegung: Über das Fach

Using the movetooverposition method (see 32 Program code Movement: Via the compartment), the motors are controlled via Modbus to move over the compartment. For this the correct bit combination is taken out of the corresponding method.

First the linear motor is written with the corresponding bits and checked whether the bits have arrived there. Then the motor is started and checked whether this has been executed. Now the start signal is switched off. Next, the commands for the movement are sent to the rotary motor using the same procedure. Finally it is checked whether the two motors have completed their movement.

```
682.          private static void movetooverposition(ModbusIpMaster master, int shel
      fNr)
683.          {
684.              var linbits = ulinshelfbool(shelfNr);
685.              var rotbits = rotshelfbool(shelfNr);
686.              ////linear motor control
687.              ///write position bits
688.              master.WriteSingleCoil(bit0register, linbits[7]); //write Bit0
689.              master.WriteSingleCoil(bit1register, linbits[6]); //write Bit1
690.              master.WriteSingleCoil(bit2register, linbits[5]); //write Bit2
691.              ///check bits for successful bit transmission into sps variable
692.              checkbit(master, controlbit0, linbits[7]);
693.              checkbit(master, controlbit1, linbits[6]);
694.              checkbit(master, controlbit2, linbits[5]);
695.              ///write motor control
696.  master.WriteSingleCoil(linearmotorregister, true); //start move co mmand
      lineal
697.   waitToComplete(master, controllin, 0);//check start for successful
       transmission
698.  master.WriteSingleCoil(linearmotorregister, false); //stop move co mmand
      lineal
699.              waitToComplete(master, controllin, 1);//check stop for successful
      transmission
700.              ///write position bits
701.              master.WriteSingleCoil(bit0register, rotbits[7]); //write Bit0
702.              master.WriteSingleCoil(bit1register, rotbits[6]); //write Bit1
703.              master.WriteSingleCoil(bit2register, rotbits[5]); //write Bit2
704.              ///check bits for successful bit transmission into sps variable
705.              checkbit(master, controlbit0, rotbits[7]);
706.              checkbit(master, controlbit1, rotbits[6]);
707.              checkbit(master, controlbit2, rotbits[5]);
708.              ///write motor control
709.  master.WriteSingleCoil(rotationmotorregister, true); //start move command
      rotation
710.   waitToComplete(master, controlrot, 0);//check start for successful
       transmission into sps variable
711.  master.WriteSingleCoil(rotationmotorregister, false); //stop move command
      rotation
712.  waitToComplete(master, controlrot, 1);//check stop for successful
      transmission into sps variable
713.              ///wait till move stop
714.  waitToComplete(master, CoilMoveCompleteLinear, 0);//wait till line ar move
      stop
715.  waitToComplete(master, CoilMoveCompleteRotation, 0);//wait till ro tation move
      stop
716.          }//move to over shelf position
717.
```

32 Programmcode Bewegung: Über das Fach

Bewegung: Ablage in das Fach

With the method movetoposition (see33 Program code Movement: Deposit into tray) the motors are controlled via Modbus to deposit objects into the tray. For this the correct bit combination is taken out of the corresponding method. Then the Linear Motor is first described with the corresponding bits and checked whether the bits have arrived there. The motor is then started and checked to see if this has been done. Now the start signal is switched off. Next, the commands for the movement are sent to the rotary motor using the same procedure. Finally, it is checked whether the two motors have completed their movement.

```
718.          private static void movetoposition(ModbusIpMaster master, int shelfNr)
719.          {
720.              var linbits = linshelfbool(shelfNr);
721.              var rotbits = rotshelfbool(shelfNr);
722.              ////linear motor control
723.              ///write position bits
724.              master.WriteSingleCoil(bit0register, linbits[7]); //write Bit0
725.              master.WriteSingleCoil(bit1register, linbits[6]); //write Bit1
726.              master.WriteSingleCoil(bit2register, linbits[5]); //write Bit2
727.              ///check bits for successful bit transmission into sps variable
728.              checkbit(master, controlbit0, linbits[7]);
729.              checkbit(master, controlbit1, linbits[6]);
730.              checkbit(master, controlbit2, linbits[5]);
731.              ///write motor control
732.  master.WriteSingleCoil(linearmotorregister, true); //start move co mmand
      lineal
733.   waitToComplete(master, controllin, 0);//check start for successful
       transmission into sps variable
734.  master.WriteSingleCoil(linearmotorregister, false); //stop move co mmand
      lineal
735.  waitToComplete(master, controllin, 1);//check stop for successful
      transmission into sps variable
736.              ///write position bits
737.              master.WriteSingleCoil(bit0register, rotbits[7]); //write Bit0
738.              master.WriteSingleCoil(bit1register, rotbits[6]); //write Bit1
739.              master.WriteSingleCoil(bit2register, rotbits[5]); //write Bit2
740.              ///check bits for successful bit transmission into sps variable
741.              checkbit(master, controlbit0, rotbits[7]);
742.              checkbit(master, controlbit1, rotbits[6]);
743.              checkbit(master, controlbit2, rotbits[5]);
744.              ///write motor control
745.  master.WriteSingleCoil(rotationmotorregister, true); //start move command
      rotation
746.   waitToComplete(master, controlrot, 0);//check start for successful
       transmission into sps variable
747.  master.WriteSingleCoil(rotationmotorregister, false); //stop move command
      rotation
748.  waitToComplete(master, controlrot, 1);//check stop for successful
      transmission into sps variable
749.              ///wait till move stop
```

```
waitToComplete(master, CoilMoveCompleteLinear, 0);//wait till line ar move stop
waitToComplete(master, CoilMoveCompleteRotation, 0);//wait till ro tation move stop
        }//move to shelf positon

```

33 Programmcode Bewegung: Ablage in das Fach

Überprüfung: Ob Bit angekommen

With the checkbit method (see 34 Program code Check: Whether bit arrived), Modbus queries whether the set bits "checkbit" have arrived on the machine. This is done by querying the Modbus registers "coil" of the machine.

```
private static void checkbit(ModbusIpMaster master, ushort coil, bool checkbit)
        {
            ushort[] state = new ushort[1];
            int bit;
            if (checkbit == true)
            {
                bit = 1;
            }
            else
            {
                bit = 0;
            }
            do
            {
                state = master.ReadInputRegisters(coil, 1);

            } while (state[0] != bit);
        }


        //bearbeiten
```

34 Programmcode Überprüfung: Ob Bit angekommen

Bewegung: Führe Werkstück Test aus

The movetest method (see 35 Program code Movement: Perform Workpiece Test) is used to have the machine perform the Workpiece Bore Test. For this purpose it is checked whether a workpiece is present. If a workpiece is present, the machine will execute the workpiece test and save and return the result in "holestate".

```
        private static ushort movetest(ModbusIpMaster master, bool state)
        {
            ushort holestate = 0;
            ushort[] check = new ushort[1];
            if (checkworkpiece(master, 2) == 1)
            {
```

```
781.                    master.WriteSingleCoil(workpiecetest, state);
782.                    if (state == true)
783.                    {
784.                        waitToComplete(master, workpiecetestcomplete, 0);
785.                        check = master.ReadInputRegisters(workpieceholecheck, 1);

786.                        holestate = check[0];
787.                    }
788.                    else
789.                    {
790.                        waitToComplete(master, workpiecetestcomplete, 1);
791.                    }
792.
793.                }
794.                return holestate;
795.            }
796.
```

35 Programmcode Bewegung: Führe Werkstück Test aus

Bewegung: Bewege Bohrer

With the movedrill method (see 36 Program code Movement: Moving Drill), if there is a part and the machine has a hole, the drill will start drilling. Depending on the direction selected, the Modbus signal is sent to move the drill up or down and then waits for the drill to move to the stop before cancelling the signal.

```
797.    private static void movedrill(ModbusIpMaster master, bool state, ushor t
        holestate)
798.            {
799.                if (checkworkpiece(master, 3) == 1)
800.                {
801.                    if (state == true && holestate == 1)
802.                    {
803.                        master.WriteSingleCoil(drillingmachineup, true);
804.                        waitToComplete(master, drillingmachineupcheck, 0);
805.                        master.WriteSingleCoil(drillingmachineup, false);
806.                    }
807.                    else if (state == false)
808.                    {
809.                        master.WriteSingleCoil(drillingmachinedown, true);
810.                        waitToComplete(master, drillingmachinedowncheck, 0);
811.                        master.WriteSingleCoil(drillingmachinedown, false);
812.                    }
813.                }
814.
815.            }
816.
```

36 Programmcode Bewegung: Bewege Bohrer

Bewegung: Bewege Festhalte Arm

With the method movetight (see 37 Program code Movement: Movement Hold Arm) the holding mechanism of the machine is actuated. The state to be assumed is sent via Modbus. Then the machine is waited until it has assumed the state.

```
817.          private static void movetight(ModbusIpMaster master, bool state)
818.          {
 819.             if (checkworkpiece(master, 3) == 1)
 820.             {
 821.                 master.WriteSingleCoil(workpiecetighten, state);
 822.                 if (state == true)
 823.                 {
 824.                     waitToComplete(master, workpiecetightencomplete, 0);
 825.                 }
 826.                 else
 827.                 {
 828.                     waitToComplete(master, workpiecetightencomplete, 1);
 829.                 }
 830.             }
 831.
832.          }
```

37 Programmcode Bewegung: Bewege Festhalte Arm

Bewegung: Bewege Tisch

The method movetable (see 38 Program code Movement: Moving table) serves to rotate the table of the machine. For this, the command to start the rotation is sent via Modbus of the machine. Then Modbus asks if the table is moving and then cancels the rotation command. Thereupon it is queried whether the table has taken its end position.

```
833.          private static void movetable(ModbusIpMaster master, bool state)
834.          {
835.
836.              master.WriteSingleCoil(tablego, state);
837.              waitToComplete(master, tablegocomplete, 0);
838.              waitToComplete(master, tabledrivecheck, 0);
839.              master.WriteSingleCoil(tablego, false);
840.              waitToComplete(master, tablegocomplete, 1);
841.
842.          }
843.
```

38 Programmcode Bewegung: Bewege Tisch

Überprüfung: Gegenstand in Werkbänken

The checkworkpiece method (see 39 Program Code Check: Object in Workbenches) checks whether a workpiece is present in the respective work areas of the machine in order to store and return the status in the "check" variable.

```
844.          private static ushort checkworkpiece(ModbusIpMaster master, int place)
845.          {
846.              ushort[] check = new ushort[1];
847.
848.              if (place == 1){
849.                  check = master.ReadInputRegisters(workpieceinplace, 1);
850.
851.              }
852.              else if(place == 2){
853.                  check = master.ReadInputRegisters(workpiecetesterplace, 1);
854.
855.              }
856.              else if (place == 3){
857.                  check = master.ReadInputRegisters(workpiecedrillingplace, 1);
858.
859.              }
860.              return check[0];
861.          }
862.
```

