



BAUMÜLLER

Operation manual Template Motion-Axis Web and Foil

Brief description:

The Motion-Axis Web and Foil template consists of a ProMaster project, which contains a prepared PLC project and an HMI project. In the basic configuration there is an axis with which the template can be operated. This can of course be expanded as you wish. A large number of functions are prepared here that are suitable as the basis of a complete application project for a web-carrying machine. The aim is to be able to concentrate on the programming of the process with this template.

Version: 2.0 from 28.04.2023

Status: Release

Author: [VT]

1. Contents

2.	History	3
3.	Introduction.....	3
4.	Definition of terms	3
5.	Notes relevant to safety.....	4
6.	Requirements	5
7.	System components	5
7.1	System structure basic components	5
8.	Overview visualization.....	6
8.1	Layout and function.....	6
8.2	Navigation bar	6
8.3	Function keys.....	6
8.4	Status bar	7
8.5	Control bar	8
8.6	Machine condition.....	8
8.7	Communication status	9
9.	HMI project	10
9.1	Content and components.....	10
10.	ProProg project.....	11
10.1	Libraries.....	11
10.2	Data types.....	11
10.3	Logical POUs.....	17
10.4	Tasks.....	20
11.	Machine functions.....	22
11.1	User level management	22
11.2	Recipe management.....	23
11.3	Touch-Probe.....	25
11.4	Cam switch	30
11.5	Alarming	36
11.6	Service / diagnosis.....	40
12.	State machine.....	46
12.1	General description	46
12.2	Homing	47
12.3	Automatic.....	47
12.4	Manual	48
12.5	State monitor	49

2. History

version	date	Surname	modification
1.0	11/04/2021	VT	Initial version
2.0	04/28/2023	VT	Added QR-Codes

3. Introduction






This document provides an overview of the existing software functions of the Motion-Axis Web & Foil technology solution. For better illustration, functions are partly shown and explained using illustrations of the user interface.

Technical information on the hardware can be found in the corresponding operating instructions or parameter manuals on the [Baumüller website](https://www.baumueller.de) can be found in the Service \ Downloads area.

4. Definition of terms

expression	definition
HW	hardware
POE	Program organization unit
PLC	Programmable logic controller
TEQ	Design page of the HMI project

5. Notes relevant to safety

DANGER	
	... indicates an imminently dangerous situation that will lead to death or serious injuries if it is not avoided.
WARNING	
	... warns of a possibly dangerous situation that can lead to death or serious injuries if it is not avoided
ATTENTION	
	... warns of a possibly dangerous situation that can lead to slight or minor injuries if it is not avoided
A NOTICE	
	... warns of a potentially dangerous situation that can lead to property damage if it is not avoided
Note!	
	... indicates useful tips and recommendations as well as information for efficient, trouble-free operation

6. Requirements

The following software and hardware components are required so that the template can be operated:

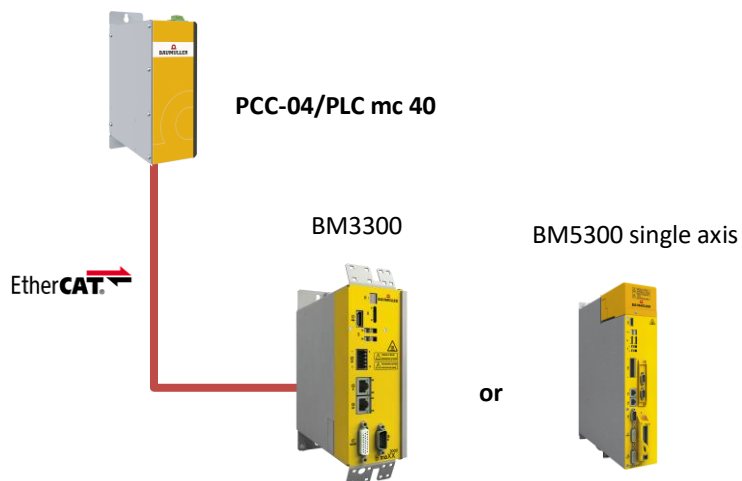
- SCADA editor (from V7.02.00.20)
- ProMaster (from 1.21.1.34 - release)
- PCC-04/PLC mc 40
- bmaXX3300 or bmaXX5000 as a single axis

7. System components

This section describes the system structure.

7.1 System structure basic components

The template in its basic configuration contains the configuration for the following components:



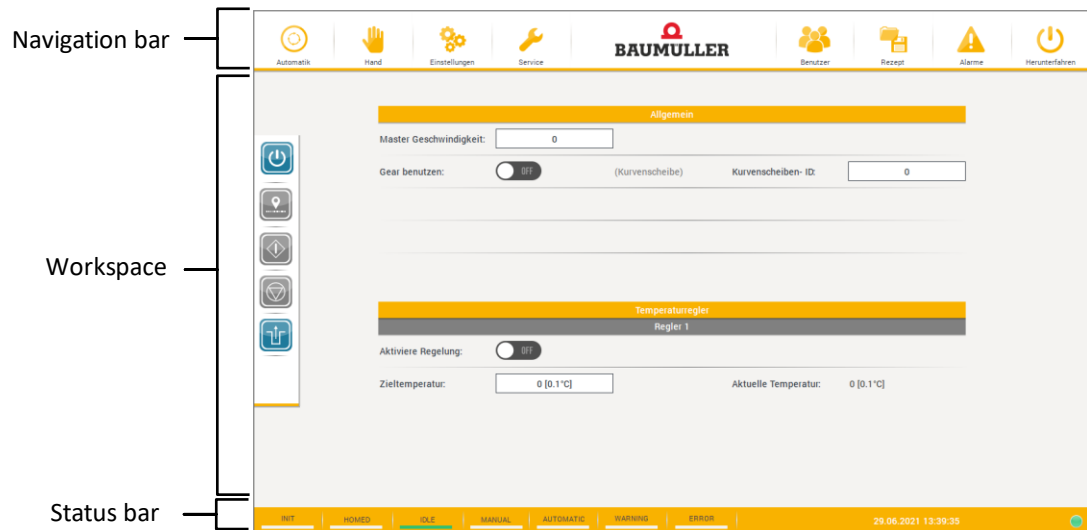
The template is delivered with this system configuration and it is recommended to first start up the template with these basic components in order to check and get to know the basic functions. The template can of course be expanded later.



8. Overview visualization

8.1 Layout and function

Each page of the user interface is divided into 3 parts:











8.2 Navigation bar



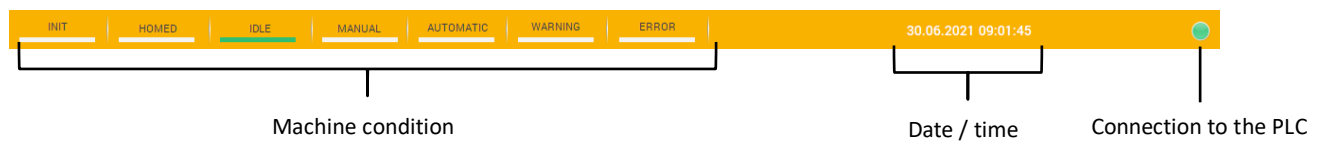
At the top of the user interface is the navigation bar, which you can use to access the individual machine functions, such as automatic and manual operation. The content of this bar is always the same, regardless of the machine function called.

8.3 Function keys

Symbol	Description
 Automatic	Automatic mode: Starting the virtual master and the slave axis via a gear or a cam. Display of the temperature controller.
 Manual	Manual: Moving the individual machine axes by hand
 Settings	Settings: Settings for e.g. recipes, language, cams, touch probes, temperature controllers

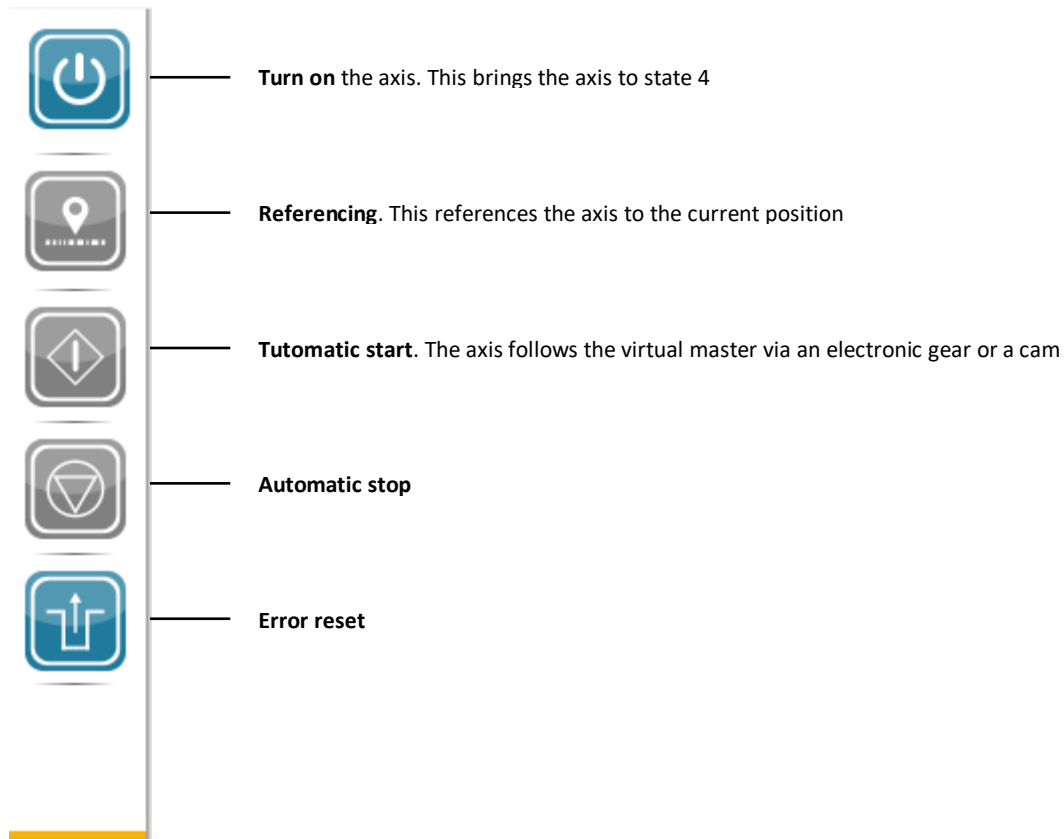
 Service	Service: Diagnostic functions of the machine
 User	User: Login with the corresponding password to activate the different operator levels
 Recipe	Recipe: The recipes can be managed here
 Alarms	Alarms: Alarm history of the individual error messages with time stamp (occurred, resolved)
 Shutdown	Shut down: enables the machine to be switched off or restarted

8.4 Status bar



The status bar provides information about the current status of the machine. In addition, the current date and time are displayed, as well as whether the connection to the PLC is active.

8.5 Control bar





8.6 Machine condition

The machine status is displayed in the status bar of the visualization. There are following states:

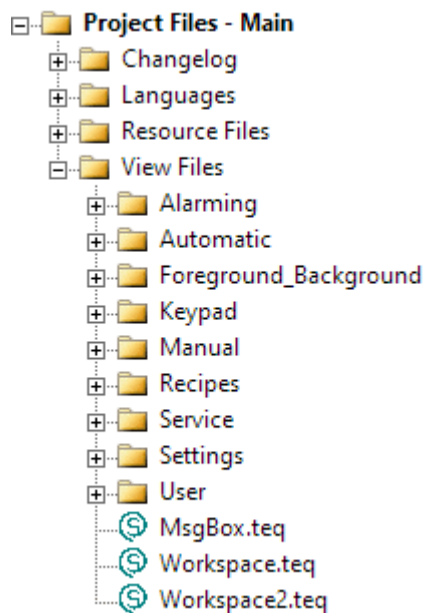
Status	Description
INIT	The state machine is being initialized. This is where initial values are set in the controller
HOMED	Indicates whether the machine is referenced
IDLE	Indicates whether the machine is idling. From this state, actions can be initiated that result in a movement of the axis.
MANUAL	Manual mode / jog active or inactive
AUTOMATIC	Automatic mode active
WARNING	There is at least one warning pending
ERROR	There is at least one error

8.7 Communication status

symbol	description
	Status: Communication with the controller is offline. (No data exchange)
	Status: Communication with the control is online (data exchange active)

9. HMI project

9.1 Content and components



Directory	Brief description
+ Changelog	A changelog can be kept here for version management
+ Language	Contains all language files
+ Resource Files	Contains program-specific files which should not be edited by the user!
- View files	All files in which the visualization interface is designed are stored here.
+ Alarming	Display of alarms / error messages
+ Automatic	Display of the automatic page
+ Foreground_Background	Foreground and background side of the visualization: Foreground => notes, messages, control bar Background => navigation bar, status bar
+ Keypad	The appearance of the mouse and keyboard is designed here A notice: These files are only used by the microbrowser if there are no "keypad.teq" and no "alphapad.teq" in the directory of the microbrowser on the target system!
+ Manual	Display of the page for manual operation
+ Recipes	Display of the recipe management
+ Service	Views for the diagnostic functions (EtherCAT, drive, system)
+ Settings	Views for the settings (cam controller, temperature control)
+ User	Login page for user level management
Workspace / 2	The pages created here can be used as a "drawing area" if you simply need an empty drawing area.

