BAUMÜLLER

Bedienungsanleitung
Template Motion-Axis Web and Foil

Kurzbeschreibung:
Das Template Motion-Axis Web and Foil besteht aus einem ProMaster-Projekt, welches ein vorbereitetes PLC-Projekt, sowie ein HMI-Projekt beinhaltet. In der Grundkonfiguration ist eine Achse vorhanden, mit der das Template betrieben werden kann. Dies kann natürlich nach Belieben erweitert werden. Es sind hier eine Vielzahl von Funktionen vorbereitet, die als Grundlage eines vollständigen Applikationsprojektes für eine bahnführende Maschine geeignet sind. Ziel ist es, sich mit diesem Template auf die Programmierung des Prozesses konzentrieren zu können.

Version: 2.0 vom 28.04.2023

Status: Release

Autor: [VT]

1. Inhalt

2. Historie

Version	Datum	Name	Änderung
1.0	04.11.2021	VT	Initial Version
2.0	28.04.2023	VT	QR-Codes hinzugefügt

3. Einleitung

Dieses Dokument gibt einen Überblick über die vorhandenen Software-Funktionalitäten der Technologielösung Motion-Axis Web & Foil. Zur besseren Veranschaulichung werden Funktionen teilweise anhand von Abbildungen der Bedienoberfläche dargestellt und erklärt.
Technische Informationen zur Hardware sind in den entsprechenden Bedienungsanleitungen bzw. Parameterhandbüchern auf der Baumüller Website im Bereich Service\Downloads zu finden.

4. Begriffsdefinition

Begriff	Definition
HW	Hardware
POE	Programmorganisationseinheit
SPS	Speicherprogrammierbare Steuerung
Statemachine	Zustandsmaschine
TEQ	Designseite des HMI-Projektes

5. Sicherheitsrelevante Hinweise

GEFAHR	
	...weist auf eine unmittelbar gefährliche Situation hin, die zum Tod oder zu schweren Verletzungen führt, wenn sie nicht gemieden wird.

WARNUNG	
	... warnt vor einer möglicherweise gefährlichen Situation, die zum Tod oder zu schweren Verletzungen führen kann, wenn sie nicht gemieden wird

ACHTUNG	
	... warnt vor einer möglicherweise gefährlichen Situation, die zu leichten oder geringfügigen Verletzungen führen kann, wenn sie nicht gemieden wird

HINWEIS	
	... warnt vor einer möglicherweise gefährlichen Situation, die zu Sachschäden führen kann, wenn sie nicht gemieden wird

Notiz!	
	... weist auf nützliche Tipps und Empfehlungen sowie auf Informationen für einen effizienten, störungsfreien Betrieb hin

6. Voraussetzungen

Damit das Template betrieben werden kann, werden folgende Software-, und Hardwarekomponenten vorausgesetzt:

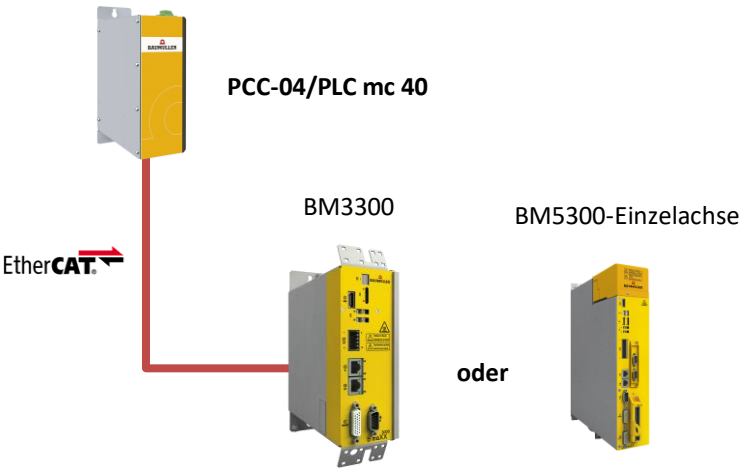
- SCADA-Editor (ab V7.02.00.20)
- ProMaster (ab 1.21.1.34 - Release)
- PCC-04/PLC mc 40
- bmaXX3300 oder bmaXX5000 als Einzelachse

7. Systemkomponenten

In diesem Abschnitt ist der Systemaufbau beschrieben.

7.1 Systemaufbau Basiskomponenten

Das Template in seiner Grundausstattung beinhaltet die Konfiguration für folgende Komponenten:

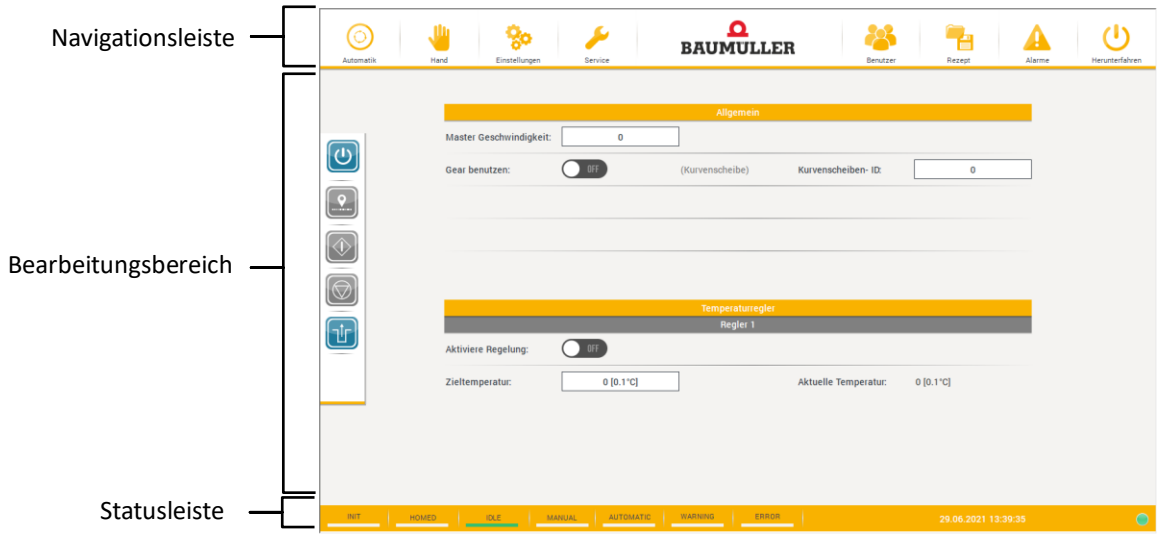


Mit dieser Systemkonfiguration wird das Template ausgeliefert und es wird empfohlen das Template vorerst mit diesen Basiskomponenten in Betrieb zu nehmen, um die grundlegenden Funktionen zu prüfen und kennenzulernen. Später kann das Template natürlich erweitert werden.

8. Überblick Visualisierung

8.1 Aufbau und Funktion

Jede Seite der Bedienoberfläche ist in 3 Teile gegliedert:

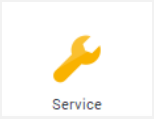


8.2 Navigationsleiste

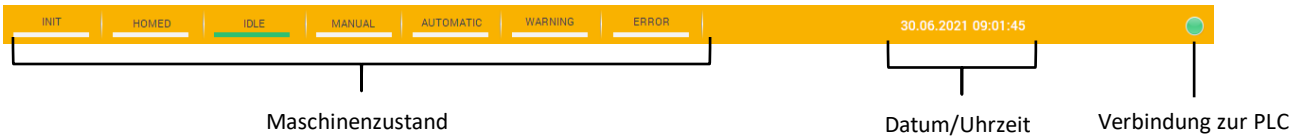
Am oberen Rand der Bedienoberfläche befindet sich die Navigationsleiste, über die man zu den einzelnen Maschinenfunktionen, wie z.B. Automatik- & Handbetrieb gelangt. Der Inhalt dieser Leiste ist immer gleich, unabhängig von der aufgerufenen Maschinenfunktion.

8.3 Funktionstasten

Symbol	Beschreibung
Automatik	**Automatikbetrieb:** Starten des virtuellen Masters und der Slave-Achse über ein Getriebe oder eine Kurvenscheibe. Anzeige des Temperaturreglers.
Hand	**Handbetrieb:** Verfahren der einzelnen Maschinenachsen von Hand
Einstellungen	**Einstellungen:** Einstellungen für z.B. Rezepte, Sprache, Nocken, Messtaster, Temperaturregler

	**Service:** Diagnosefunktionen der Maschine
	**Benutzer:** Login mit dem entsprechenden Passwort zum freischalten der unterschiedlichen Bedienerebenen
	**Rezept:** Hier können die Rezepte verwaltet werden
	**Fehlermanagement:** Alarmhistorie der einzelnen Fehlermeldungen mit Zeitstempel (Aufgetreten, Behoben)
	**Herunterfahren:** ermöglicht das Ausschalten bzw. Neustarten der Maschine

8.4 Statusleiste



Die Statusleiste gibt Aufschluss über den aktuellen Zustand der Maschine. Zudem wird das aktuelle Datum und die Uhrzeit angezeigt, sowie ob die Verbindung zur PLC aktiv ist.

8.5 Steuerleiste

Einschalten der Achse. Damit wird die Achse in den Zustand 4 gebracht

Referenzieren. Damit wird die Achse auf der aktuellen Position referenziert

Automatik Start. Die Achse folgt über einen elektronisches Getriebe oder einer Kurvenscheibe dem virtuellen Master

Automatik Stopp

Fehlerreset

8.6 Maschinenzustand

Der Maschinenzustand wird in der Statusleiste der Visualisierung angezeigt. Dabei gibt es folgende Zustände:

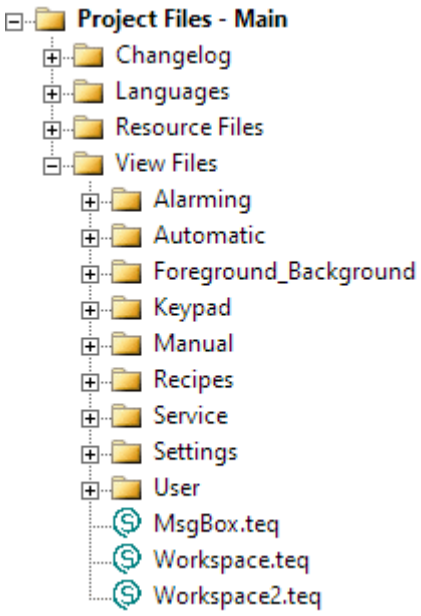
Status	Beschreibung
INIT	Die Zustandsmaschine befindet sich in der Initialisierung. Hier werden Initialwerte in der Steuerung gesetzt
HOMED	Zeigt an ob die Maschine referenziert ist
IDLE	Zeigt an ob die Maschine im Leerlauf ist. Aus diesem Zustand können Aktionen angestoßen werden, die eine Bewegung der Achse zur Folge haben.
MANUAL	Handbetrieb/Jog aktiv oder inaktiv
AUTOMATIC	Automatikmodus aktiv
WARNING	Es steht mindestens eine Warnung an
ERROR	Es steht mindestens ein Fehler an

8.7 Kommunikationsstatus

Symbol	Beschreibung
	**Status:** Die Kommunikation mit der Steuerung ist Offline. (Kein Datenausstauch)
	**Status:** Die Kommunikation mit der Steuerung ist Online (Datenaustausch aktiv)

9. HMI-Projekt

9.1 Inhalt und Komponenten

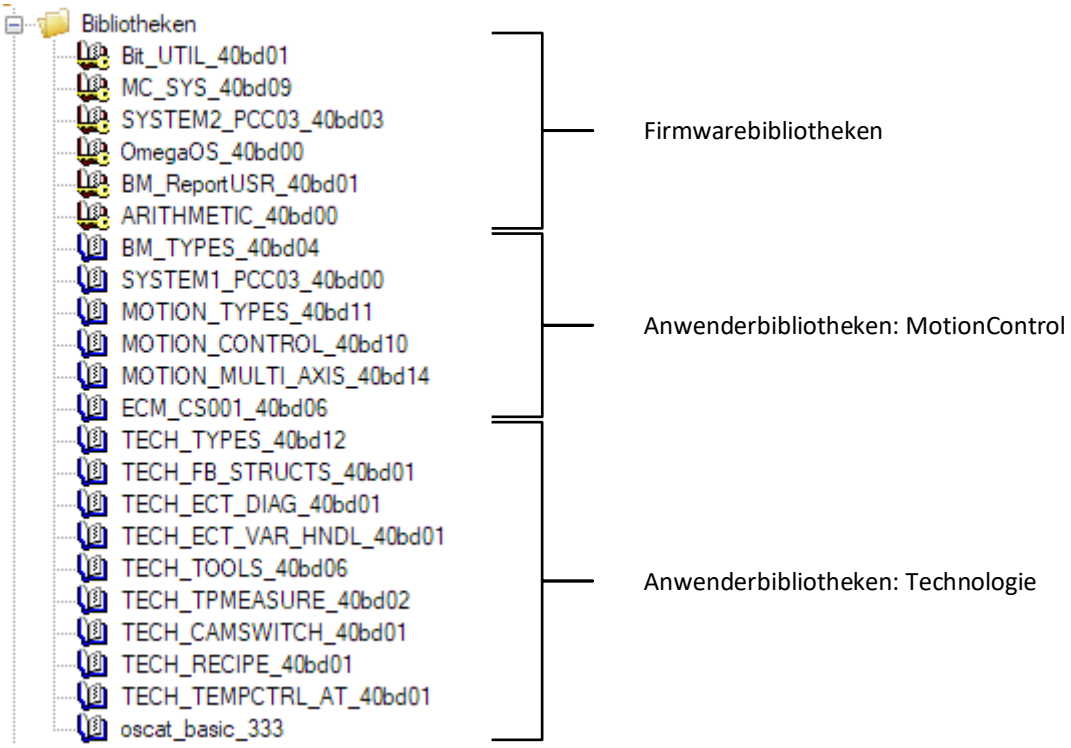


Verzeichnis	Kurzbeschreibung
+ Changelog	Hier kann für die Versionsverwaltung ein Changelog geführt werden
+ Language	Beinhaltet alle Sprachdateien
+ Resource Files	Beinhaltet programmspezifische Dateien, welche vom Benutzer nicht bearbeitet werden sollten!
- View Files	Hier sind alle Dateien hinterlegt, in welchen die Visualisierungsoberfläche gestaltet wird.
+ Alarming	Anzeige der Alarme/Fehlermeldungen
+ Automatic	Anzeige der Automatikseite
+ Foreground_Background	Vordergrund-, und Hintergrundseite der Visualisierung: Vordergrund => Hinweise, Meldungen, Steuerleiste Hintergrund => Navigationsleiste, Statusleiste
+ Keypad	Das Aussehen der Maus und Tastatur wird hier designed **Hinweis:** Diese Dateien werden vom Microbrowser nur verwendet, wenn auf dem Zielsystem im Verzeichnis des Microbrowsers **keine** „keypad.teq“ und keine „alphapad.teq“ vorhanden sind!
+ Manual	Anzeige der Seite für den Handbetrieb
+ Recipes	Anzeige der Rezeptverwaltung
+ Service	Views für die Diagnosefunktionen (EtherCAT, Antrieb, System)
+ Settings	Views für die Einstellungen (Nockenschaltwerk, Temperaturregelung)
+ User	Login-Seite für das Benutzerlevelmanagement
Workspace/2	Die hier angelegten Seiten können als „Zeichnungsfläche“ benutzt werden, wenn man einfach eine leere Zeichenfläche benötigt.

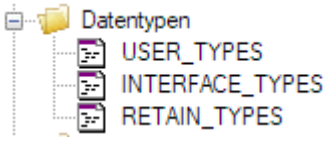
10. ProProg-Projekt

Notiz!	
	Es wird hier nicht auf jede einzelne Funktion der Funktionsbausteine innerhalb dieser POEs eingegangen. **Die einzelnen Funktionen können in der jeweiligen Bausteinhilfe nachgelesen werden**. Jedoch wird in den maschinenfunktionsspezifischen Kapiteln noch näher auf den Zusammenhang mit den einzelnen Bausteinen eingegangen.