39 Programmcode Überprüfung: Gegenstand in Werkbänken

Kommando: Bewege Tester, Prüfung und Event

With the method execMoveTest (see 40 Program Code Command: Move Tester with Check and Event) the Json command of the Move Test is processed and returned with corresponding events.

```
863.          private void execMoveTest(ModbusIpMaster master, PlcCommand cmd)
864.          {
865.              if (cmd.Args.ContainsKey("Test") == false)
866.  throw new ArgumentException("Command MoveArm must contain argu ment
      'Direction' of boolean type");//Todo
867.
868.  m_TraceEventService(TextResources.CmdMoveArm, new object[] { cmd.A rgs["Test"]
      });//Todo
869.              holestatus = movetest(master, (bool)cmd.Args["Test"]);
870.   m_TraceEventService(TextResources.CmdMoveArmFinish, new object[] {
       cmd.Args["Test"] });//Todo
871.
872.              m_OnEventCallback(this, 1, DateTime.Now, (bool)cmd.Args["Test"], n
      ull);//Todo
```

```
873.            }
```

40 Programmcode Kommando: Bewege Tester mit Prüfung und Event

Kommando: Bewege Bohrer, Prüfung und Event

With the method execMoveDrill (see 41 Program Code Command: Move Drill with Check and Event) the Json command of the move drill is processed and returned with corresponding events.

```
874.            private void execMoveDrill(ModbusIpMaster master, PlcCommand cmd)
875.            {
876.                if (cmd.Args.ContainsKey("Drill") == false)
877.    throw new ArgumentException("Command MoveArm must contain argu ment
        'Direction' of boolean type");//Todo
878.
879.    m_TraceEventService(TextResources.CmdMoveArm, new object[] { cmd.A
        rgs["Drill"] });//Todo
880.                movedrill(master, (bool)cmd.Args["Drill"],holestatus);
881.     m_TraceEventService(TextResources.CmdMoveArmFinish, new object[] {
         cmd.Args["Drill"] });//Todo
882.
883.                m_OnEventCallback(this, 1, DateTime.Now, (bool)cmd.Args["Drill"],
        null);//Todo
884.            }
```

41 Programmcode Kommando: Bewege Bohrer mit Prüfung und Event

Kommando: Bewege Tisch, Prüfung und Event

The method execMoveTable (see 42 Program Code Command: Move Table with Check and Event) processes the Json command of the Move Table and returns it with corresponding events.

```
885.            private void execMoveTable(ModbusIpMaster master, PlcCommand cmd)
886.            {
887.                if (cmd.Args.ContainsKey("Table") == false)
888.    throw new ArgumentException("Command MoveArm must contain argu ment
        'Direction' of boolean type");//Todo
889.
890.    m_TraceEventService(TextResources.CmdMoveArm, new object[] { cmd.A
        rgs["Table"] });//Todo
891.                movetable(master, (bool)cmd.Args["Table"]);
892.     m_TraceEventService(TextResources.CmdMoveArmFinish, new object[] {
         cmd.Args["Table"] });//Todo
893.
894.                m_OnEventCallback(this, 1, DateTime.Now, (bool)cmd.Args["Table"],
        null);//Todo
895.            }
```

42 Programmcode Kommando: Bewege Tisch mit Prüfung und Event

Kommando: Bewege Befestigung, Prüfung und Event

By the method execMoveTight (see 43 Program Code Command: Move Attachment with Check and Event40 Program Code Command: Move Tester with Check and Event) the Json command of the Move Hold is processed and returned with corresponding events.

```
896.            private void execMoveTight(ModbusIpMaster master, PlcCommand cmd)
897.            {
898.                if (cmd.Args.ContainsKey("Tight") == false)
899.    throw new ArgumentException("Command MoveArm must contain argu ment
        'Direction' of boolean type");//Todo
900.
901.    m_TraceEventService(TextResources.CmdMoveArm, new object[] { cmd.A
        rgs["Tight"] });//Todo
902.                movetight(master, (bool)cmd.Args["Tight"]);
903.     m_TraceEventService(TextResources.CmdMoveArmFinish, new object[] {
         cmd.Args["Tight"] });//Todo
904.
905.                m_OnEventCallback(this, 1, DateTime.Now, (bool)cmd.Args["Tight"],
        null);//Todo
906.            }
907.            //bearbeiten
```

43 Programmcode Kommando: Bewege Befestigung mit Prüfung und Event

Überprüfung der Befehle

The CanInterprete method (see 44 Checking the Program Code of Commands) is used to return a command that cannot be interpreted.

```
908.            public bool CanInterprete(DeviceCommandMessage deviceCommand)
909.            {
 910.               if (deviceCommand.CommandType == DeviceCommandMessage.CmdPlcProgra
        m)
 911.                   return true;
 912.               else
 913.                   return false;
914.            }
915.
916.
```

44 Programmcode Überprüfung der Befehle

Ausgabe der Befehlsanzahl

The GetNumOfSequences method (see 45 Program code output of the command count) is used to count and return the program steps.

```
917.            public int GetNumOfSequences(DeviceCommandMessage deviceCommand)
918.            {
```

```
919.                if (deviceCommand.CommandType == DeviceCommandMessage.CmdPlcProgra
      m)
920.                {
921.                    PlcProgram program = JsonConvert.DeserializeObject<PlcProgram>
      (deviceCommand.Data.ToString());
922.                    if (program.ProgramSequence.Count == 2)//DEMO
923.                    {
924.                        return 5000;//DEMO
925.                    }
926.                    else
927.                    {
928.                        return program.ProgramSequence.Count;
929.                    }
930.
931.                }
932.                else
933.                    return -1;
934.            }
```

45 Programmcode Ausgabe der Befehlsanzahl

Slot Cloud Initialisierung

The GetInitialState method (see 46 GetInitialState program code) is used to return the information of the slots from the cloud to the user interface.

```
935.            public object GetInitialState(DeviceCommandMessage deviceCommand)
936.            {
 937.              if (deviceCommand.CommandType == DeviceCommandMessage.CmdPlcProgram)
 938.              {
 939.                    PlcProgram program = JsonConvert.DeserializeObject<PlcProgram>
      (deviceCommand.Data.ToString());
 940.                    return program.SlotDefinitions;
 941.                }
 942.                else
 943.                    return null;
944.            }
945.        }
946.    }
```

46 Programmcode GetInitialState

JSON Kommandos

This part of the program (see 47 Program code Json libraries and warehouses with sequence) contains the libraries and objects used, as well as the Festo storage station and a command example for the one-time reference run.

```
1.      using System;
2.      using System.Collections.Generic;
3.      using System.Dynamic;
4.      using System.Web.Http;
5.      using Daenet.Diagnostics;
6.      using Daenet.SecurityManager.Owin;
```

```
7.      using PLC.RemoteControl.Entities;
8.      using System.Security.Permissions;
9.      using Daenet.IoT;
10.     using Newtonsoft.Json.Linq;
11.     using System.Linq;
12.
13.     namespace PLC.RemoteControl.Controllers
14.     {
15.         public static class Commands
16.         {
17.             public const string ConnectToSlave = "ConnectToSlave";
18.
19.             public const string MoveArm = "MoveArm";
20.
21.             public const string MoveClaw = "MoveClaw";
22.
23.             public const string MoveToOverPosition = "MoveToOverPosition";
24.
25.             public const string MoveToPosition = "MoveToPosition";
26.
27.             public const string Demo = "Demo";
28.
29.             //bearbeiten
30.             public const string MoveTest = "MoveTest";
31.
32.             public const string MoveDrill = "MoveDrill";
33.
34.             public const string MoveTight = "MoveTight";
35.
36.             public const string MoveTable = "MoveTable";
37.             //bearbeiten
38.         }
39.
40.        // [Authorize]
41.         public class PlcController : ApiController
42.         {
43.             private static PlcModel m_Model;
44.
45.             /// <summary>
46.             /// Gets the list of all available programs for all machines (devices)
47.             /// </summary>
48.             /// <returns></returns>
49.     #if USE_AAD
50.             [ScmAuthorizeAttribute("PLC.Operator")]
51.     #else
52.             [PrincipalPermission(SecurityAction.Demand, Role = "PLC.Operator")]
53.     #endif
54.             [HttpGet]
55.             public PlcModel QueryPrograms()
56.             {
57.                 try
58.                 {
59.                     if (m_Model == null)
60.                     {
61.
62.                         m_Model = new PlcModel()
63.                         {
64.
65.                             Devices = new List<Device>()
```

```
66.                         {
67.
68.                             new Device()
69.                             {
70.                                DeviceId = "FESTOLAGER01",
71.                                 Description = "Festo Lager System",
72.                                 Location = "Hannover Messe",
73.
74.
75.                                ProgramList = new List<PlcProgram>()
76.                                {
77.                                   new PlcProgram()
78.                     {
79.                                        Name = "Homing",
80.                                    Description = "Reset the Position",
81.                                    Status = "Running",
82.                               ProgramSequence = new List<PlcCommand>()
83.                             {
84.
85.                                    new PlcCommand()
86.                                    {
87.
88.                         // Following sequence should be interpreted as:
89.                 //ModbusIpMaster master = ModbusIpMaster.CreateIp("127.0.0.1", 502);
90.                                        CommandName = Commands.ConnectToSlave,
91.                                        Args = new Dictionary<string, object>()
92.                                        {
93.                                            {"Address", "10.2.3.217" },
94.                                                {"Port", 502 },
95.                                        },
96.                                    },
97.
98.                                    new PlcCommand()
99.                                    {
100.                              // Following sequence should be interpreted as:
101.                               //  master.WriteSingleCoil(16390, false); //move arm in
102.                                       CommandName = Commands.MoveArm,
103.                                       Args = new Dictionary<string, object>()
104.                                       {
105.                                           {"Direction", false }//In,
106.                                       },
107.                                   },
108.
109.                                   new PlcCommand()
110.                                   {
111.                              // Following sequence should be interpreted as:
112.                              //  master.WriteSingleCoil(16389, false); //claw open
113.                                       CommandName = Commands.MoveClaw,
114.                                       Args = new Dictionary<string, object>()
115.                                       {
116.                                           {"Condition", false}//Open
117.                                       },
118.                                   },
119.
120.                                   new PlcCommand()
121.                                   {
122.                            // Following sequence should be interpreted as:
123.                    // movetooverposition(master, InshelfNr);//move over begin shelf
124.                                       CommandName = Commands.MoveToPosition,
```

```
125.                                        Args = new Dictionary<string, object>()
126.                                       {
127.                                            {"shelfNr", 19 },
128.                                       },
129.                                   },
130.
131.                               },
132.                               },
```