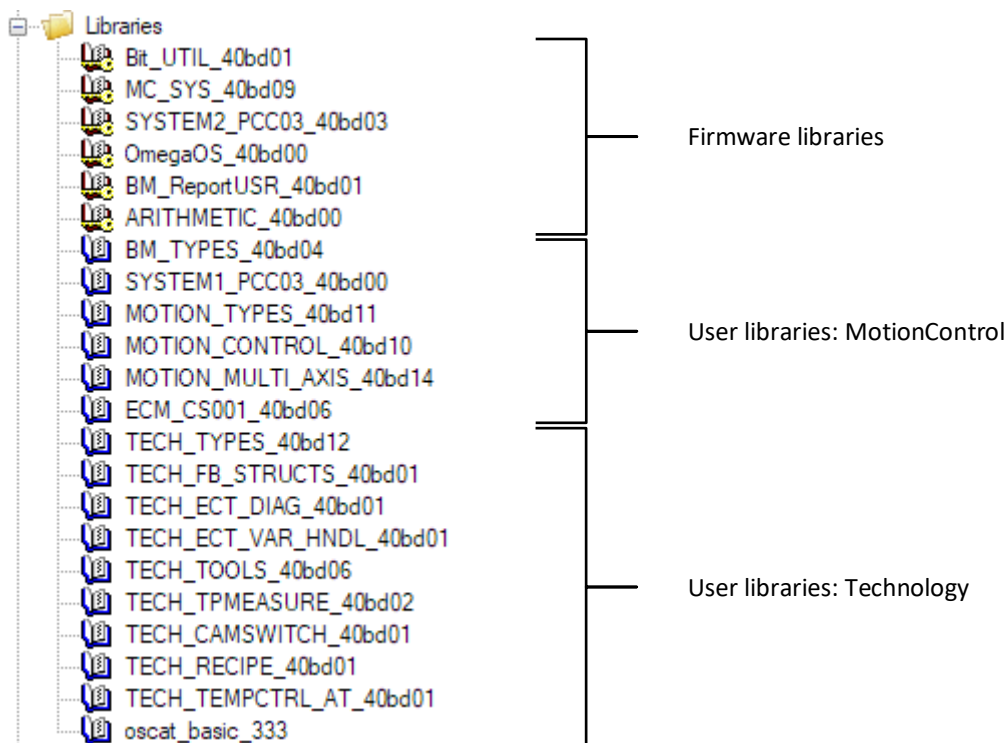
10. ProProg project

Note!

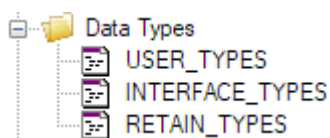


Every single function of the function blocks within these POU's is not discussed here. The individual functions can be looked up in the respective module help. However, in the machine function-specific chapters, the relationship with the individual blocks is discussed in more detail.

10.1 Libraries

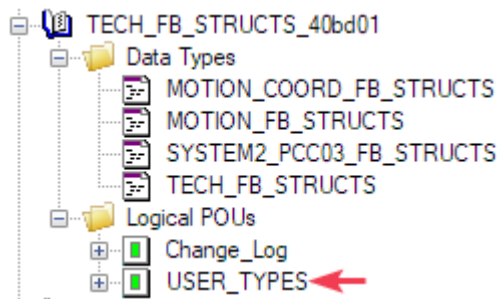


10.2 Data types



10.2.1 USER_TYPES

The data types contained here are copied from the "TECH_FB_STRUCTS" library:



This data type worksheet contains all the data types required by the user with which he can determine the number of the individual structure data types belonging to the blocks. The number is grouped in the form of arrays and can be changed flexibly. Put simply, this means the following:

Example "TB_RecipeHandler"

Block instance 1 => connection with data type structure array 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 Block instance no.
- 2 Field index in the data type structure
- 3 Maximum number of block instances that can be used in the project

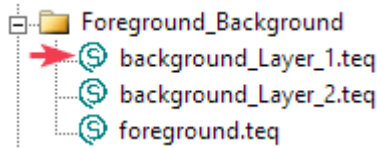
10.2.2 INTERFACE_TYPES

All data types that are used for communication with the visualization are located here. The top structure level is divided into the following sub-areas:

```
(* 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;
```

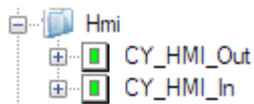
10.2.2.1 x_Enable_Hmi_Interface

This tag enables communication between the HMI and the PLC. This tag is also the source of the display for active communication in the status bar of the HMI. The variable is written cyclically in the visualization during runtime by a repaint event. This happens in the "background_Layer_1.teq":



=> Enable HMI Interface

A double click on the small white box opens the settings of this painter. This means that the HMI interface is automatically activated at runtime when the visualization is started. This variable is then used in the HMI POU's as a condition for reading / writing:



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

10.2.2.2 _Cmd

Direction: HMI => PLC

All commands that go from the HMI to the PLC are in this structure. This is, for example, the command for switching on the axes, an error reset, or jogging.

This structure is structured in such a way that there is an area whose tags are automatically reset after a PLC cycle and one in which this does not happen. This can be seen in the comments:

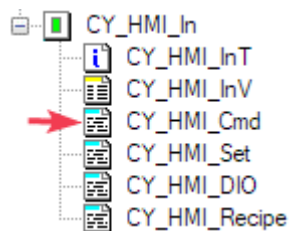
```

(* 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;

```

Explained in more detail using the example "x_PowerOn":

If the "x_PowerOn" signal is set to TRUE by the visualization, this is recognized in the PLC and reset directly. This means that there is a clean handshake. This step is done in the POU "CY_HMI_In" in the worksheet "CY_HMI_Cmd":

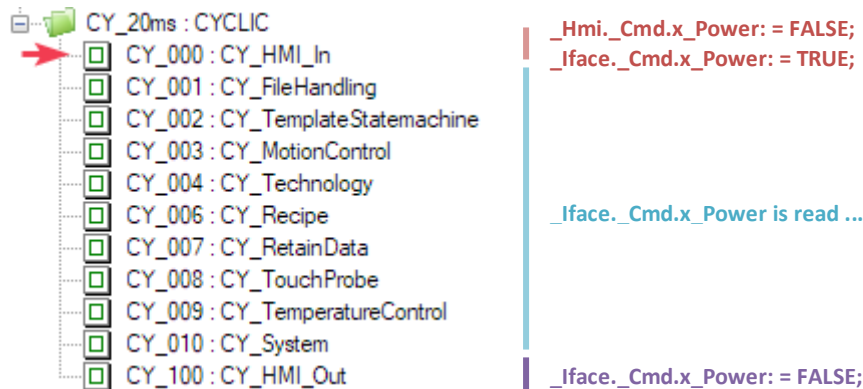


```

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

```

The structure variable "_Iface ..." is then set. This all happens in the first program instance of the cyclic task:



In the program instances after "CY_HMI_In", the interface variable "_Iface._Cmd.x_PowerOn" is used. This is then finally reset AUTOMATICALLY in the last program instance "CY_HMI_Out" (in the worksheet "CY_HMI_Reset"). This is done with a memory management module. The programmer no longer has to make any adjustments here.

```
(* 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;

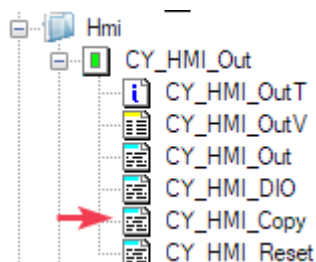
(* Set to 0 *)
w_Memset := MEMSET(i_Memset_Err, b_Memset_Val, uint_to_dint(ud_Size), _Iface._Cmd.b_MemSetStart);
```

10.2.2.3 _Sts

Direction: PLC => HMI

All status messages that go to the visualization can be found in this structure. Typically, the machine statuses can be found here, for example.

All status variables are automatically copied from the "_Iface" structure to the "_Hmi" structure. This is done in the "CY_HMI_Copy" worksheet in the "CY_HMI_Out" POU:



10.2.2.4 _Pro

Direction: HMI => PLC

Process-relevant data is stored here, such as the adjustable cycle speed or the bag length for film machines.

10.2.2.5 _Set

Direction: HMI => PLC

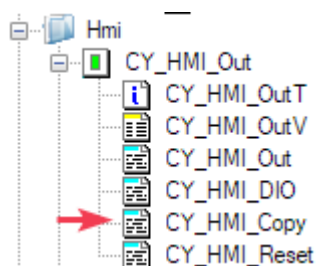
All machine settings can be stored here. These would be gear factors, speeds, accelerations, for example.

10.2.2.6 _Act

Direction: PLC => HMI

Current values and data that are sent from the direction of the PLC to the visualization are contained in this structure.

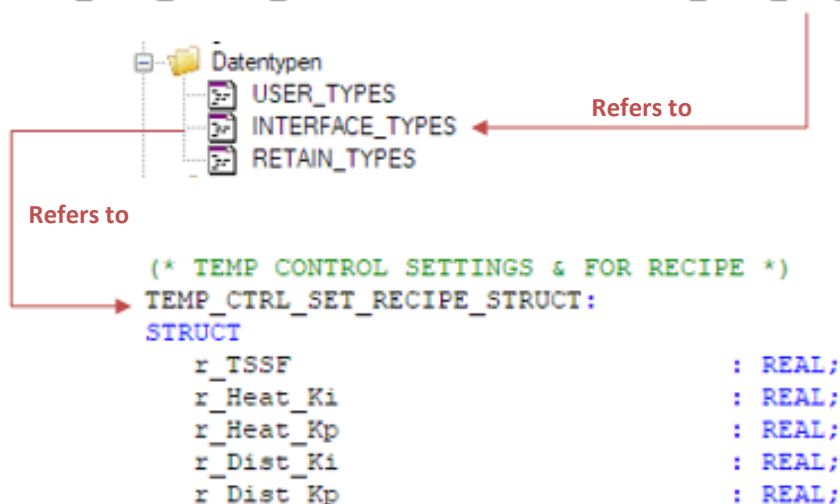
All "_Act" variables are automatically copied from the "_Iface" structure into the "_Hmi" structure. This is done in the "CY_HMI_Copy" worksheet in the "CY_HMI_Out" POU:




10.2.3 RETAIN_TYPES

This data type worksheet is for the structures that are taken into account for retentive data management. In relation to the template, the settings for the temperature control are stored here as an example.

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



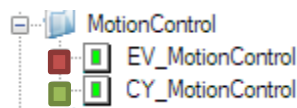
In the state machine of the temperature control you will find the variable for the data type, which is declared as "Retain":

Name	Type	Usage	Description	Address	Init	Retain
Default						
a_TempCtrl_Retain	TEMP_CTRL_RETAIN_ARR	VAR				 <input checked="" type="checkbox"/>

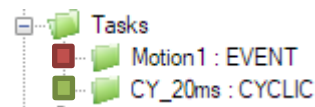
10.3 Logical POU


10.3.1 Differentiation between EV and CY

In some POU's there is a distinction between "EV_" and "CY_":



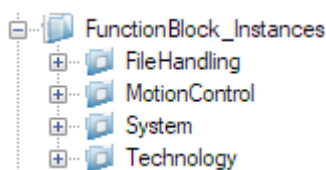
"EV" means that the blocks instantiated here are also running in the associated "EV"-Task:



Note!	
	<p>So that is EVENT-Task which is executed in the fieldbus cycle. All function blocks that have to run in the fieldbus cycle and are therefore time-critical are inserted here.</p> <p>"CY" means that all modules run here that can run in non-time-critical tasks. These execute commands in a time that does not have to be executed within a fieldbus cycle. The associated task is the "CY_20ms: CYCLIC"-Task. See also Tasks</p>

10.3.2 FunctionBlock_Instances

Each function block that is used in the project must be instantiated at one point in the project. Ultimately, this means inserting the block into the project. This is the point at which the function block is also called and processed. These modules are divided into categories within the template, which ultimately result in subdirectories.