10.1 Bibliotheken

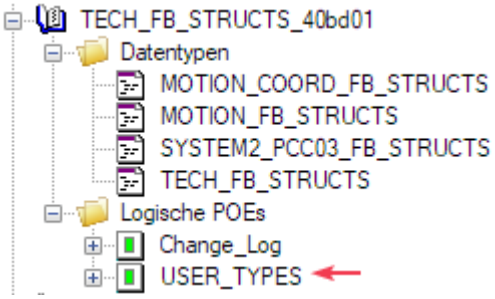


10.2 Datentypen



10.2.1 USER_TYPES

Die hier enthaltenen Datentypen sind aus der Bibliothek „TECH_FB_STRUCTS“ kopiert:



Dieses Datentyparbeitsblatt beinhaltet alle für den Benutzer notwendigen Datentypen, mit welchen er die Anzahl der einzelnen zu den Bausteinen zugehörigen Strukturdatentypen bestimmen kann. Die Anzahl ist in Form von Arrays gruppiert und kann flexibel geändert werden. Vereinfacht ausgedrückt heißt das folgendes:

Beispiel „TB_RecipeHandler“

Bausteininstanz 1 **=>** Verbindung mit Datentypstrukturarray 0

```
(* -- TECH_RECIPE ------------------------------------------------------------ *)
TB_RECIPE_ARRAY                         : ARRAY [0..1] OF TB_RECIPE_TMPL;
(* ====================================== *)
(* = TB_RecipeHandler
(* ====================================== *)

TB_RecipeHandler_1(
(*  IN   *)
(* =>|   *)Data               :=_File_TMPL.a_DataDummy[0],                    (* FILE_TMPL STRUCT *)
(* =>|   *)a_RecipeList       :=_Tech_TMPL._RecipeHandler[0]._InOut.a_RecipeList,   (* Array of Recipes *)
(* ->|   *)x_Enable           :=_Tech_TMPL._RecipeHandler[0]._In.x_Enable,          (* Enable FB *)
(* ->|   *)s_Path             :=_Tech_TMPL._RecipeHandler[0]._In.s_Path,            (* Path to recipes on PCC *)
(* ->|   *)i_RecipeIndex      :=_Tech_TMPL._RecipeHandler[0]._In.i_RecipeIndex,     (* Index of recipe in a_RecipeList to use *)
(* ->|   *)x_ListRecipes      :=_Tech_TMPL._RecipeHandler[0]._In.x_ListRecipes,     (* List all recipes from s_Path *)
(* ->|   *)x_LoadRecipe       :=_Tech_TMPL._RecipeHandler[0]._In.x_LoadRecipe,      (* Load recipe at i_RecipeIndex in a_RecipeList *)
(* ->|   *)x_SaveRecipe       :=_Tech_TMPL._RecipeHandler[0]._In.x_SaveRecipe,      (* Save recipe at i_RecipeIndex in a_RecipeList *)
(* ->|   *)x_DeleteRecipe     :=_Tech_TMPL._RecipeHandler[0]._In.x_DeleteRecipe);   (* Delete recipe at i_RecipeIndex in a_RecipeList *)
(*  OUT  *)
(*   |=> *)_File_TMPL.a_DataDummy[0]                    :=TB_RecipeHandler_1.Data;          (* FILE_TMPL STRUCT *)
(*   |=> *)_Tech_TMPL._RecipeHandler[0]._InOut.a_RecipeList  :=TB_RecipeHandler_1.a_RecipeList;  (* Array of Recipes *)
(*   |-> *)_Tech_TMPL._RecipeHandler[0]._Out.x_Done     :=TB_RecipeHandler_1.x_Done;        (* Action done *)
(*   |-> *)_Tech_TMPL._RecipeHandler[0]._Out.x_Busy     :=TB_RecipeHandler_1.x_Busy;        (* Action busy *)
(*   |-> *)_Tech_TMPL._RecipeHandler[0]._Out.w_ErrorID  :=TB_RecipeHandler_1.w_ErrorID;     (* FB ErrorID *)
(*   |-> *)_Tech_TMPL._RecipeHandler[0]._Out.x_Error    :=TB_RecipeHandler_1.x_Error;       (* FB Error *)
```

1. Bausteininstanz-Nr.
2. Feldindex in der Datentypstruktur
3. Anzahl der maximal im Projekt verwendbaren Bausteininstanzen

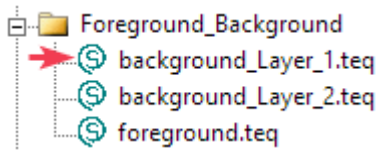
10.2.2 INTERFACE_TYPES

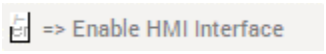
Hier befinden sich alle Datentypen, welche der Kommunikation mit der Visualisierung dienen. Dabei ist die oberste Strukturebene in folgende Teilbereiche unterteilt:

```
(* STRUCT *)
INTERFACE_STRUCT:
STRUCT
   x_Enable_Hmi_Interface          : BOOL;                 (* ENABLE / DISABLE HMI INTERFACE  *)
   _Cmd                            : COMMAND_STRUCT;       (* COMMAND FROM HMI                *)
   _Sts                            : STATUS_STRUCT;        (* STATUS TO HMI                   *)
   _Pro                            : PROCESS_STRUCT;       (* PROCESS DATA                    *)
   _Set                            : SETTINGS_STRUCT;      (* SETTINGS TO RETAIN              *)
   _Act                            : ACTUAL_STRUCT;        (* ACTUAL VALUES                   *)
END_STRUCT;
```

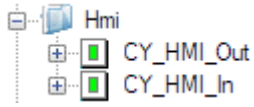
10.2.2.1 **x_Enable_Hmi_Interface**

Mit dieser Variable wird die Kommunikation zwischen HMI und PLC freigegeben. Diese Variable ist auch Quelle der Anzeige für die aktive Kommunikation in der Statusleiste des HMI. Die Variable wird in der Visualisierung zur Laufzeit durch ein Repaint-Event zyklisch geschrieben. Das geschieht in der „background_Layer_1.teq“:





Ein Doppelklick auf das kleine weiße Kästchen öffnet die Einstellungen dieses Painters. Damit wird das HMI-Interface zur Laufzeit automatisch mit dem Start der Visualisierung aktiviert. Diese Variable wird dann in den HMI-POE’s als Bedingung des Lesens/Schreibens verwendet:



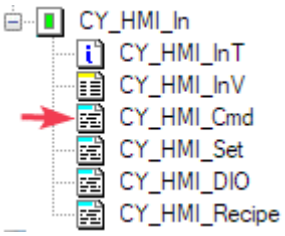

```
(* Enable / Disable Interface *)
IF _Hmi.x_Enable_HMI_Interface THEN
```

10.2.2.2 **_Cmd**

Richtung: HMI => PLC
Alle Kommandos, welche vom HMI in Richtung PLC gehen, befinden sich in dieser Struktur. Das ist z.B. das Kommando für das Einschalten der Achsen, ein Fehlerreset, oder Tippen.
Diese Struktur ist so aufgebaut, dass es einen Bereich gibt, dessen Variablen nach einem PLC-Zyklus automatisch zurückgesetzt werden und einen, in dem dies nicht passiert. Dies ist an den Kommentaren zu erkennen:

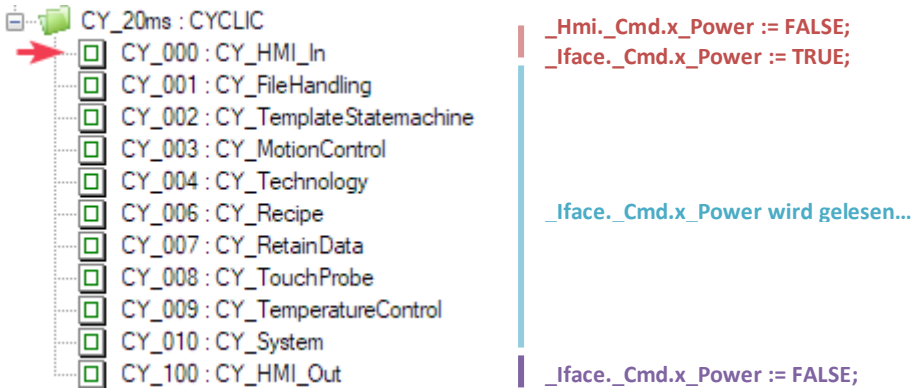
```
(* COMMANDS *)
COMMAND_STRUCT:
STRUCT
    (* (v) NOT RESETTED IN CY_HMI_OUT *)
    _Node                                   : NODE_IO_ARRAY;
    x_Jog_Fwd                               : BOOL;
    x_Jog_Bwd                               : BOOL;
    (* (v) RESETTED IN CY_HMI_OUT *)
    b_MemSetStart                           : BYTE;
    _Recipe                                 : RECIPE_CMD_STRUCT;
    _TP                                     : TP_CMD_STRUCT;
    _CS_Pos                                 : CS_POS_CMD_ARRAY;
    _CS_Time                                : CS_TIME_CMD_ARRAY;
    _CS_Pos_DT                              : CS_POS_DT_CMD_ARRAY;
    _TempCtrl                               : TEMP_CTRL_CMD_STRUCT;
    x_PowerOn                               : BOOL;
    x_Start                                 : BOOL;
    x_ErrorReset                            : BOOL;
    x_Home                                  : BOOL;
    x_Stop                                  : BOOL;
    x_UseGear                               : BOOL;
    x_Dummy                                 : BOOL;
    b_MemSetEnd                             : BYTE;
END_STRUCT;
```

An dem Beispiel „x_PowerOn“ näher erläutert:
Wird das Signal „x_PowerOn“ seitens Visualisierung auf TRUE gesetzt, dann wird dieses in der Steuerung erkannt und direkt zurückgesetzt. Damit ist ein sauberer Handshake vorhanden. Dieser Schritt wird in der POE „CY_HMI_In“ im Arbeitsblatt „CY_HMI_Cmd“ gemacht:




```
(* Listen for Power event *)
IF _Hmi._Cmd.x_PowerOn THEN
    _Hmi._Cmd.x_PowerOn    := FALSE;
    _Iface._Cmd.x_PowerOn := TRUE;
END_IF;
```

Daraufhin wird die Strukturvariable „_Iface…“ gesetzt. Dies geschieht alles in der ersten Programminstanz der zyklischen Task:

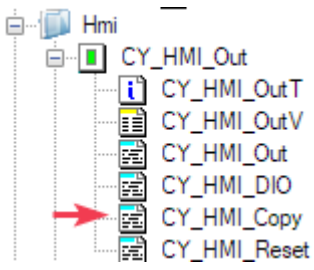


In den Programminstanzen nach der „CY_HMI_In“ wird dann mit der Interfacevariable „_Iface._Cmd.x_PowerOn“ gearbeitet. Diese wird dann letztendlich in der letzten Programminstanz „CY_HMI_Out“ (Im Arbeitsblatt „CY_HMI_Reset“) AUTOMATISCH zurückgesetzt. Dies passiert mit einem Speichermanagement-Baustein. Hier muss der Programmierer also keine Anpassungen mehr vornehmen.

```
(* Get Start of Reset byte *)
MC_GetPointer_1(
SRC                         :=_Iface._Cmd.b_MemSetStart);
_Iface._Cmd.b_MemSetStart :=MC_GetPointer_1.SRC;
ud_PtrStart                 :=MC_GetPointer_1.SRC_Ptr;
ud_SizeStart                :=MC_GetPointer_1.SRC_Length;

(* Get End of Reset byte *)
MC_GetPointer_2(
SRC                       :=_Iface._Cmd.b_MemSetEnd);
_Iface._Cmd.b_MemSetEnd :=MC_GetPointer_2.SRC;
ud_PtrEnd                 :=MC_GetPointer_2.SRC_Ptr;
ud_SizeEnd                :=MC_GetPointer_2.SRC_Length;

(* Calc size of bytes to reset *)
ud_Size := ud_PtrEnd - ud_PtrStart;

(* Set to 0 *)
w_Memset :=MEMSET(i_Memset_Err,b_Memset_Val,udint_to_dint(ud_Size),_Iface._Cmd.b_MemSetStart);
```

10.2.2.3 **_Sts**

Richtung: PLC => HMI
Alle Statusmeldungen, welche an die Visualisierung gehen, können in dieser Struktur vorgefunden werden. Klassischerweise können hier beispielsweise die Maschinenzustände gefunden werden.
Alle Statusvariablen werden automatisch aus der „_Iface“-Struktur in die „_Hmi“-Struktur kopiert. Dies geschieht in dem Arbeitsblatt „CY_HMI_Copy“ in der POE „CY_HMI_Out“:



10.2.2.4 **_Pro**

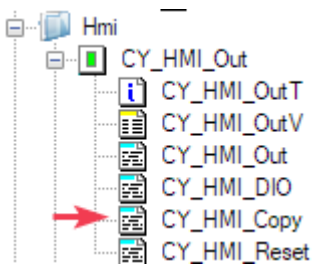
Richtung: HMI => PLC
Hier werden prozessrelevante Daten abgelegt, wie beispielsweise die einstellbare Taktgeschwindigkeit, oder die Beutellänge bei Folienmaschinen.

10.2.2.5 **_Set**

Richtung: HMI => PLC
Alle Maschineneinstellungen können hier abgelegt werden. Das wären z.B. Getriebefaktoren, Geschwindigkeiten, Beschleunigungen.

10.2.2.6 **_Act**

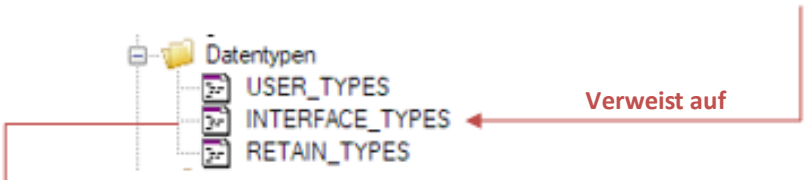
Richtung: PLC => HMI
Aktuelle Werte und Daten, welche aus Richtung der PLC an die Visualisierung gesendet werden, sind in dieser Struktur enthalten.
Alle „_Act“-Variablen werden automatisch aus der „_Iface“-Struktur in die „_Hmi“-Struktur kopiert. Dies geschieht in dem Arbeitsblatt „CY_HMI_Copy“ in der POE „CY_HMI_Out“:



10.2.3 **RETAIN_TYPES**

Dieses Datentyparbeitsblatt ist für die Strukturen, welche für die remanente Datenhaltung in Betracht gezogen werden.
Auf das Template bezogen werden hier als Beispiel die Einstellungen für die Temperaturregelung abgelegt.

```
TEMP_CTRL_RETAIN_ARR : ARRAY[0..1] OF TEMP_CTRL_SET_RECIPE_STRUCT;
```



Verweist auf

```
(* TEMP CONTROL SETTINGS & FOR RECIPE *)
TEMP_CTRL_SET_RECIPE_STRUCT:
STRUCT
   r_TSSF                              : REAL;
   r_Heat_Ki                           : REAL;
   r_Heat_Kp                           : REAL;
   r_Dist_Ki                           : REAL;
   r_Dist_Kp                           : REAL;
```

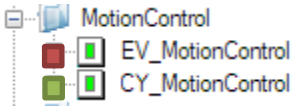
In der Statemachine der Temperaturregelung findet man dann die Variable zu dem Datentyp, welche als „Remanent“ deklariert ist:

Name	Typ	Verwendung	Beschreibung	Adresse	Anfangsw...	Remanent
Default						
a_TempCtrl_Retain	TEMP_CTRL_RETAIN_ARR	VAR				☑

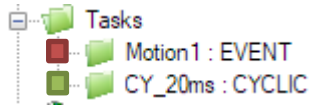
10.3 Logische POEs

10.3.1 Unterscheidung EV und CY

In einigen POEs gibt es die Unterscheidung zwischen „EV_“ und „CY_“:



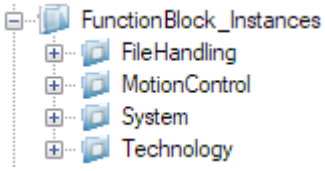
„EV“ bedeutet, dass die hier instanziierten Bausteine auch in der dazugehörigen „EV“-Task laufen:



Notiz!	
	Damit ist die **EVENT**-Task gemeint, welche im Feldbustakt ausgeführt wird. Hier sind also alle Funktionsbausteine eingefügt, welche im Feldbustakt laufen müssen und demnach zeitkritisch sind. „**CY**“ bedeutet, dass hier alle Bausteine laufen, welche in zeitunkritischen Tasks laufen können. Diese führen Befehle in einer Zeit aus, welche nicht innerhalb eines Feldbuszyklus ausgeführt werden müssen. Die dazugehörige Task ist die „**CY_20ms: CYCLIC**“-Task. Siehe auch Tasks

10.3.2 FunctionBlock_Instances

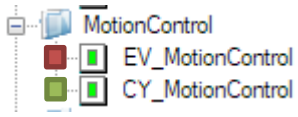
Jeder Funktionsbaustein, der im Projekt verwendet wird, muss an einer Stelle im Projekt instanziiert werden. Damit ist letztendlich das Einfügen des Bausteins in das Projekt gemeint. Das ist die Stelle, an der der Funktionsbaustein auch aufgerufen und abgearbeitet wird. Diese Bausteine sind innerhalb des Templates in Kategorien, welche letztendlich in Unterverzeichnissen resultieren, unterteilt.



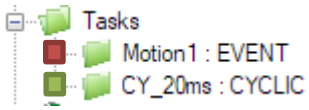
FileHandling
Hier sind alle Bausteine instanziiert, welche für die Verwaltung von Dateien auf der PLC genutzt werden können. Der Name der POE spiegelt auch immer den Baustein wieder, welcher hier hauptsächlich verwendet wird:

POE/Funktionsbaustein	Kurzbeschreibung
BM_FileOpen	Öffnet eine Datei
BM_FileClose	Schließt eine Datei
BM_FileList	Listet Dateien oder Verzeichnisse innerhalb eines Verzeichnisses
BM_FileDelete	Löscht eine Datei
BM_FileRead	Liest eine Datei, welche vorher mit BM_FileOpen geöffnet wurde
BM_FileWrite	Schreibt Daten in eine Datei, welche vorher mit BM_FileOpen geöffnet wurde

MotionControl
Hier gibt es die Unterscheidung zwischen **EV_MotionControl** und **CY_MotionControl**:



„**EV**“ bedeutet, dass die hier instanziierten Bausteine auch in der dazugehörigen „**EV**“-Task laufen:



Notiz!	
	Damit ist die **EVENT**-Task gemeint, welche im Feldbustakt ausgeführt wird. Hier sind also alle Funktionsbausteine eingefügt, welche im Feldbustakt laufen müssen und demnach zeitkritisch sind. „**CY**“ bedeutet, dass hier alle Bausteine laufen, welche in zeitunkritischen Tasks laufen können. Diese führen Befehle in einer Zeit aus, welche nicht innerhalb eines Feldbuszyklus ausgeführt werden müssen. Die dazugehörige Task ist die „**CY_20ms: CYCLIC**“-Task. Siehe auch Tasks

System
Hier befinden sich Bausteine, welche systemrelevante Funktionen ausführen können.

POE/Funktionsbaustein	Kurzbeschreibung
BM_CallWindowsApp	Führt eine ausführbare Datei auf der Windows-Seite der PLC aus
BM_MC_Control	Ermöglicht es den Feldbus neu zu starten

Technology
Dieser Teil beinhaltet alle Technologiebausteine, welche für die unterschiedlichsten Maschinenfunktionen benötigt, bzw. benutzt werden können.