47 Programmcode Json Bibliotheken und Lager mit Sequenz

JSON Kommando Beispiel Lagern

This part of the program (see 48 Program code Json command page removal) is assigned to the Festo storage station. It contains a command example for the operating sequence when an object is to be retrieved.

```
334.            new PlcProgram()
 335.                   {
 336.                                       Name = "Auslagern",
 337.                                       Description = "Clean up storage 5",
 338.                                       Status = "Running",
 339.                                       ProgramSequence = new List<PlcCommand>()
 340.
 341.                       {
 342.                                   new PlcCommand()
 343.                           {
 344.                                  // Following sequence should be interpreted as:
345.           //ModbusIpMaster master = ModbusIpMaster.CreateIp("127.0.0.1", 502);
 346.                                   CommandName= Commands.ConnectToSlave,
 347.                                    Args = newDictionary<string, object>()
 348.                                   {
 349.                                       {"Address", "10.2.3.217" },
 350.                                        {"Port", 502 },
 351.                                   },
 352.                           },
 353.
 354.                            new PlcCommand()
 355.                           {
 356.                                 // Followingsequence should be interpreted as:
 357.                  // master.WriteSingleCoil(16390, false); //move arm in
 358.                                   CommandName= Commands.MoveArm,
 359.                                    Args = newDictionary<string, object>()
 360.                                   {
 361.                                       {"Direction", false }//In,
 362.                                   },
 363.                           },
 364.
 365.                            new PlcCommand()
 366.                           {
 367.                                 // Followingsequence should be interpreted as:
 368.                  //  master.WriteSingleCoil(16389, false); //claw open
 369.                                   CommandName= Commands.MoveClaw,
 370.                                    Args = newDictionary<string, object>()
```

```
371.                                {
372.                                    {"Condition", false}//Open
373.                                },
374.                            },
375.
376.                            new PlcCommand()
377.                            {
378.                                // Following sequence should be interpreted as:
379.                // movetooverposition(master, InshelfNr);//move over begin shelf
380.                                CommandName = Commands.MoveToOverPosition,
381.                                  Args = new Dictionary<string, object>()
382.                                {
383.                                    {"shelfNr", 5 },
384.                                },
385.                            },
386.
387.                             new PlcCommand()
388.                            {
389.                                // Following sequence should be interpreted as:
390.                                // armout(master);
391.                                CommandName = Commands.MoveArm,
392.                                  Args = new Dictionary<string, object>()
393.                                {
394.                                    {"Direction", true },//Out
395.                                },
396.                            },
397.                           new PlcCommand()
398.                            {
399.                                // Following sequence should be interpreted as:
400.                    // movetoposition(master, InshelfNr);//move to begin shelf
401.                                CommandName = Commands.MoveToPosition,
402.                                  Args = new Dictionary<string, object>()
403.                                {
404.                                    {"shelfNr", 5 },
405.                                },
406.                            },
407.                           new PlcCommand()
408.                            {
409.                                // Following sequence should be interpreted as:
410.                    //   master.WriteSingleCoil(16389, true); //claw close
411.                                CommandName = Commands.MoveClaw,
412.                                  Args = new Dictionary<string, object>()
413.                                {
414.                                    {"Condition", true}//Close
415.                                },
416.                            },
417.                          new PlcCommand()
418.                            {
419.                                // Following sequence should be interpreted as:
420.                //movetooverposition(master, InshelfNr);//move over begin shelf
421.                                CommandName = Commands.MoveToOverPosition,
422.                                 Args = new Dictionary<string, object>()
423.                                {
424.                                    {"shelfNr", 5 },
425.                                },
426.                            },
427.                          new PlcCommand()
428.                            {
429.                                // Following sequence should be interpreted as:
```

```
                            // armin(master);
                             CommandName = Commands.MoveArm,
                              Args = new Dictionary<string, object>()
                              {
                                  {"Direction", false},//In
                              },
                        },
                    new PlcCommand()
                        {
                            // Following sequence should be interpreted as:
        //movetooverposition(master, OutshelfNr);//move over target shelf
                             CommandName = Commands.MoveToOverPosition,
                              Args = new Dictionary<string, object>()
                              {
                                  {"shelfNr", 0 },
                              },
                        },
                    new PlcCommand()
                        {
                            // Following sequence should be interpreted as:
                            // armout(master);
                             CommandName = Commands.MoveArm,
                              Args = new Dictionary<string, object>()
                              {
                                  {"Direction", true },//Out
                              },
                        },
                    new PlcCommand()
                        {
                            // Following sequence should be interpreted as:
          // movetoposition(master, OutshelfNr);//move to target shelf
                             CommandName = Commands.MoveToPosition,
                              Args = new Dictionary<string, object>()
                              {
                                  {"shelfNr", 0 },
                              },
                        },
                    new PlcCommand()
                        {
                            // Following sequence should be interpreted as:
             //   master.WriteSingleCoil(16389, false); //claw open
                             CommandName = Commands.MoveClaw,
                              Args = new Dictionary<string, object>()
                              {
                                  {"Condition", false}//Open
                              },
                        },
                    new PlcCommand()
                        {
                            // Following sequence should be interpreted as:
        // movetooverposition(master, OutshelfNr);//move over target shelf
                             CommandName = Commands.MoveToOverPosition,
                              Args = new Dictionary<string, object>()
                              {
                                  {"shelfNr", 0 },
                              },
                        },
                     new PlcCommand()
                        {
```

```
489.                            // Following sequence should be interpreted as:
490.                            // armin(master);
491.                              CommandName = Commands.MoveArm,
492.                              Args = new Dictionary<string, object>()
493.                              {
494.                                   {"Direction", false },//In
495.                              },
496.                         },
497.
498.                              },
499.                              },
```

48 Programmcode Json Kommando Auslagern

JSON Kommando Bearbeiten

This part of the program (see 49 Program code Json command Edit with sequence) is assigned to the Festo machining station. It contains a command example for the operating sequence when an object is to be tested and drilled.

```
6016.                          new Device()
6017.                           {
6018.                              DeviceId = "FESTOBEARBEITEN01",
6019.                               Description = "Festo Bearbeitungs System",
6020.                               Location = "Frankfurt University of Applied
                                                                    Science",
6021.
6022.                              ProgramList = new List<PlcProgram>()
6023.                              {
6024.                                 new PlcProgram()
6025.                 {
6026.                                        Name = "One Workpiece",
6027.                                     Description = "Program for only one
                                                                  Workpiece",
6028.                                     Status = "Running",
6029.                       ProgramSequence = new List<PlcCommand>()
6030.                      {
6031.
6032.                                   new PlcCommand()
6033.                                   {
6034.
6035.
6036.
6037.                                     CommandName = Commands.ConnectToSlave,
6038.                                     Args = new Dictionary<string, object>()
6039.                                     {
6040.                                         {"Address", "10.2.3.217" },
6041.                                            {"Port", 502 },
6042.                                     },
6043.                                   },
6044.                                   new PlcCommand()
6045.                                   {
6046.                                     CommandName = Commands.MoveTable,
6047.                                     Args = new Dictionary<string, object>()
6048.                                       {
6049.                                          {"Table", true },//In
```

```
6050.                                   },
6051.                               },
6052.                               new PlcCommand()
6053.                               {
6054.                                   CommandName = Commands.MoveTest,
6055.                                   Args = new Dictionary<string, object>()
6056.                                     {
6057.                                        {"Test", true },//In
6058.                                     },
6059.                               },
6060.                               new PlcCommand()
6061.                               {
6062.                                   CommandName = Commands.MoveTest,
6063.                                   Args = new Dictionary<string, object>()
6064.                                     {
6065.                                        {"Test", false },//In
6066.                                     },
6067.                               },
6068.                               new PlcCommand()
6069.                               {
6070.                                   CommandName = Commands.MoveTable,
6071.                                   Args = new Dictionary<string, object>()
6072.                                     {
6073.                                        {"Table", true },//In
6074.                                     },
6075.                               },
6076.                               new PlcCommand()
6077.                               {
6078.                                   CommandName = Commands.MoveTight,
6079.                                   Args = new Dictionary<string, object>()
6080.                                     {
6081.                                        {"Tight", true },//In
6082.                                     },
6083.                               },
6084.                               new PlcCommand()
6085.                               {
6086.                                   CommandName = Commands.MoveDrill,
6087.                                   Args = new Dictionary<string, object>()
6088.                                     {
6089.                                        {"Drill", true },//In
6090.                                     },
6091.                               },
6092.                               new PlcCommand()
6093.                               {
6094.                                   CommandName = Commands.MoveDrill,
6095.                                   Args = new Dictionary<string, object>()
6096.                                     {
6097.                                        {"Drill", false },//In
6098.                                     },
6099.                               },
6100.                               new PlcCommand()
6101.                               {
6102.                                   CommandName = Commands.MoveTight,
6103.                                   Args = new Dictionary<string, object>()
6104.                                     {
6105.                                        {"Tight", false },//In
6106.                                     },
6107.                               },
6108.                               new PlcCommand()
```

```
6109.                                   {
6110.                                       CommandName = Commands.MoveTable,
6111.                                       Args = new Dictionary<string, object>()
6112.                                         {
6113.                                            {"Table", true },//In
6114.                                         },
6115.                                   },
6116.                                   new PlcCommand()
6117.                                   {
6118.                                       CommandName = Commands.MoveTable,
6119.                                       Args = new Dictionary<string, object>()
6120.                                         {
6121.                                            {"Table", true },//In
6122.                                         },
6123.                                   },
6124.                                   new PlcCommand()
6125.                                   {
6126.                                       CommandName = Commands.MoveTable,
6127.                                       Args = new Dictionary<string, object>()
6128.                                         {
6129.                                            {"Table", true },//In
6130.                                         },
6131.                                   },
6132.                                   new PlcCommand()
6133.                                   {
6134.                                       CommandName = Commands.MoveTable,
6135.                                       Args = new Dictionary<string, object>()
6136.                                         {
6137.                                            {"Table", true },//In
6138.                                         },
6139.                                   },
6140.
6141.
6142.                               },
6143.                           },
6144.                       }
6145.                   }
```

49 Programmcode Json Kommando Bearbeiten mit Sequenz

Festo Lager Motoren

The following figure shows an overview of the project: Warehouse20131001 (see Figure 29 Festo warehouse project setting). All components can be configured here. This software (Festo Configuration Tool FCT) is provided by Festo for configuration purposes..