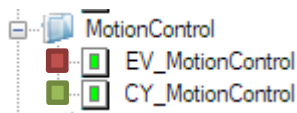
FileHandling

All blocks that can be used to manage files on the PLC are instantiated here. The name of the POU always reflects the block that is mainly used here:

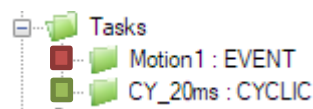
POU / function block	Brief description
BM_FileOpen	Opens a file
BM_FileClose	Closes a file
BM_FileList	Lists files or directories within a directory
BM_FileDelete	Deletes a file
BM_FileRead	Reads a file that was previously opened with BM_FileOpen
BM_FileWrite	Writes data to a file that was previously opened with BM_FileOpen


MotionControl

There is a distinction here between **EV_MotionControl** and **CY_MotionControl**:



"**EV**" means that the blocks instantiated here are also running in the associated "**EV**"-Task:



Note!	
	<p>So that is EVENT-Task which is executed in the fieldbus cycle. All function blocks that have to run in the fieldbus cycle and are therefore time-critical are inserted here.</p> <p>"CY" means that all modules run here that can run in non-time-critical tasks. These execute commands in a time that does not have to be executed within a fieldbus cycle. The associated task is the "CY_20ms: CYCLIC"-Task. See also Tasks</p>

System

Here are blocks that can execute system-relevant functions.

POU / function block	Brief description
BM_CallWindowsApp	Executes an executable file on the Windows side of the PLC
BM_MC_Control	Enables the fieldbus to be restarted

Technology

This part contains all technology modules that are required or can be used for a wide variety of machine functions.

The most important function blocks are explained in more detail in the later chapters in connection with the machine functions. A detailed description of the individual blocks can be looked up in the corresponding block help.

10.3.3 Hmi

Basic function of communication

The communication from and to the HMI is structured in such a way that there is ONE variable "_Hmi" which communicates directly with the visualization. There is also a "_Iface" variable, which has the same data type as the "_Hmi" variable. Both can be found in the global variable list in the "HMI" group:

Name	Typ	Verwendung	Beschreibung	Adresse	Anfangswert	Reman...	PDD	OPC
HMI								
_Iface	INTERFACE_STRUCT	VAR_GLOBAL	PLC-Internal values for HMI communication			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
_Hmi	INTERFACE_STRUCT	VAR_GLOBAL	PDD Interface to webServer (PCC)			<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

The "PDD" checkbox shows that only the "_Hmi" tag is enabled for HMI communication. "PDD" means "Process Data Directory" and is based on an internal name resolution. This interface works solely with the names of the variables and, in contrast to OPC, does not require any addresses.

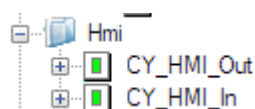
In the POU's explained below, a handshake is ultimately made between the variable "_Hmi" and "_Iface".

The following picture should clarify the structure again:



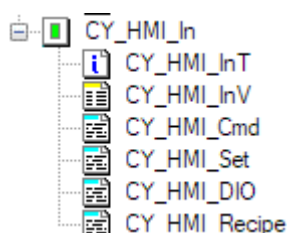
The interface to and from the HMI is located here. All data that either come from the HMI or also go to the HMI are transferred here. This allows the data to be validated again and checked for validity before receiving or sending.

The name of the POU shows which direction the POU is used for:



CY_HMI_Out => Everything that is sent to the HMI
CY_HMI_In => Data that are sent from the HMI to the PLC

For better structuring, the worksheets are subdivided within the POU's and named in such a way that the corresponding function can also be derived from the names.



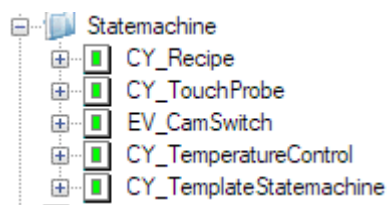
CY_HMI_Cmd	=>	Commands from the HMI (MouseUp, MouseDown, Toggle)
CY_HMI_Set	=>	Setting values from the HMI
CY_HMI_DIO	=>	Management of the digital inputs / outputs from the HMI
CY_HMI_Recipe	=>	All recipe-relevant data that can be controlled in the HMI

10.3.4 RetainHandling


There is only one block in this POU which can be used to manually initiate the saving of the retentive data. This is necessary, for example, if a PCC-04 is operated without a UPS, as there is no NOVDRAM here that automatically saves the data. With the current version of the PLCmc, however, this is no longer necessary and therefore obsolete, but it can still be used.

10.3.5 State machine

All state machines that are required for the project are available here:

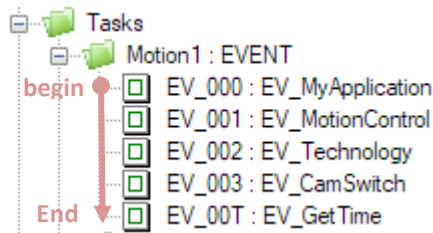


State machine	Brief description
CY_Recipe	State machine for handling the Recipe data
CY_TouchProbe	The probe function is dealt with here
EV_CamSwitch	State machine for handling the cam controller
CY_TemperatureControl	The temperature control is managed here
CY_TemplateStatemachine	The "main" state machine of the project. The state machine of the machine is mapped here.

Note!	
	The individual state machines are explained in more detail in the respective machine function-specific chapters

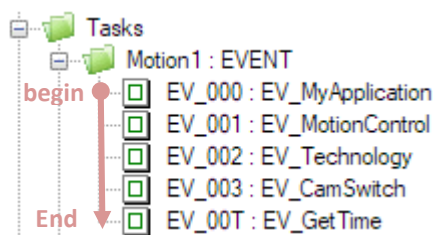
10.4 Tasks

In general, all tasks are processed from top to bottom. This is important because it plays an important role in the "CYCLIC" task in relation to the HMI program instances.



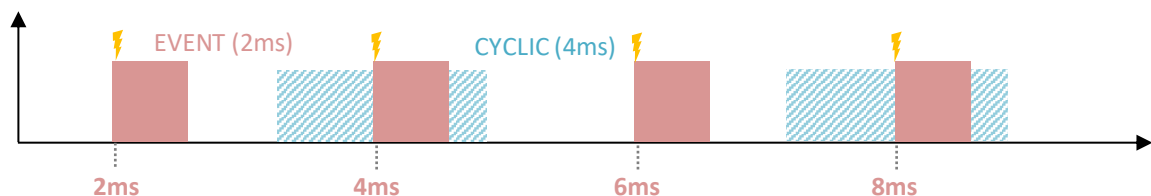
10.4.1 Motion1: EVENT

This task contains all program instances that begin with the prefix "EV_". These tasks are time-critical and are executed in the fieldbus cycle.

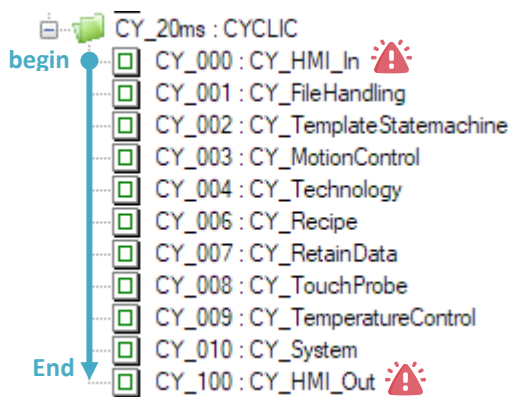


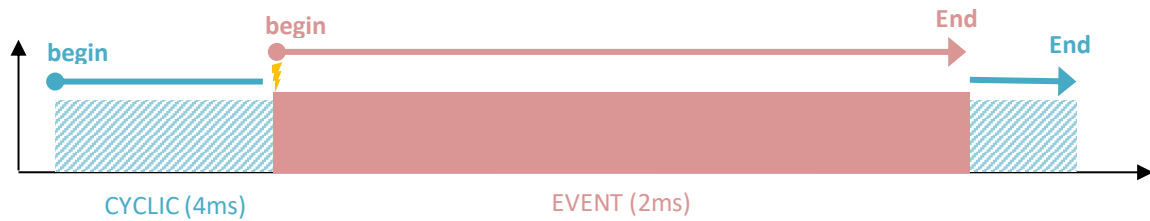
10.4.2 CY_20ms: CYCLIC

All program instances that are not time-critical and are therefore not called up in the fieldbus cycle are located here. These instances are called every 20ms, but interrupted by a fieldbus interrupt, as this has a higher priority:



In this example, the CYCLIC task is called every 4 ms (for illustration purposes only) and the EVENT task every 2 ms. You can see that the CYCLIC task can be interrupted by the EVENT task and continues where it left off. Therefore, in most cases it makes sense not to mix the data from the CYCLIC with the data from the EVENT.





The program instance "CY_HMI_In" **has to be located** at the beginning of the task and "CY_HMI_Out" at the end of the task. The reason for this is that all data that are sent from the HMI to the controller are copied into the _Iface variables in the first task and these _Iface variables are then used in the subsequent tasks. At the end of the CYCLIC task, the data is then copied from the _Iface variables into the "_Hmi" structure.

11. Machine functions

11.1 User level management

The user level management includes all components to manage the preset maximum of four user levels. Management takes place exclusively in the visualization. So there is no interface to the PLC implemented here. However, this can be retrofitted if necessary.

For this there is a subdirectory "User" in the HMI project with the view "Login.teq":



Here you can see the login / logout window, which can be edited if necessary.

In the top left corner you can see the currently logged in user via access to the CONTAINER variable "userLevel" in connection with an HTML tag "CurrentUser_", which serves as a prefix:

Login

Current user

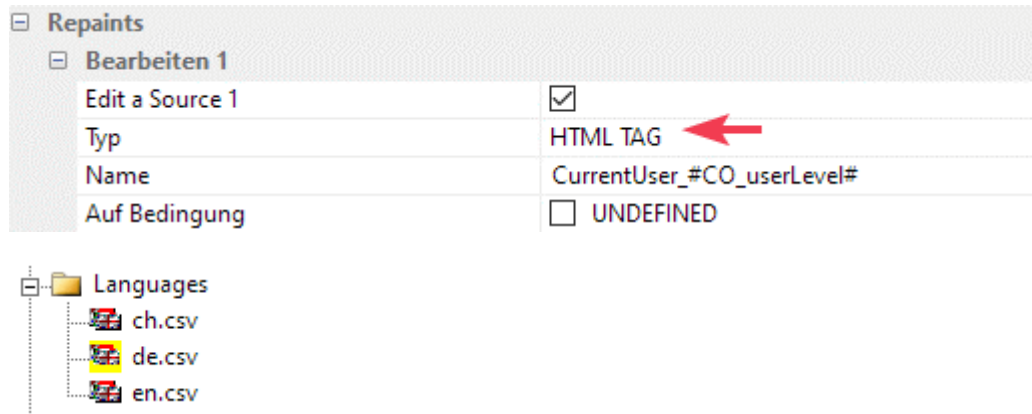
CurrentUser_#CO_userLevel# ←

Password SHA256PasswordConstant_6_40_00b

myPassword

Logout

The corresponding user is then displayed here at runtime depending on the value in "userLevel". Since this variable is of the "HTML TAG" type, it is also entered in the language files during the build:



Current user;Current user
 CurrentUser_#CO_userLevel#;CurrentUser_#CO_userLevel#
 CurrentUser_0;Operator
 CurrentUser_1;Production manager
 CurrentUser_2;Setter
 CurrentUser_3;Service
 CurrentUser_4;Developer

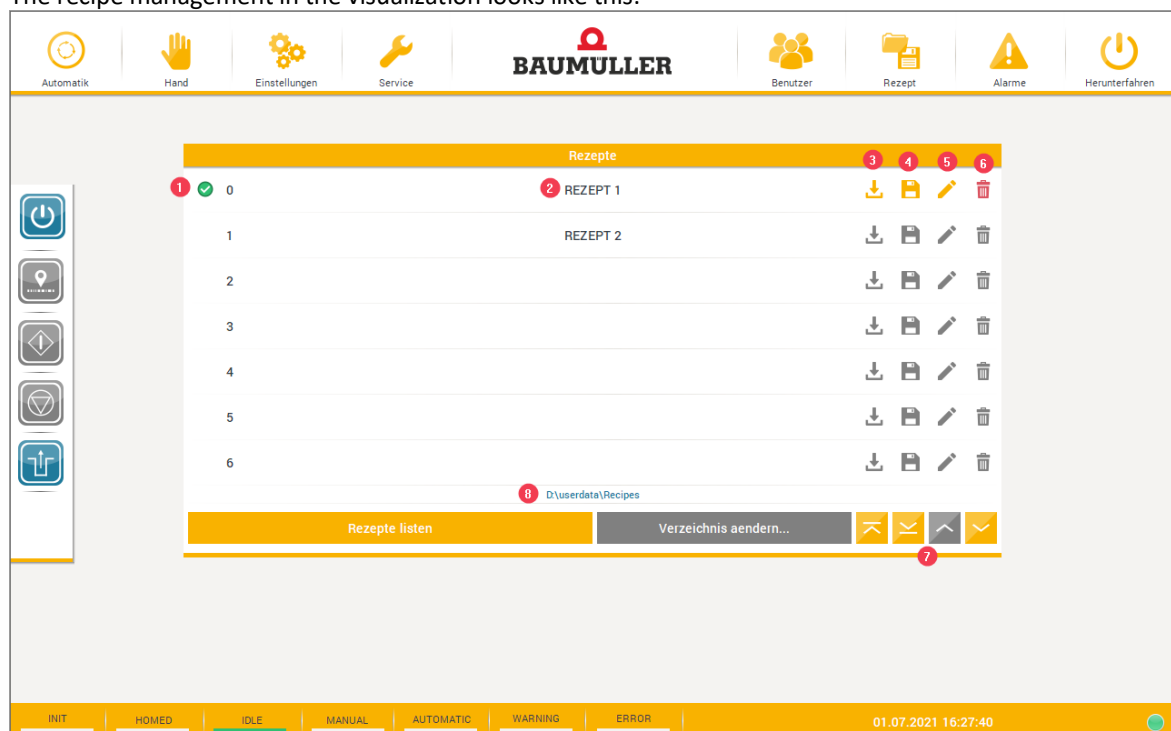


11.2 Recipe management

The recipe management consists of a PLC and a visualization part. The management of the recipes is done in the PLC. The visualization only serves as a display or input mask.