Die wichtigsten Funktionsbausteine werden in den späteren Kapiteln in Verbindung mit den Maschinenfunktionen näher erklärt. Eine Detailbeschreibung der einzelnen Bausteine kann in der entsprechenden Bausteinhilfe nachgeschlagen werden.

10.3.3 Hmi

Grundlegende Funktion der Kommunikation
Die Kommunikation vom und zum HMI ist so aufgebaut, dass es hier EINE Variable „_Hmi“ gibt, welche direkt mit der Visualisierung kommuniziert. Zusätzlich gibt es eine „_Iface“-Variable, welche den gleichen Datentyp hat, wie die „_Hmi“-Variable. Beide findet man in der globalen Variablenliste in der Gruppe „HMI“:

Name	Typ	Verwendung	Beschreibung	Adresse	Anfangswert	Reman...	PDD	OPC
⊟ HMI								
_Iface	INTERFACE_STRUCT	VAR_GLOBAL	PLC-Internal values for HMI communication			☐	☐	☐
_Hmi	INTERFACE_STRUCT	VAR_GLOBAL	PDD Interface to webServer (PCC)			☐	☑	☐

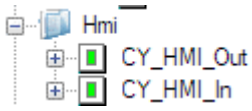
An der Checkbox „PDD“ ist zu erkennen, dass nur die Variable „_Hmi“ für die HMI-Kommunikation freigegeben ist. „PDD“ bedeutet „Process Data Directory“ und basiert auf einer internen Namensauflösung. Diese Schnittstelle arbeitet also alleine mit den Namen der Variablen und benötigt, im Gegensatz zu OPC, keine Adressen.

In den unten erklärten POEs wird letztendlich ein Handshake zwischen der Variable „_Hmi“ und „_Iface“ gemacht.

Folgendes Bild soll die Struktur nochmal verdeutlichen:

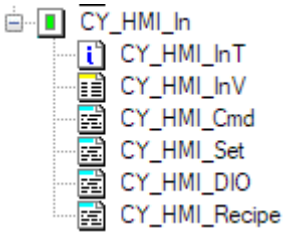
Hier befindet sich die Schnittstelle vom und zum HMI. Alle Daten, die entweder vom HMI kommen, oder aber auch an das HMI gehen, werden hier übergeben. Damit kann man vor dem Empfangen, bzw. Senden die Daten nochmals validieren und auf ihre Gültigkeit hin prüfen.

Anhand der Bezeichnung der POE lässt sich erkennen für welche Richtung die POE verwendet wird:



CY_HMI_Out	=>	Alles was an das HMI geschickt wird
CY_HMI_In	=>	Daten die vom HMI an die PLC geschickt werden

Für eine bessere Strukturierung sind die Arbeitsblätter innerhalb der POEs nochmals unterteilt und so benannt, dass sich auch hier anhand der Namen die entsprechende Funktion ableiten lässt.



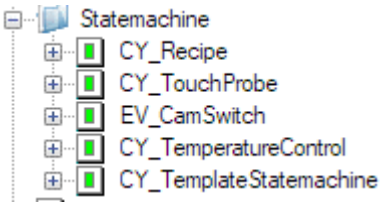
CY_HMI_Cmd	=>	Kommandos vom HMI (MouseUp, MouseDown, Toggle)
CY_HMI_Set	=>	Einstellwerte vom HMI
CY_HMI_DIO	=>	Verwaltung der digitalen Eingänge/Ausgänge vom HMI
CY_HMI_Recipe	=>	Alle rezeptrelevanten Daten welche im HMI gesteuert werden können

10.3.4 RetainHandling

In dieser POE befindet sich lediglich ein Baustein, welcher dazu genutzt werden kann um das Abspeichern der remanenten Daten manuell anzustoßen. Dies ist z.B. notwendig wenn man einen PCC-04 ohne eine USV betreibt, da hier kein NOVRAM vorhanden ist, welches die Daten automatisch speichert. Mit der aktuellen Version der PLCmc ist dies allerdings nicht mehr notwendig und damit obsolet, kann allerdings weiter verwendet werden.

10.3.5 Statemachine

Hier sind alle Zustandsmaschinen vorhanden, welche für das Projekt benötigt werden:

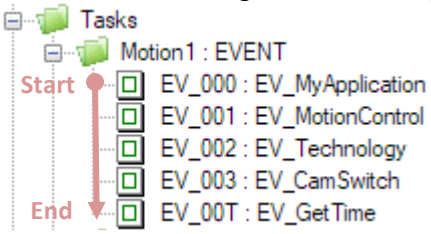


Zustandsmaschine	Kurzbeschreibung
CY_Recipe	Statemachine für das Handling der Rezepturdaten
CY_TouchProbe	Hier wird die Messtasterfunktion gehandelt
EV_CamSwitch	Statemachine für das Handling des Nockenschaltwerks
CY_TemperatureControl	Die Temperaturregelung wird hier verwaltet
CY_TemplateStatemachine	Die „Haupt“-Statemachine des Projektes. Hier wird die Zustandsmaschine der Maschine abgebildet.

Notiz!	
	Die einzelnen Zustandsmaschinen werden in den jeweiligen maschinenfunktionsspezifischen Kapiteln näher erläutert

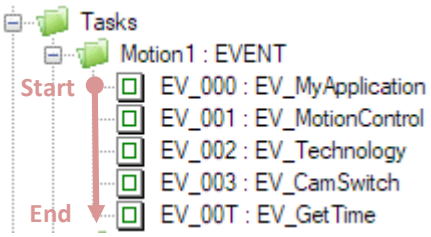
10.4 Tasks

Generell werden alle Tasks von oben nach unten abgearbeitet. Dies ist wichtig, da es besonders bei der „CYCLIC“-Task in Bezug auf die HMI-Programminstanzen eine wichtige Rolle spielt.



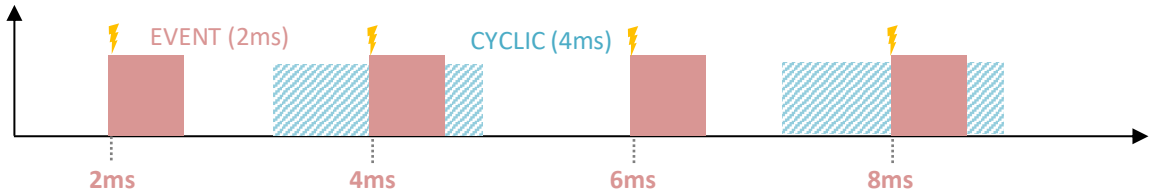
10.4.1 Motion1 : EVENT

In dieser Task befinden sich alle Programminstanzen, welche mit dem Präfix „EV_“ beginnen. Diese Tasks sind zeitkritisch und werden im Zyklus des Feldbustaktes ausgeführt.

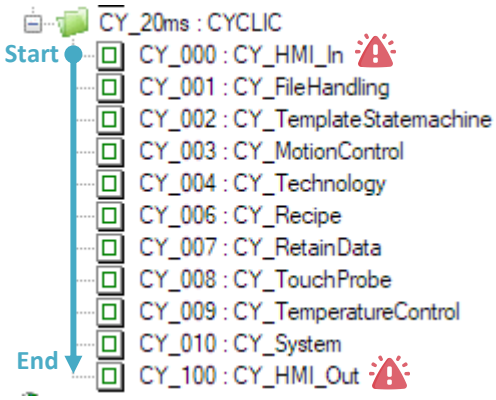


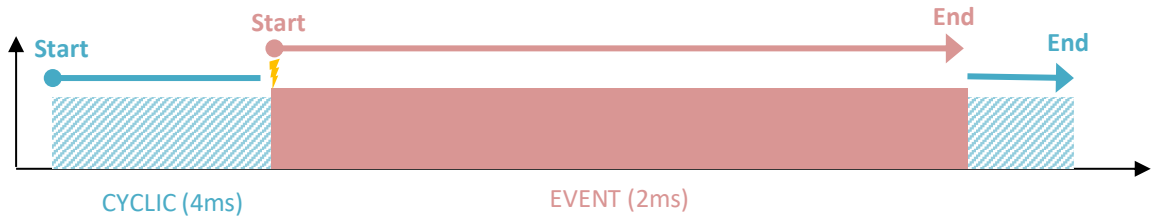
10.4.2 CY_20ms : CYCLIC

Hier befinden sich alle Programminstanzen, welche nicht zeitkritisch sind und somit nicht im Feldbustakt aufgerufen werden. Diese Instanzen werden alle 20ms aufgerufen, aber durch einen Feldbus-Interrupt unterbrochen, da dieser eine höhere Priorität hat:



In diesem Beispiel wird die **CYLCLIC-Task alle 4ms** aufgerufen (nur für die Veranschaulichung) und die **EVENT-Task alle 2ms**. Man sieht, dass die CYCLIC-Tasks durch die EVENT-Task unterbrochen werden kann und dort weiter macht, wo sie aufgehört hat. Deshalb macht es in den meisten Fällen auch Sinn die Daten aus der CYCLIC nicht mit den Daten aus der EVENT zu mischen.



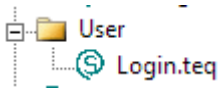


Die Programminstanz „CY_HMI_In“ **muss** am Anfang der Task stehen und die „CY_HMI_Out“ am Ende der Task. Das ist dadurch begründet, dass alle Daten, welche vom HMI an die Steuerung gesendet werden, in der ersten Task in die _Iface-Variablen kopiert werden und diese _Iface-Variablen dann in den darauf folgenden Tasks benutzt werden. Am Ende (End) der CYCLIC-Task werden dann die Daten aus den _Iface-Variablen in die „_Hmi“-Struktur kopiert.

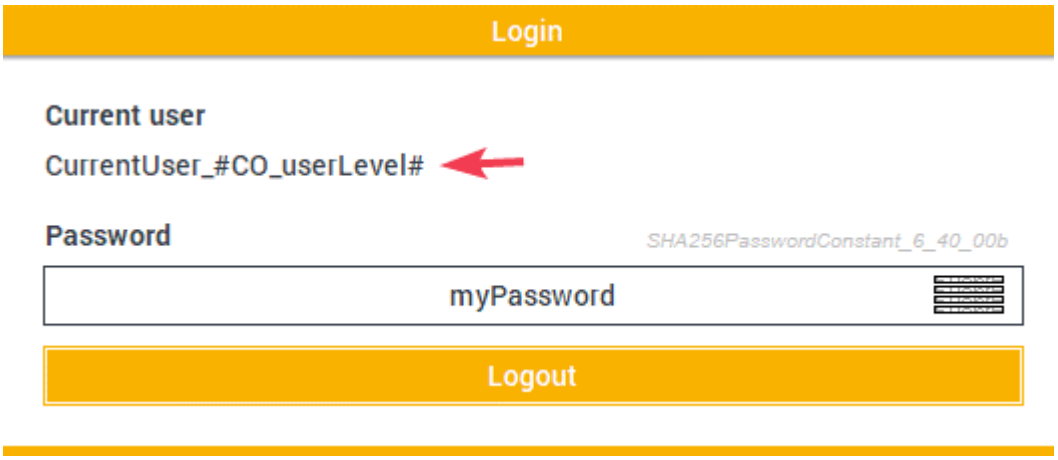
11. Maschinenfunktionen

11.1 Benutzerlevelmanagement

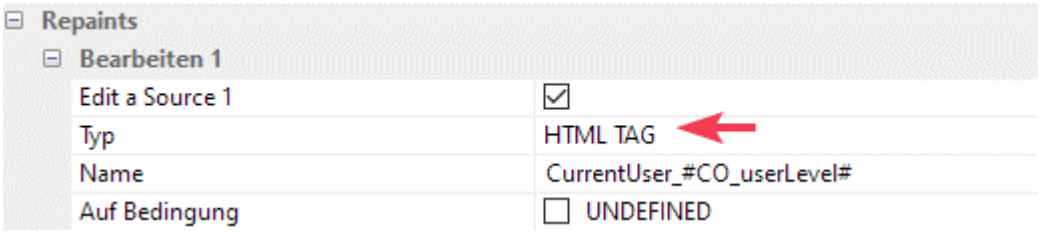
Das Benutzerlevelmanagement beinhaltet alle Komponenten um die voreingestellten maximal vier Benutzerlevel zu verwalten. Die Verwaltung findet ausschließlich in der Visualisierung statt. Es ist hier also keine Schnittstelle zur PLC implementiert. Diese kann aber bei Bedarf nachgerüstet werden.
Hierfür gibt es im HMI-Projekt ein Unterverzeichnis „User“ mit der View „Login.teq“:

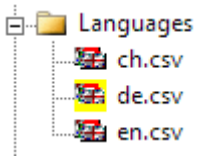


Hier sieht man das Login/Logout-Fenster, welches bei Bedarf editiert werden kann.
Im linken oberen Eck sieht man den aktuell angemeldete Benutzer über den Zugriff auf die CONTAINER-Variable „userLevel“ in Verbindung mit einem HTML-Tag „CurrentUser_“, welcher als Präfix dient:



Hier wird dann zur Laufzeit in Abhängigkeit des Wertes in „userLevel“ der Entsprechende Benutzer dargestellt. Da diese Variable vom Typ „HTML TAG“ ist, wird diese auch in die Sprachdateien beim Build eingetragen:





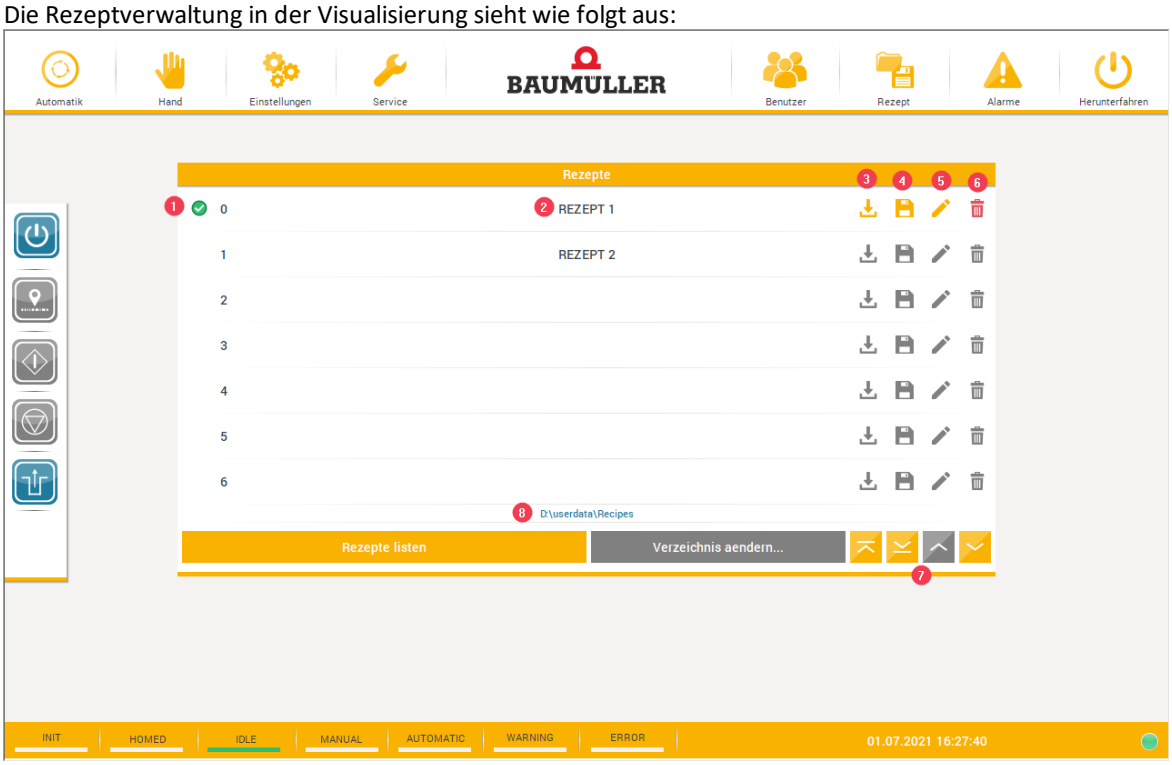
```
CurrentUser_#CO_userLevel#;CurrentUser_#CO_userLevel#
CurrentUser_0;Bediener
CurrentUser_1;Produktionsmanager
CurrentUser_2;Einsteller
CurrentUser_3;Service
CurrentUser_4;Entwickler
```

11.2 Rezeptverwaltung

Die Rezeptverwaltung besteht aus einem PLC-, und einem Visualisierungsteil. Das Management der Rezepte wird aber in der PLC gemacht. Die Visualisierung dient lediglich als Anzeige-, oder Eingabemaske.

11.2.1 HMI

Die Rezeptverwaltung in der Visualisierung sieht wie folgt aus:

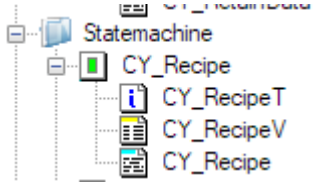


Nr	Beschreibung
1	Aktuelle Auswahl
2	Rezeptname
3	Rezept laden
4	Rezept speichern
5	Rezept umbenennen
6	Rezept löschen

7	Rezepte scrollen
8	Rezeptverzeichnis auf der PLC

11.2.2 **PLC**

In der PLC gibt es für die Rezeptverwaltung eine Statemachine:



State 0 ist immer der Wartezustand in dem auf eine Eingabe, oder ein Befehl durch die Visualisierung gewartet wird:

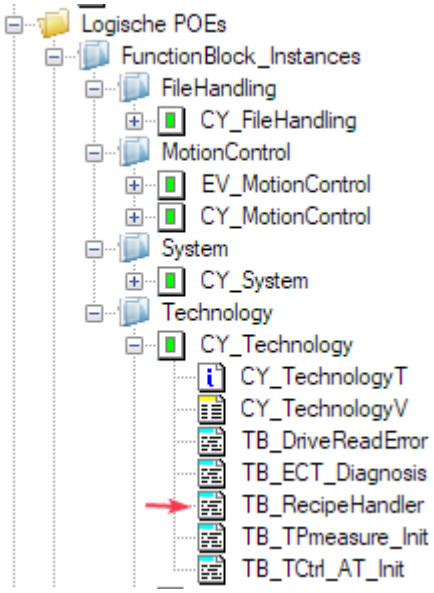
```
(* ------------------------------------------------------------------------------ *)
0: s_State := '0: Wait for command';
(* ------------------------------------------------------------------------------ *)
```

Wird ein Kommando erkannt, so wird die dazugehörige Aktion ausgeführt:

```
IF _Iface._Cmd._Recipe.x_DeleteRecipe THEN
   _Tech_TMPL._RecipeHandler[0]._In.x_DeleteRecipe    := FALSE;
   i_State                 := INT#200;
END_IF;
```

Hier wird das aktuell ausgewählte Rezept gelöscht.