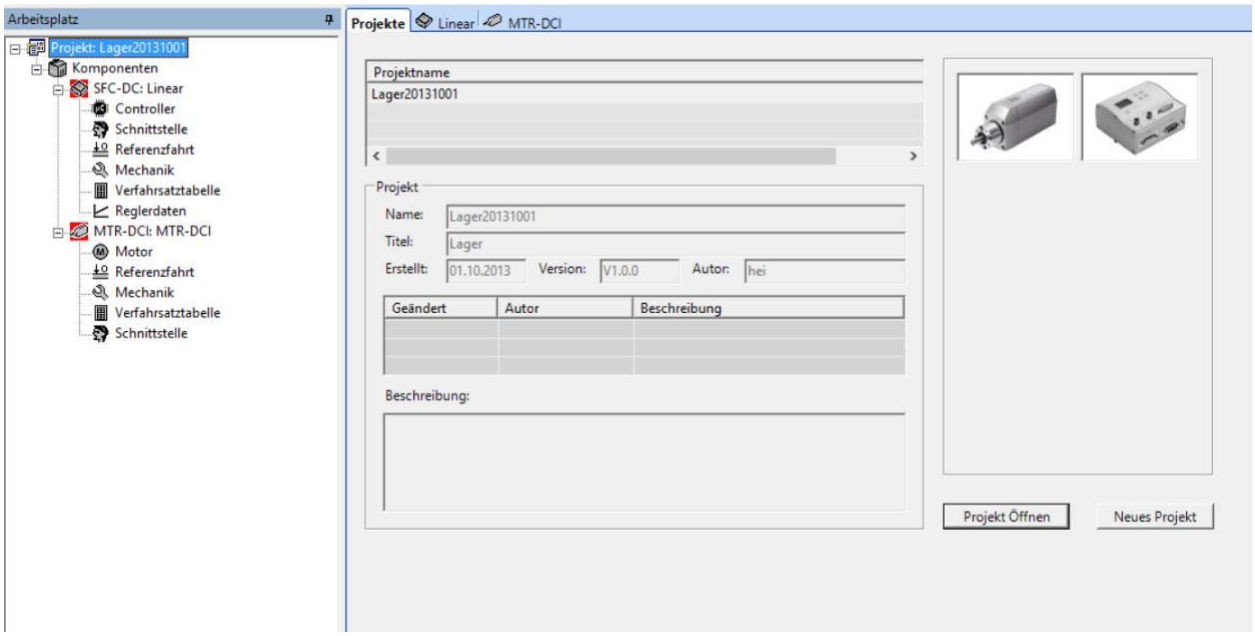


Abbildung 29 Festo Lager Projekt Einstellung

The following figure shows how to configure the controller type "SFC-DC-VC-3-E-H2-IO" (see Figure 30 Festo bearing linear controller setting). This is responsible for the linear motor.

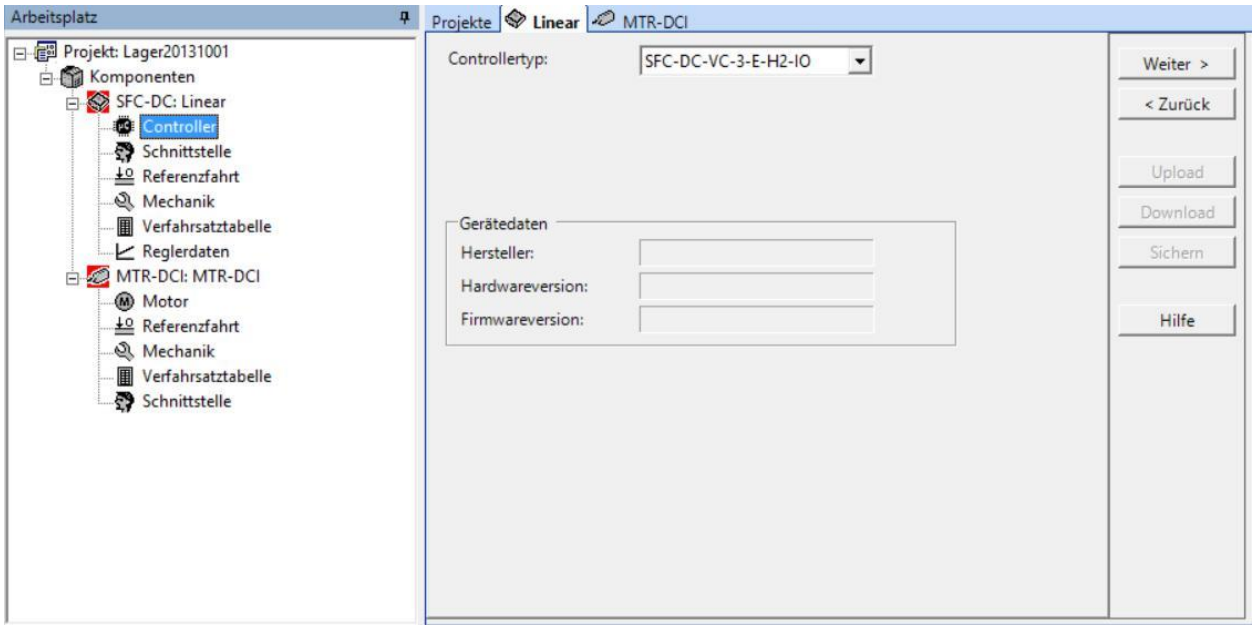


Abbildung 30 Festo Lager Linear Controller Einstellung

The following diagram shows which interface "IO Multipol I/O" is used (see Figure 31 Festo bearing linear interface setting).

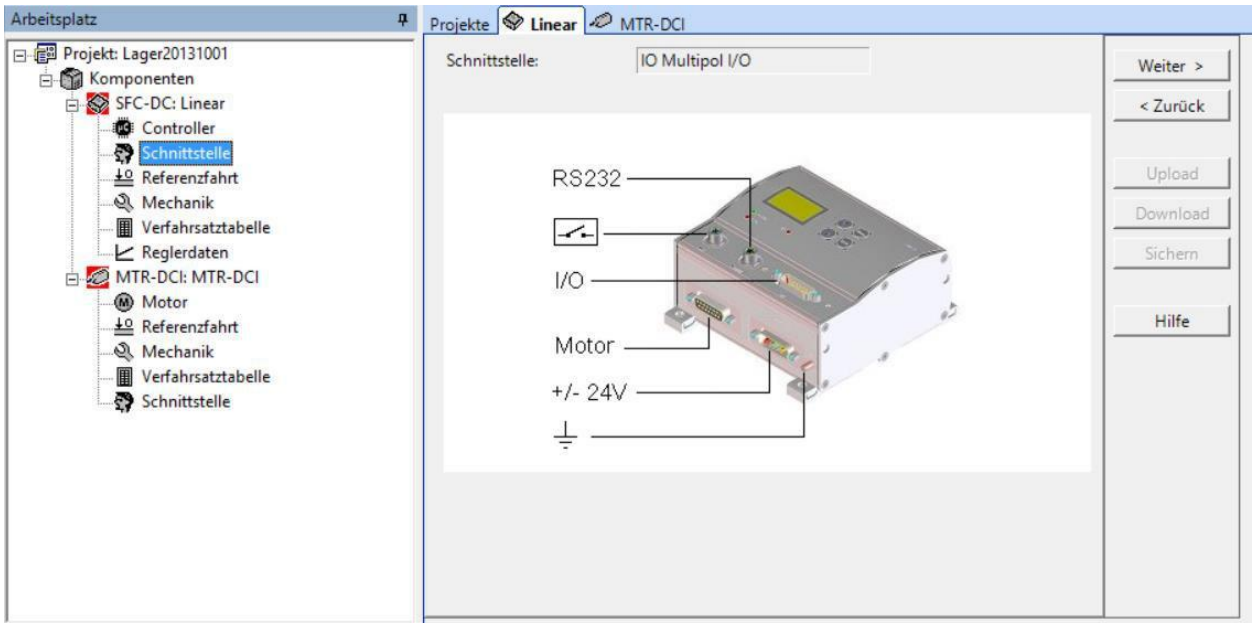


Abbildung 31 Festo Lager Linear Schnittstelle Einstellung

There are many configuration options for homing under the Homing item (see Figure 32 Festo Bearing Linear homing setting). The "Ref. method" is the way in which the homing movement is to determine the values for the motor. The "axis zero point" describes the minimum distance of the extended arm. The two parameters "Ref. speed. V_rp" and "Ref. speed". V_zp" describe the traversing speeds for searching the reference point and the traversing speeds for approaching the axis zero point. The parameter "Relative motor current" defines the relative current value for the motor