11.2.1 HMI

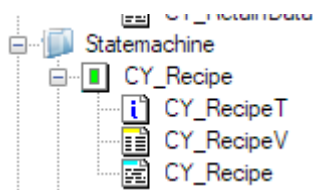
The recipe management in the visualization looks like this:



No	description
1	Current selection
2	Recipe name
3	Load recipe
4	Save recipe
5	Rename recipe
6	Delete recipe
7	Scroll recipes
8	Recipe directory on the PLC

11.2.2 PLC

There is a state machine in the PLC for recipe management:



State 0 is always the waiting state in which the visualization waits for an input or a command:

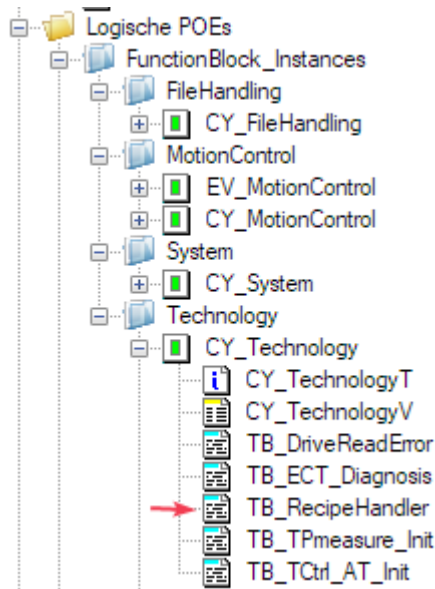
```
(* ----- *)
0: s_State := '0: Wait for command';
(* ----- *)
```

If a command is recognized, the associated action is carried out:

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

The currently selected recipe is deleted here.

The associated function block is located here:



11.3 Touch-Probe

11.3.1 General functional description

The measuring probe works in such a way that an optical sensor is connected to an input of the bmaXX. For this purpose, special fast inputs are provided, which are designated with the abbreviation "MT" for measuring probes:

HW	Invert.	Log	Sonderfunktion	Trigger-Modus	Ziel-Parameter	Wert	Parameter-Bezeichnung
1	<input checked="" type="radio"/> MT1	<input type="checkbox"/>	Keine Sonderfunktion	Flanke	000.000.0.0	...	
Bit-Operationen							
2	<input checked="" type="radio"/> MT2	<input type="checkbox"/>	Keine Sonderfunktion	Flanke	000.000.0.0	...	
Bit-Operationen							
3	<input type="radio"/>	<input type="checkbox"/>	Keine Sonderfunktion	Flanke	000.000.0.0	...	
Bit-Operationen							
4	<input checked="" type="radio"/> IF1	<input type="checkbox"/>	Keine Sonderfunktion	Flanke	000.000.0.0	...	
Bit-Operationen							

These inputs can be used for the probe function. In addition, the detailed function of these inputs can be specified in the drive.

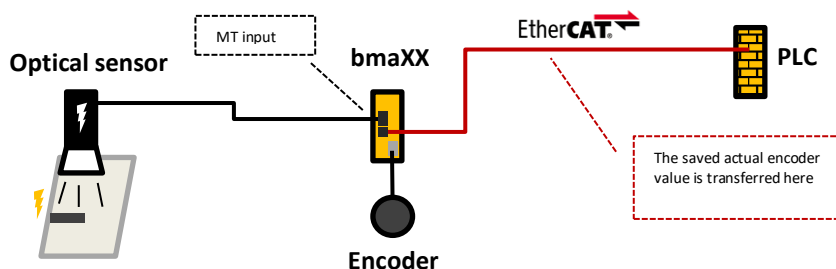
The configuration options can be seen under Configuration -> Touch probe -> Touch probe encoder 1, for example:



The settings that can be seen here are set in the template by the plc / visualization. This means that you do not have to make any settings here manually. The representation serves purely for information.

If the probe is activated and an edge is detected, the associated position value of the encoder is saved and made available to the plc. In the plc, the distance is then calculated from at least 2 signal edges that are separated from one another.

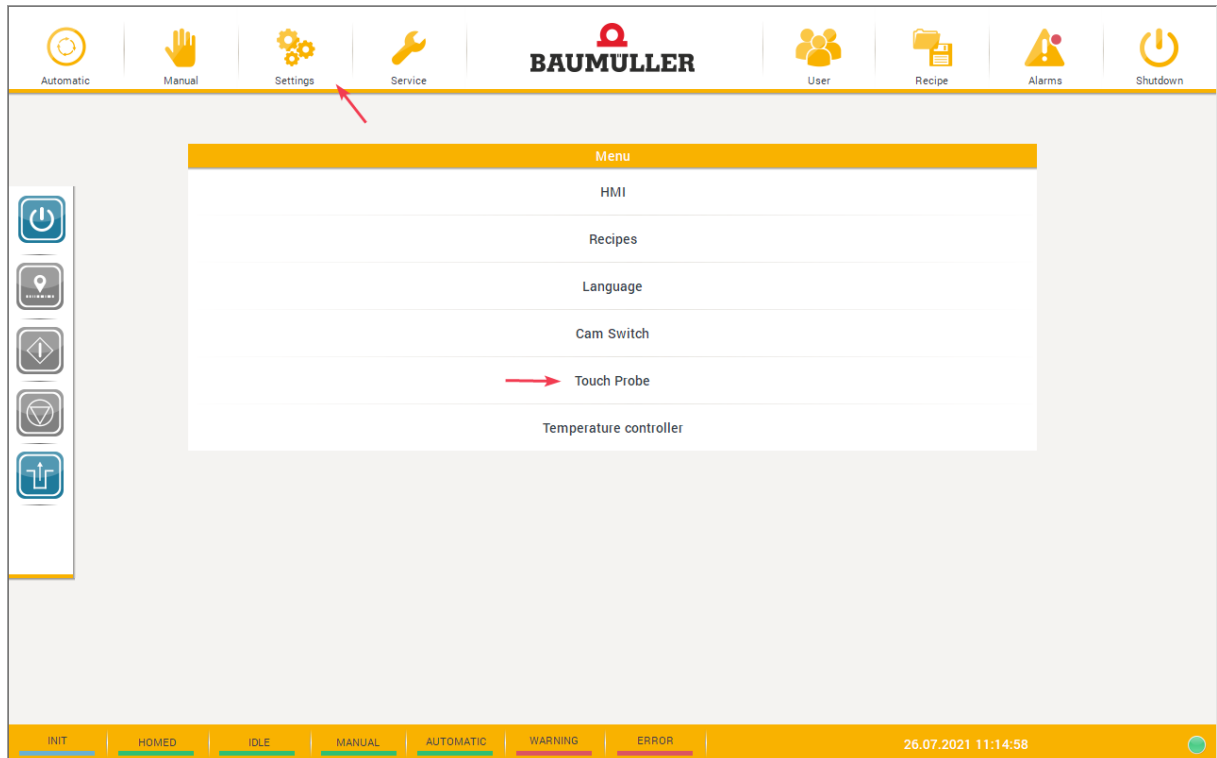
Here is the scheme graphically prepared for clarification:



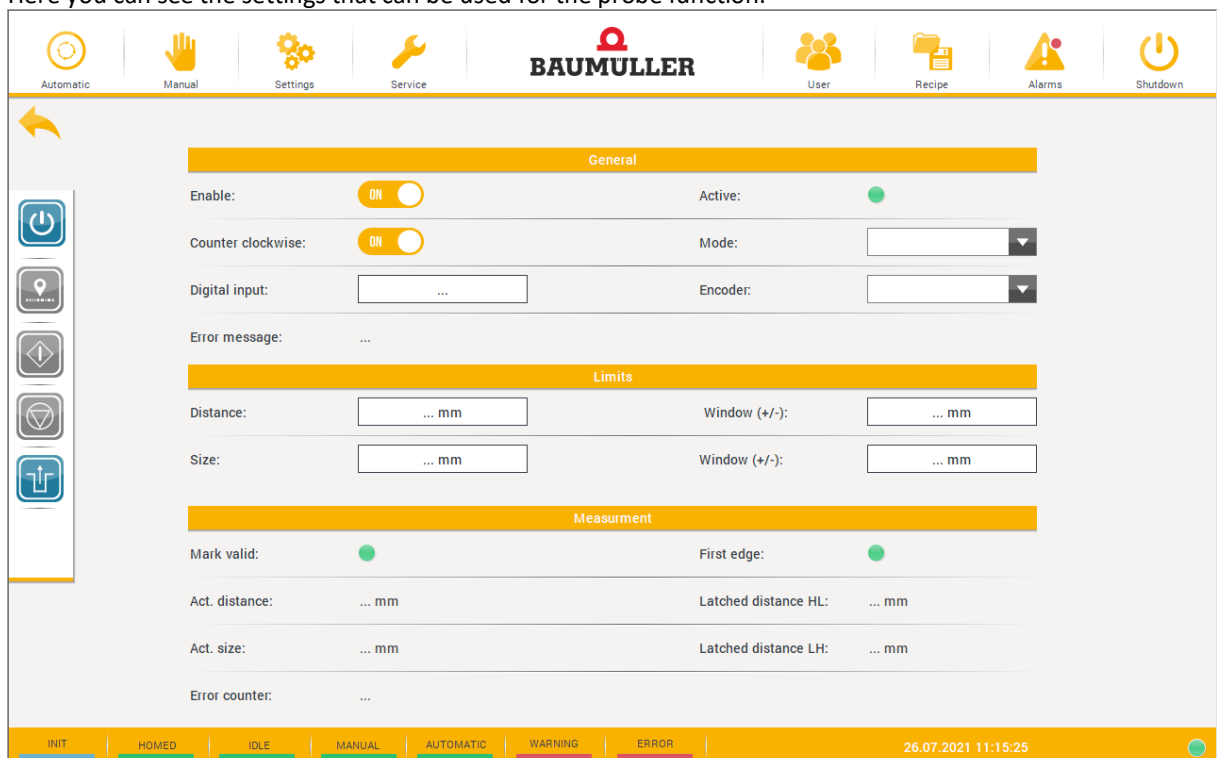
For the touch probe function there is a part in the visualization and a part in the PLC.

11.3.2 HMI

The probe settings can be found under Settings -> Probe:

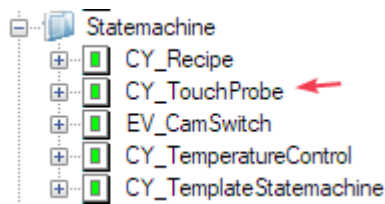


Here you can see the settings that can be used for the probe function:



11.3.3 PLC

There is a state machine for this in the PLC project and of course the associated function blocks.
The state machine can be found here:



In state 0, the Visu waits for the "Enable" command:

```
(* ----- *)
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;
```

This corresponds to the toggle button in the visualization:




Before activating, the settings for the measuring probe should be set. Here you can, for example, select the digital input of the controller and which encoder should be used for it.