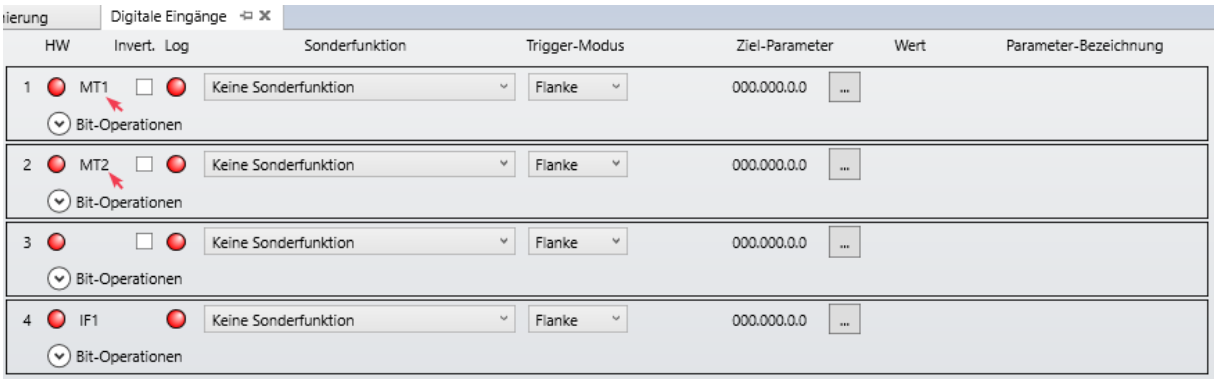
Der dazugehörige **Funktionsbaustein** befindet sich hier:



11.3 Messtaster

11.3.1 Allgemeine Funktionsbeschreibung

Der Messtaster funktioniert so, dass ein optischer Sensor an einem Eingang des bmaXX angeschlossen wird. Hierfür sind spezielle schnelle Eingänge vorgesehen, welche mit dem Kürzel „MT“ für Messtaster bezeichnet sind:



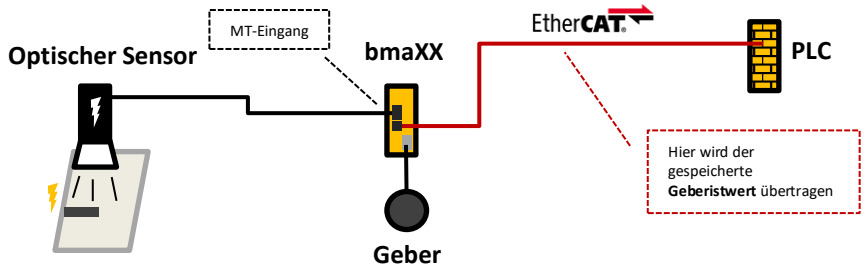
Diese Eingänge können für die Messtasterfunktion genutzt werden. Zusätzlich kann im Antrieb die detaillierte Funktion dieser Eingänge festgelegt werden.

Unter Konfiguration -> Messtaster -> Messtaster Geber 1 bspw. sind die Konfigurationsmöglichkeiten zu sehen:

Die Einstellungen, die hier zu sehen sind, werden in dem Template durch die Steuerung/Visualisierung gesetzt. Das bedeutet, dass man hier manuell keine Einstellungen vornehmen muss. Die Darstellung dient rein der Information.

Ist der Messtaster aktiviert und es wird eine Flanke erkannt, dann wird der dazugehörige Positionswert des Gebers gespeichert und der Steuerung zur Verfügung gestellt. In der Steuerung wird dann der aus mindestens 2 voneinander entfernten Signalflanken die Distanz berechnen.

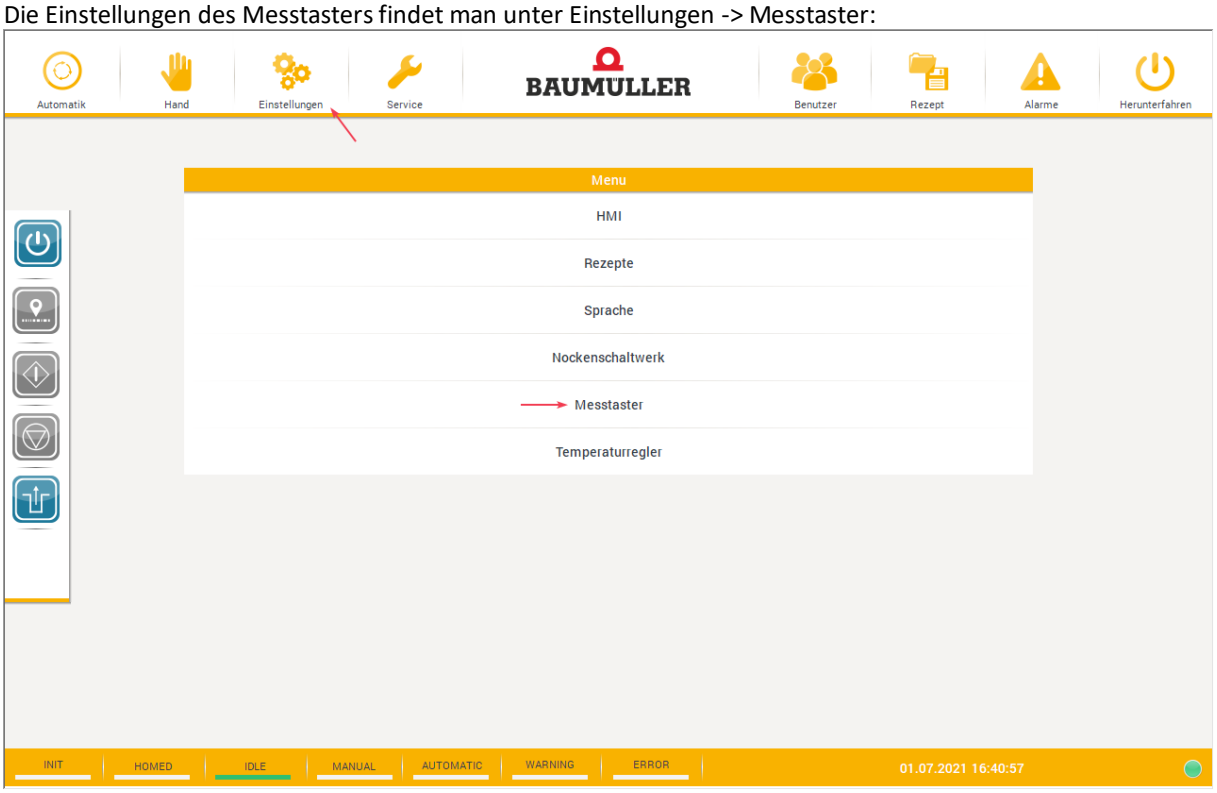
Hier das Schema zur Verdeutlichung grafisch aufbereitet:



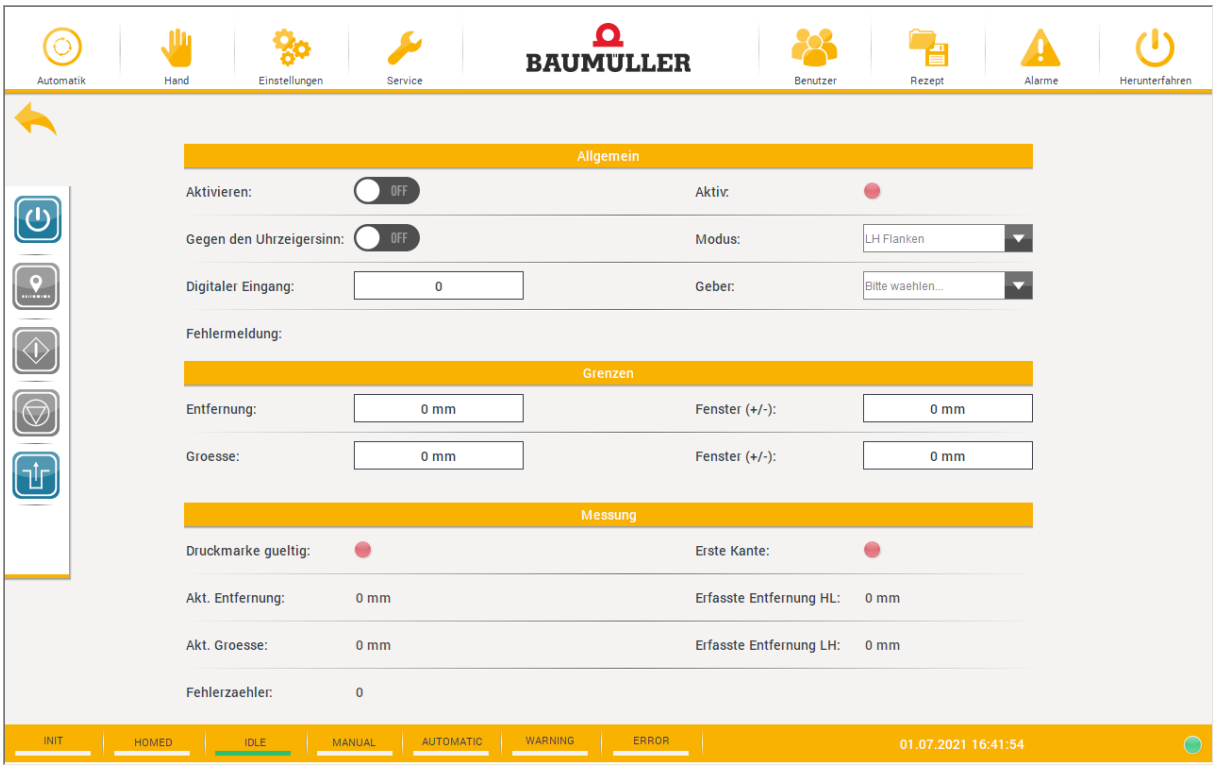
Für die Messtasterfunktion gibt es einen Teil in der Visualisierung und einen Teil in der PLC.

11.3.2 HMI

Die Einstellungen des Messtasters findet man unter Einstellungen -> Messtaster:

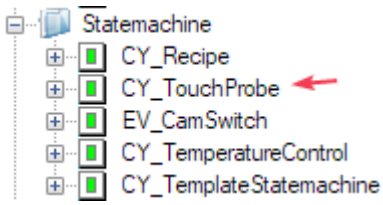


Hier sieht man die Einstellungen, welche man für die Messtasterfunktion nutzen kann:



11.3.3 **PLC**

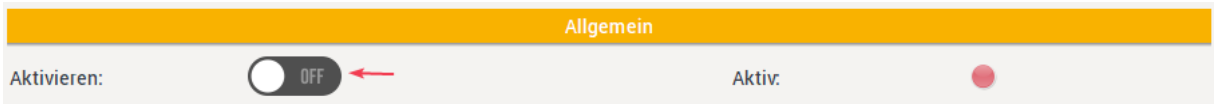
Im PLC-Projekt gibt es dazu eine Statemachine und natürlich auch die dazugehörigen Funktionsbausteine. Die Statemachine findet man hier:



Im Zustand 0 wird auf das Kommando „Aktivieren“ durch die Visu gewartet:

```
(* ------------------------------------------------------------------------ *)
0: s_State := '0: Wait for command';
(* ------------------------------------------------------------------------ *)

   _Tech_TMPL._TPMeasure[0]._In.x_Enable        := FALSE;
   _Tech_TMPL._TPMeasure_Init[0]._In.x_Enable   := FALSE;

   IF _Iface._Cmd._TP.x_Enable THEN
      _Iface._Act._TP.s_ErrorMessage := '';
      i_State := 100;
   END_IF;
```

Dies entspricht dem Toggle-Button in der Visualisierung:

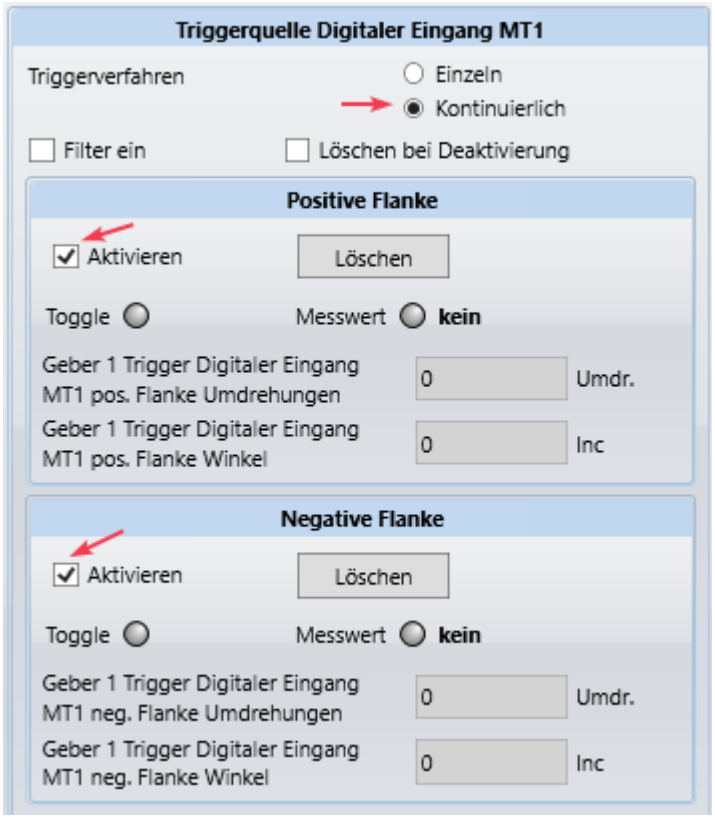


Vor dem Aktivieren sollten die Einstellungen zu dem Messtaster gesetzt werden. Hier kann man z.B. den digitalen Eingang des Reglers wählen und welcher Geber dafür benutzt werden soll.

Wurde der Messtaster erfolgreich initialisiert, sieht man das an der „Aktiv“-LED:

Notiz!	
	Alle Einstellungen sind in der Steuerung mit den dazugehörigen Variablen für die Funktionsbausteine verknüpft. Diese kann man im Detail in den Hilfedateien der Bausteine nachlesen.

In diesem Schritt wurde nun die Messtasterfunktion im Regler entsprechend der Einstellungen, die gesetzt wurden, initialisiert. Das kann man mit ProDrive prüfen:



Die Statemachine befindet sich jetzt im Zustand „TPmeasure“:

```
(* ---------------------------------------------------------------------------- *)
200: s_State := '200: TPmeasure';
(* ---------------------------------------------------------------------------- *)
```

Hier werden die Einstellungen für Distanz zwischen den Druckmarken und die Größe der Druckmarken, sowie die Toleranzfenster an den Baustein übergeben:

```
_Tech_TMPL._TPMeasure[0]._In.di_DefDist         := _Iface._Set._TP.di_Distance;
_Tech_TMPL._TPMeasure[0]._In.di_DefSize         := _Iface._Set._TP.di_Size;
_Tech_TMPL._TPMeasure[0]._In.di_WinDist         := _Iface._Set._TP.di_DistanceWindow;
_Tech_TMPL._TPMeasure[0]._In.di_WinSize         := _Iface._Set._TP.di_SizeWindow;
_Tech_TMPL._TPMeasure[0]._In.x_CounterClockwise := _Iface._Sts._TP.x_CounterClockwise;
```

Zusätzlich muss man hier noch die Skalierung der Einheiten anpassen. Per Default ist hier eine Auflösung von 100 mm pro Motorumdrehung eingestellt:

```
(* Change Scaling on demand / Default is axis scaling *)
_Tech_TMPL._TPMeasure[0]._In.ud_Units     := UDINT#100;    (* mm *)
_Tech_TMPL._TPMeasure[0]._In.u_Revolution := UINT#1;       (* Turns *)
```

Notiz!	
	Hier muss dann natürlich die Einstellung auf die Maschine unter Berücksichtigung der Getriebefaktoren eingestellt werden.

Anschließend wird der Baustein aktiviert:

```
_Tech_TMPL._TPMeasure[0]._In.x_Enable  := TRUE;
```

Dreht sich nun der Motor und wird dabei die Druckmarke erkannt, dann wird die gemessene Entfernung und die Größe entsprechend unter „Messung“ ausgegeben. Sollten Fehlerhafte Druckmarken erkannt werden (z.B. nicht im Fenster erfasst), dann wird dies auch über einen Fehlerzähler dargestellt.