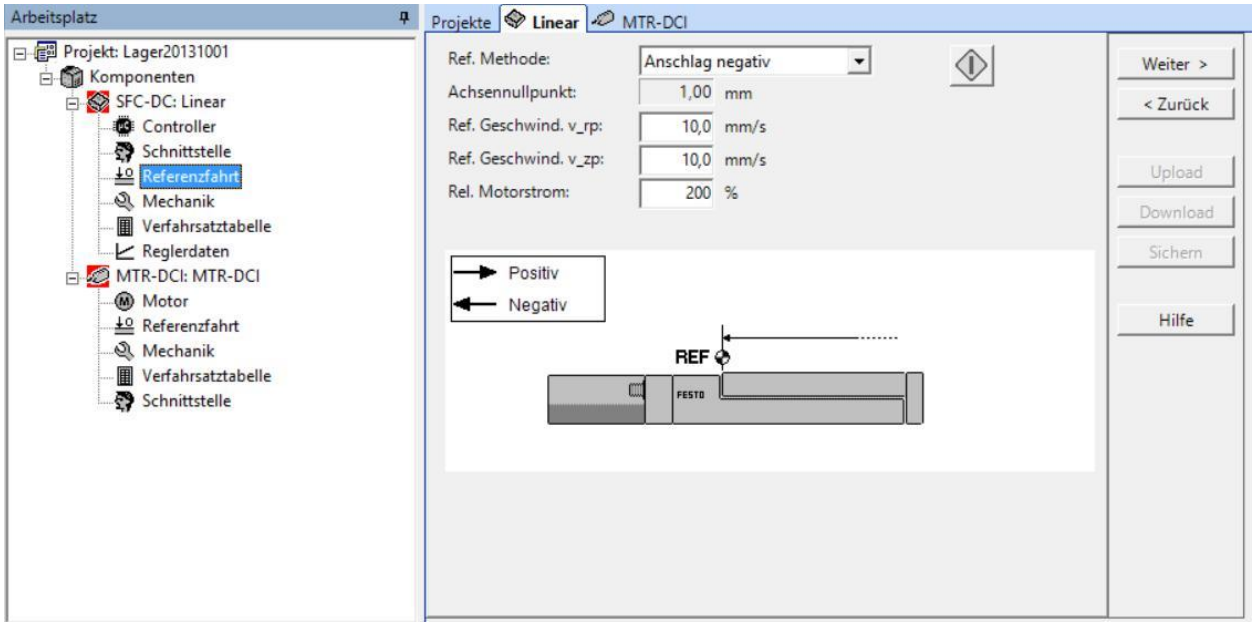


Abbildung 32 Festo Lager Linear Referenzfahrt Einstellung

The following figure shows how the mechanics can be configured with various parameters (see Figure 33 Festo Bearing Linear Mechanics Adjustment). The item "Type of axis" describes the type of axis used "Festo Mini slide SLTE". This includes the "size" of the axis type "SLTE-16-150-LS7.5-DC-VCS". The parameter "Working stroke" describes the maximum extendable distance of the arm. The point "Axis zero point" describes the minimum extendable distance. The parameters "SW end position pos" and "SW end position neg" move within the range defined in the "Useful stroke" point. The point "Gear factor" describes the transmission ratio of the gearbox. The parameter "Feed constant" describes the distance covered by the arm during one revolution of the motor.

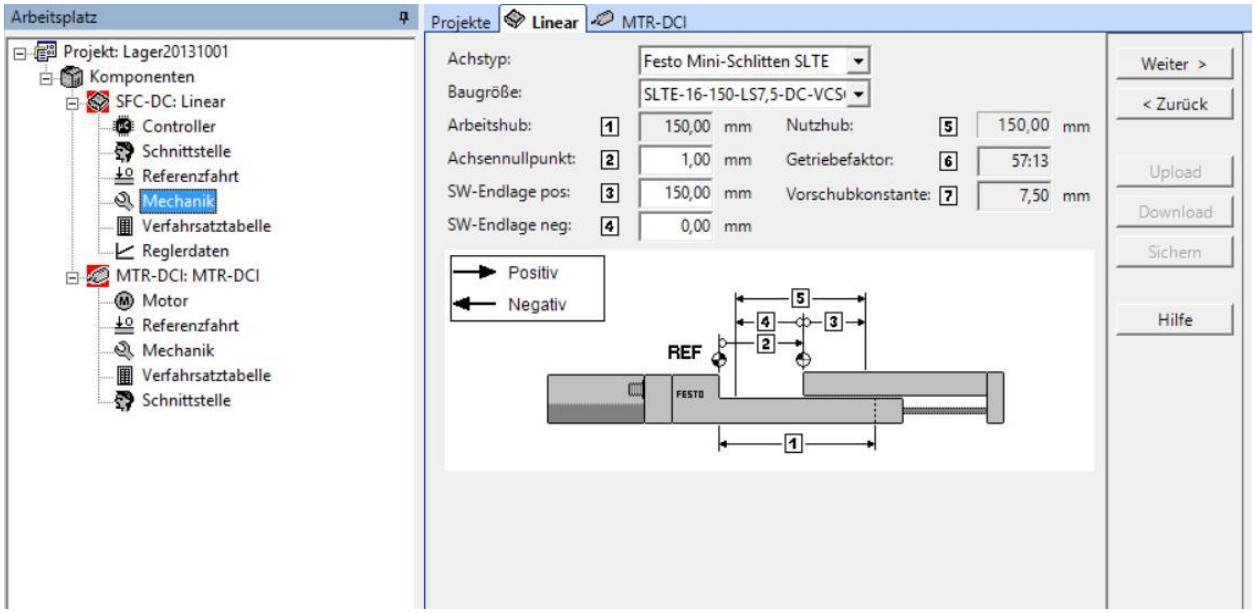


Abbildung 33 Festo Lager Linear Mechanik Einstellung

The travel set table is shown below (see Figure 34 Festo bearing Linear travel set table Adjustment). This table contains the positions and speeds for the respective compartment. Note that this is the linear motor that controls the vertical movement. Each rack row is backed with 2 vertical positions..

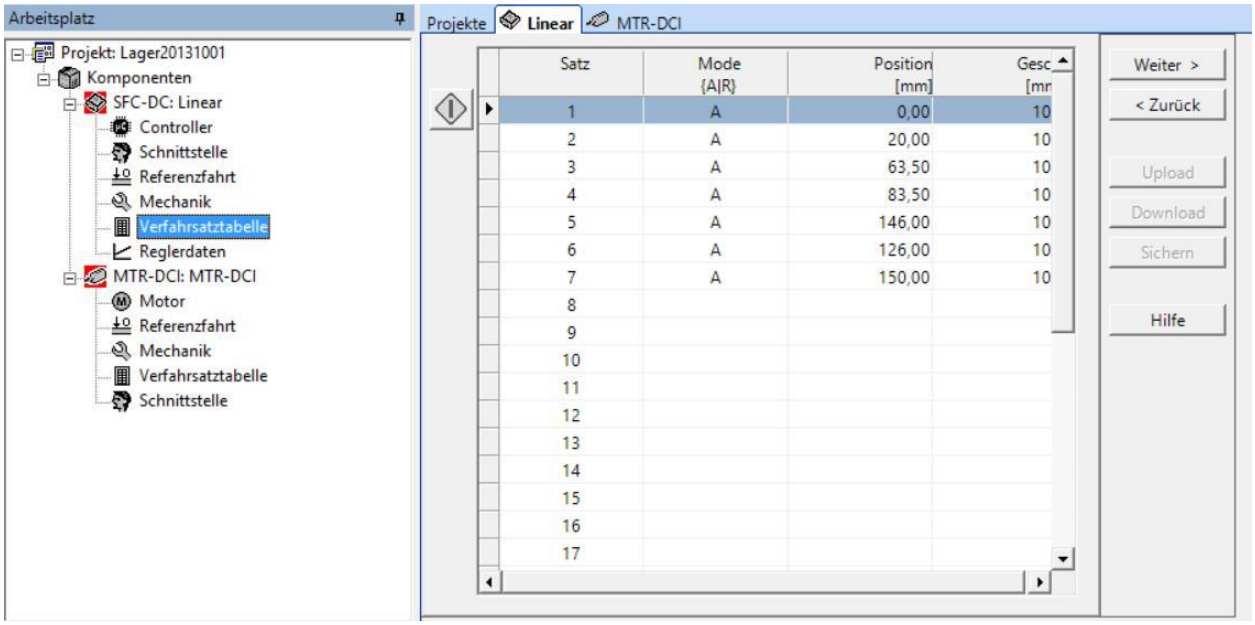


Abbildung 34 Festo Lager Linear Verfahrsatztabelle Einstellung

Im Folgenden werden die Reglerdaten definiert (siehe Abbildung 35 Festo Lager Linear Reglerdaten Einstellung). Diese werden bei den Standardeinstellungen belassen.

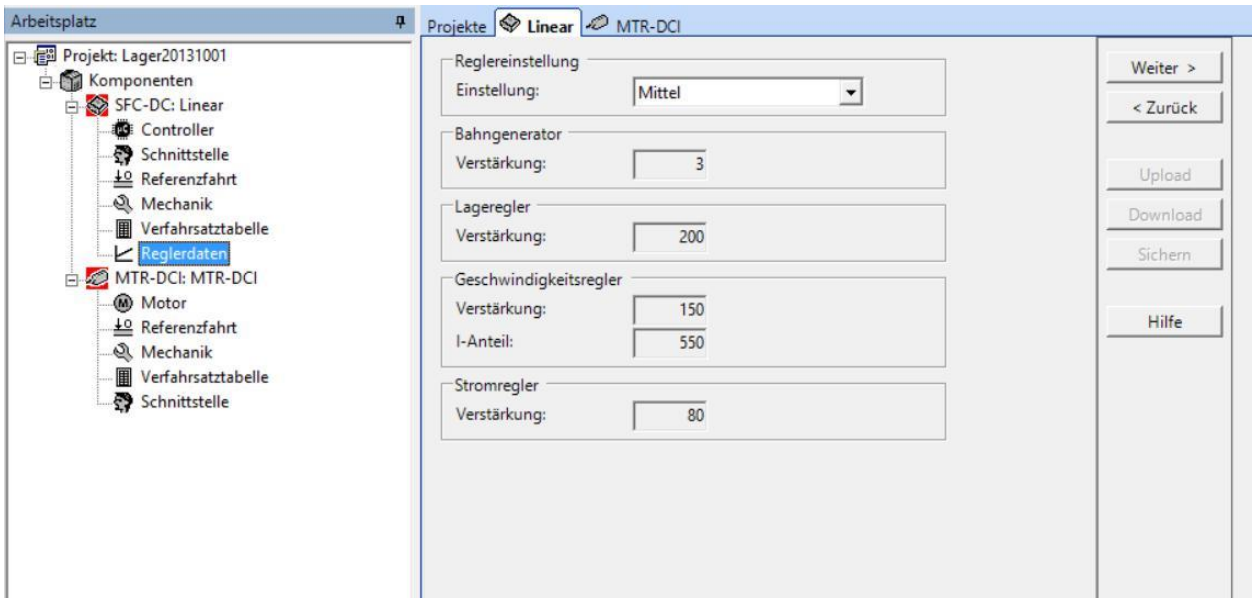


Abbildung 35 Festo Lager Linear Reglerdaten Einstellung

The following figure shows the configuration for the rotary motor (see Figure 36 Festo bearing rotation motor setting). The "Motor type" parameter contains the "MTR-DCI-42x-G14-y2IO" motor used. The point "Size" contains the used size of the motor. The "Gear" parameter contains the type of gear used. In the item "Interface" the used interface has to be configured.