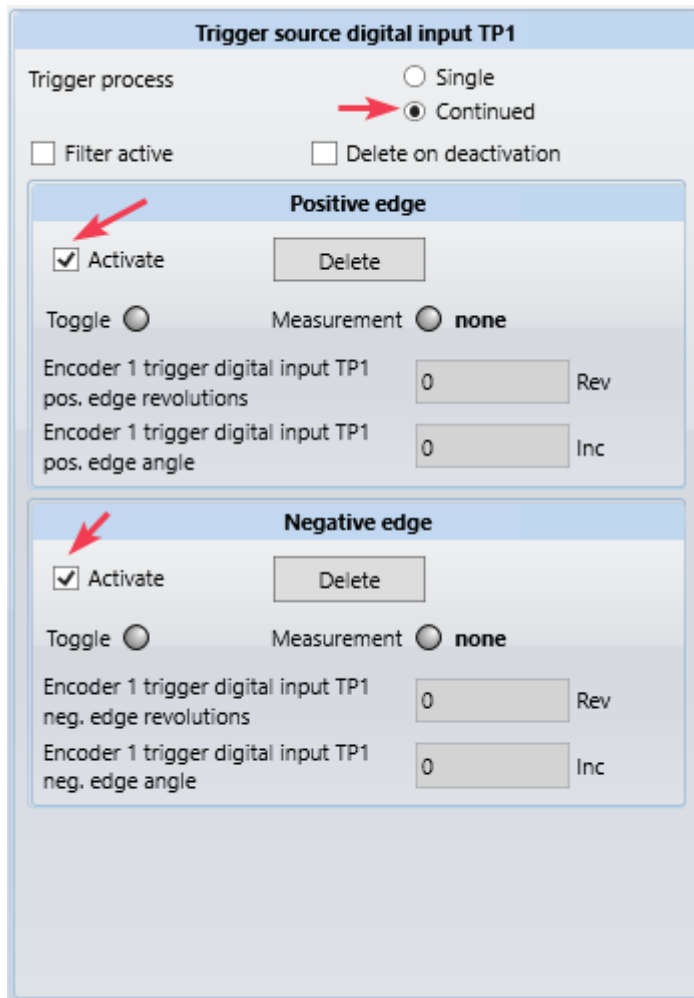


If the measuring probe has been successfully initialized, this can be seen from the "Active" LED:



Note!	
	<p>All settings are linked in the plc with the associated variables for the function blocks. You can read this in detail in the help files of the blocks.</p>

In this step, the probe function in the controller was initialized according to the settings that were made. You can check that with ProDrive:



Trigger source digital input TP1

Trigger process: ☐ Single ☒ Continued

☐ Filter active ☐ Delete on deactivation

Positive edge

☒ Activate

Toggle ☐ Measurement ☐ none

Encoder 1 trigger digital input TP1 pos. edge revolutions: 0 Rev

Encoder 1 trigger digital input TP1 pos. edge angle: 0 Inc

Negative edge

☒ Activate

Toggle ☐ Measurement ☐ none

Encoder 1 trigger digital input TP1 neg. edge revolutions: 0 Rev

Encoder 1 trigger digital input TP1 neg. edge angle: 0 Inc

The state machine is now in the "TPmeasure" state:


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

The settings for the distance between the print marks and the size of the print marks, as well as the tolerance window, are transferred to the block here:

```
_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;
```

In addition, you have to adjust the scaling of the units here. By default, a resolution of 100 mm per motor revolution is set here:

```
(* 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 *)
```

Note!	
	<p>Here, of course, the setting must be made to suit the machine, taking the gearbox factors into account.</p>

The block is then activated:

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

If the motor now turns and the print mark is recognized, the measured distance and the size are output accordingly under "Measurement". If faulty print marks are recognized (e.g. not recorded in the window), this is also displayed via an error counter.



11.4 Cam switch

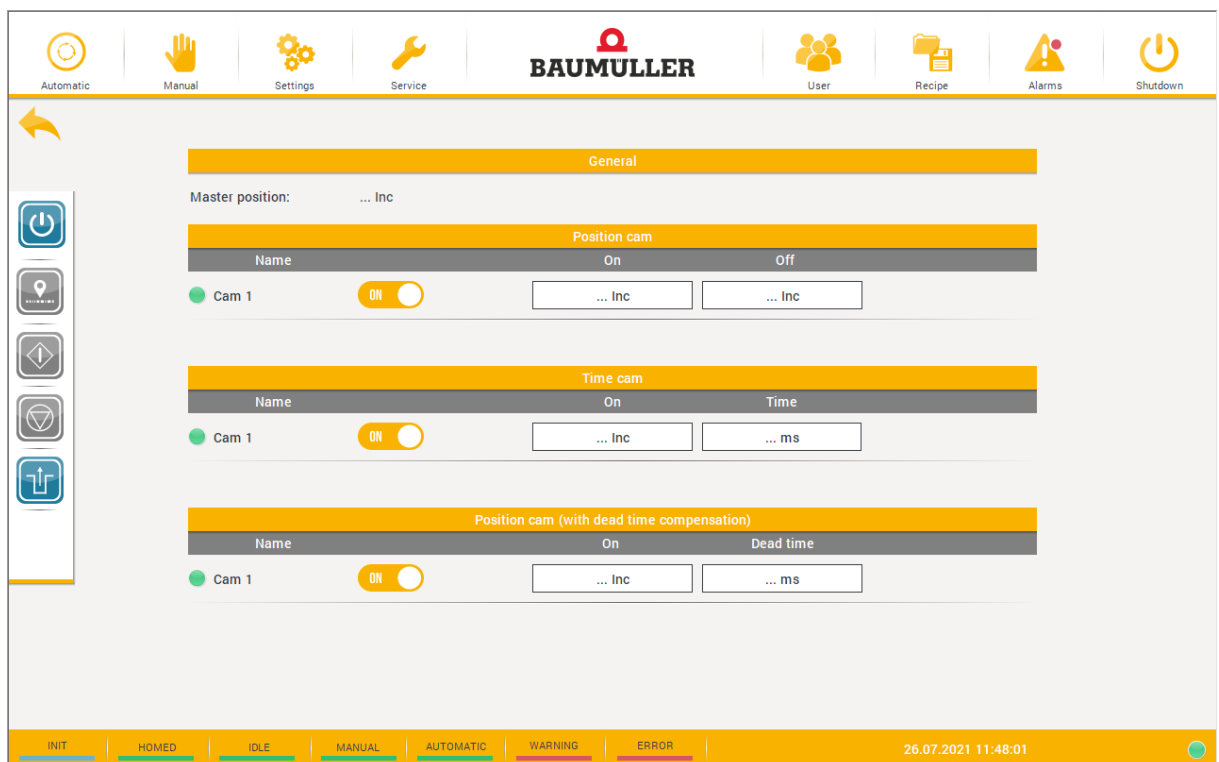
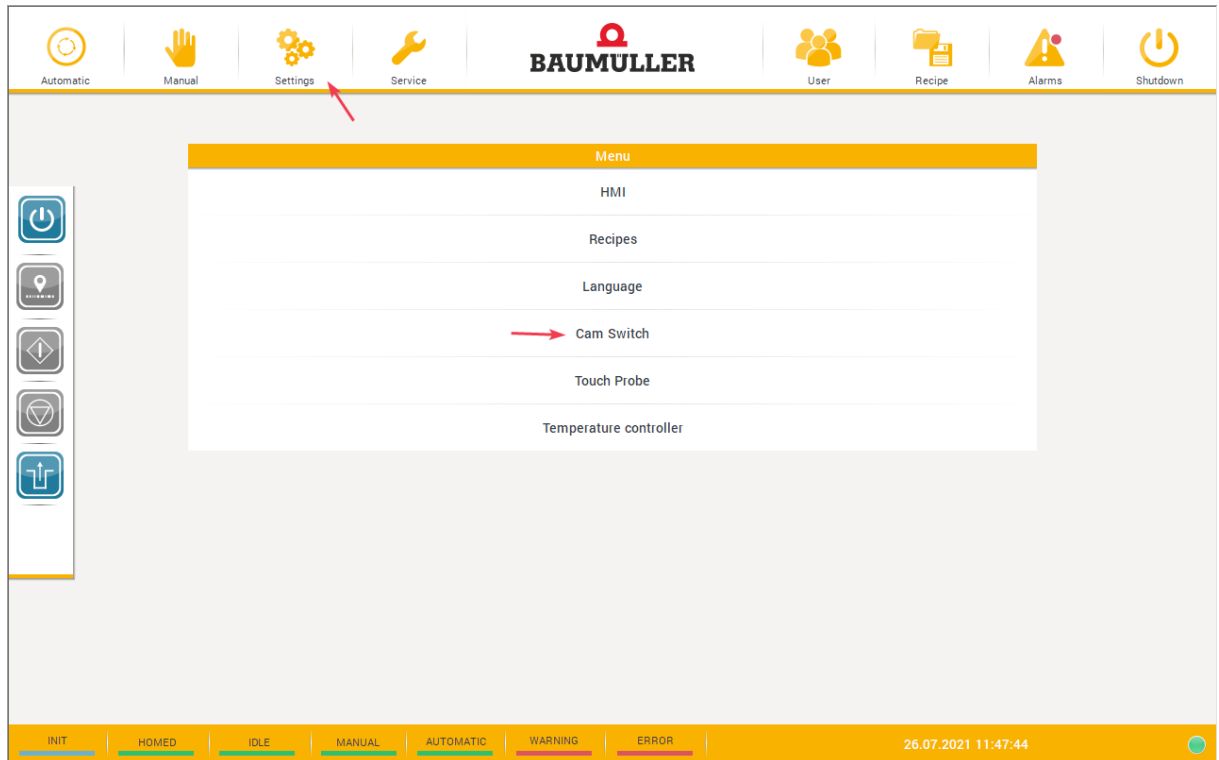
11.4.1 General functional description

During the process, the cam controller can be used to carry out actions at certain times or positions in relation to an axis. This axis can be real or virtual. Pneumatic valves, for example, can be controlled here. These valve controls usually have a dead time, which must also be taken into account during the process. If a valve is activated, it does not mean that this valve has also carried out the action in the next cycle. The time difference between activation and execution of the action corresponds to the dead time.




11.4.2 HMI

The settings and the "cam switch" function are located under Settings -> Cam switch:

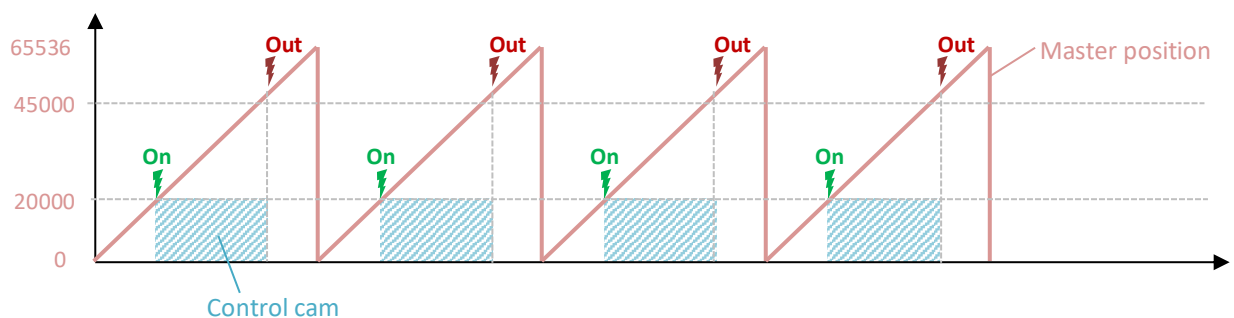
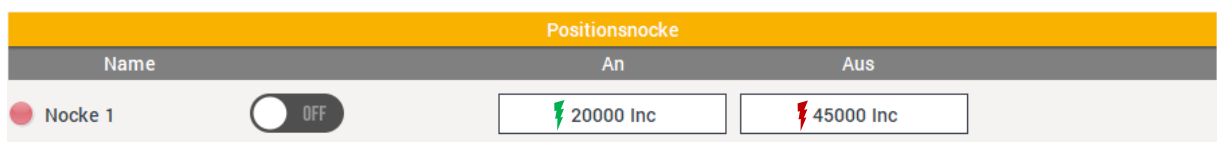


The cams that can be set here serve as an illustrative example and can be expanded as required. These cams are based on the master position, which is displayed under "General". As soon as a cam is active, the associated LED turns green.