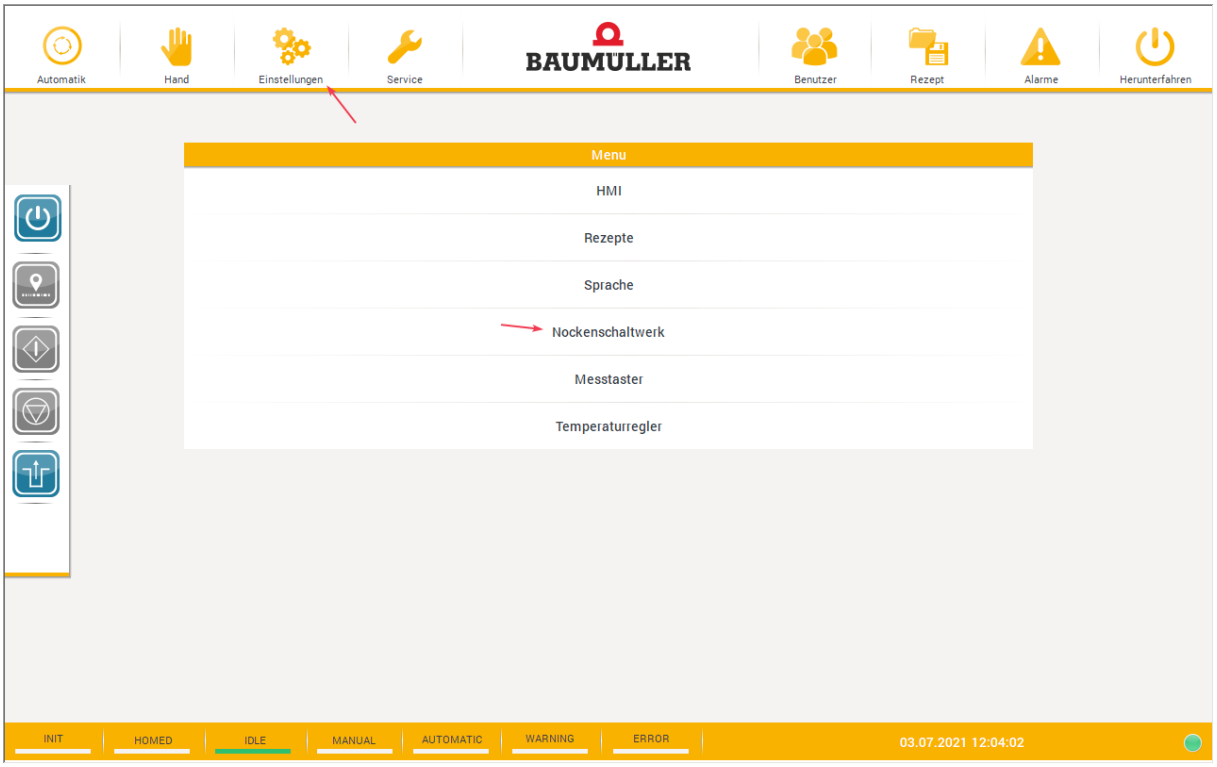
11.4 Nockenschaltwerk

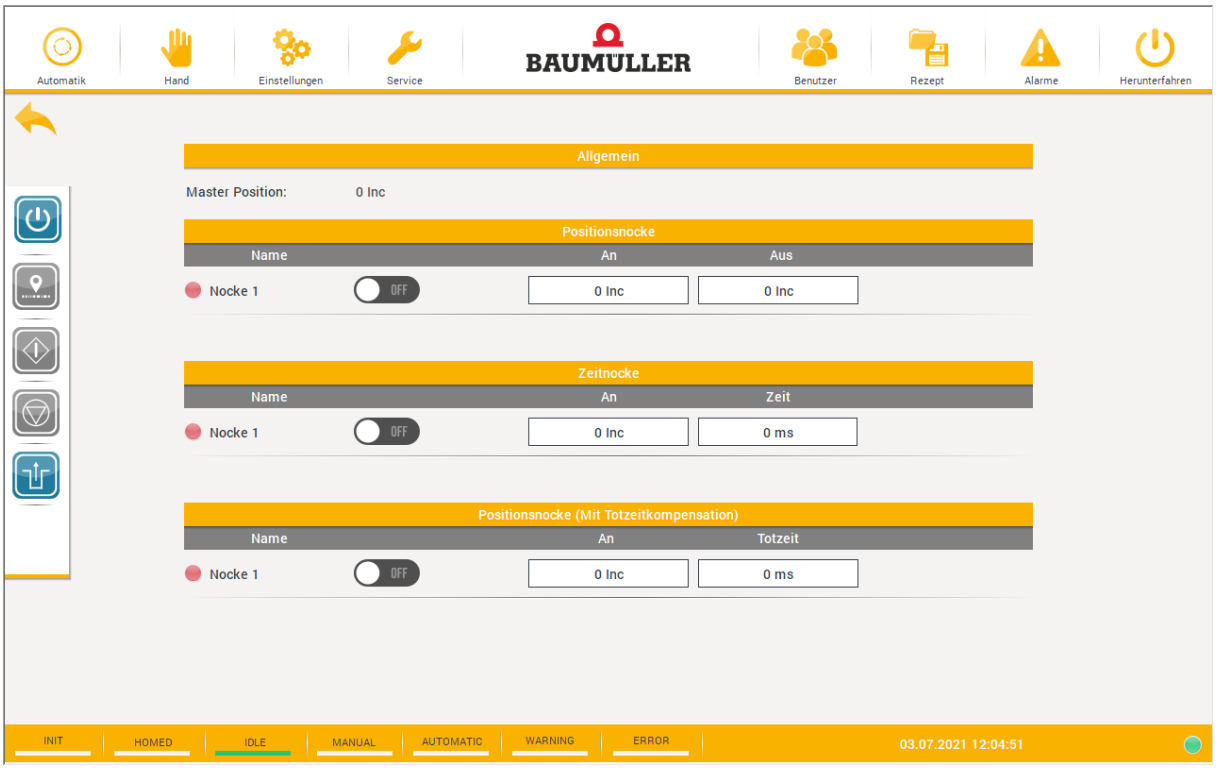
11.4.1 Allgemeine Funktionsbeschreibung

Mit dem Nockenschaltwerk können während des Prozesses zu bestimmten Zeitpunkten, bzw. Positionen, bezogen auf eine Achse Aktionen ausgeführt werden. Diese Achse kann real oder virtuell sein. Hierbei können beispielsweise Pneumatikventile angesteuert werden. Diese Ventilansteuerungen besitzen in der Regel eine Totzeit, welche während des Prozesses auch berücksichtigt werden muss. Wird ein Ventil angesteuert, heißt es nicht, dass dieses Ventil auch im nächsten Zyklus die Aktion ausgeführt hat. Die Zeitdifferenz von der Ansteuerung bis hin zum Ausführen der Aktion entspricht der Totzeit.

11.4.2 HMI

Die Einstellungen und die Funktion „Nockenschaltwerk“ befinden sich unter Einstellungen -> Nockenschaltwerk:





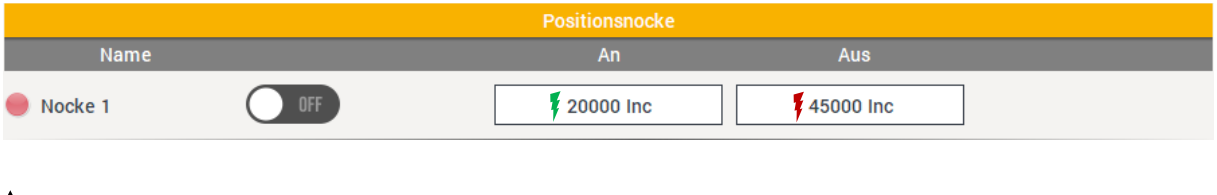
Die hier einstellbaren Nocken dienen als Anschauungsbeispiel und können beliebig erweitert werden. Diese Nocken orientieren sich an der Masterposition, welche unter „Allgemein“ eingeblendet wird. Sobald eine Nocke aktiv ist, wird die dazugehörige LED grün.

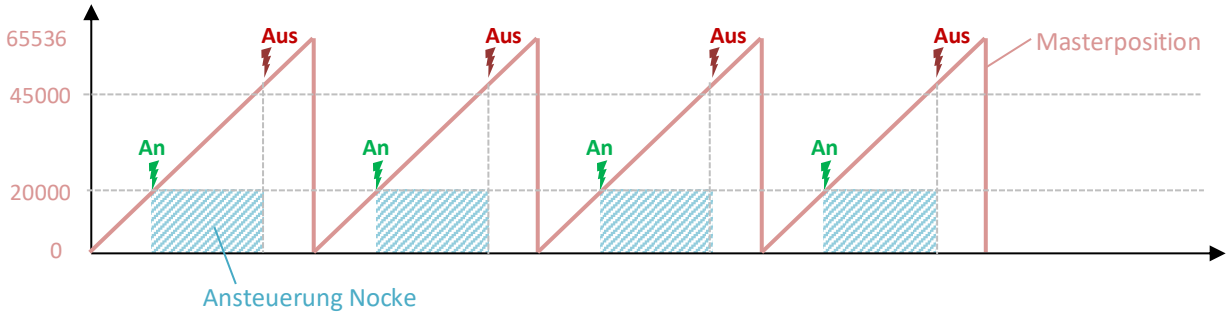
Notiz!	

	Bei der Positionsnocke mit Totzeitkompensation allerdings wird die Nocke lediglich für einen Zyklus gesetzt. Das ist zu schnell um es in der Visualisierung zu sehen. Deshalb wurde hierfür im Code ein Zähler integriert, damit man sieht ob der Ausgang schaltet.
	
	`(* Check CamOut *)`
	`IF _Tech_TMPL._CamSwitch[0]._Out.x_CamOut THEN`
	`  i_CamCounter := i_CamCounter + 1;`
	`END_IF;`

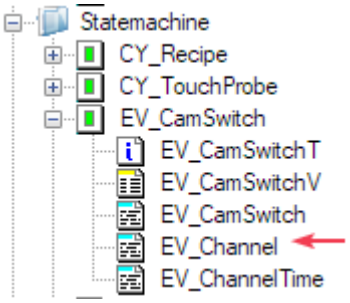
11.4.3 Positionsnocke

Die Positionsnocke ist eine Nocke, welche in Abhängigkeit einer Position einschaltet und in Abhängigkeit einer anderen Position abschaltet. Hier gibt es keine Totzeitkompensation. Es wird lediglich auf die Position geschaut, welche hier die des virtuellen Masters ist:





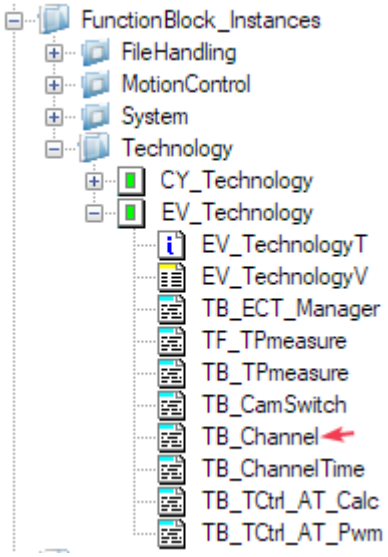
Der in der Steuerung dafür verwendete Funktionsbaustein ist der „TB_Channel“. Für das Nockenschaltwerk gibt es generell eine Statemachine, welche das Ein-, und Ausschalten der Bausteine verwaltet:



Wobei man hierbei sagen muss, dass sowohl der „TB_Channel“, als auch der „TB_ChannelTime“ keine Statemachine benötigen, sondern nur der Vollständigkeit halber hier mit aufgenommen wurden.
Diese Funktionsbausteine benötigen aufgrund ihres einfachen Handlings keine komplexe Verwaltung.
Das wird klar, wenn man sich das Arbeitsblatt „EV_Channel“ genauer ansieht:

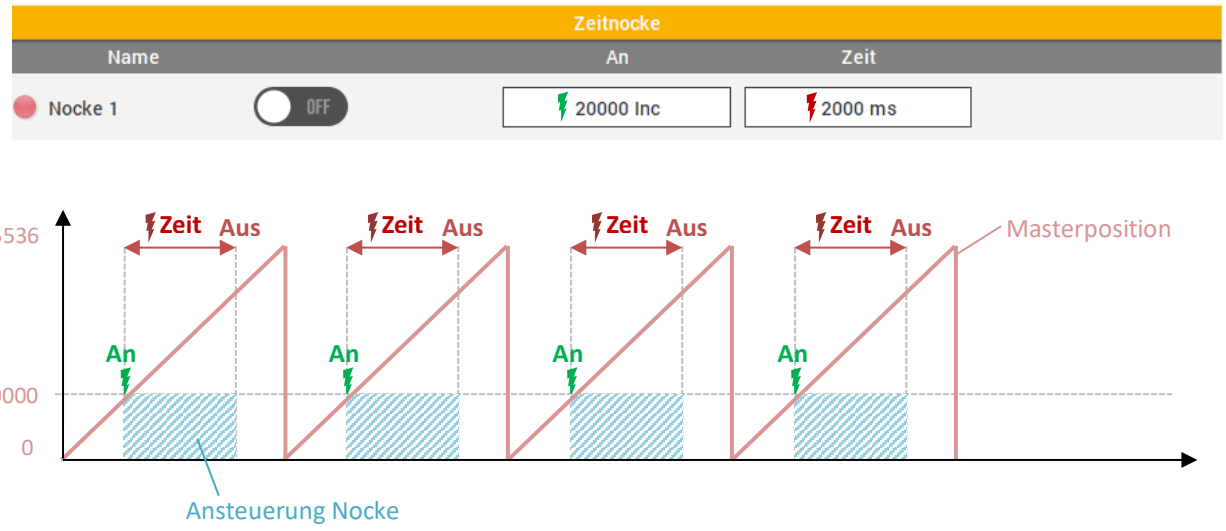
```
(* 1. Cam switch position *)
IF _Iface._Sts._CS_Pos[0].x_Enabled THEN
  _Tech_TMPL._Channel[0]._In.ud_On     := REAL_TO_UDINT(_Iface._Set._CS_Pos[0].r_On);  (* On value *)
  _Tech_TMPL._Channel[0]._In.ud_Off    := REAL_TO_UDINT(_Iface._Set._CS_Pos[0].r_Off); (* Off value *)
   _Tech_TMPL._Channel[0]._In.ud_Pos   := _MyMaster_TMPL._Out.ud_CyclePosition;        (* Switch by master position *)
END_IF;
```

Die Instanz des dazugehörigen Funktionsbausteines findet man hier:

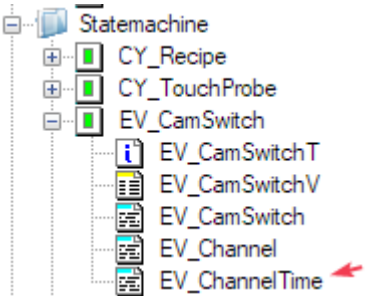


11.4.4 Zeitnocke

Mit der Zeitnocke kann ein eine Nocke zu einer bestimmten Position für eine einstellbare Zeit angesteuert werden. Auch hier wird keine Totzeit berücksichtigt.



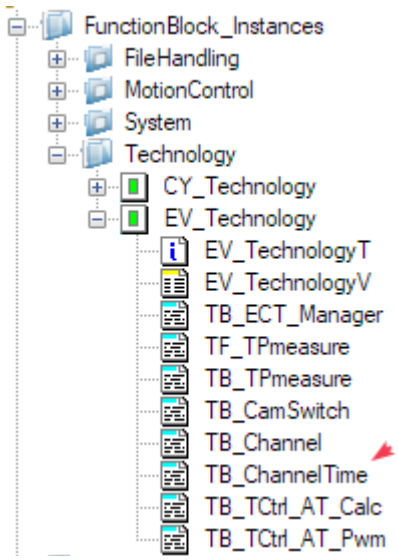
In der PLC gibt es hierfür unter Statemachine in der POE „EV_CamSwitch“ das Arbeitsblatt „EV_ChannelTime“:



Auch hier ist, wie bei dem „TB_Channel“ keine Statemachine notwendig:

```
(* 1. Cam switch time *)
IF _Iface._Sts._CS_Time[0].x_Enabled THEN
  _Tech_TMPL._ChannelTime[0]._In.ud_On            := REAL_TO_UDINT(_Iface._Set._CS_Time[0].r_On);          (* On value *)
  _Tech_TMPL._ChannelTime[0]._In.ud_Off_Time      := REAL_TO_UDINT(_Iface._Set._CS_Time[0].r_Time);        (* Time *)
   _Tech_TMPL._ChannelTime[0]._In.u_Interrupt_Time := u_MC_CommCycle_M01;                                  (* Event cycle *)
   _Tech_TMPL._ChannelTime[0]._In.ud_Pos_Res       := UDINT#65535;                                         (* 16 Bit Resolution of Master*)
   _Tech_TMPL._ChannelTime[0]._In.ud_Pos           := _MyMaster_TMPL._Out.ud_CyclePosition;                (* Master position *)
END_IF;
```

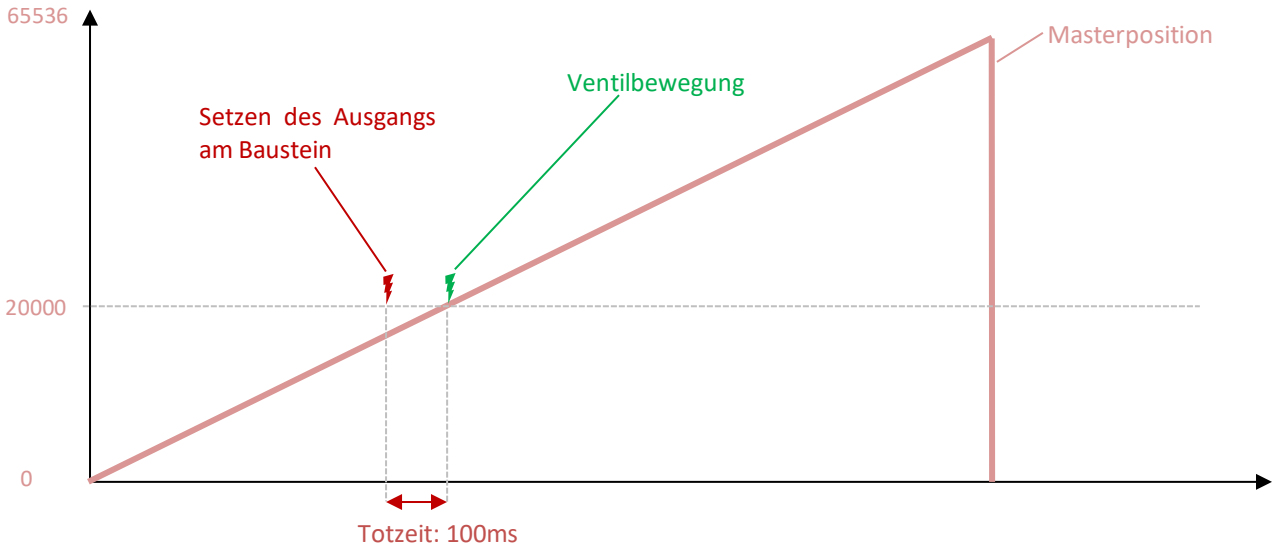
Die dazugehörige Bausteininstanz findet man hier:



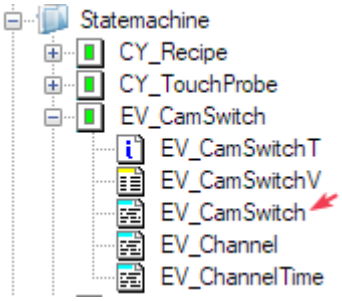
11.4.5 Positionsnocke (mit Totzeitkompensation)

Bei der Positionsnocke mit Totzeitkompensation handelt es sich um eine Nocke, welche zu einer bestimmten Position unter Berücksichtigung einer Totzeit für einen Zyklus angesteuert werden kann. Da diese nur für einen Zyklus aktiv ist, sieht man die Ansteuerung nicht über die LED. Dafür gibt es in der PLC allerdings einen Zähler, der bei jeder Ansteuerung hochzählt. (siehe HMI)

Anhand des folgenden Bildes wird diese Funktion näher erläutert:



In der Steuerung gibt es für die Verwaltung dieser Funktion eine Statemachine:



In State 0 wird auf die Aktivierung der Funktion über das HMI gewartet:

```
(* ------------------------------------------------------------------------ *)
0: s_State := '0: Wait for command';
(* ------------------------------------------------------------------------ *)
```

Dabei wird auch die aktuelle Position des Masters als Startposition für den Funktionsbaustein festgelegt:

```
IF _Iface._Sts._CS_Pos_DT[0].x_Enabled THEN
   _Tech_TMPL._CamSwitch[0]._In.di_StartPos  := UDINT_TO_DINT(_MyMaster_TMPL._Out.ud_CyclePosition);
   i_State := 100;
END_IF;
```

Anschließend werden die Parameter aus der Visualisierung an den Baustein übergeben, sowie die Masterposition. Dabei wird noch von Inc/Zyklus nach Inc/s umskaliert.

```
(* Convert from Inc/Cycle => Inc/s *)
_Tech_TMPL._CamSwitch[0]._In.di_VelSet_s     := REAL_TO_DINT(DINT_TO_REAL(_MyMaster_TMPL._In.di_Velocity) * 1000.0 (*ms/s*) / (UINT_TO_REAL(u_MC_CommCycle_M01) / 1000.0 (*µs/ms*)));
_Tech_TMPL._CamSwitch[0]._In.di_VelAct_s     := REAL_TO_DINT(DINT_TO_REAL(_MyMaster_TMPL._Out.di_CycleVelocity) * 1000.0 (*ms/s*) / (UINT_TO_REAL(u_MC_CommCycle_M01)/ 1000.0 (*µs/ms*)));
```

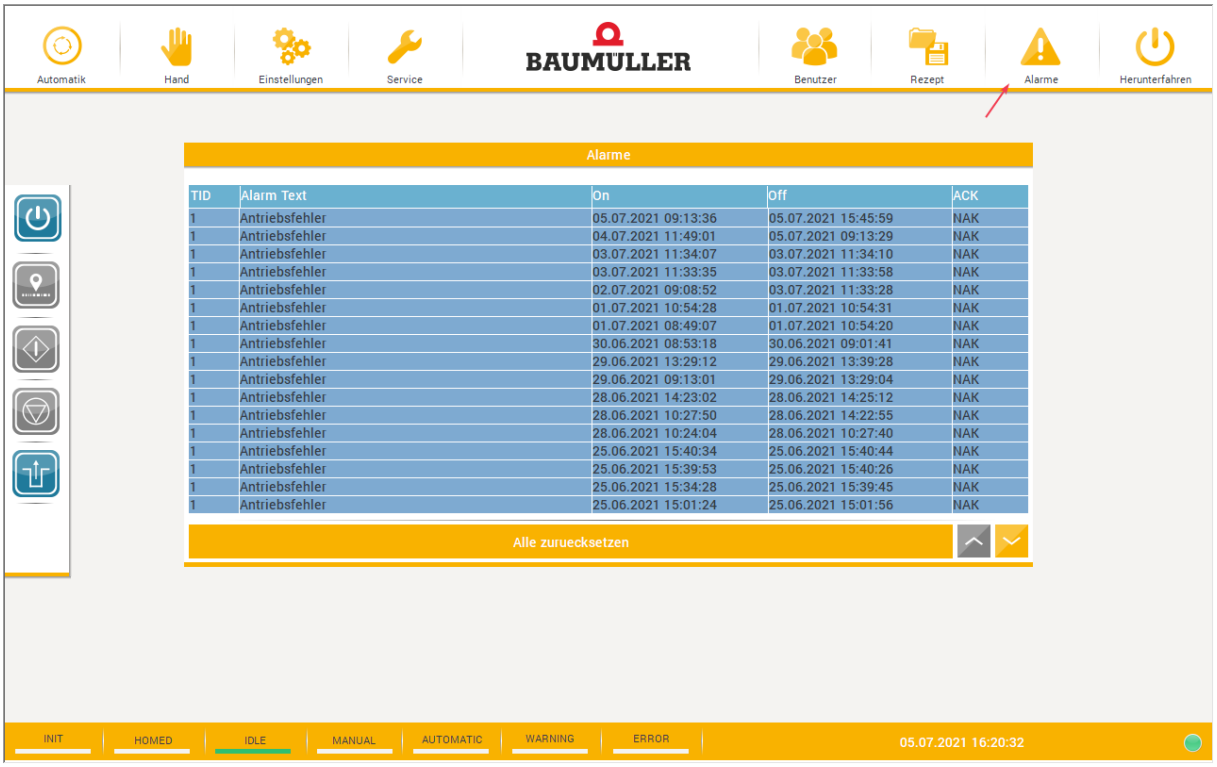
Mit dem Zähler kann man sehen, wie oft, bzw. auch ob der Ausgang für die Nockenanschaltung aktiviert wurde:

```
(* Check CamOut *)
IF _Tech_TMPL._CamSwitch[0]._Out.x_CamOut THEN
    i_CamCounter := i_CamCounter + 1;
END_IF;
```

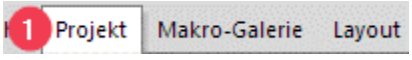
11.5 Alarming

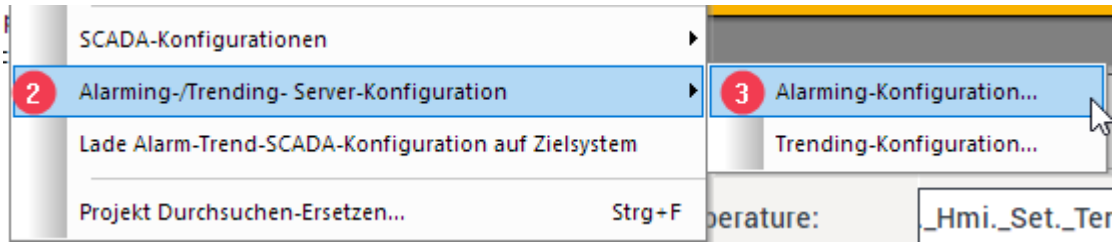
11.5.1 HMI

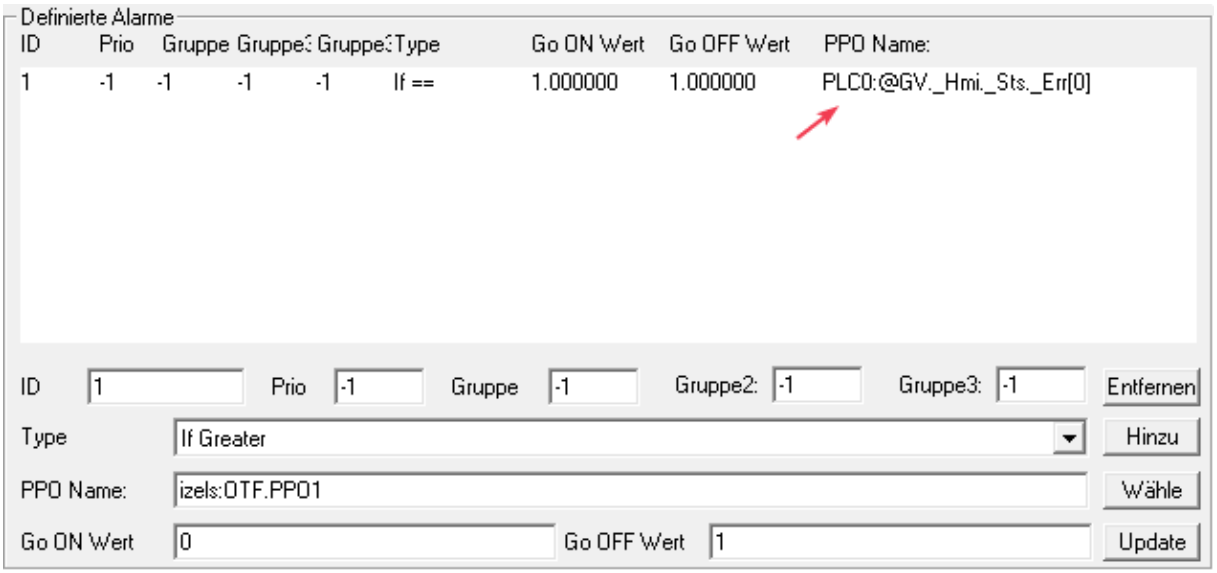
Für das Alarming gibt es in der Visualisierung die Seite „Alarme“:



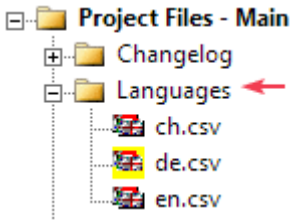
Hier werden alle Alarme angezeigt und auch die Historie. Die Variablen für die Alarme sind über den SCADA-Editor einstellbar. Für das Template ist als Beispiel eine Variable hinterlegt.
Diese Einstellung findet man im SCADA-Editor unter







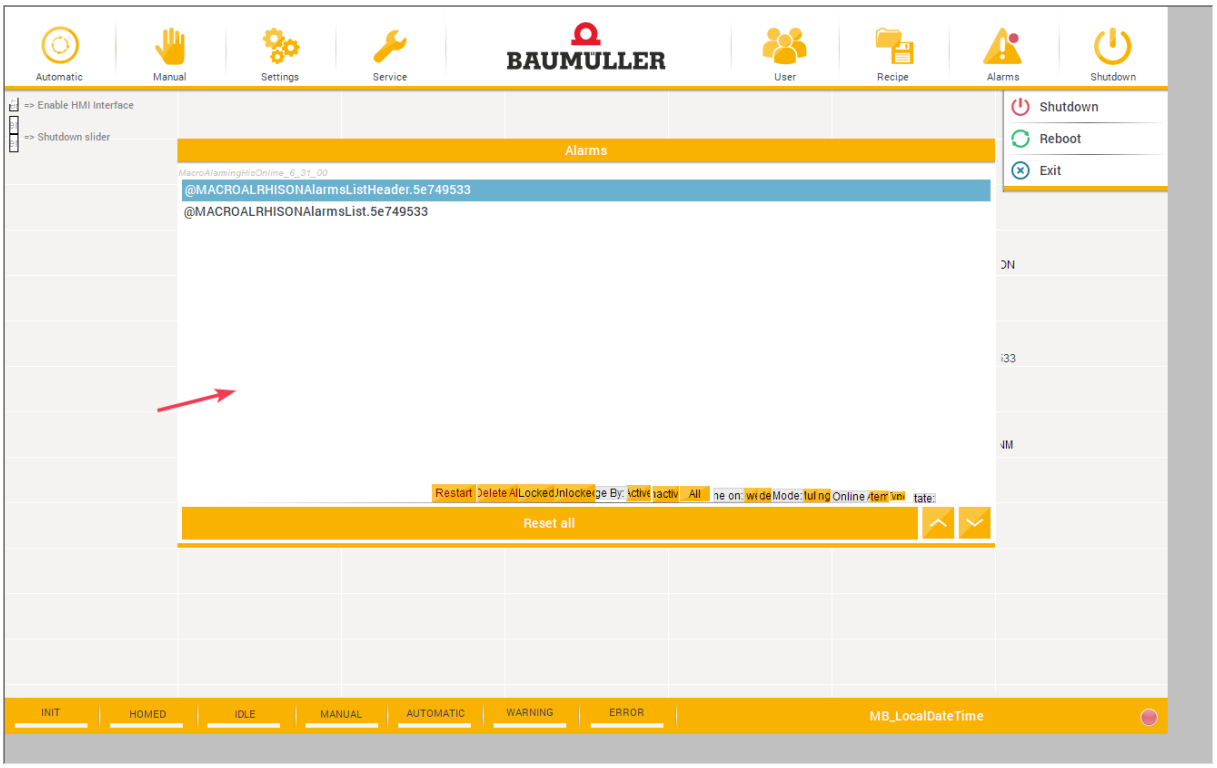
Den dazugehörigen Fehlertext findet man in den Sprachdateien:



Hier dann der Text für die „ALRLIST0“ mit der Fehler-ID 1:

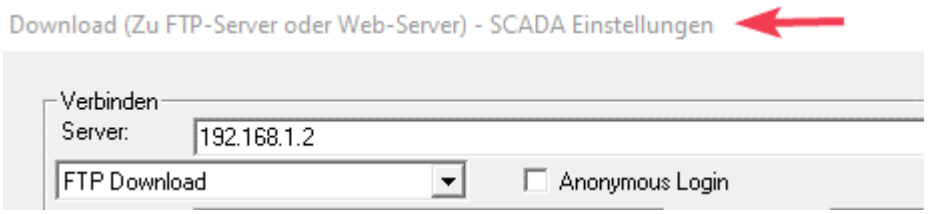
```
=> Shutdown slider;=> Shutdown sl
ABC;ABC
ALRLIST0_1;Antriebsfehler
ALRLIST0_10;ALRLIST0_10
```

Um das Alarming-Makro zu bearbeiten, genügt es einen Doppelklick auf das Alarmfenster zu machen:



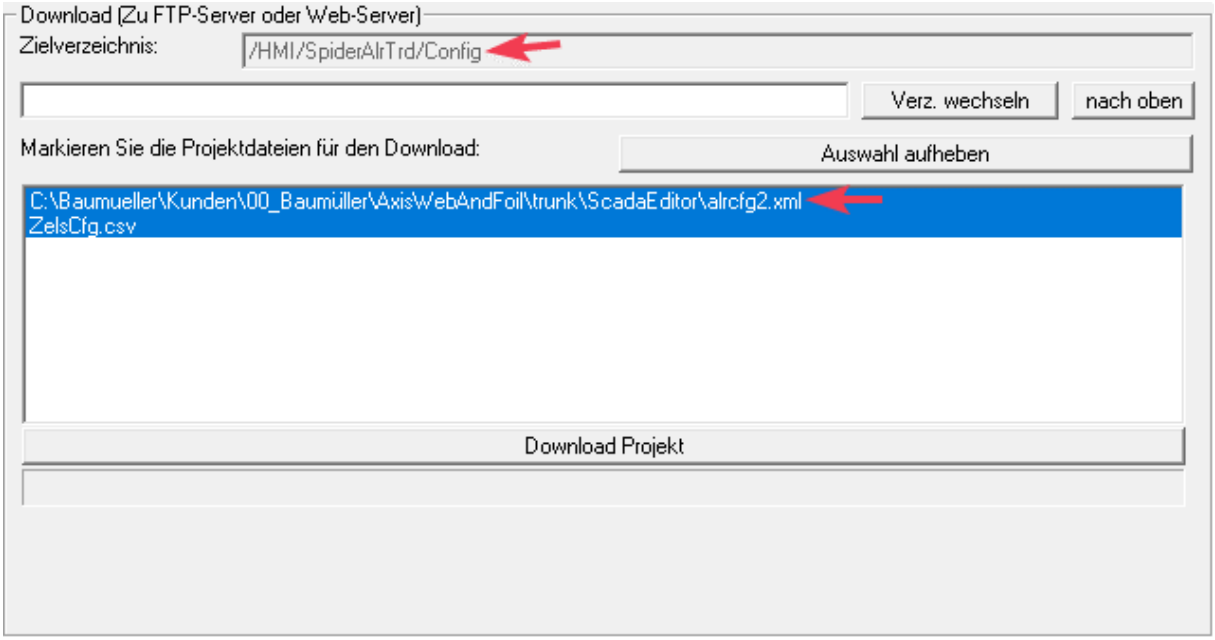
Damit gelangt man in das Konfigurationsfenster für die Einstellungen dieses Makros. Hier kann man z.B. den Alarmfilter, oder die Farben für aktive/inaktive Alarme anpassen.