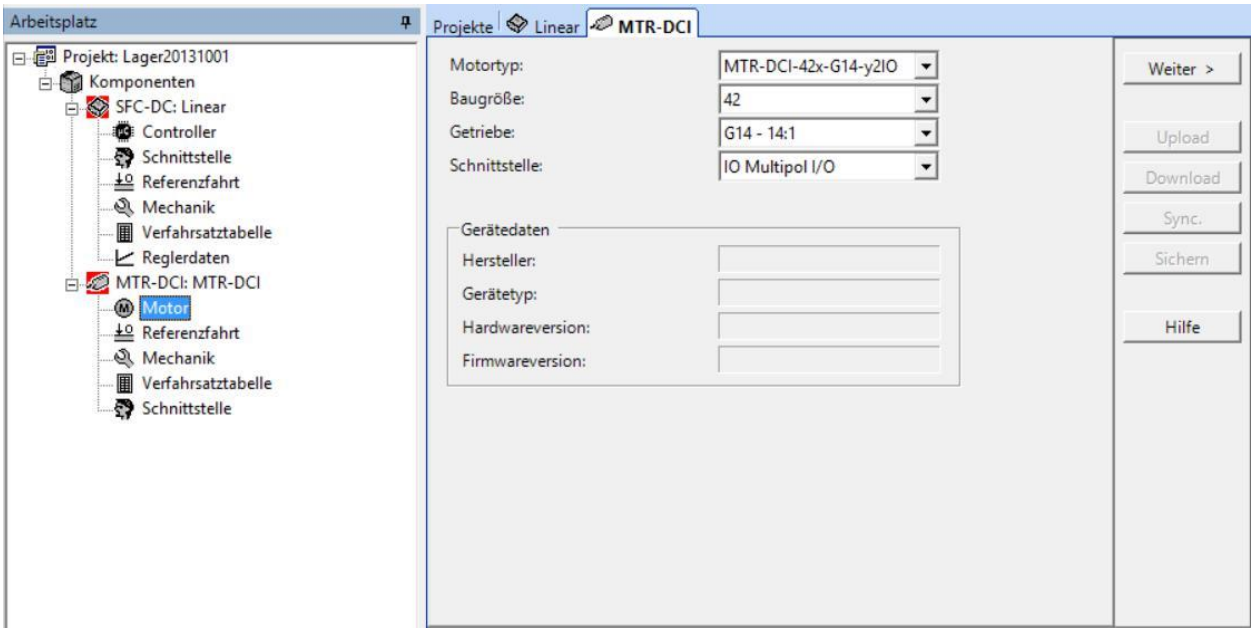


Abbildung 36 Festo Lager Rotation Motor Einstellung

The following figure shows the configuration of the reference run for the rotary motor (see Figure 37 Festo bearing rotation reference run setting). In the "Ref. method" parameter, the "Ref. switch positive" method used for homing is configured. The "axis zero point" is the minimum rotatable distance. The two parameters "Ref. speed. V_rp" and "Ref. speed". V_zp" describe the traversing speeds for searching the reference point and the traversing speeds for approaching the axis zero point. The parameter "Relative motor current" defines the relative current value for the motor.

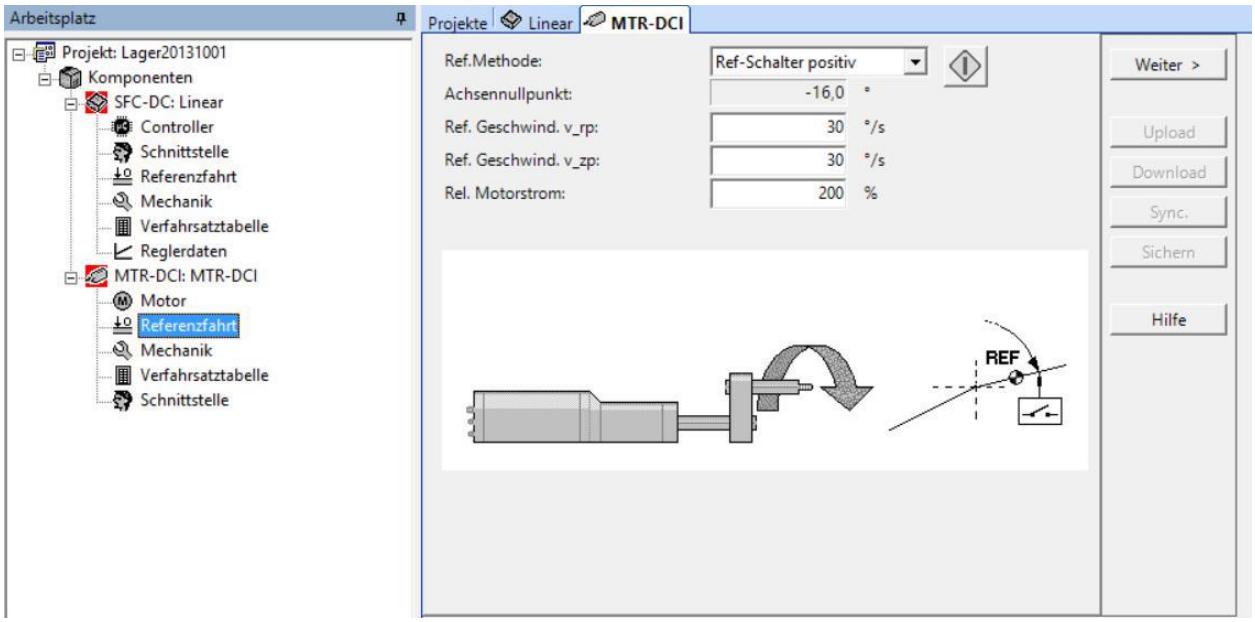


Abbildung 37 Festo Lager Rotation Referenzfahrt Einstellung

The following figure shows configurations for adjusting the rotary motor (see Figure 38 Festo Bearing Rotation Mechanical Adjustment). The "Axis type" parameter describes the axis type used. The "Working stroke" parameter describes the maximum swivel distance of the arm. The "Axis zero point" describes the minimum distance of the rotated arm. The parameters "SW end position pos" and "SW end position neg" move within the range defined in point "Useful stroke". The point "Gear factor" describes the transmission ratio of the gear. The parameter "Feed constant" describes the distance covered by the arm during one revolution of the motor.

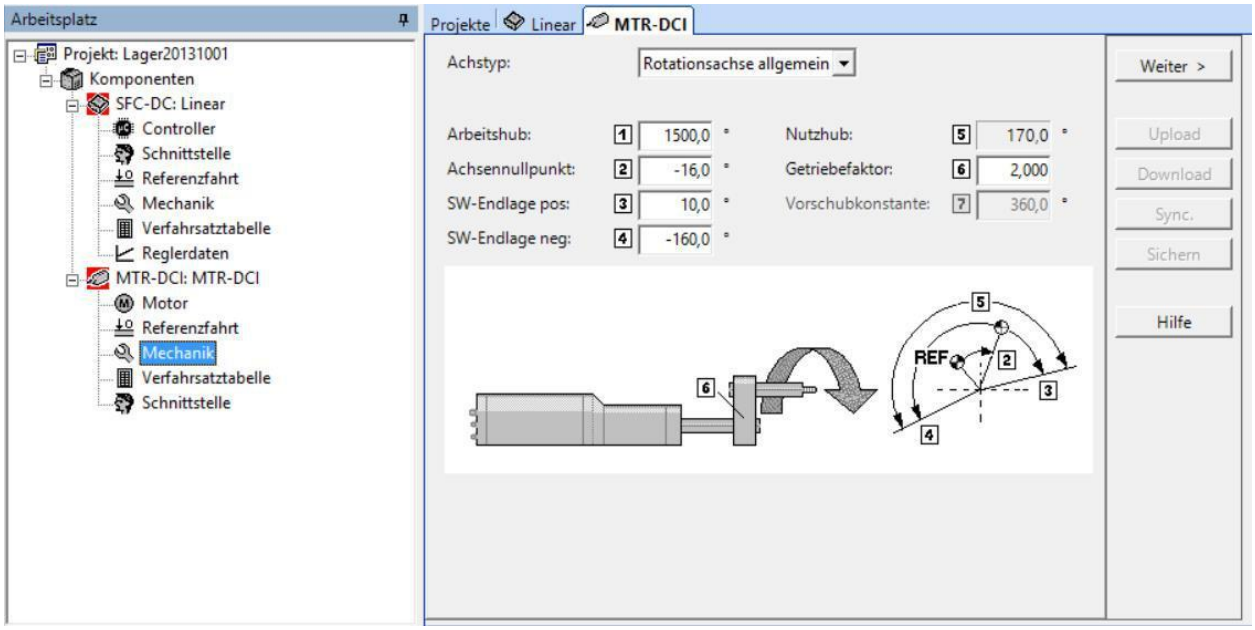


Abbildung 38 Festo Lager Rotation Mechanik Einstellung

The travel set table is shown below (see Figure 39 Festo bearing rotation travel set table setting). This table shows the positions and speeds for each compartment. Please note that this is the rotary motor which controls the horizontal movement. Each shelf column is stored with 1 horizontal position.

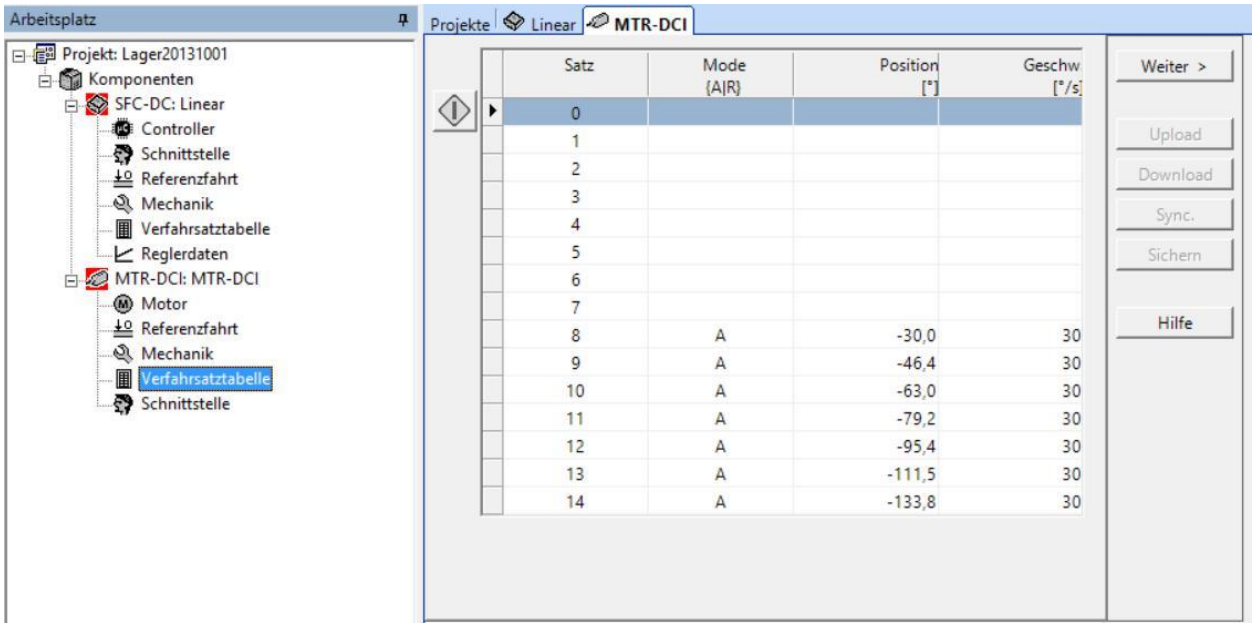


Abbildung 39 Festo Lager Rotation Verfahrsatztabelle Einstellung

The following figure shows the interface of the rotary motor (see Figure 40 Festo bearing rotation interface setting).