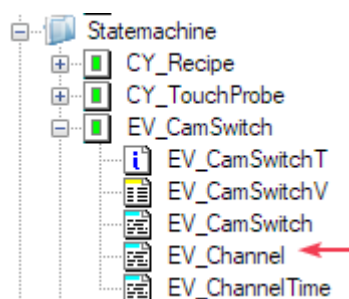
Note!	
	<p>In the Position cam with dead time compensation, however, the cam is only set for one cycle. It's too fast to see in the visualization. Therefore a counter was integrated in the code so that you can see whether the output is switching.</p> <pre> (* Check CamOut *) IF _Tech_TMPL._CamSwitch[0]._Out.x_CamOut THEN i_CamCounter := i_CamCounter + 1; END_IF; </pre>

11.4.3 Position cam

The position cam is a cam which switches on depending on one position and switches off depending on another position. There is no dead time compensation here. It is only looked at the position, which here is that of the virtual master:



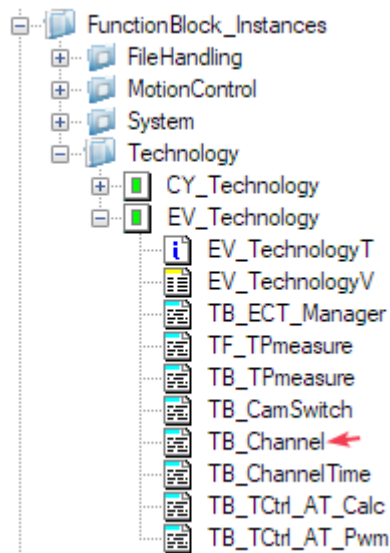
The function block used in the controller for this is the "TB_Channel". There is generally a state machine for the cam controller, which manages the switching on and off of the modules:



Whereby it has to be said that both the "TB_Channel" and the "TB_ChannelTime" do not require a state machine, but have only been included here for the sake of completeness. Due to their simple handling, these function blocks do not require any complex administration. This becomes clear if you take a closer look at the "EV_Channel" worksheet:


```
(* 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;
```

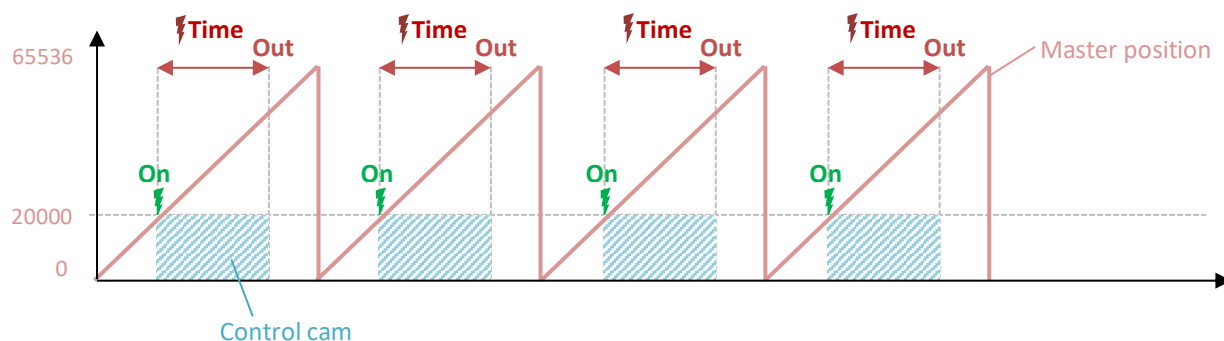
The instance of the associated function block can be found here:



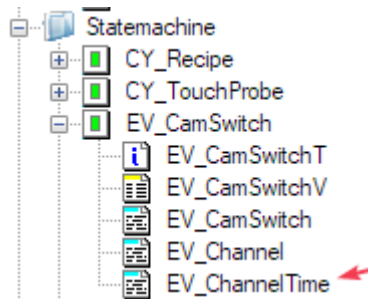
11.4.4 Time cam

The time cam can be used to control a cam at a specific position for an adjustable time. No dead time is taken into account here either.

Zeitnacke		
Name	An	Zeit
<input checked="" type="radio"/> Nocke 1 <input type="radio"/> OFF	<input type="text" value="20000 Inc"/>	<input type="text" value="2000 ms"/>



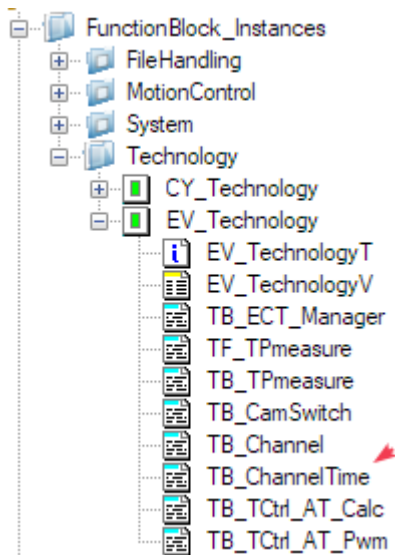
In the PLC there is the worksheet "EV_ChannelTime" under State machine in the POU "EV_CamSwitch":



As with the "TB_Channel", no state machine is necessary here either:

```
(* 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.ud_Interruption_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;
```

The associated block instance can be found here:

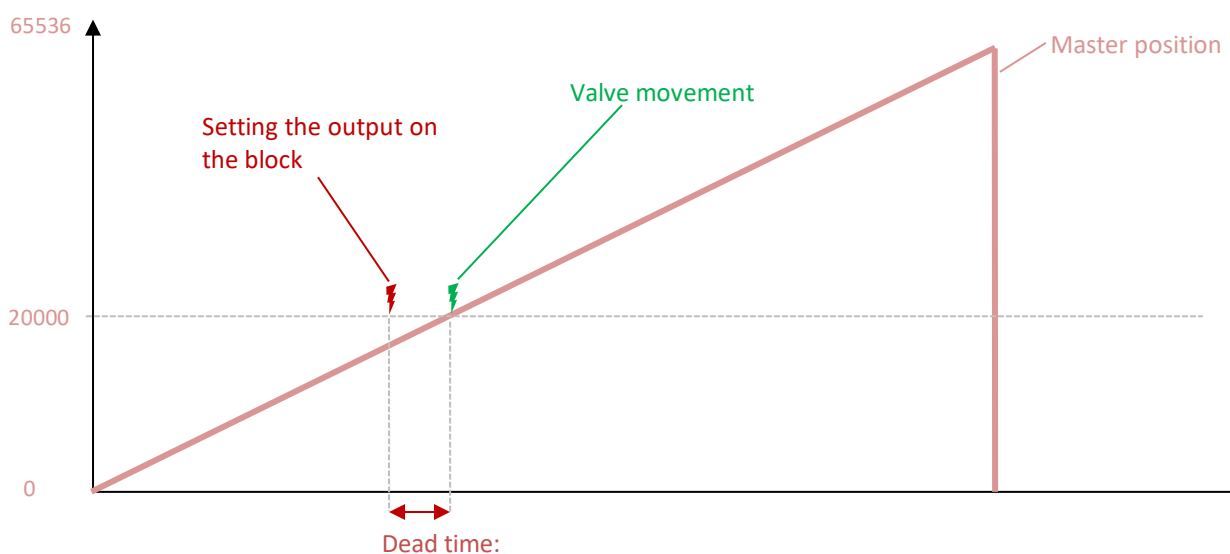


11.4.5 Position cam (with dead time compensation)

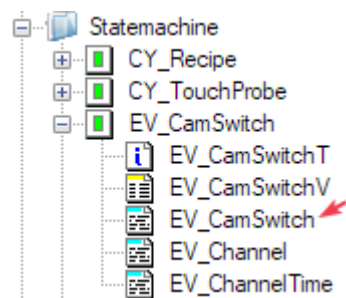
The position cam with dead time compensation is a cam that can be controlled to a specific position, taking into account a dead time for a cycle. Since this is only active for one cycle, the control cannot be seen via the LED. For this, however, there is a counter in the PLC that counts up each time it is activated. (please refer [HMI](#))

Position cam (with dead time compensation)		
Name	On	Dead time
Cam 1	ON <input type="checkbox"/>	... Inc <input type="text"/> ... ms <input type="text"/>

This function is explained in more detail using the following figure:



There is a state machine in the control for managing this function:



In state 0, the activation of the function via the HMI is waited for:

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

The current position of the master is also defined as the start position for the function block:

```
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;
```

Then the parameters from the visualization are transferred to the block, as well as the master position. It is also rescaled from Inc / cycle to Inc / s.

```
(* 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 (*us/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 (*us/ms*)));
```

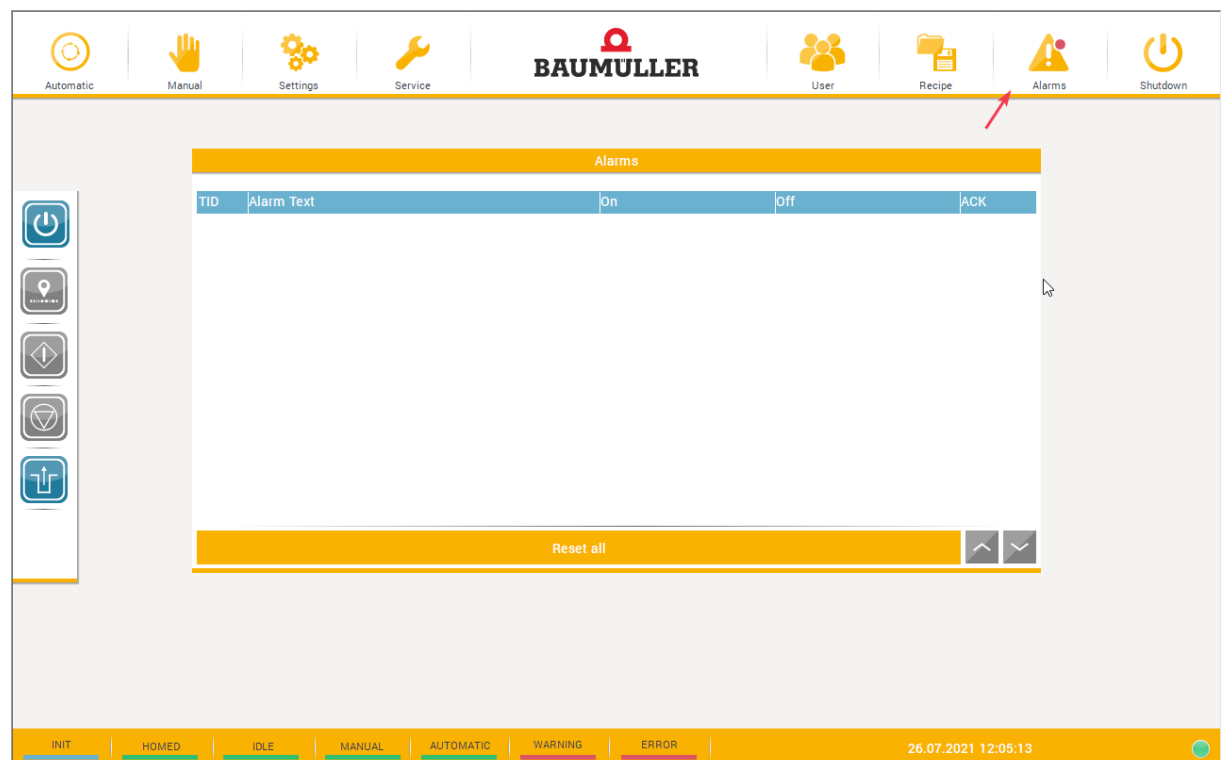
With the counter you can see how often or whether the output for the cam connection has been activated:

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

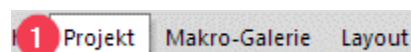
11.5 Alarming

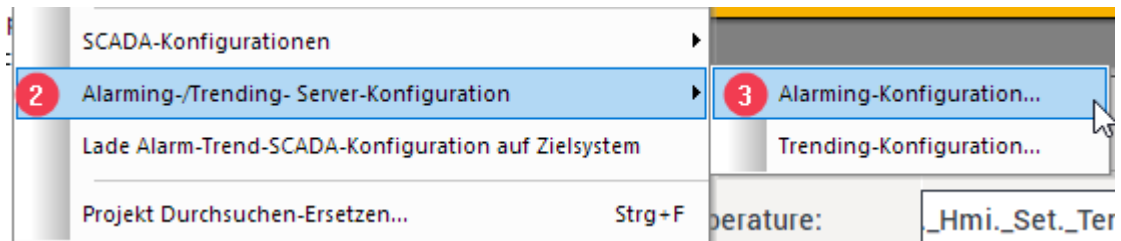
11.5.1 HMI

The "Alarms" page is available in the visualization for alarming:



All alarms are displayed here, as well as the history. The variables for the alarms can be set using the SCADA editor. A variable is stored as an example for the template. This setting can be found in the SCADA editor under





ID	Prio	Gruppe	Gruppe2	Gruppe3	Type	Go ON Wert	Go OFF Wert	PPO Name:
1	-1	-1	-1	-1	If ==	1.000000	1.000000	PLC0:@GV_Hmi_Sts_Err[0]

Definierte Alarme

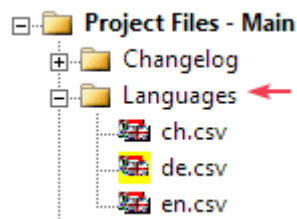
ID: 1 Prio: -1 Gruppe: -1 Gruppe2: -1 Gruppe3: -1 Entfernen