Wichtig für die Alarmfunktion ist beim Projektdownload der zweite Downloaddialog, der sich öffnet, sobald der erste Download abgeschlossen ist:



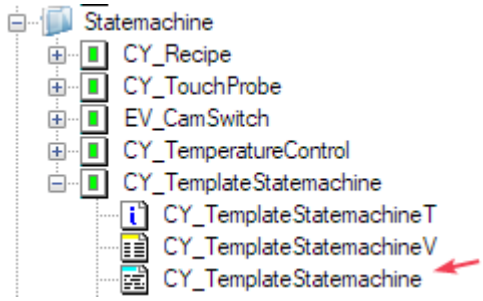
Login-Daten:
Benutzer: BM_ProViz
Passwort: 12345678

Hier sieht man auch die Einstellungen welche gesetzt sein müssen, damit das Alarming auf dem Zielsystem in das korrekte Verzeichnis kopiert wird und die Datei, welche letztendlich die Alarmkonfiguration enthält:



11.5.2 **PLC**

Die Variable „_Iface._Sts._Err[0]“ wird in der PLC für einen Antriebsfehler genutzt. Das Setzen der Variable findet hier statt:



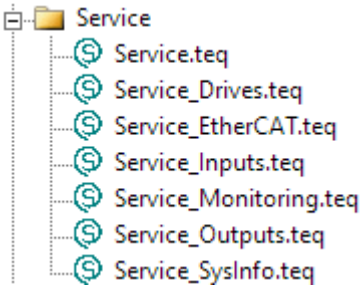

```
(* -- DRIVE ERROR ------------------------------------------------------------ *)

(* Check all drives for error *)
FOR i_AxisIndex := 1 TO i_LastAxisIndex DO
   x_Clk_RTrig_1 := _Tech_TMPL._DriveReadError[i_AxisIndex]._Out.x_DriveError;
   IF x_Clk_RTrig_1 THEN
      EXIT;
   END_IF;
END_FOR;

(* Trigger Error *)
R_TRIG_1(CLK:= x_Clk_RTrig_1);

(* Check if initialization is done *)
IF R_TRIG_1.Q AND i_State > 0 THEN
   i_State := 800;
   _Iface._Sts._Err[0] := TRUE;   (* Alarm *)
END_IF;
```

11.6 Service/Diagnose

Folgende Service-, und Diagnosemöglichkeiten sind in dem Template vorhanden:

- Antriebsdiagnose, um Fehlerdetails auszuwerten
- EtherCAT-Diagnose
- Anzeige und Forcen der Zustände der digitalen Eingänge
- Anzeige und Forcen der Zustände der digitalen Ausgänge
- Trend-Anzeige mit Sinus und Cosinus
- Systeminformationen

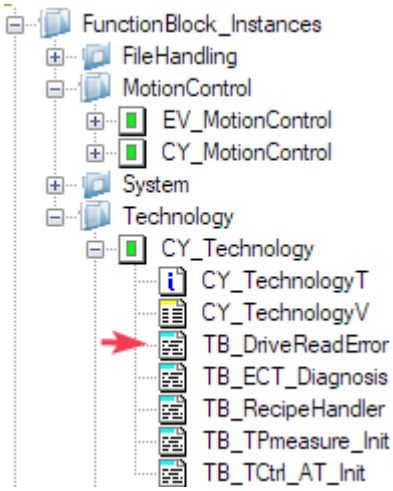
Diese Funktionen sind im HMI-Projekt in dem Unterverzeichnis „Service“ zu finden:



Notiz!	
	Im PLC-Projekt gibt es für keine der Seiten eine Statemachine. Die Informationen auf der Seite „Service_Sysinfo“ werden auch rein über CONTAINER-Variablen realisiert.

11.6.1 Antriebsdiagnose

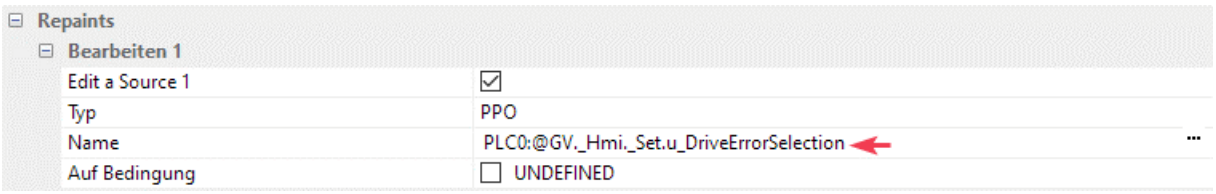
Die Antriebsdiagnose wird über den Funktionsbaustein „TB_DriveReadError“ realisiert. Dieser befindet sich hier:



Die Informationen werden in der POE „CY_HMI_Out“ an die Visualisierung übergeben:

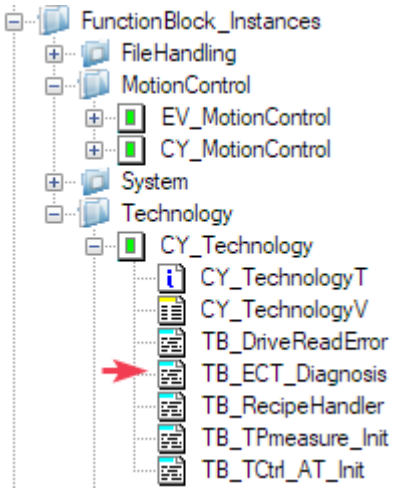
```
(* Drive Erros *)
u_DriveErrorSelection := _Iface._Set.u_DriveErrorSelection;

_Iface._Act.u_NmbrOfDriveErrors     := INT_TO_UINT(_Tech_TMPL._DriveReadError[u_DriveErrorSelection]._Out.a_ErrorList[0]);
_Iface._Sts.x_DriveError            := _Tech_TMPL._DriveReadError[u_DriveErrorSelection]._Out.x_DriveError;
_Iface._Sts.x_DriveWarning          := _Tech_TMPL._DriveReadError[u_DriveErrorSelection]._Out.x_DriveWarning;
_Iface._Act.a_DriveErrors           := _Tech_TMPL._DriveReadError[u_DriveErrorSelection]._Out.a_ErrorList;
```

In Abhängigkeit der Variable „u_DriveErrorSelection“ wird die dazugehörige Bausteininstanz ausgewählt:



11.6.2 **EtherCAT-Diagnose**

Die EtherCAT-Diagnose wird über den Funktionsbaustein „TB_ECT_Diagnosis“ realisiert. Dieser befindet sich hier:

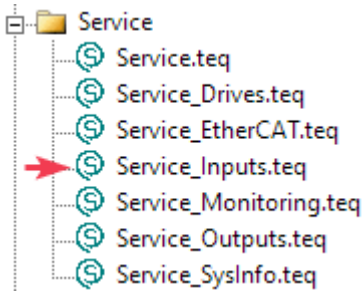


Die Informationen werden in der POE „CY_HMI_Out“ an die Visualisierung übergeben:

```
(* EtherCAT Diagnosis *)
_Iface._Sts._EtherCAT.x_EtherCAT_OK          := _Tech_TMPL._EctDiagnosis[0]._Out.x_ECT_is_OK;
_Iface._Act._EtherCAT.s_ECT_Kernel_ErrorInfo := _Tech_TMPL._EctDiagnosis[0]._Out.s_ECT_Kernel_ErrorInfo;
_Iface._Act._EtherCAT.u_ECT_Slaves_inOP      := _Tech_TMPL._EctDiagnosis[0]._Out.u_ECT_Slaves_inOperational;
_Iface._Act._EtherCAT.u_ECT_SlavesFound      := _Tech_TMPL._EctDiagnosis[0]._Out.u_ECT_Slaves_Found;
_Iface._Act._EtherCAT.w_ECT_KernelVersion    := _Tech_TMPL._EctDiagnosis[0]._Out.w_ECT_Master_KernelVersion;
```

11.6.3 Anzeige und Forcen der Zustände der digitalen Eingänge

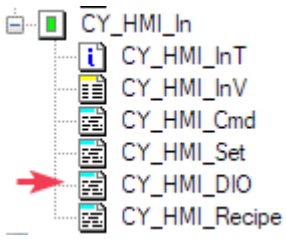
Die digitalen Eingänge (z.B. eines E-Bus Koppelmoduls) können über die Seite „Service_Inputs.teq“ in der Visualisierung gesteuert und ausgewertet werden:



Hier gibt es für jeden Eingang eine Anzeige, für dessen Hardwarestatus und der logische Status:

Der logische Status entspricht dem Hardwarestatus, solange der Eingang auf „AUTO“ geschaltet ist. Wird er über „ON“ oder „OFF“ manuell gesetzt, dann wird der Hardware-Eingang nicht mehr auf den logischen Eingang kopiert.

In dem PLC-Projekt werden die digitalen Eingänge hier verwaltet:



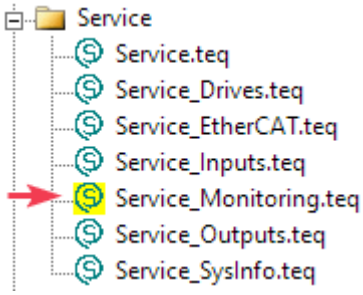

```
(* Digital Input 1 *)
CASE _Iface._Cmd._Node[1]._DI_LOG[1] OF
   0:   _Iface._Sts._Node[1]._DI_LOG[1] := BOOL_TO_INT(ix_01_01_01_HW_Sts);(* 0 = PLC controlled *)
   1:   _Iface._Sts._Node[1]._DI_LOG[1] := 1;                              (* 1 = Forcing ON     *)
   2:   _Iface._Sts._Node[1]._DI_LOG[1] := 2;                              (* 2 = Forcing OFF    *)
END_CASE;
```

11.6.4 Anzeige und Forcen der Zustände der digitalen Ausgänge

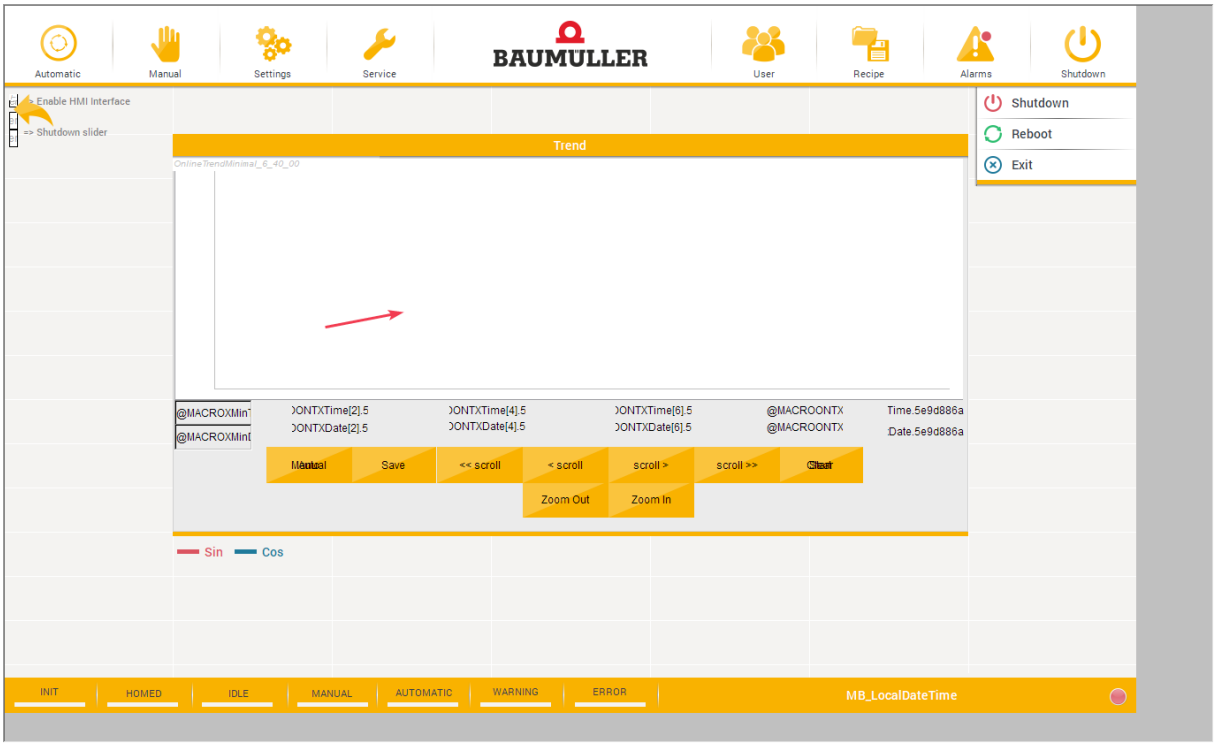
Die Funktion entspricht der, der digitalen Eingänge, bis auf das es bei den digitalen Ausgängen keinen logischen Ausgang gibt.

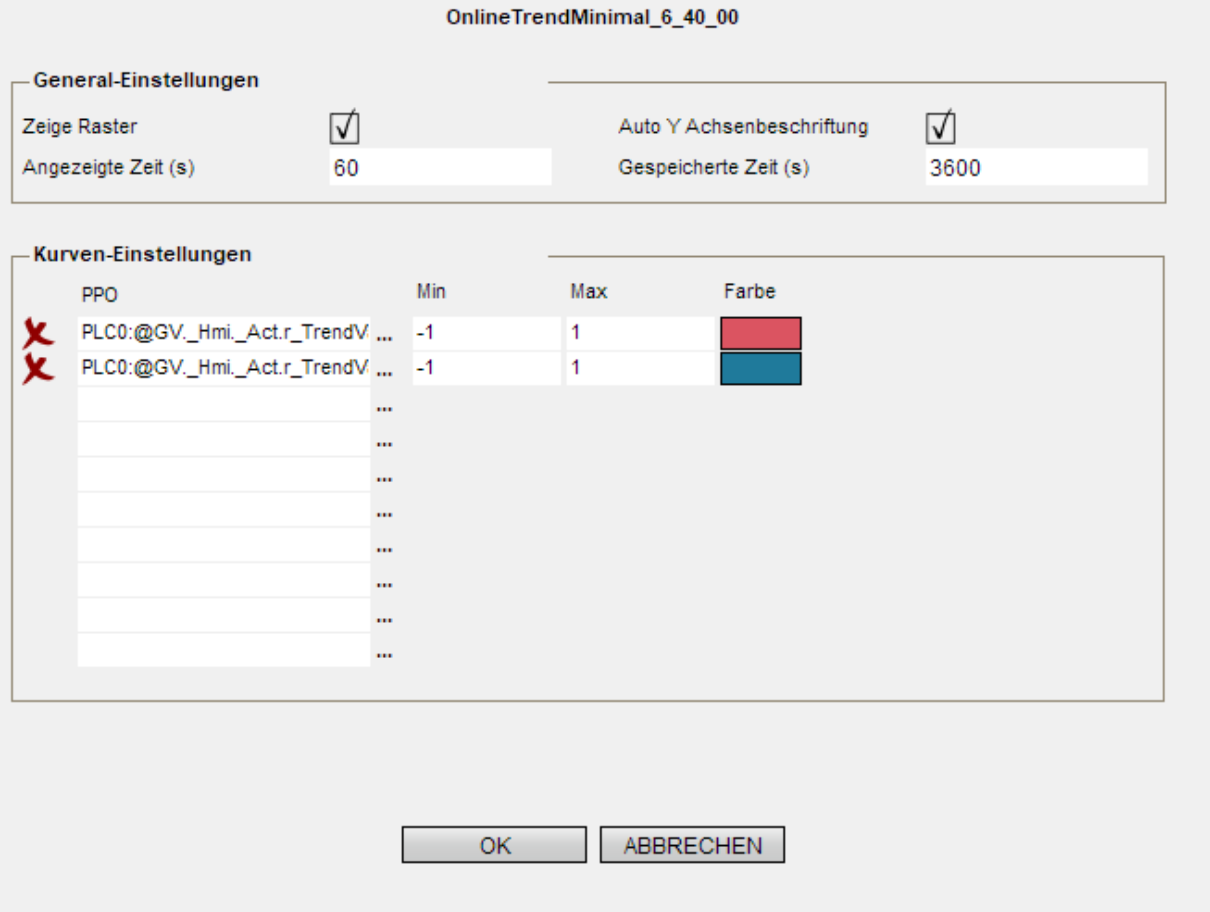
11.6.5 Trend-Anzeige mit Sinus und Cosinus

Die Trendanzeige findet man im HMI-Projekt unter „Service“ > „Service_Monitoring“:



Hier ist ein Online-Trend-Makro instanziiert, welches durch einen Doppelklick die Einstellungen anzeigt:





Hier sind die Trendvariablen zu sehen, welche mit dem Makro verbunden sind. Die Farbe der einzelnen Linien, sowie die Skalierung kann hier angepasst werden.

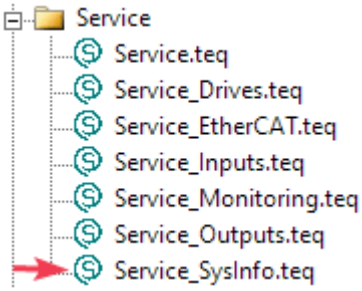
Im PLC-Projekt gibt es hierfür in der POE „CY_HMI_Out“ und den gleichnamigen Arbeitsblatt einen Bereich, welcher die Trendvariablen berechnet und in das Interface kopiert:

```
(* TREND EXAMPLE *)
r_Rad := r_Rad + 0.01;

IF r_Rad >= (r_2PI) THEN
   r_Rad := 0.0;
END_IF;

_Iface._Act.r_TrendValues[1] := SIN(r_Rad);   (* Sin *)
_Iface._Act.r_TrendValues[2] := COS(r_Rad);   (* Cos *)
```

11.6.6 Systeminformationen

Die Systeminformationsseite beinhalt Informationen über die Version des Microbrowsers, das Betriebssystem auf dem dieser läuft, die HMI-Version (welche einstellbar ist) und ebenfalls Informationen zu den Netzwerkeinstellungen auf dem System.

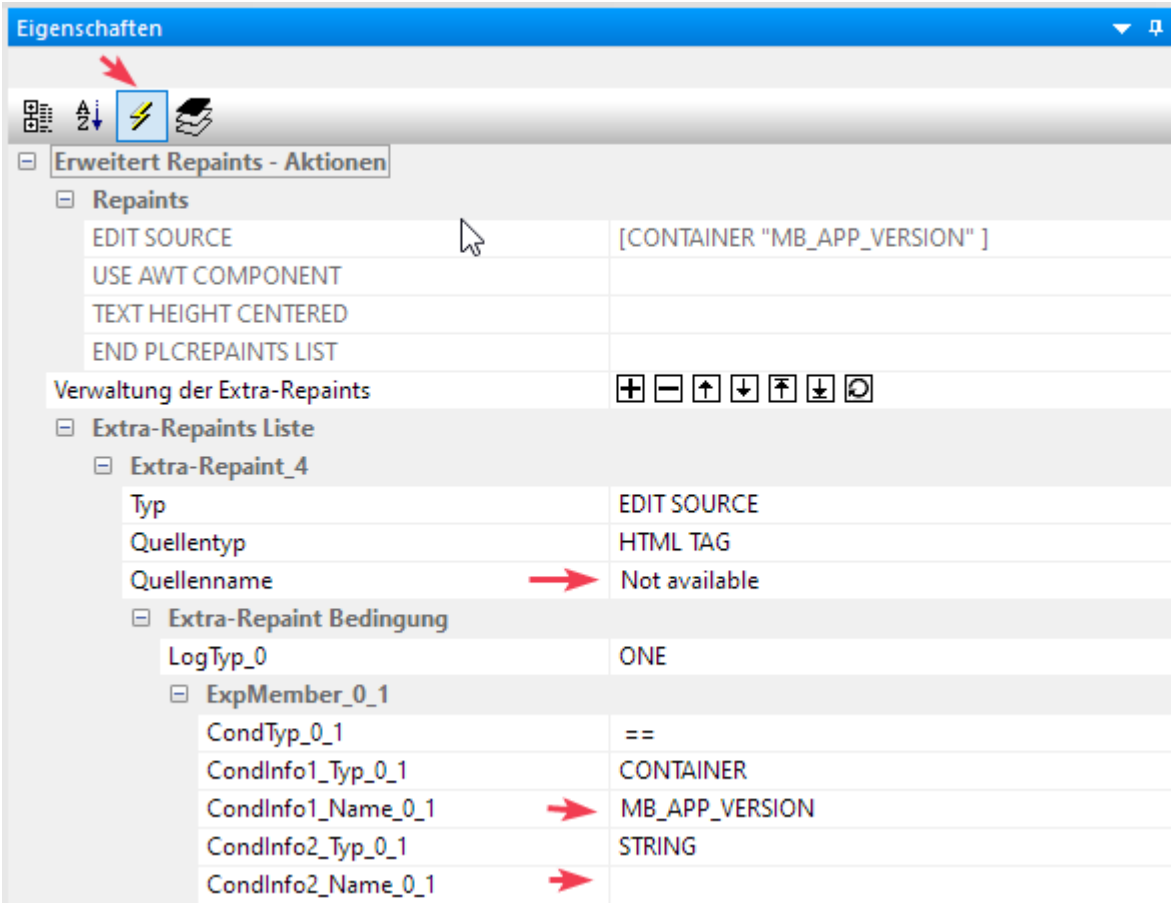
Die Seite der Systeminformationen ist im SCADA-Editor-Projekt hier zu finden:



Die hier angezeigten Variablen zeigen in der Entwicklungsumgebung alle nur „Not available“ an. Das liegt daran, dass das ein Wert ist, welche die Variablen annehmen können, wenn der Wert auf dem Zielsystem nicht abgerufen werden kann:

	System
Microbrowser-Version:	Not available

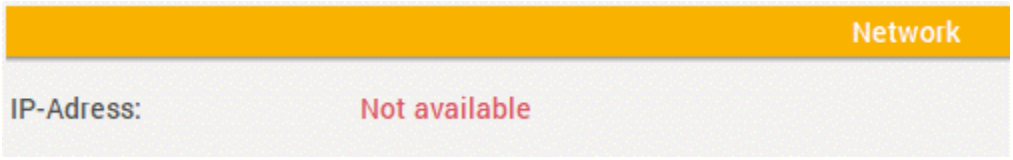
In den Eigenschaften unter den erweiterten Repaints (das gelbe Blitz Icon) sieht man genauer, was hier passiert. Hier ist die Bedingung gesetzt, wenn die Container-Variable „MB_APP_VERSION“ gleich eines leeren Strings ist, dann soll „Not available“ angezeigt werden.



Das „Extra-Repaint_5 sorgt dann noch für eine Farbumschaltung:

⊟ Extra-Repaints Liste	
⊞ Extra-Repaint_4	
⊟ Extra-Repaint_5	
Typ	USE OUTLINE COLOR
Quellentyp	STRING
Quellenname	219,84,97
⊟ Extra-Repaint Bedingung	
LogTyp_0	ONE
⊟ ExpMember_0_1	
CondTyp_0_1	==
CondInfo1_Typ_0_1	CONTAINER
CondInfo1_Name_0_1	MB_APP_VERSION
CondInfo2_Typ_0_1	STRING
CondInfo2_Name_0_1	

Zur Laufzeit sieht das dann so aus:

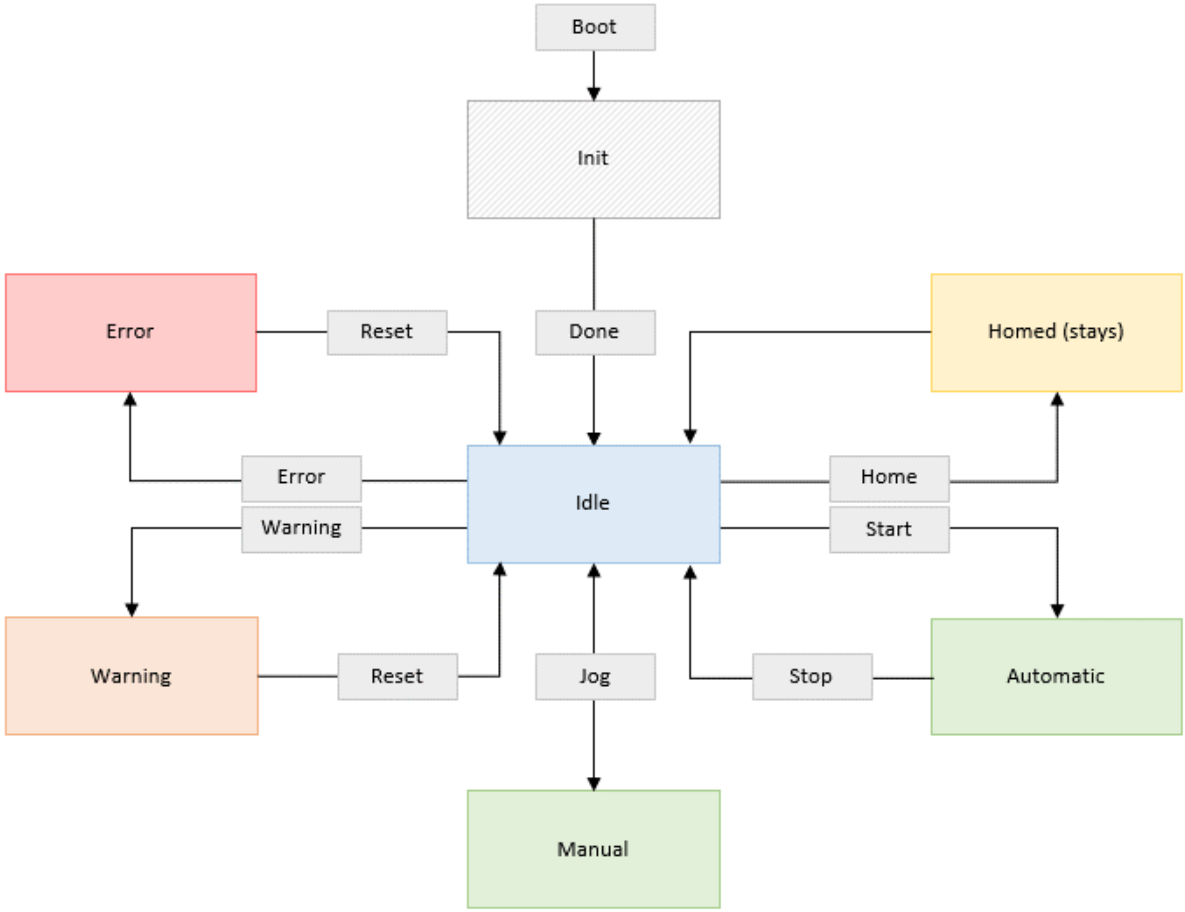


Hier wird mit CONTAINER-Variablen gearbeitet, welche in der Hilfe des Editors zu finden sind.

12. Statemachine

12.1 Allgemeine Beschreibung

In der Statemachine wird der Zustand der Maschine abgebildet. Anhand folgender Grafik wird die Statemachine, welche im Template integriert ist, näher erläutert:



Notiz!
Der Zustand „Homed“ bleibt nach dem Referenzieren für die Anzeige erhalten. Der Zustand „Warning“ ist lediglich in der Visu als Anzeige vorhanden, allerdings nicht ausprogrammiert.

In der Visualisierung wird die Statemachine in der Fußzeile angezeigt:

12.2 Homing

Die Referenzierung der Achse wird gestartet, sobald man in der Visualisierung den Button „Referenzieren“ in der Sidebar betätigt. Hierbei wird als Beispiel die Referenziermethode „Referenzpunkt setzen“ im Antrieb ausgeführt. Dabei wird auf den Funktionsbaustein „MC_Homenit_SetPosition“ zurückgegriffen. Dies kann natürlich nach Belieben geändert werden.
Im Zustand „S100_Idle“ sieht man was passiert, wenn das Homing, bzw. die Referenzierung gestartet wird:

```
(* -- Home -- *)
IF _Iface._Cmd.x_Home AND x_PoweredOn THEN
   i_State               := 300;
END_IF;
```

Zuerst wird die Referenzierungsmethode im Antrieb gesetzt:

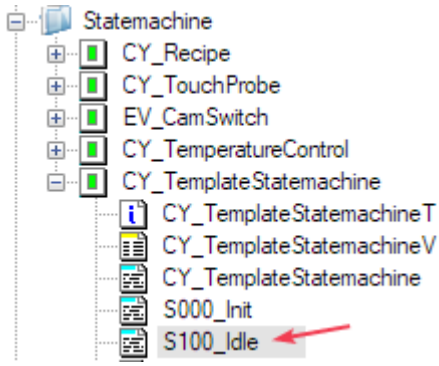
```
(* ------------------------------------------------------------------------ *)
300: s_State := '300: Enable MC_HomeInit_SetPos';
(* ------------------------------------------------------------------------ *)
```

Anschließend wird mit dem Funktionsblock „MC_Home“ das Homing ausgeführt:

```
(* ------------------------------------------------------------------------ *)
310: s_State := '310: Enable MC_Home';
(* ------------------------------------------------------------------------ *)
```

12.3 Automatic

Die Statemachine geht automatisch in den Automatikmodus sobald der „Start“-Button in der Steuerleiste betätigt wurde. Es muss hier also nicht manuell in den Automatikmodus umgeschaltet werden. In der Statemachine im State „S100_Idle“ kann man sehen, was passiert, sobald der Startbutton betätigt wurde und welche Bedingung damit verknüpft ist:




```
(* -- Start -- *)
IF _Iface._Cmd.x_Start AND x_PoweredOn THEN

   IF _Iface._Sts.x_GearUsed THEN
      i_State         := 600;                    (* Use MC_GearIn  *)
   ELSE
      i_State         := 500;                    (* Use MC_CamIn   *)
   END_IF;

END_IF;
```

Hier sieht man, dass je nachdem ob das Gear in der Visualisierung aktiviert wurde, oder nicht, in unterschiedliche Zustände gesprungen wird.

12.3.1 Gear

Wird das Gearing aktiviert, so wird erst die MasterEngine gestartet und anschließend der Baustein MC_GearIn, welcher die Slave-Achse an den Master mit einem voreingestellten Getriebefaktor von 1 koppelt:

```
(* ------------------------------------------------------------------------ *)
620: s_State := '620: Enable MC_GearIn';
(* ------------------------------------------------------------------------ *)

   (* -- Step Command -- *)
   FOR i_AxisIndex := 1 TO i_LastAxisIndex DO
      a_Axis_TMPL[i_AxisIndex]._GearIn._In.u_RatioDenominator := UINT#1000;
      a_Axis_TMPL[i_AxisIndex]._GearIn._In.i_RatioNumerator   := INT#1000;
      a_Axis_TMPL[i_AxisIndex]._GearIn._In.x_Execute          := TRUE;
   END_FOR;
```

12.3.2 Cam

Wird das Gearing in der Visualisierung nicht aktiviert, dann wird die Achse über den Funktionsbaustein „MC_CamIn“ an die MasterEngine gekoppelt. Auch hier ist ein Getriebefaktor von 1 per default voreingestellt:

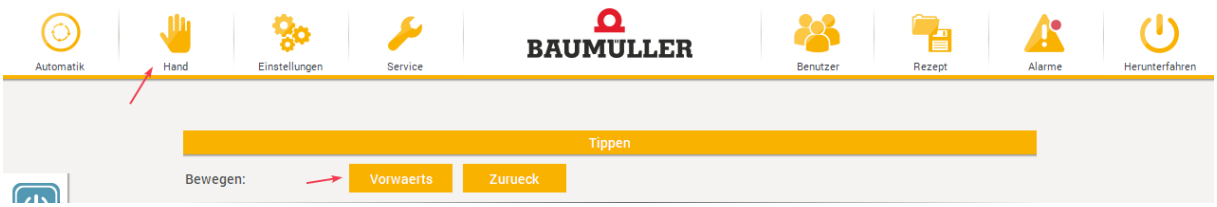
```
(* ------------------------------------------------------------------------ *)
520: s_State := '520: Enable MC_CamIn';
(* ------------------------------------------------------------------------ *)

   (* -- Step Command -- *)
   FOR i_AxisIndex := 1 TO i_LastAxisIndex DO
      a_Axis_TMPL[i_AxisIndex]._CamIn._In.u_CamTableID          := _Iface._Pro.u_CamTableID;
      a_Axis_TMPL[i_AxisIndex]._CamIn._In.u_MasterScaling       := UINT#1000;
      a_Axis_TMPL[i_AxisIndex]._CamIn._In.u_SlaveScaling        := UINT#1000;
      a_Axis_TMPL[i_AxisIndex]._CamIn._In.x_Execute             := TRUE;
   END_FOR;
```

Dieser kann natürlich jederzeit geändert werden.
Hier wird die CamTableID eingestellt, welche in der Visualisierung gesetzt ist. Da es allerdings keine CamTable mit der ID 0 gibt, wird hier in der POE „CY_HMI_In/CY_HMI_Set“ eine Überprüfung der Eingabe gemacht und ggf. der Wert auf 1 gesetzt:

```
(* Validate Cam ID *)
IF _Hmi._Pro.u_CamTableID < UINT#1 THEN
   _Iface._Pro.u_CamTableID := UINT#1;
ELSE
   _Iface._Pro.u_CamTableID := _Hmi._Pro.u_CamTableID;
END_IF;
```

12.4 Manual

Der Manual-Zustand wird aktiv, sobald die Achse über die Seite „Hand“ in der Visualisierung getippt wird. Auch hier muss, analog zum Automatikmodus, keine manuelle Umschaltung in den Handmodus erfolgen. Natürlich kann das allerdings später noch angepasst bzw. geändert werden.



In der Statemachine in der PLC kann man unter dem Arbeitsblatt „S100_Idle“ sehen was genau passiert, sobald die Achse getippt wird:

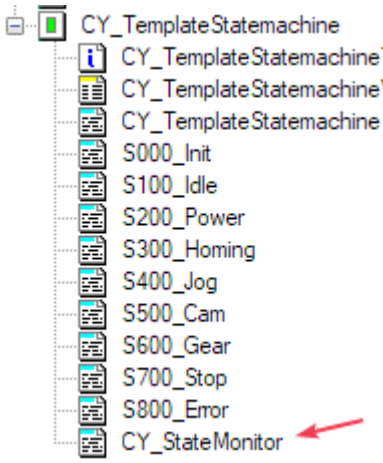
```
(* -- Jog -- *)
IF _Iface._Cmd.x_Jog_Fwd OR _Iface._Cmd.x_Jog_Bwd AND x_PoweredOn THEN
   i_State            := 400;
END_IF;
```

In dem Arbeitsblatt „S400_Jog“ findet man dann den dazugehörigen Prozess. Hier wird der Funktionsbaustein „BM_Jog_EV“ genutzt um die Achse zu tippen:

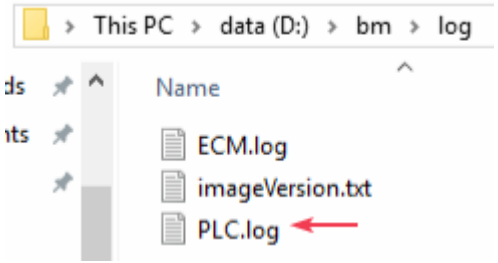
```
(* ------------------------------------------------------------------------------ *)
400:  s_State := 'MANUAL - 400: Enable BM_Jog_EV';
(* ------------------------------------------------------------------------------ *)
```

12.5 Statemonitor

Mit dem Statemonitor ist es möglich, alle durchlaufenen States zu überwachen und diese auch auf der PLC in einer Datei mitzuloggen. Den Statemonitor findet man im letzten Arbeitsblatt der „CY_TemplateStatemachine“ und der „CY_TemperatureControl“ POE’s:



Letztendlich handelt es sich hierbei um einen Baustein, welcher viele Funktionen zum Loggen der Aktivitäten einer Statemachine bietet. Details können aus den Kommentaren, bzw. aus der Bausteinhilfe entnommen werden.
Die Log-Dateien können auf der PLC in folgendem Verzeichnis gefunden werden:


This PC
data (D:)
bm
log
Name
ECM.log
imageVersion.txt
PLC.log