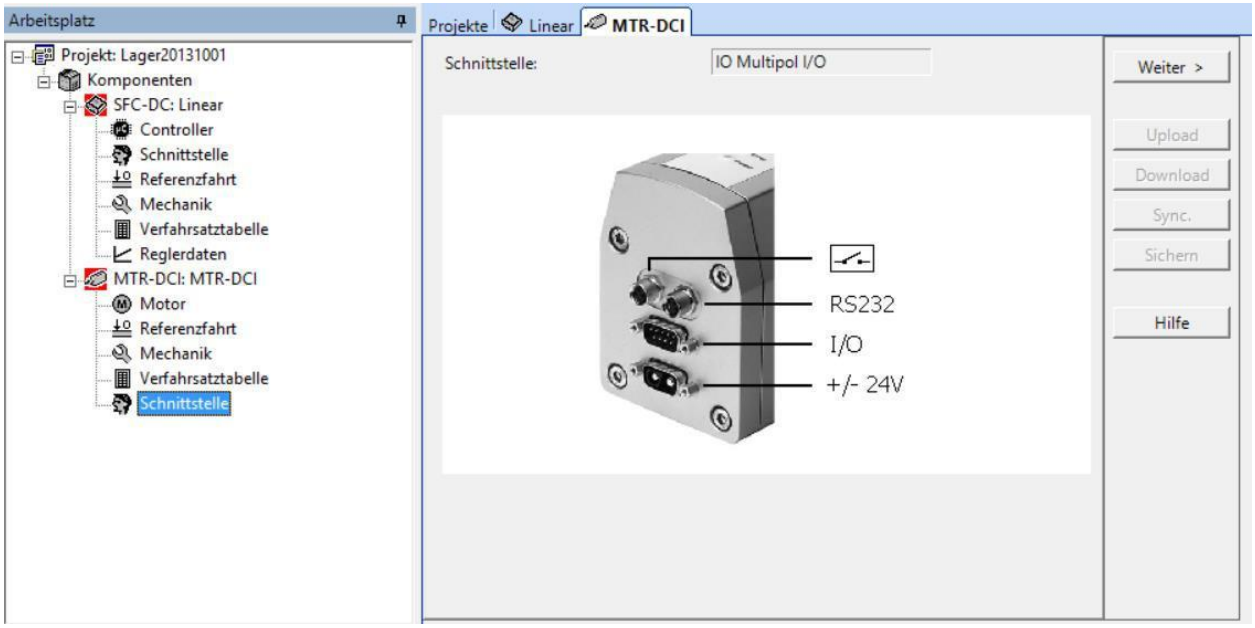


Abbildung 40 Festo Lager Rotation Schnittstelle Einstellung

Anwendung

The following figure shows the Web Dashboard in development (see Figure 41 Web Dashboard). The tachometer-shaped display shows the current status as a percentage during an operation. The bars to the right show the usage frequency of the respective machine parts. To get information if a part has to be replaced from a maintenance point of view. The point "machines" enables the control of different machines. The picture shows the selected FESTOLAGER01. The item "programs" lists the possible pre-programmed sequences which the user can select to execute.

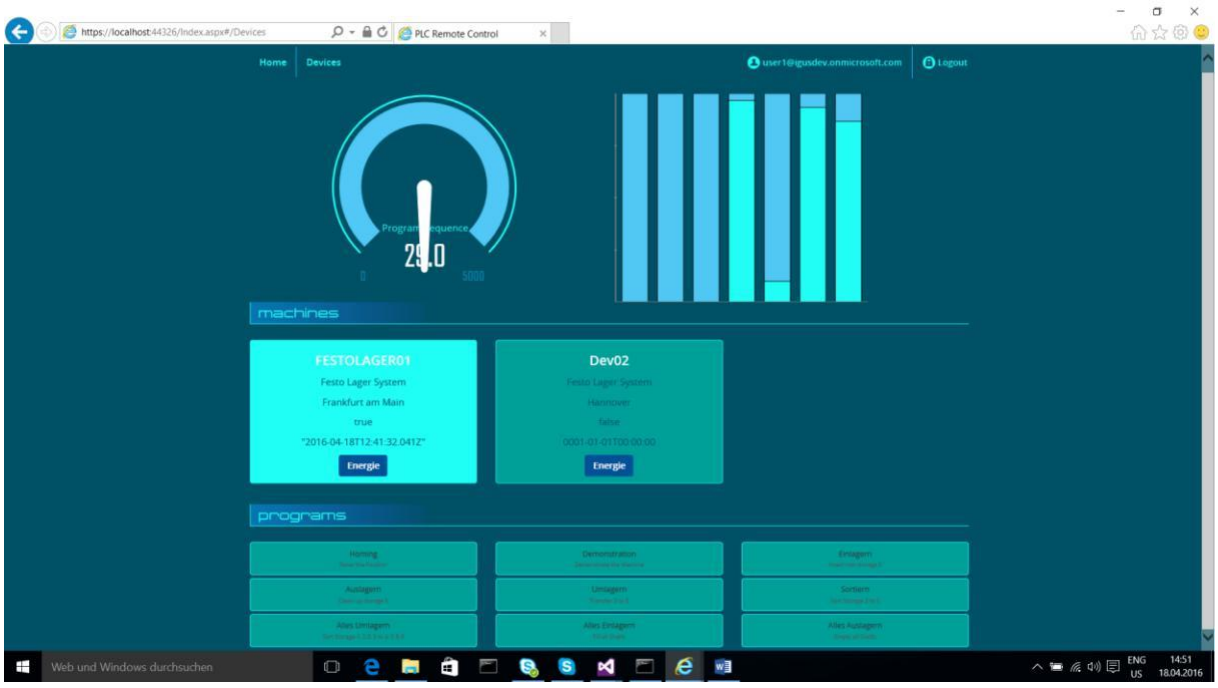


Abbildung 41 Web Dashboard

The following figure shows the possibility to change the executable Json code. With this the program flow in the web interface can be adapted (see Figure 42 Web Json Commands Version 1).

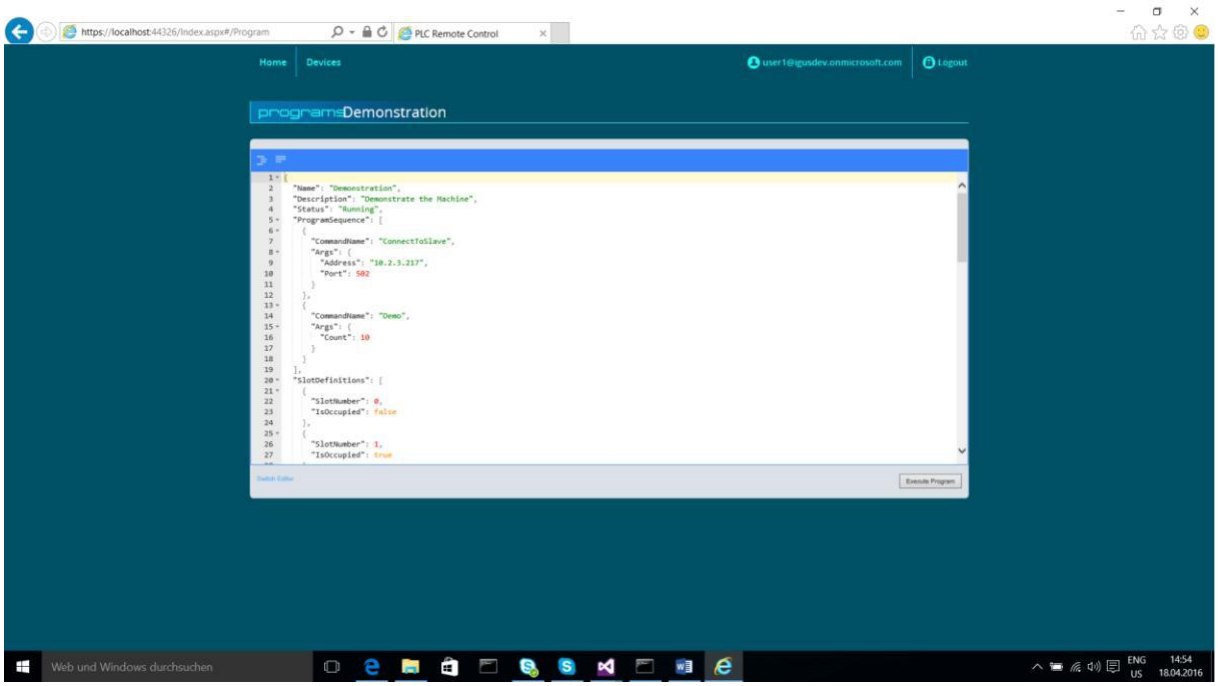


Abbildung 42 Web Json Kommandos Version 1

The following figure shows the possibility to change only selected parameters in the Json code. This way the program flow in the web interface can be adapted (see Figure 43 Web Json Commands Version 2). The white checkbox on the ISOccupied can be changed, the rest of the source code cannot be changed.