Type: If Greater Hinzu

PPO Name: izels:OTF.PP01 Wähle

Go ON Wert: 0 Go OFF Wert: 1 Update

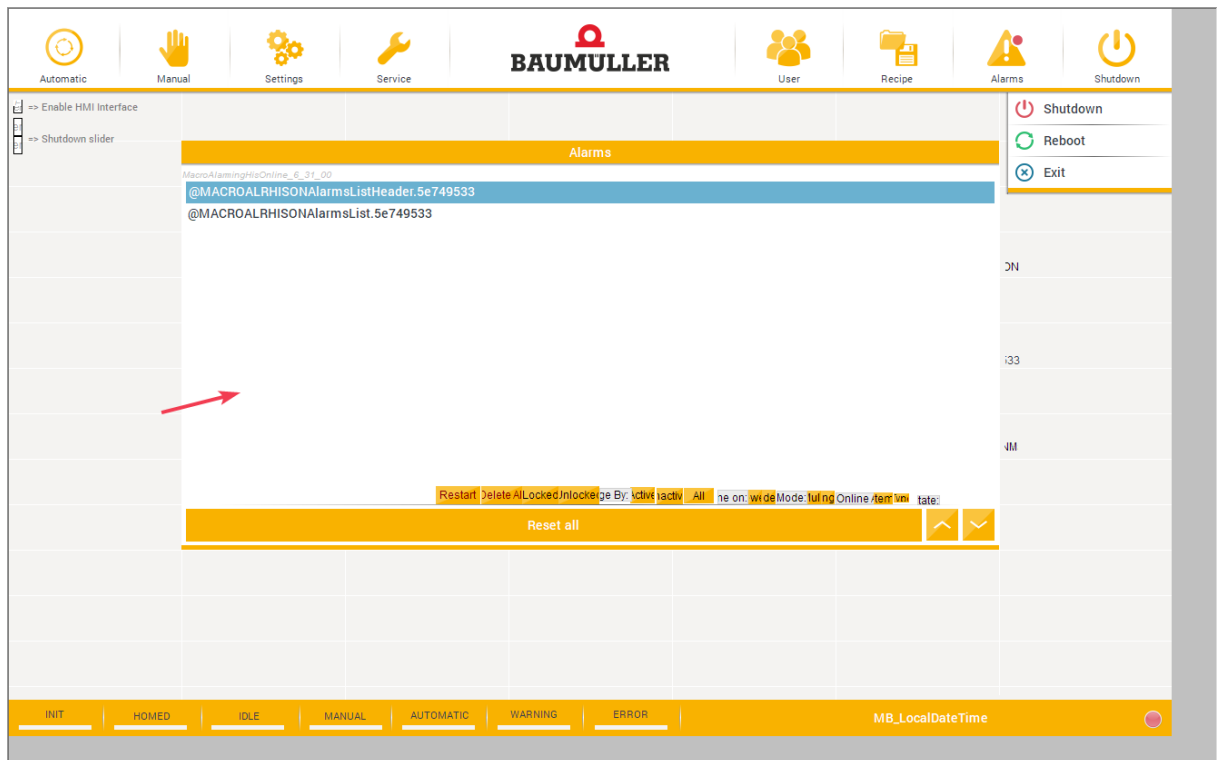
The associated error text can be found in the language files:



Here then the text for the "ALRLIST0" with the error ID 1:

```
=> Shutdown slider;=> Shutdown slider
A;A
ABC;ABC
ALRLIST0_1;Drive Error
ALRLIST0_10;ALRLIST0_10
```


To edit the alarming macro, simply double-click the alarm window:



This takes you to the configuration window for the settings of this macro. Here you can, for example, adjust the alarm filter or the colors for active / inactive alarms.

The second download dialog, which opens as soon as the first download is completed, is important for the alarm function when downloading a project:

Download (Zu FTP-Server oder Web-Server) - SCADA Einstellungen 

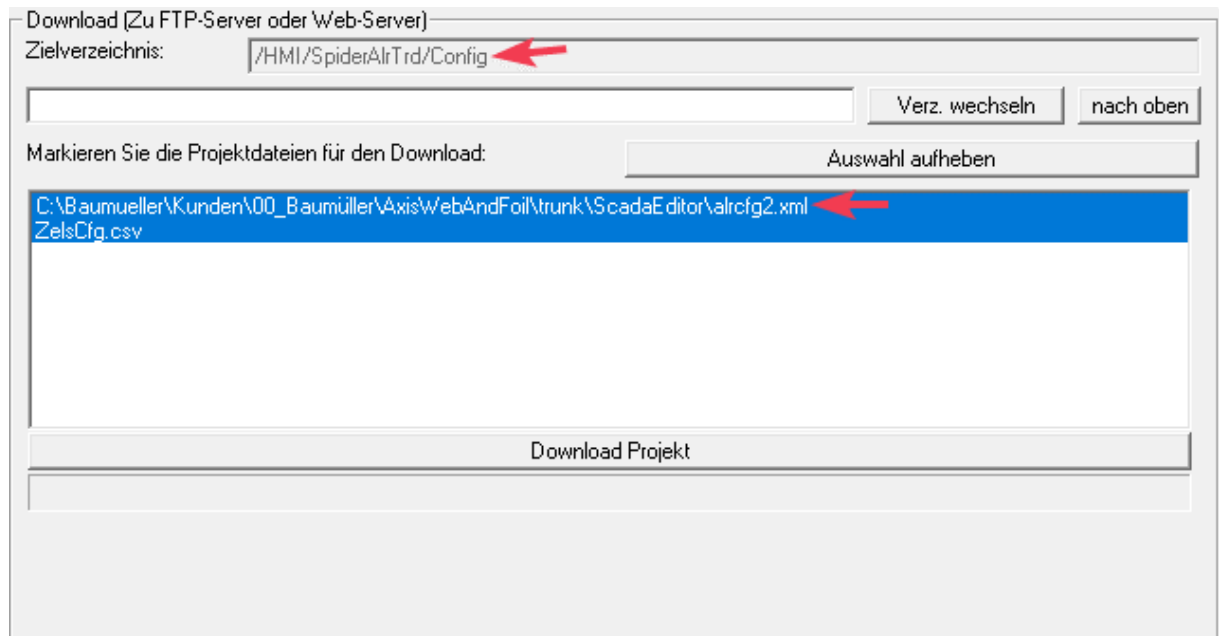


Login data:

User: BM_ProViz

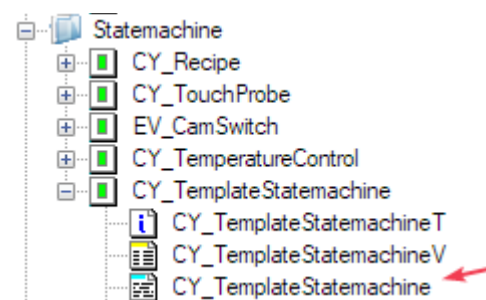
Password: 12345678

Here you can also see the settings which must be set so that the alarming is copied into the correct directory on the target system and the file which ultimately contains the alarm configuration:



11.5.2 PLC

The "_Iface._Sts._Err [0]" tag is used in the PLC for a drive error. The variable is set here:



```
(* -- 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);

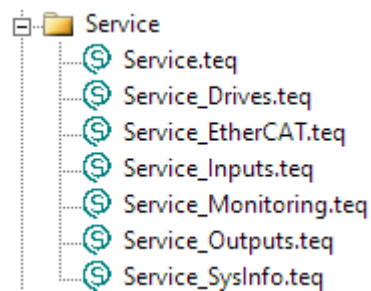
(* Check if initialization is done *)
IF R_TRIG_1.Q AND i_State > 0 THEN
    i_State := 800;
    _Iface._Sts._Err[0] := TRUE; (* Alarm *)
END_IF;
```

11.6 Service / diagnosis

The following service and diagnosis options are available in the template:

- Drive diagnostics to evaluate error details
- EtherCAT diagnosis
- Display and forcing the states of the digital inputs
- Display and forcing of the states of the digital outputs
- Trend display with sine and cosine
- System information

These functions can be found in the HMI project in the "Service" subdirectory:



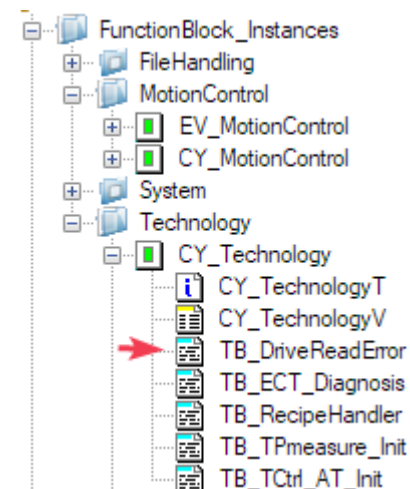
Note!



There is no state machine for any of the pages in the PLC project. The information on the "Service_Sysinfo" page is also implemented purely via CONTAINER variables.

11.6.1 Drive diagnosis

The drive diagnosis is implemented using the "TB_DriveReadError" function block. This is located here:



The information is transferred to the visualization in the "CY_HMI_Out" POU:


```
(* Drive Errors *)
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;
```

The associated block instance is selected depending on the "u_DriveErrorSelection" tag:

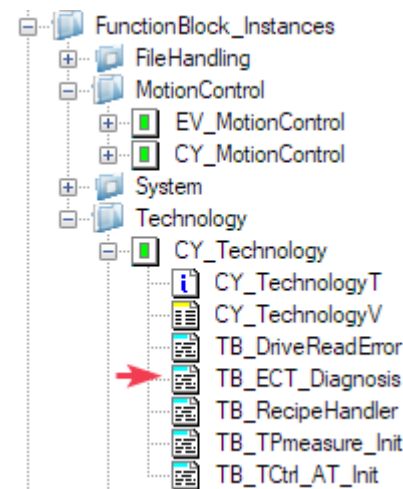
General

Select Drive: PLC0:@GV_Hmi_Set.u_D ▼ ←

Repaints	
<div> <div>Bearbeiten 1</div> <div> <div>Edit a Source 1</div> <div><input checked="" type="checkbox"/></div> </div> </div>	
Typ	PPO
Name	PLC0:@GV_Hmi_Set.u_DriveErrorSelection ←
Auf Bedingung	<input type="checkbox"/> UNDEFINED

11.6.2 EtherCAT diagnosis

The EtherCAT diagnosis is implemented using the "TB_ECT_Diagnosis" function block. This is located here:

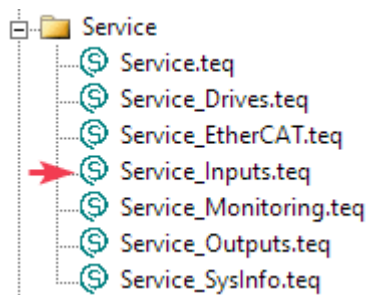


The information is transferred to the visualization in the "CY_HMI_Out" POU:

```
(* 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 Display and forcing the states of the digital inputs

The digital inputs (e.g. an E-bus coupling module) can be controlled and evaluated via the "Service_Inputs.teq" page in the visualization:

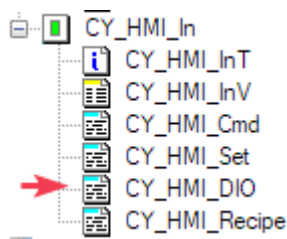


There is a display for each input, for its hardware status and the logical status:



The logical status corresponds to the hardware status as long as the input is switched to "AUTO". If it is set manually via "ON" or "OFF", the hardware input is no longer copied to the logical input.

In the PLC project, the digital inputs are managed here:



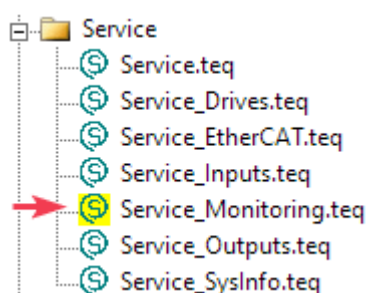
```
(* Digital Input 1 *)
CASE _Iface._Cmd._Node[1]._DI_LOG[1] OF
0:  _Iface._Sts._Node[1]._DI_LOG[1] := BOOL_TO_INT(ix_01_01_01_HW_Sts); (* 0 = PLC controlled *)
1:  _Iface._Sts._Node[1]._DI_LOG[1] := 1;                               (* 1 = Forcing ON *)
2:  _Iface._Sts._Node[1]._DI_LOG[1] := 2;   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

11.6.4 Display and forcing of the states of the digital outputs

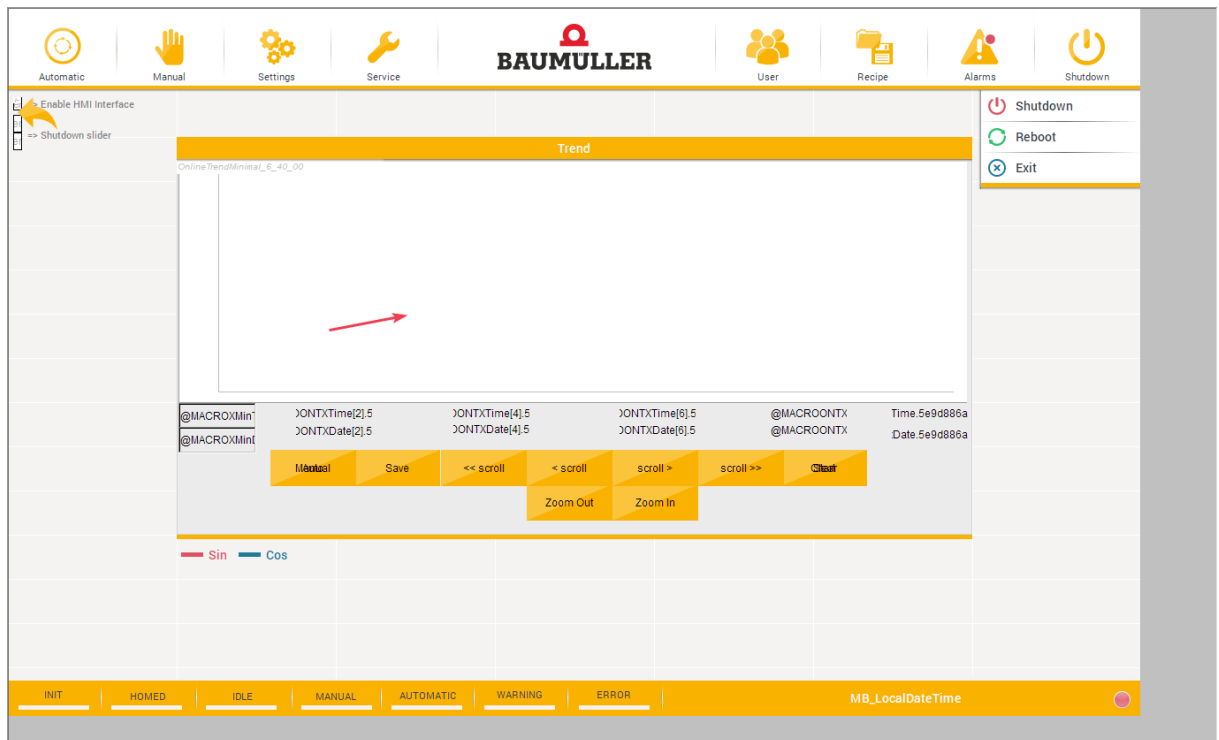
The function corresponds to that of the digital inputs, except that there is no logical output for the digital outputs.