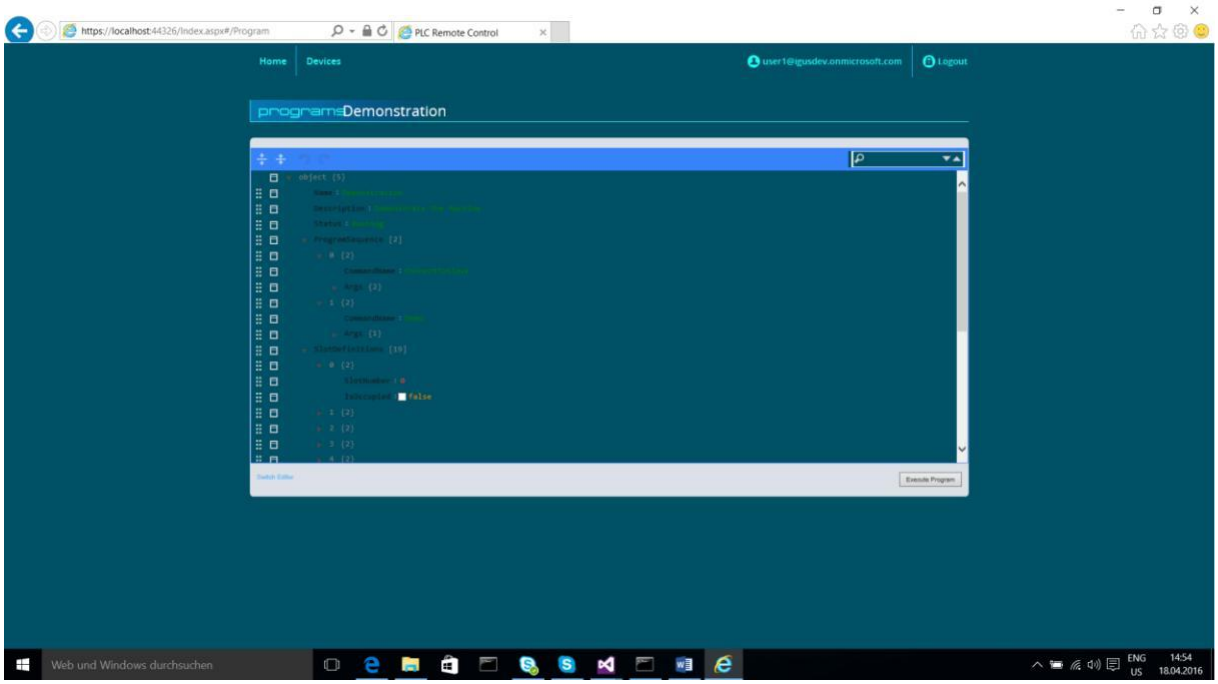


Abbildung 43 Web Json Kommandos Version 2

The following figure shows the status display of the warehouse in the gateway program (see Figure 44 Gateway application). Under the item "activity status" there is a button to start and stop the gateway. You can also use the button to delete the commands and stop the program. The illustrations for Storage, Shelf01, Shelf02 and Shelf03 visualize the status of the respective compartments. The item "log" shows the current status.

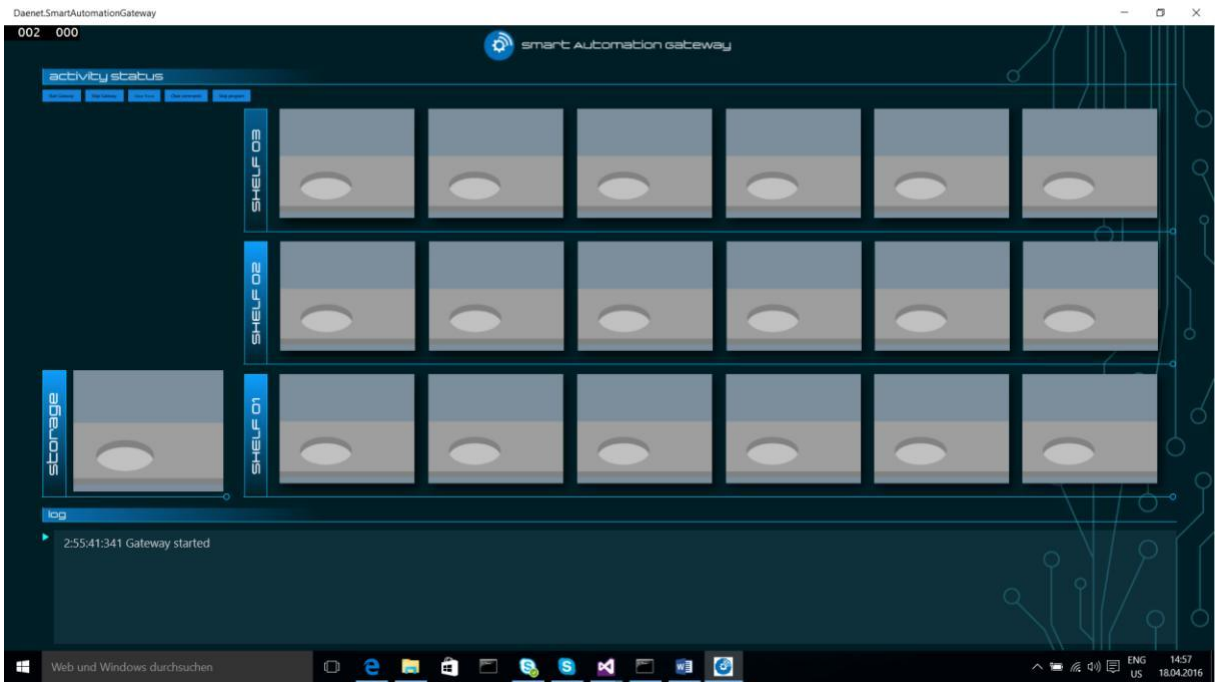


Abbildung 44 Gateway Anwendung

The following figure shows the entire structure of the Festo storage station with the B&R PLC (see Figure 45 Festo storage structure). The storage station is mounted on the cart. The PLC is located at the bottom inside the cart. This is connected to the storage station via the inputs and outputs. The green Ethernet cable is connected to the Raspberry Pi..

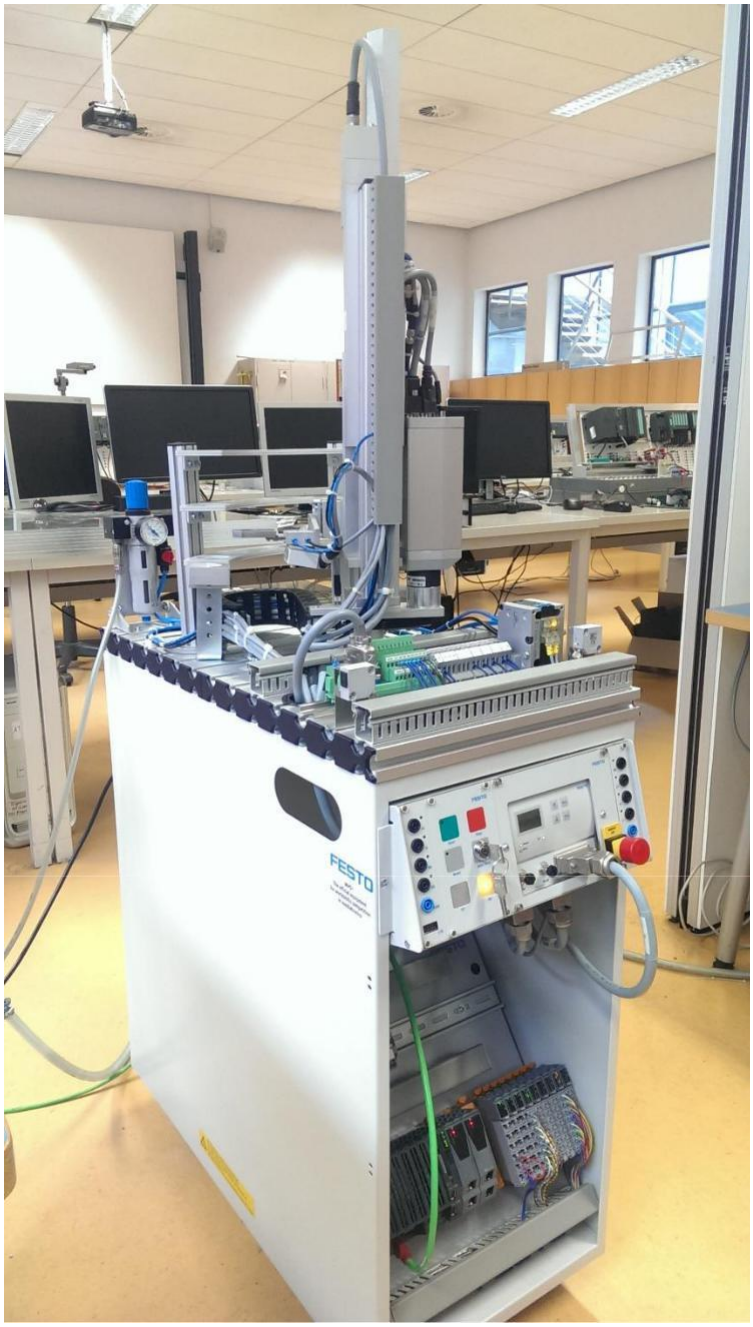


Abbildung 45 Festo Lager Aufbau

Fazit

The following results can be summarised. Due to the control of the Raspberry Pi, the technical implementation is no longer real-time capable. Because the Raspberry Pi, due to its microprocessor, does not allow cyclic work steps. This does not influence the working process as long as the microprocessor is not overloaded by parallel program sequences. An additional problem is the connection to the cloud. Since here the gateway cannot receive any more commands if there is no connection and feedback to the cloud is not possible. A good connection to the cloud is essential. This in particular offers great flexibility, since any device capable of web browsing can be used to control the storage station. By connecting to the Internet, information about the controlled machines can also be output with little effort. Thus, maintenance and logistics can be made more efficient, as they can directly receive information about the processes. The implementation represents a certain transitional solution, as the PLC system is extended by a cloud service. In order to avoid the high costs associated with a PLC system, it is also conceivable to replace it completely with a single board computer. The test setup would only have to be used to expand the inputs and outputs of the Raspberry Pi. A further addition would be to address the PLC systems using a different protocol in order to cover a larger product range.

Gewonnene Erkenntnisse

The in-depth project provides for a deepening of understanding in various programming languages as well as in communication technology in the field of cloud computing. The knowledge of the programming and information processing modules was expanded during the in-depth project. Since a practice-oriented experimental setup was implemented, much of the theory learned could be put into practice in a real environment. It was also possible to allow realistic periods of time to work on the tasks and this was always precisely defined. Good research often saves a lot of time. Therefore, you should always plan realistically and search for further goal-oriented alternatives.

Literaturverzeichnis

Im aktuellen Dokument sind keine Quellen vorhanden.

Abbildungsverzeichnis

Programmcodeverzeichnis