11.6.5 Trend display with sine and cosine

The trend display can be found in the HMI project under "Service"> "Service_Monitoring":



An online trend macro is instantiated here, which displays the settings by double-clicking:



OnlineTrendMinimal_6_40_00

General-Einstellungen

Zeige Raster	<input checked="" type="checkbox"/>	Auto Y Achsenbeschriftung	<input checked="" type="checkbox"/>
Angezeigte Zeit (s)	60	Gespeicherte Zeit (s)	3600

Kurven-Einstellungen

PPO	Min	Max	Farbe
PLC0:@GV_Hmi_Act.r_TrendV ...	-1	1	
PLC0:@GV_Hmi_Act.r_TrendV ...	-1	1	
...			
...			
...			
...			
...			
...			
...			
...			

OK
ABBRECHEN

The trend variables that are linked to the macro can be seen here. The color of the individual lines and the scaling can be adjusted here.

In the PLC project there is an area for this in the POU "CY_HMI_Out" and the worksheet of the same name, which calculates the trend variables and copies them to the interface:

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

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

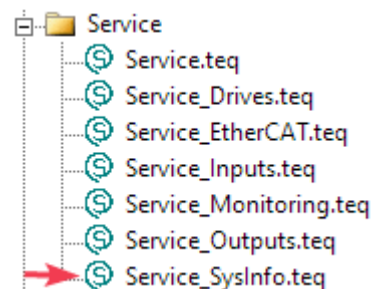
_Iface._Act.r_TrendValues[1] := SIN(r_Rad); (* Sin *)
_Iface._Act.r_TrendValues[2] := COS(r_Rad); (* Cos *)
```



11.6.6 System information

The system information page contains information about the version of the microbrowser, the operating system on which it runs, the HMI version (which can be set) and also information about the network settings on the system.

The system information page can be found here in the SCADA editor project:



The variables displayed here all only show "Not available" in the development environment. This is because this is a value that the variables can assume if the value cannot be retrieved on the target system:

System		
Microbrowser-Version:	Not available	

In the properties under the extended repaints (the yellow lightning bolt icon) you can see more precisely what is happening here. The condition is set here if the container variable "MB_APP_VERSION" is equal to an empty string, then "Not available" should be displayed.

Eigenschaften

Erweitert Repaints - Aktionen

- Repaints

EDIT SOURCE	[CONTAINER "MB_APP_VERSION"]
USE AWT COMPONENT	
TEXT HEIGHT CENTERED	
END PLCREPAINTS LIST	
- Verwaltung der Extra-Repaints

+	-	↑	↓	↕	↻
---	---	---	---	---	---
- Extra-Repaints Liste
 - Extra-Repaint_4

Typ	EDIT SOURCE
Quellentyp	HTML TAG
Quellenname	Not available
 - 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	

The "Extra-Repaint_5" also ensures a color change:

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	

At runtime it looks like this:

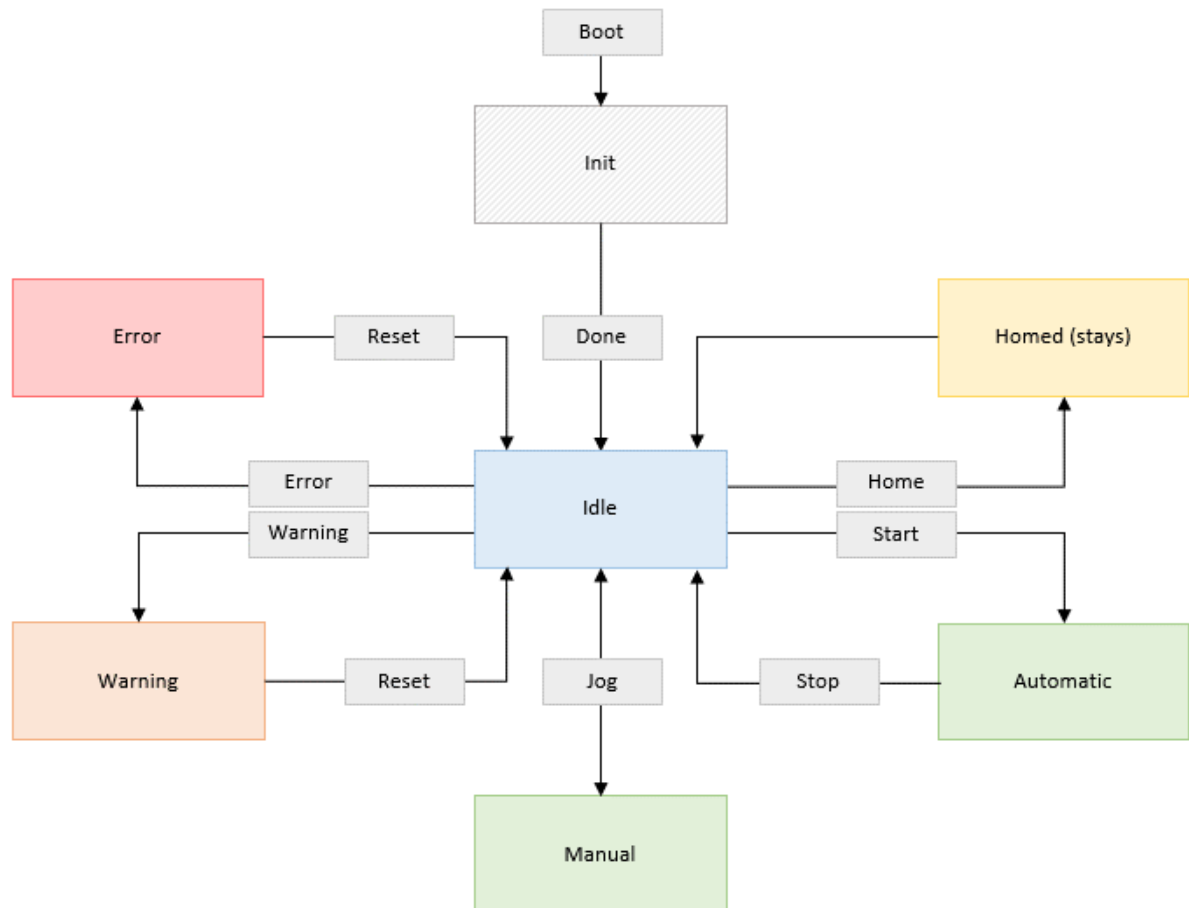
Network	
IP-Adress:	Not available

CONTAINER variables are used here, which can be found in the editor's help.

12. State machine

12.1 General description

The state of the machine is mapped in the state machine. The state machine, which is integrated in the template, is explained in more detail using the following graphic:



Note!



The "Homed" status is retained for the display after referencing.
The "Warning" state is only available as a display in the Visu, but not programmed.

The state machine is displayed in the footer of the visualization:



12.2 Homing

The referencing of the axis is started as soon as you click the "Homing" button in the sidebar in the visualization. The homing method "Set homing position" is executed in the drive as an example. The "MC_Homenit_SetPosition" function block is used for this. This can of course be changed at will. In the "S100_Idle" state you can see what happens when homing or referencing is started:

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

First the referencing method is set in the drive:

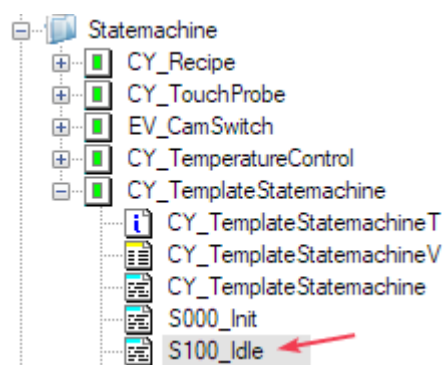
```
(* ----- *)
300: s_State := '300: Enable MC_HomeInit_SetPos';
(* ----- *)
```

Homing is then carried out with the "MC_Home" function block:

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

12.3 Automatic

The state machine automatically goes into automatic mode as soon as the "Start" button in the [Control bar](#) was actuated. You do not have to switch to automatic mode manually here. In the state machine in the state "S100_Idle" you can see what happens as soon as the start button has been pressed and which condition is linked to it:



```
(* -- 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;
```

Here you can see that depending on whether the gear was activated in the visualization or not, different states are jumped to.

12.3.1 Gear

If the gearing is activated, the master engine is started first and then the module MC_GearIn, which couples the slave axis to the master with a preset gear factor of 1:

```
(* ----- *)
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

If the gearing is not activated in the visualization, the axis is coupled to the master engine via the "MC_CamIn" function block. Here, too, a gear factor of 1 is preset by default:

```
(* ----- *)
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;
```

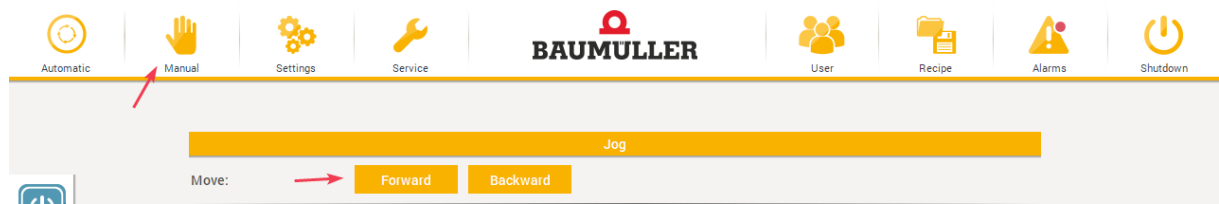
This can of course be changed at any time.

The CamTableID, which is set in the visualization, is set here. However, since there is no CamTable with ID 0, the entry is checked in the POU "CY_HMI_In / CY_HMI_Set" and the value is set to 1 if necessary:

```
(* 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

The manual state becomes active as soon as the axis is tapped on the "Hand" page in the visualization. Here too, analogous to the automatic mode, there is no need to switch to manual mode. Of course, this can be adjusted or changed later.



In the state machine in the PLC you can see under the worksheet "S100_Idle" what exactly happens as soon as the axis is jogged:

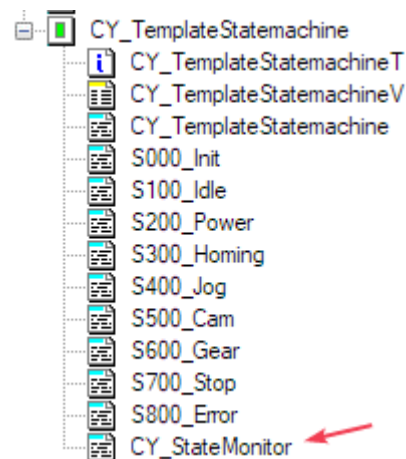
```
(* -- Jog -- *)
IF _Iface._Cmd.x_Jog_Fwd OR _Iface._Cmd.x_Jog_Bwd AND x_PoweredOn THEN
    i_State := 400;
END IF;
```

The corresponding process can then be found in the "S400_Jog" worksheet. Here the function block "BM_Jog_EV" is used to jog the axis:

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

12.5 State monitor

With the state monitor it is possible to monitor all the states run through and also to log them in a file on the PLC. The state monitor can be found in the last worksheet of the "CY_TemplateStateMachine" and the "CY_TemperatureControl" POUs:



Ultimately, this is a module that offers many functions for logging the activities of a state machine. Details can be found in the comments or in the module help.

The log files can be found on the PLC in the following directory:

