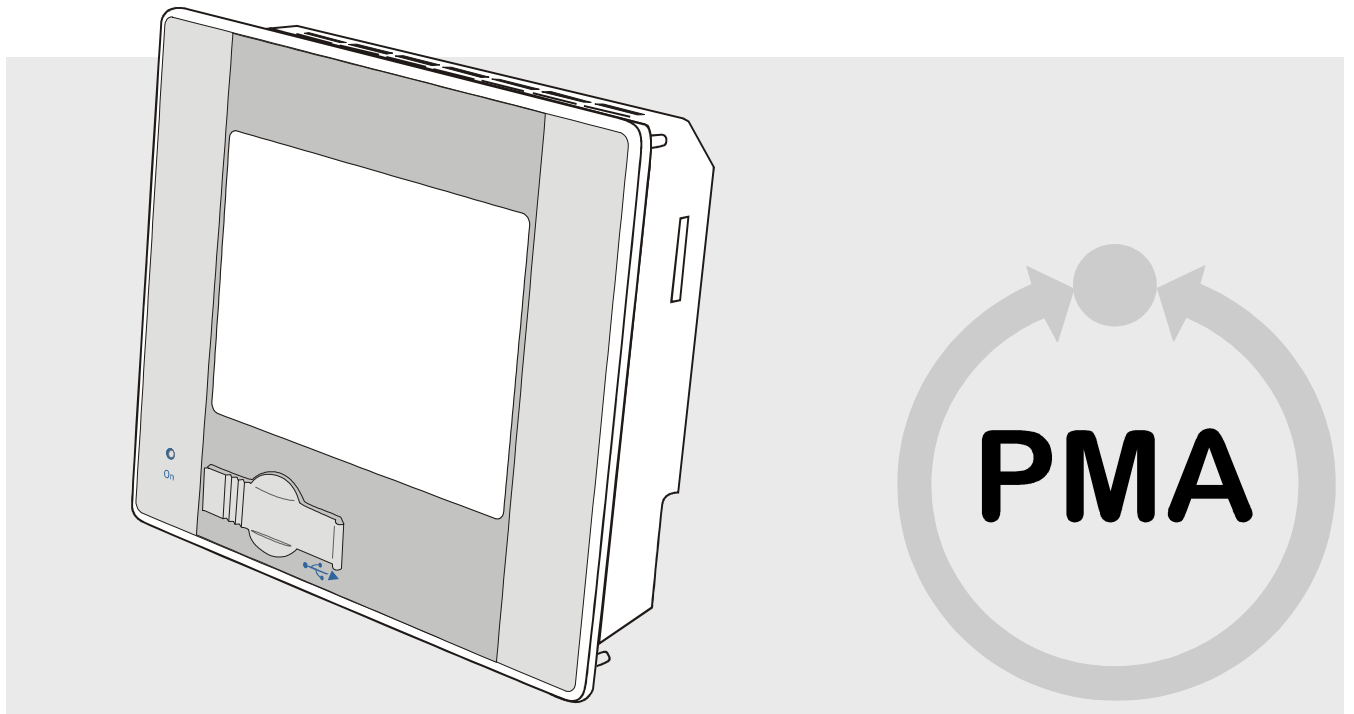


PMA Prozess- und Maschinen-Automation GmbH

# Manual Universal Programmer

**KS 108 easy**



preliminary



**Operator's guide, please read before using product**

Bestellnummer:  
9499-040-85911

© PMA  
Prozeß- und Maschinen-Automation GmbH  
Miramstraße 87  
34123 Kassel

Tel.: + 49 / 0561 / 505-0  
Fax.: + 49 / 0561 / 505-1710

[mailbox@pma-online.de](mailto:mailbox@pma-online.de)  
[www.pma-online.de](http://www.pma-online.de)

Release: 1  
Revision: 0

## Contents

|            |   |           |
|------------|---|-----------|
| <b>I</b>   | <b>Universal programmer .....</b>   | <b>5</b>  |
|            | I-1.1 General .....   | 5         |
|            | I-1.2 Prerequisites .....   | 5         |
| <b>I-2</b> | <b>Installation and configuration .....</b>   | <b>6</b>  |
|            | I-2.1 BlueDesign programming environment .....                                      | 6         |
|            | I-2.1.1 Installing BlueDesign .....   | 6         |
|            | I-2.1.2 Licencing BlueDesign .....  | 6         |
|            | I-2.1.3 Configuring BlueDesign .....  | 6         |
|            | I-2.1.4 Installing the Vario configurator .....                                     | 6         |
|            | I-2.2 BlueEdit program editor .....   | 6         |
|            | I-2.2.1 Installing, configuring and licencing BlueEdit .....                        | 6         |
| <b>I-3</b> | <b>Components for program creation .....</b>  | <b>8</b>  |
|            | I-3.1 KS 108 easy with run-time environment .....                                   | 8         |
|            | I-3.2 BlueDesign .....  | 8         |
|            | I-3.3 PMA library .....   | 9         |
|            | I-3.4 BlueSimulation .....  | 10        |
|            | I-3.5 BlueEdit .....  | 10        |
| <b>I-4</b> | <b>From the idea to the programm sequence in the device .....</b>                   | <b>12</b> |
|            | I-4.1 Preparation: Defining programs .....  | 12        |
|            | I-4.1.1 Basic program structure .....   | 12        |
|            | I-4.2 BlueDesign: Creating an engineering using the PROGRAMMER function block ..... | 13        |
|            | I-4.3 BlueEdit: recipes .....   | 27        |
|            | I-4.4 KS108 easy : Starting the programmer operation .....                          | 28        |
|            | I-4.4.1 Loading the engineering .....   | 28        |
|            | I-4.4.2 Loading a recipe .....  | 29        |
|            | I-4.4.3 Starting the programmer .....   | 30        |
| <b>I-5</b> | <b>Program management: BlueEdit .....</b>   | <b>40</b> |
| <b>I-6</b> | <b>Tutorial: A practical example of a "programmer project" .....</b>                | <b>41</b> |
|            | I-6.1 Step 1: Creating a new project .....  | 47        |
|            | I-6.2 Step 2: Creating a controller .....   | 49        |
|            | I-6.3 Step 3: Creating a programmer .....   | 54        |
|            | I-6.4 Step 4: Creating the password management .....                                | 56        |
|            | I-6.5 Step 5: Creating the simulation .....   | 58        |
|            | I-6.6 Step 6: Determining interfaces, connecting program blocks .....               | 61        |
|            | I-6.7 Step 7: Determining parameters .....  | 65        |
|            | I-6.8 Step 8: Generate a symbol file .....  | 69        |
|            | I-6.9 Step 9: Creating a recipe .....   | 70        |
|            | I-6.10 Step 10: An application test .....   | 71        |
| <b>II</b>  | <b>Index .....</b>  | <b>73</b> |



# I Universal programmer

## I-1.1 General

The following section gives an overview of the functions provided by the programmer in *KS108 easy* and how to use them. Moreover, you will learn how to handle some typical tasks (for instance, defining a program).

The required environment for the function block PROGRAMMER includes the following components:

- The KS 108 instrument with the PMA and BlueDesign run-time software
- BlueDesign/PMA library with KS108 BlueSimulation
- I/O system
- BlueEdit for KS108

In the further course of this chapter, these components are described shortly with an explanation of their interactions. The chapter presents the steps required to create an operable program for KS108 that can be practised using an application example.

## I-1.2 Prerequisites

To work with the PROGRAMMER, the following knowledge is required:

- Basic knowledge of the *Microsoft Windows*™ operating system

## I-2 Installation and configuration

### I-2.1 BlueDesign programming environment

#### I-2.1.1 Installing BlueDesign

The BlueDesign installation package can be found either on the CD delivered with *KS108 easy* or downloaded from the PMA homepage. Start the "*ibluedesign.exe*" installation software.

Follow the instructions displayed on the screen.



**NOTE!**

*A detailed description for installation of BlueDesign is given in Chapter II "Development environment" of the KS108 easy manual.*

#### I-2.1.2 Licencing BlueDesign

Without a valid licence, projects can be opened and handled in *BlueDesign* only during a short time. Enter the delivered licence under "?" in menu item „License..." of the main menu.



**NOTE!**

*Information on BlueDesign licencing is given in Chapter II, section "Development environment" of the KS108 easy manual.*

#### I-2.1.3 Configuring BlueDesign



**NOTE!**

*For information on the configuration of BlueDesign, refer to Chapter II, section II "Operation of the development environment" in the KS108 easy manual.*

#### I-2.1.4 Installing the Vario configurator

If you use the *Vario IO system*, you need the additional *VarioConfiguration* program. The installation package for *VarioConfiguration* can be found either on the CD delivered with *KS 108 easy* or downloaded from the PMA homepage. Start the "*varioconfiguration.zip*" installation package.

Follow the instructions displayed on the screen.



**NOTE!**

*For a detailed description how to install the VarioConfiguration program, refer to Chapter II "Development environment" of the KS108 easy manual.*

### I-2.2 BlueEdit program editor

#### I-2.2.1 Installing, configuring and licencing BlueEdit

The installation package for BlueEdit can be found either on the CD delivered with *KS108 easy* or downloaded from the PMA homepage. Start the "*iblueedit.exe*" installation package.

Follow the instructions displayed on the screen.

**NOTE!**

*For information on the installation of BlueEdit, refer to the BlueEdit operating manual.*

BlueEdit requires a valid licence.

**NOTE!**

*For information on the installation of BlueEdit, refer to the BlueEdit operating manual.*

**NOTE**

*For information on the configuration of BlueEdit, refer to the BlueEdit operating manual.*

## I-3 Components for program creation

The following section provides an overview of the components required to create a program for KS108 easy using the new PROGRAMMER function block.

Unless you have already worked using the PMA library or the development environment, we recommend reading the description "A practical example" in Chapter II Development environment of the *KS108 easy* manual first. In this section, the interaction of the *BlueDesign*, *BlueSimulation* and *KS108easy* components is explained at a practical example.

For program creation, a practical example for quick entrance is provided at the end of this chapter. See section Tutorial.

### I-3.1 KS 108 easy with run-time environment

Basically, *KS 108 easy* is a computer with the LINUX operating system, i.e. in principle, the instrument might be used as a "normal" LINUX computer. Making *KS 108 easy* a powerful multi-function controller is achieved mainly by three software components:

- **BlueDesign run-time environment:** Permits the execution of *BlueDesign* applications on the instrument. User programs are created in the *BlueDesign* development environment and transferred to *KS 108 easy*, where they are executed in the run-time environment of the *BlueDesign* software.
- **PMA library:** Provides powerful function blocks for system operation which are used by the *BlueDesign* applications.
- **BlueEdit program editor:** For convenient recipe creation and management. Recipes are loaded into *KS108* dependent on requirement. The recipes make the PROGRAMMER function block a powerful but easy to operate programmer.

### I-3.2 BlueDesign

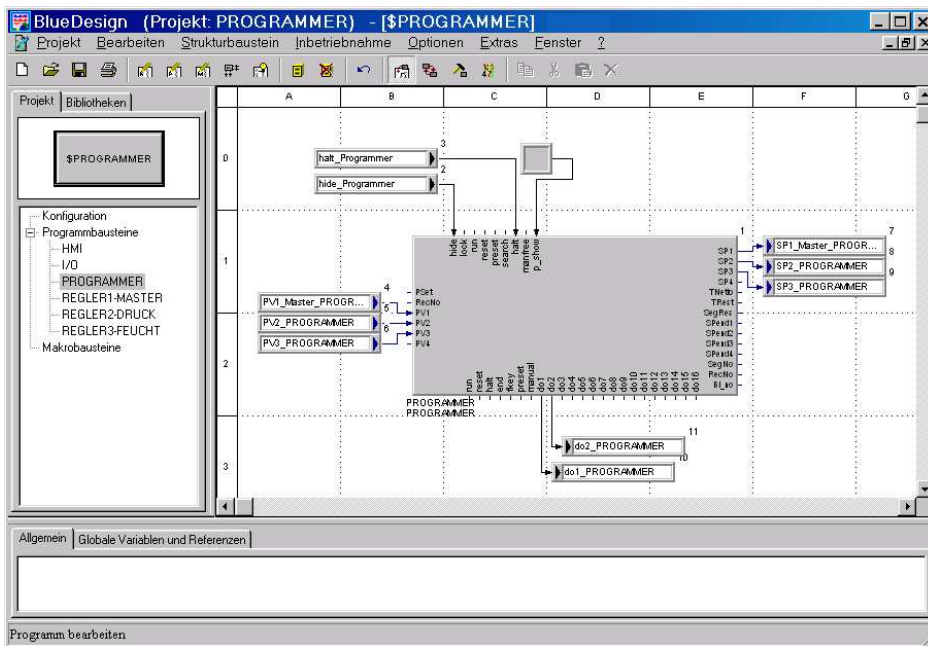


Fig. 1: BlueDesign

*BlueDesign* is a development environment for control applications. Applications are created in *BlueDesign* by selecting prefabricated components (e.g. programmers or controllers) using a graphic editor and connecting them. Programming knowledge is not required for this purpose.

Creating a project using *BlueDesign* includes the following steps: Developing the application, creating an operator interface, realizing the parameter setting and testing the application.



**NOTE!**

Detailed information how to create applications using *BlueDesign* and a practical example are given in Chapter II Development environment.

### I-3.3 PMA library

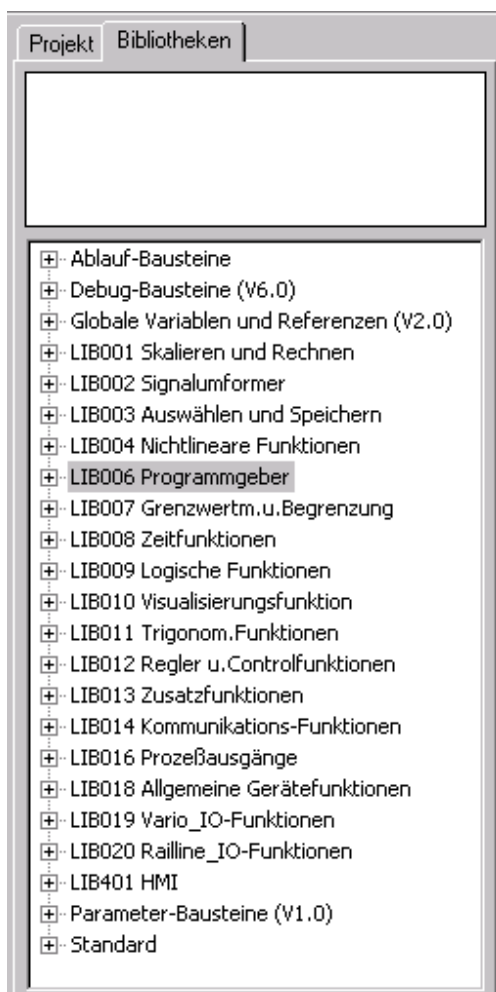


Fig. 2: PMA-library

The engineering for *KS 108 easy* is developed using the *BlueDesign* tool based on the PMA library.



**NOTE!**

For information on the PMA library, refer to Chapter II Development environment.

## I-3.4 BlueSimulation

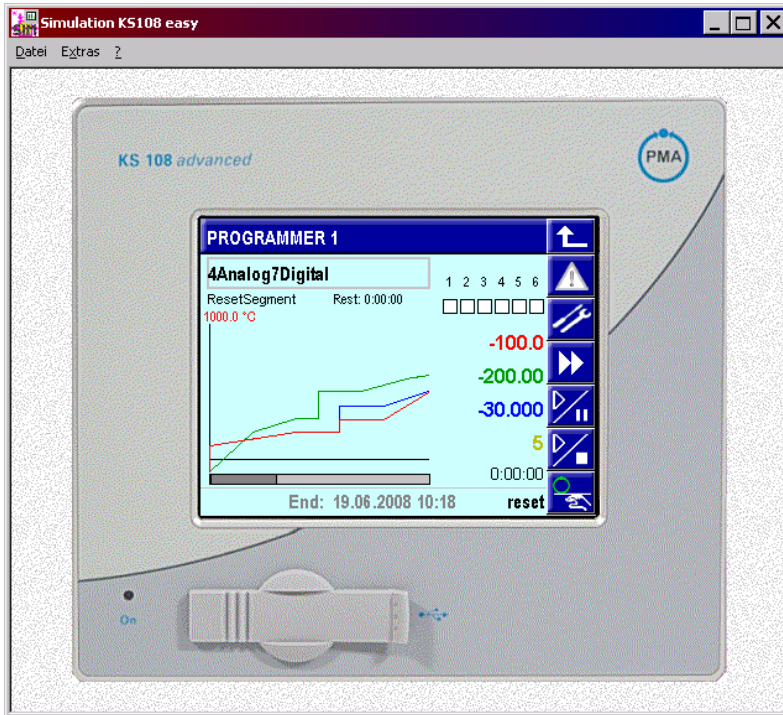


Fig. 3: BlueSimulation

The *BlueSimulation* tool simulates the *KS 108 easy* instrument. Behaviour and displays correspond to those of *KS108 easy*. Using the simulator, application development and program testing are possible also without the instrument.

## I-3.5 BlueEdit

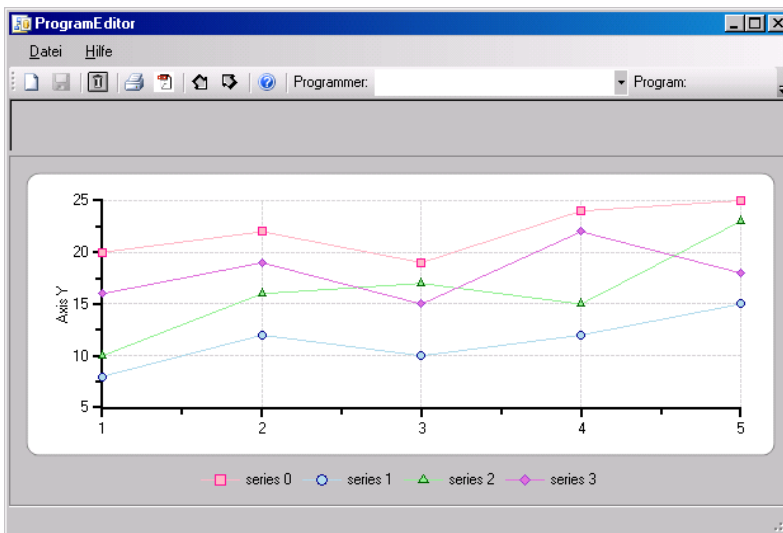


Fig. 4: BlueEdit with displayed profiles

The *BlueEdit* program editor is used for recipe and program creation, handling and management.

The basic parameters for the recipes are read out from an existing *BlueDesign* engineering which contains at least one PROGRAMMER function block.

These include the number of available PROGRAMMER function blocks, whereby each of these function blocks comprises the recipe list, the number of analog and digital tracks. This is the framework for the recipes of this engineering for *KS108 easy*.

The recipes are created using *BlueEdit* and transferred into the *KS108 easy*. They can also be tested using the simulation. To work with the recipes, the engineering must run in *KS108 easy* with the correct parameters.

**NOTE**

*Subsequent changes of the parameters (recipe list, number of tracks) in BlueDesign, KS108 easy or BlueEdit must be added manually, i.e. the parameters in the recipe and in the instrument must be identical.*

# I-4 From the idea to the programm sequence in the device

## General

The programmer comprises a function block **PROGRAMMER** and at least one recipe file with a program. The program files are created conveniently using the *BlueEdit* program editor, however, they can be changed in the instrument. The programmer (**PROGRAMMER**) may have up to 4 analog and up to 16 digital tracks.

Each **PROGRAMMER** file includes a recipe with a program comprising an arbitrary number of segments. Additional recipes are added as one file for each recipe. The number of segments or recipes is limited only by the available memory size.

The directory in which the recipes are stored is determined in the engineering (use an own directory for each **PROGRAMMER!**). Recipe selection is possible on the operating page or via the analog input **RecNo**.

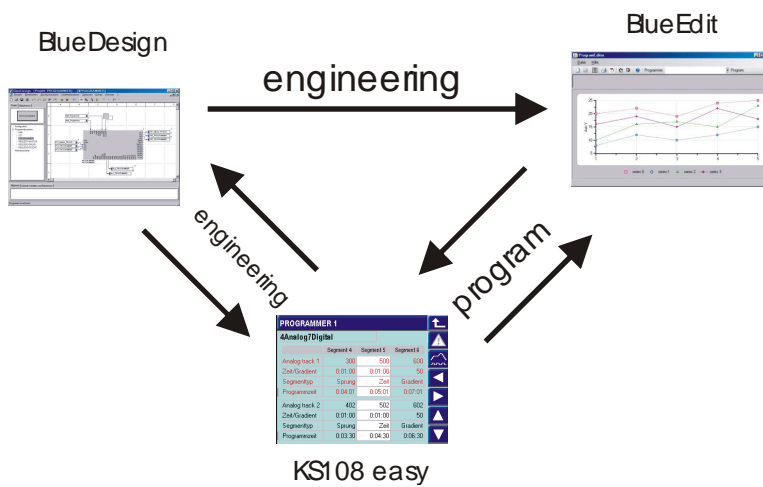


Fig. 5: Interaction of BlueDesign and KS108 easy

## I-4.1 Preparation: Defining programs

### I-4.1.1 Basic program structure

For using the universal programmer (FB PROGRAMMER), some basic parameters must be determined. These must be identical in the engineering and in the program and should be defined at the very beginning. Subsequent changes during the further course are possible. However, they require manual adaptations in the other components because these settings must be identical in the recipe and in the programmer configuration.

#### Programmer directory

A programmer may have a single recipe, however, a choice of several ones is normally offered. To enable the programmer in *KS108 easy* to find the programs and to change only its own ones, an own directory with a unique name must be specified for each programmer (i.e. for each PROGRAMMER function block). This directory must be entered into BlueDesign during configuration of the function block.

When loading the engineering into *KS108 easy*, this directory is created, unless it exists already.

The *BlueEdit* program editor reads out the directory for each programmer function block from the engineering and can then store the related recipes for each programmer in the right location in *KS108 easy*.

### Number of analog tracks and master track

Each programmer can have up to 4 analog tracks. The first track is always the master track. The other tracks are coupled to this track either via the segment or via the time. For this reason, it is important to know the relationship of the tracks.

The names defined for the tracks should be as clear as possible.

Moreover, the operation of the programmer offers the possibility to assign a colour to each analog track so that the operator can realize the curves and values at a single glance.

### Number of digital tracks: control tracks

Each programmer can have up to 16 digital tracks. The tracks are coupled to the master track via the segment and via the time.

The names defined for the tracks should be as clear and meaningful as possible.

During operation, the first six control tracks are displayed on the main page. For this reason, it is purposeful to assign the most important signals to control tracks DO1 to DO6.

Tip: To make an own segment for each switch point unnecessary, the user can set the status of the digital tracks independent of the relevant segment start and end by means of a switch-on delay and a duty cycle.

## I-4.2 BlueDesign: Creating an engineering using the PROGRAMMER function block

The programmer in the engineering realizes the recipe-dependent program sequences represented by the controllers and by the remaining engineering with the application.

The most important settings and requirements for the programmer in the engineering are:

- Signal soft-wiring, i.e. connection of the PROGRAMMER to the overall engineering
- Configuration and parameter setting
- If necessary: user levels and passwords
- If necessary: HMI user visualization

Insert the PROGRAMMER function block into the engineering.

The programmer must be soft-wired dependent on requirements, whereby the following considerations must be taken into account:

- Are there any process values and where do they come from (e.g. search)?
- Which control inputs are required?
  - Which steps do you need to control via operation?
  - Does the operation have to be available at any time, or do you want to lock or suppress the display of operating pages when defined conditions arise?
  - Is manual operation required?
  - Do you need to make the program editing page available to permit modification of program data such as segment times or bandwidths?
- Which analog and digital tracks (SP1...SP4, do1 ... do16) are used, and where do you want to connect them?
- Which additional displays and signals do you want to use in the engineering?
- How many analog and how many digital tracks are provided?
- Which programmer behaviour after start-up and at the program end do you need to configure?

- Which are the couplings between the analog tracks?
  - Which names do you want to use on the operating pages?
  - What is the name of the recipe directory (unique name in *KS108 easy*)?
- Which users do you need to take into account? Is the engineering already provided with user (level) management via a PASSWORD function block, or do you have to create it now?
- Which names do you want to assign to the users?
  - Which passwords (pass numbers) do you want to use?
  - Is the user management via the PASSWORD operating page accessible at any time?

The key data are the PROGRAMMER themselves and for every of these functionblocks the list of recipes and the amount of analog and digital tracks. If these are known and listed in the engineering, the creation of recipes can be started.

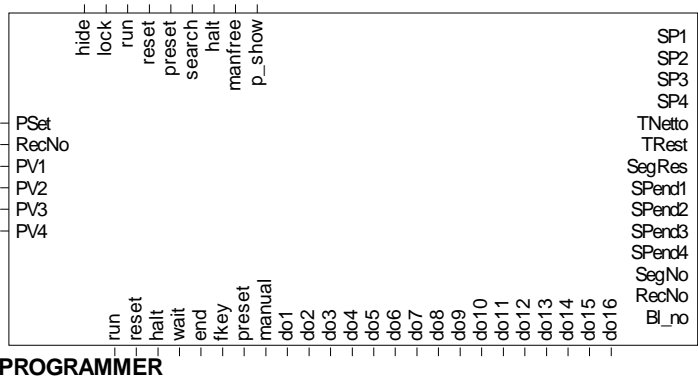


Fig. 6

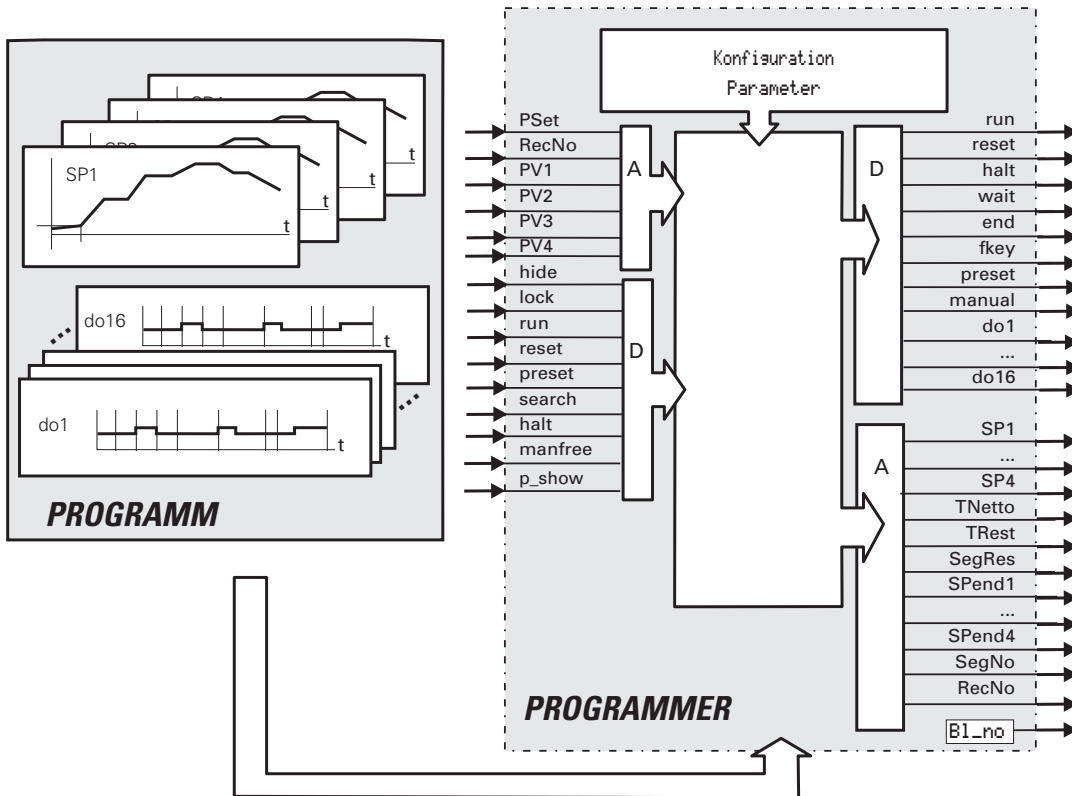


Fig. 7

If an error is detected when opening a recipe file, the reset value is output (status display on operating page: 'Error'). After an engineering download, the **Reset** status is active. If input **run** isn't soft-wired, **stop** is used.

### Master and slave tracks

The first analog track is always the master track. All other tracks are slave tracks.

| In-/Outputs PROGRAMMER |       |  |
|------------------------|-------|--|
| Name                   | Type  | Description  |
| PSet                   | Float | Preset value (time or segment, dependent on parameter PMode)   |
| RecNo                  | Float | Required recipe number. The recipe number (setpoint) determines which recipe should be started next. Running recipes are not influenced. The selected recipe is activated only after the next reset. |
| PV1                    | Float | Process value 1 for search run (master)  |
| PV2 ...<br>PV4         | Float | Process value 2 for search run   |
| hide                   | Bool  | Display suppression. hide = 1: the page is not displayed in the operation.   |
| lock                   | Bool  | Value adjustment blocking via operation. lock = 0: adjustment enabled, lock = 1: adjustment disabled.  |
| run                    | Bool  | Program stop/run. run = 0: stop, run = 1: run  |
| reset                  | Bool  | Continue/reset program. reset = 0: continue, reset = 1: reset  |
| preset                 | Bool  | Program preset, 1 = preset   |
| search                 | Bool  | Start programm search run, 1 = search run (operates on all analog tracks)  |
| halt                   | Bool  | Program interruption (e.g. due to an exceeded bandwidth detected outside the programmer). Output "run" remains active. halt = 0: program is not halted. halt = 1: program is halted.                 |
| manfree                | Bool  | Disable manual mode. manfree = 0: switch-over to manual mode is not permitted, manfree = 1: switch-over to manual mode is permitted.   |
| p_show                 | Bool  | Enable program editing. Display and adjustment of all segment parameters pertaining to a recipe. Calling up this page is done from the main operating page.  |

| Name                 | Type  | Description  |
|----------------------|-------|--|
| SP1                  | Float | Programmer setpoint 1 (master)   |
| SP2 ... SP4          | Float | Programmer setpoint 2  |
| TNetto               | Float | Net program time. Elapsed program time without halt/stop times. (master) |
| TRest                | Float | Remaining programmer time (master)                                       |
| SegRes               | Float | Remaining segment time (master)  |
| SPend1               | Float | End value of current segment of analog track 1 (master)                  |
| SPend2 ...<br>SPend4 | Float | End value of current segment of analog track 2                           |
| SegNo                | Float | Current segment number (master)  |
| RecNo                | Float | Current recipe number  |
| BI_no                | Float | Own block number (e.g. for calling the operating page via CALLPG)        |

|                 |      |   |
|-----------------|------|---|
| run             | Bool | Program stop/run status; 0 = program stop, 1 = program run  |
| reset           | Bool | Program reset status; 1 = program reset   |
| wait            | Bool | Program wait status; 1 = Wait (one analog track has stopped the programmer at the end of a segment, because of a segment type with wait at the end).  |
| halt            | Bool | Program halt status (master); 1 = Program interruption (because of an external halt, an exceeding of the bandwidth or a wait on the other segment coupled slaves, which are still running)  |
| end             | Bool | Program end status (master); 1 = program end reached  |
| fkey            | Bool | F key status. Pressing the key causes a pulse.  |
| preset          | Bool | Single preset command: a pulse is output for the duration of one cycle (dependent on the programmer cycle time). Continuous preset command: the output is always active.<br>preset = 0: no preset status, preset = 1: PROGRAMMER stands in preset status. |
| manual          | Bool | Indication of manual mode, all tracks; manual = 0: none of the tracks of PROGRAMMER works in automatic mode, manual = 1: at least one track of PROGRAMMER works in manual mode.   |
| do1 ...<br>do16 | Bool | Status bit 1  |

| Parameter |             |          |   |        |         |       |     |
|-----------|-------------|----------|---|--------|---------|-------|-----|
| ID        | Name        | Type     | Description   | Access | Default | Range | Off |
| PMode     | Preset mode | Enum     | Preset mode for input PSet or interface: Preset to segment or to time.  | r/w    | 1       |       |     |
|           |             | Segment  | Preset to segment. Value at PSet input is used as target segment during a preset command (activated via preset input).  |        | 0       |       |     |
|           |             | Time     | Preset to time. Value at PSet input is used as target program time during a preset command (activated via preset input).  |        | 1       |       |     |
| SMode     | Search mode | Enum     | Behaviour when activating search via search input or at program start   | r/w    | 0       |       |     |
|           |             | Segment  | Search run in the segment. When starting the search run, setpoint SP is set to the value of input PV and approach to the segment end value is with the current gradient (TPrio = Grad.Prio) or in the current remaining segment time (TPrio). |        | 0       |       |     |
|           |             | Programm | Search run in the program or program section. Search only in segments the gradients of which have the same sign. (Hold segment is neutral). The search cycle time may be longer than one  |        | 1       |       |     |

|                       |                 |          |  |     |        |         |  |
|-----------------------|-----------------|----------|--|-----|--------|---------|--|
|                       |                 |          | processing cycle.  |     |        |         |  |
|                       |                 | Off      | Search run switched off  |     | 2      |         |  |
| TPrio                 | Start priority  | Enum     | Start mode in search run determines the higher priority of gradient or segment/time.   | r/w | 0      |         |  |
|                       |                 | Gradient | Start mode in search run: Gradient has priority. When starting the search run, setpoint SP is set to the value of input PV and goes to the segment end value with the current gradient.                |     | 0      |         |  |
|                       |                 | Time     | Start mode in search run: Time has priority. When starting the search run, setpoint SP is set to the value of input PV and approach to the segment end value is in the current remaining segment time. |     | 1      |         |  |
| Dp1 ...<br>Dp4        | Decimals 1      | Int      | Number of digits behind the decimal point for display of the setpoint 1  | r/w | 3      | 0 ... 3 |  |
| SPl01<br>...<br>SPl04 | Min. setpoint 1 | Float    | Lower limit of set-point 1   | r/w | 0.0    |         |  |
| SPh1<br>...<br>SPh4   | Max.setpoint 1  | Float    | Upper limit of set-point 1   | r/w | 1000.0 |         |  |

| Configuration |                     |          |   |        |         |          |     |
|---------------|---------------------|----------|---|--------|---------|----------|-----|
| ID            | Name                | Type     | Description   | Access | Default | Range    | Off |
| ATracks       | Analog track number | Int      | Number of the used analog tracks  | r/w    | 1       | 1 ... 4  |     |
| DTracks       | Contr.output number | Int      | Number of the used control outputs  | r/w    | 6       | 0 ... 16 |     |
| PwrUp         | On mains recovery   | Enum     | Behaviour after mains recovery  | r/w    | 0       |          |     |
|               |                     | Continue | Continue program at the point it was stopped by power failure.  |        | 0       |          |     |
|               |                     | Search   | Search run after mains recovery: forward and backward search from the point of failure, search function depends on parameter SMode. |        | 1       |          |     |
|               |                     | Continue | Continue at actual time. Search run after mains   |        | 2       |          |     |

|                               |                       |                  |  |     |   |  |  |
|-------------------------------|-----------------------|------------------|--|-----|---|--|--|
|                               |                       | at time          | recovery: forward and backward search from the program time in which the program would be without power failure, search function depends on parameter SMode. Hint: If the program time from power failure to power recovery includes min. one segment with wait status at the end, the search run in the segment in which the program would be without power failure is omitted and the program stops at the first wait status without search run. |     |   |  |  |
| PEnd                          | On Program end        | Enum             | Behaviour at program end   | r/w | 0 |  |  |
|                               |                       | Stop             | After program end: stop (the setpoint of the last segment remains unchanged).  |     | 0 |  |  |
|                               |                       | Reset            | After program end: reset (the start setpoint becomes active. The program restarts automatically, when the run condition has remained unchanged.)   |     | 1 |  |  |
|                               |                       | Reset + Stop     | After program end: reset+stop (continuous reset, start setpoint with reset und stop).  |     | 2 |  |  |
| Coupling2<br>...<br>Coupling4 | Mast/slave<br>coupl.2 | Enum             | Coupling between slave track 2 and master track, slave 2 is coupled with time or segment of the master track.  | r/w | 0 |  |  |
|                               |                       | Time coupling    | Slave works on the same program time like the master (segments can be different).  |     | 0 |  |  |
|                               |                       | Segment coupling | Slave works every time in the same segment like the master (program time can be different).  |     | 1 |  |  |
| SpScale2<br>...<br>SpScale4   | Setpoint<br>scaling 2 | Enum             | Define scaling for the graph on the main page for track 2 independent from master or take over master scaling.   | r/w | 0 |  |  |
|                               |                       | Own              | Own graphic scaling: SpLo ...  |     | 0 |  |  |

|                      |                     |                |  |     |                  |                   |  |
|----------------------|---------------------|----------------|--|-----|------------------|-------------------|--|
|                      |                     | scaling        | SpHi   |     |                  |                   |  |
|                      |                     | Master scaling | Graphic scaling from master: SpLo1 ... SpHi1                                 |     | 1                |                   |  |
| Color1 ... Color4    | Setpoint color 1    | Enum           | Color of graph on main page  | r/w | 0                |                   |  |
|                      |                     | Red            | Red  |     | 0                |                   |  |
|                      |                     | Blue           | Blue   |     | 1                |                   |  |
|                      |                     | Green          | Green  |     | 2                |                   |  |
|                      |                     | Yellow         | Yellow   |     | 3                |                   |  |
| TChart               | Visible time [min]  | Float          | Visible time for graph on main page  | r/w | 60.0             | 1.0 ... 1800000.0 |  |
| A1Title ... A4Title  | Name analog track 1 | Text           | Name for analog track 1  | r/w | Analog track 1   |                   |  |
| Unit1 ... Unit4      | Unit 1              | Text           | Unit of setpoint 1   | r/w | Unit             |                   |  |
| D1Title ... D16Title | Name contr. output1 | Text           | Name for control output 1  | r/w | Control output 1 |                   |  |
| RecDir               | Recipe directory    | Text           | Name of the directory, in which the recipes of the function block are stored | r/w | PROGRAMME R      |                   |  |

### Segment types

| Name         | Type        | Description   | Access | Default | Range |
|--------------|-------------|---|--------|---------|-------|
| Segment type | Enum        | Setpoint behaviour in the segment. The setpoint can be held or changed with a ramp or a step change. Continuation is automatic or manual ("Wait for operation"; configurable).  | r/w    | 8       |       |
|              | Time        | Time segment: The setpoint changes linearly in the segment duration from the start value to the target setpoint of the relevant segment. The gradient is a function of this change (start value = end value of the previous segment)              |        | 0       |       |
|              | Rate        | Rate to setpoint: The setpoint changes linearly with the adjusted gradient from the start value to the target value of the relevant segment. The segment duration is a function of this change (start value = end value of the previous segment). |        | 1       |       |
|              | Dwell       | Dwell: The end setpoint of the previous segment is kept constant for the time.  |        | 2       |       |
|              | Step        | Step to setpoint: The setpoint goes to the adjusted target setpoint immediately.  |        | 3       |       |
|              | Time + wait | Time to setpoint and wait. The setpoint changes linearly from the start value to the target setpoint in the duration of the segment. The programmer goes to stop condition at the end of segment.   |        | 4       |       |

|              |  |   |  |
|--------------|--|---|--|
| Rate + wait  | Rate to setpoint and wait: The setpoint changes linearly from the start value to the target setpoint with the adjusted gradient. The segment duration is a function of this change (start value = end value of the previous segment). At the segment end, the programmer goes to the stop condition. | 5 |  |
| Dwell + wait | Dwell and wait: Dwell segment: The end setpoint of the previous segment is kept constant during the segment time. At the end of the segment, the programmer goes to the stop condition.  | 6 |  |
| Step + wait  | Step and wait: Step segment: The programmer goes to the adjusted target setpoint immediately and waits at the segment end.   | 7 |  |
| End          | The last segment in a program is the end segment. When reaching the end segment, the setpoint output last is held.   | 8 |  |

**Additional parameters that are determining for a program step:**

| Name                                  | Type  | Description  | Access | Default | Range           |
|---------------------------------------|-------|--|--------|---------|-----------------|
| (Segment) time / gradient<br><br>TpGr | Float | Time or gradient for the segment. The duration of a segment can be determined directly, or as a gradient and a setpoint difference SP - segment start setpoint. Whether segment time or gradient are concerned is determined in parameter Segment type (Type). | r/w    | 0.0     | 0.0 ... 1800000 |

| Name               | Type  | Description  | Access | Default | Range |
|--------------------|-------|--|--------|---------|-------|
| Setpoint<br><br>SP | Float | End value for the segment. Target value at the end of the first segment. Approach to this value is from the last valid setpoint (from the process value at the beginning of the 1 <sup>st</sup> segment). After elapse of the program, the controller continues controlling using the target setpoint adjusted last. | r/w    | 0.0     |       |

**Synchronization**

The analog slave tracks can be selected either as coupling by segment or by time to the master. A **time coupled** slave follows the master time, also if the master stops or changes its time because of search or preset. So it runs independently from the other slaves to its program end, where it is waiting until all tracks reach their end. After that the programmer executes its end function, which depends on the PEnd configuration.

If **segment coupling** is selected all tracks wait ('halt' state) for each other at the end of each segment, until all the tracks have ended this segment. This can happen, if the tracks have different segment times, if a search or a bandwidth violation has occurred or if a track is in a wait state at segment end.

All digital tracks are coupled by segment to the master. The segment times of these tracks are taken from the master, they do not have separate segment times.

**Analog tracks**



*HINT!*

*The first analog track is always the master track. All other tracks are slave tracks.*

### Start setpoint

The start setpoint is part of the recipe: setpoint of the 1st segment (segment with segment number 0). Start setpoint is active, if PROGRAMMER is in status **reset**. If no recipe is selected or the selected recipe is not available, e.g. the recipe does not exist or is not correct, or it does not match the configuration, the effective setpoint is set 0.

### Operation preparation and end position

Each program starts at an initial position. This position is used and maintained with reset or first programmer set-up.

With program start from rest position, the first programmer segment runs. The program starts from the instantaneous process value at the time of start command, if the corresponding process value was soft-wired at **PV** of the **PROGRAMMER** and **search run** was configured.

With step change mode, the setpoint of the first segment is activated immediately.

At program end, depending on configuration (PEND), either

- 0=Stop: the setpoint of the last segment is maintained

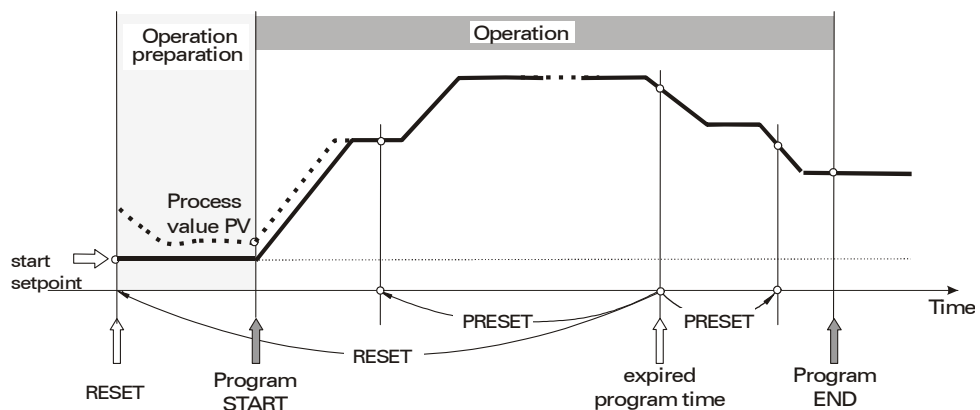


Fig. 8 Profile, with the end position kept unchanged

- 1 = Reset: the programmer goes to rest position **SP** of the first segment (segment 0). The program restarts automatically, if run is still valid.

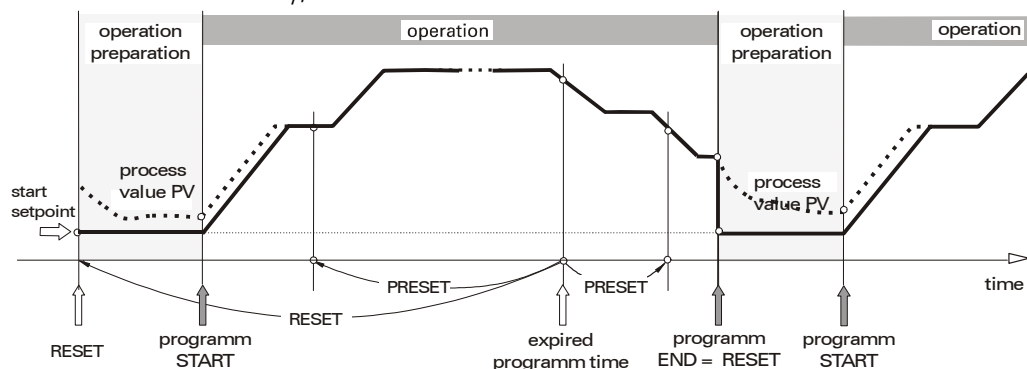


Fig. 9

- 2 = Reset + Stop: Programmer goes to rest condition **SP** of the first segment (segment 0) with reset and stops there.

At program end, the number of the last segment increased by 1 is output as active segment number (**SegNo** output). This is required to bring the slave block to the end status safely with a segment preset

## Search Run

The term *search run* describes the behaviour of the programmer to reach the end value starting from the current process value. A search run can be performed in the current program segment or over several program segments.

A search run is used in the following cases:

- After program start: via the operator interface or via the **run** input.
- After a program restart using the **reset** command on the programmer operating page, or via the **reset** input.
- Via the **search** input.

## Configuring the Programmer

In the *BlueDesign* parameter dialogue, you can determine, if a programmer can perform search runs and which type of search can be performed.

For this purpose, the *SMode* parameter offers the following options:

- Segment: search runs are made in the program segment.
- Program: search runs are made in the program section. A program section is defined as a sequence of program sections with gradient of the same direction (only upwards or only downwards). Change in direction limits a program section.
- Off: No search runs are made.

| Search mode |          |
|-------------|----------|
| 0:          | Segment  |
| 1:          | Programm |
| 2:          | Off      |

Fig. 10: Parameter "SMode"

## Search Run in the Program Segment

With a search run, the process value (*PV*) is set as programmer set-point first. Then the process value is controlled to the segment end value either using the current gradient or using the current remaining segment time (depending on the programmer configuration).

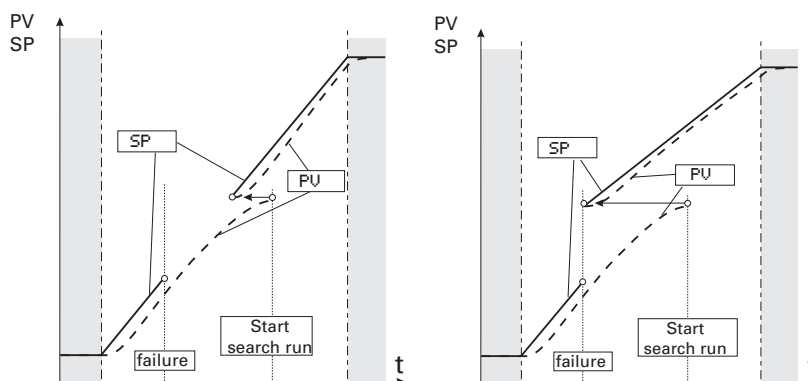


Fig. 11: Search run with the current gradient/the current remaining segment time

Configuring the programmer is done using the *TPrio* parameter in the *BlueDesign* parameter dialogue. When making a search run, the following information must be taken into account:

- If setting "TPrio = Gradient" is used for the search run and the search value is out of the current segment, the program is continued at the segment point next to the search value.
- If the start value is equal to the end value (a segment without gradient) the program is continued at the segment start.
- With a jump segment, start is always at the beginning of the segment. The target set-point is assigned to the process value *PV*.

### Search Run in the Program Section

A search run in the program section is limited to a section of several segments that have the same sign as the gradient, whereby hold segments are not considered as changes of sign. If the current program section contains hold segments, a search run is performed only, if this program section contains at least one more program segment which isn't a hold segment. If this segment is directly preceded or followed by another hold segment, the search run is realized only in the current segment.

Depending on the number of segments included in the search run, the run-time may be very long. For this reason, searching is divided into several partial runs. Each partial run checks only one segment. This is an automatic procedure which needs no interaction from operator.



#### NOTE!

*If the value of parameter "TPrio" is "1: time", the search run is always limited to the current program segment.*

*Program segments with a final waiting state (e.g. "time + wait") do not limit the search range (exception: if a search run after a power failure is concerned).*

*A search run can lead to termination of the program.*

### Particularities of a search run with the PROGRAMMER

Dependent on master/slave coupling, a search run can have various effects:

| Master/slave coupling | Search run function   | Search run of a single track  | Simultaneous search run of all tracks  |
|-----------------------|-----------------------|---|--|
| Time coupling         | Search run in segment | Search run is permitted only for the master. As all slave tracks follow the master track time, no search run occurs with the slave tracks.              |  |
|                       | Search run in program |   |  |
| Segment coupling      | Search run in segment | A search run is performed for the track. At the segment end, all tracks with segment coupling wait for each other before starting the next segment. (*) |  |
|                       |                       |   | A search run is performed for all tracks with segment coupling. At the segment end, they wait for each other before starting the next segment. (*) |

|                       |   |
|-----------------------|---|
| Search run in program | <p>Permitted only for the master track.</p> <p>The master track performs a search run. All tracks with segment coupling follow the master into the new segment and perform a search run in this segment. If this segment search run isn't successful, the track continues at the segment start or end value, dependent on the setpoint (start or end) which is closer to its process value. If the process value (input <b>PV</b>) of a slave track with coupled segments is not soft-wired, the slave track continues at the value of the elapsed time of the master track in the segment.</p> <p>At the segment end, all tracks with segment coupling wait for each other before starting the next segment (*).</p> |
|-----------------------|---|

(\*) With segment coupling of at least one analog track, the master track operates according to the same principle.

Dependent on parameter **SMode** (no search run / search run in the segment / search run in the program), the **search** input acts on all analog tracks simultaneously. Via the operating pages or the interface, a search run in the segment or in the program can be started independently of **SMode**. The search run for a single track can be started in the same way.

During a step or a dwell segment, the search run is without effect on this track.

If a slave track with segment coupling following the master with a search run in the program arrives at a dwell or step segment, the slave track continues at the value of the elapsed time of the master track in the segment and it waits there for the other tracks, before the next segment is started in common.

### Preset

A preset is used to change to a defined time in the program.

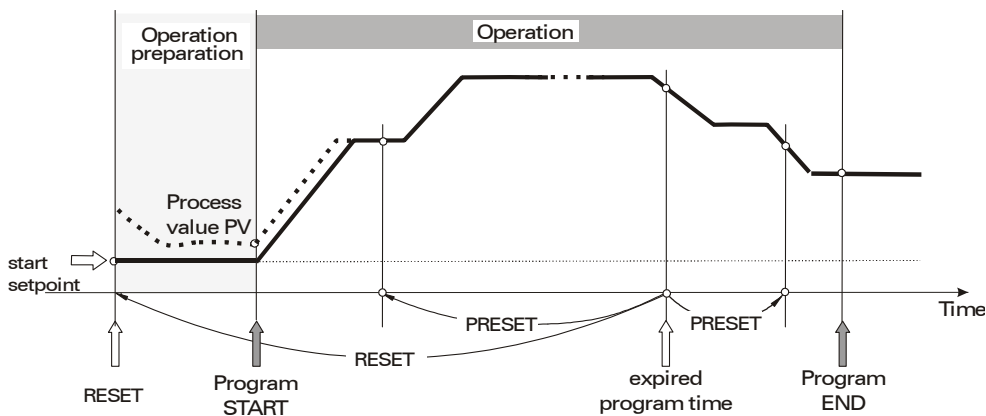


Fig. 12

For a detailed description how to call up the preset via operation, see operating page "Review profiles".

- A slave track with time coupling follows the master track.
- A slave track with segment coupling changes into the new segment of the master track and follows the time of the master track within this segment.

### Bandwidth monitoring

To prevent the separation between setpoint and process value from becoming too large, bandwidth monitoring is used. This function checks, if the control deviation is within the permissible limits and stops the programmer ("**Halt status**"), as soon as the separation becomes too big. As soon as the control deviation has decreased sufficiently, the bandwidth monitoring terminates the **halt** status and the program is continued.

If the separation between setpoint **SP** and process value **PV** exceeds the bandwidth (**BW**) parameter, **halt** is activated:

$$HALT \leftarrow ( |SP - PV| > bandwidth )$$

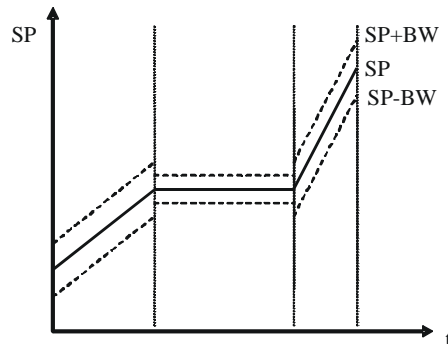


Fig. 13 Bandwidth around the setpoint SP

- Each analog track has its own process value (PV) input and its own bandwidth.
- There is one bandwidth for each segment. Value **BW** determines the separation between process value **PV** and setpoint **SP** without triggering the **halt** status.
- If the bandwidth is exceeded, a track with time coupling stops, until **PV** is within the permissible bandwidth again. During this time, all tracks with time coupling including the master track are stopped. A track with segment coupling shows the same bandwidth behaviour, but the other tracks continue up to their segment end. At the segment end, these tracks including the master track wait, until all tracks with coupled segments have reached the segment end. While the bandwidth is exceeded, **halt** is signalled at the **halt** output and on the corresponding operating pages (main page and detailed page).
- Bandwidth monitoring can be switched off by setting the function to "OFF".

### Control tracks (digital tracks)

#### Start values

The start values of the control tracks are included in the recipe: the values of the first segment (segment no. 0). The start values are active, when the **PROGRAMMER** is in the **reset** status. If no recipe is selected, or if the selected recipe is not available, e.g. because the recipe doesn't exist or isn't o.k., or because it doesn't suit the configuration, all control tracks are set to 0.

#### Segment time

The control tracks are coupled to the master track via the segment numbers. To reduce the division into segments, the two parameters **t+** and **t-** can be used:

- Delay **t-** : delay after which the control track reaches its adjusted value. Up to this time, the control track remains on the inverted value.
- Switch-on time **t+** : The control track remains on its adjusted value, until switch-on time **t+** has elapsed. If **t+** is set to "0", the switch-on time ends at the segment end.

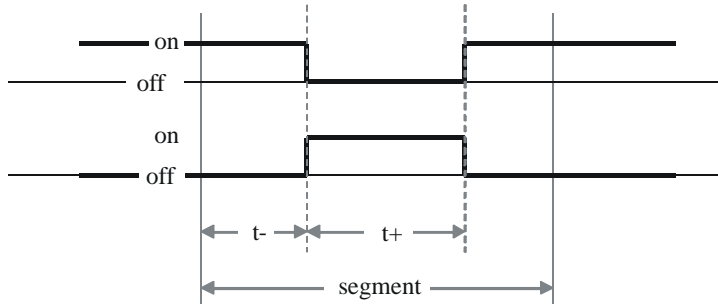


Fig. 14 Switch-on and off times within a segment

## Recipes

The recipe management, i.e. creating, deleting and editing, is performed using the BlueEdit program editor. A program can be changed also using the PROGRAMMER editing page.

Fill in the basic data for the recipes in BlueDesign: Recipe directory (each PROGRAMMER must have an own directory), number of required analog tracks, number of required digital tracks.



### CAUTION!

When using several PROGRAMMER function blocks, the recipes can be changed alternately.

As the PROGRAMMER function blocks start with the same recipe directory when edited in BlueDesign (default setting), they can access the same recipes. For this reason:

- A unique recipe directory name must be used for each PROGRAMMER function block.

For creating the recipes, the engineering must be read in using BlueEdit. Subsequently, suitable recipes can be created and edited. These are uploaded into the instrument as required. Recipes in the instrument can be downloaded, edited, stored and reloaded.

Each PROGRAMMER uses its own directory specified in the **RecDir** configuration. The recipe file names have a defined structure of recipe number (001 ... 999) and recipe name: "RecNo\_RecName .prf". During the start-up phase or after downloading the engineering, the PROGRAMMER generates its recipe directory, if it still doesn't exist.



### NOTE!

A recipe directory that isn't required any more is not deleted automatically. It must be deleted using an external program (e.g. FTP tool), for instance the program editor.

## Temporary recipe change

While PROGRAMMER is running, the user can change the active recipe temporarily, i.e. until the next **reset**.



### CAUTION!

Changes of the active program made on the editing page are effective **IMMEDIATELY**.

This may cause unpredictable and dangerous situations in the process. For this reason:

- Consider the effects before making changes in the current program and take measures to prevent potentially hazardous situations.

The temporary change is made on the PROGRAMMER editing page and changes will become active only during the future execution of the recipe (see editing page).

**NOTE!**

When quitting the editing page, a memory dialogue for permanent storage of recipe changes opens. If only a temporary change is required, "No" (= don't save changes permanently) must be selected.

**Recipe change – program selection**

The required recipe can be selected externally via the analog **RecNo** input, or internally using the recipe number adjusted via operation/interface.

**NOTE!**

During an active program sequence, switch-over to a different recipe on the programmer operating page is not possible. Recipe changing is possible only during the reset status.

**Halt status**

1. For example, used for external bandwidth monitoring

The **halt** status can be switched on and off using control input **halt**. Unlike the **stop** status, the **run** status remains unchanged during the **halt** status (the **run** output remains active). Status display is **halt**.

2. Internal bandwidth monitoring

The **halt** status can be triggered by internal bandwidth monitoring (see section Bandwidth monitoring). Status **halt** is displayed.

3. Synchronization of tracks with coupled segments

The **halt** status can be triggered when a track must wait for another track, because the tracks are coupled via segment coupling to the master (see section Synchronization). Status display is **halt**.

**I-4.3 BlueEdit: recipes****NOTE**

For information on recipe creation and handling, refer to the BlueEdit manual.

**I-4.4 KS108 easy : Starting the programmer operation****I-4.4.1 Loading the engineering**

In principle, you can address various target devices with BlueDesign. However, it is only possible to work with one device at a time. To address the device, BlueDesign needs information where to find the device .

**NOTE**

Detailed information on how to build up the communication is given in Chapter II Development environment "Logging on to the target system" in the KS108 easy manual.

Open the required engineering created in BlueDesign. To load it into *KS108 easy* log on to the target system from the Run menu and establish the connection.

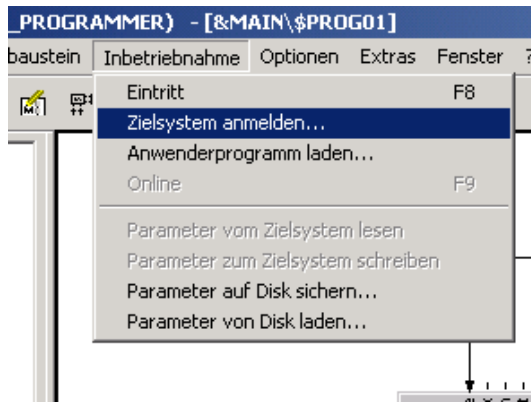


Fig. 15: Connecting to the target system

### Status indication

The status of connection to the device is displayed by symbols (the simulator display is shown as an example below):

| Symbol                | Meaning  |
|-----------------------|--|
| Simulation KS108 easy | The connection is active..                                       |
| Simulation KS108 easy | The connection was configured, but it is not used. konfiguriert. |
| Simulation KS108 easy | The connection isn't active.                                     |

### Loading the user program



#### DANGER!

This symbol is a warning against injury or against a hazard of material damage by unpredictable functions and movements in the process.

When using *KS 108 easy* in connection with other instruments/facilities, the program transmission may cause a risk of reactions of these instruments/actuators, etc. For this reason:

- Before connecting, consider the effects of a program update and take suitable protective measures.
- Prior to establishing the connection, make sure that the right program is loaded and
- ensure that the program is free of faults.

1. Open menu "Download ...":

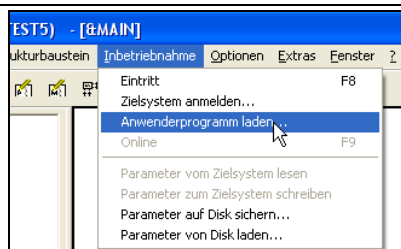


Fig. 16: Download

## I-4.4.2 Loading a recipe

In principle, you can address various devices with the *BlueEdit* program editor. To address the *KS108 easy*, you must provide *BlueEdit* with the information where to find the device.



### NOTE!

Detailed information on how to establish the connection is given in the *BlueEdit* operating manual.

Open the required recipe created in *BlueEdit*. To load it into *KS108 easy* establish the connection.



### DANGER!

This symbol is a warning against injury or against a hazard of material damage due to unpredictable functions and movements in the process.

When using *KS 108 easy* in connection with other instruments/facilities, the program transfer may cause a risk of reactions of these instruments/actuators, etc. For this reason:

- Before connecting, consider the effects of a program update and take suitable protective measures.
- Prior to establishing the connection, make definitely sure that the right program is loaded
- and ensure that the program is free of faults.

### 1. Open menu "Download ..."

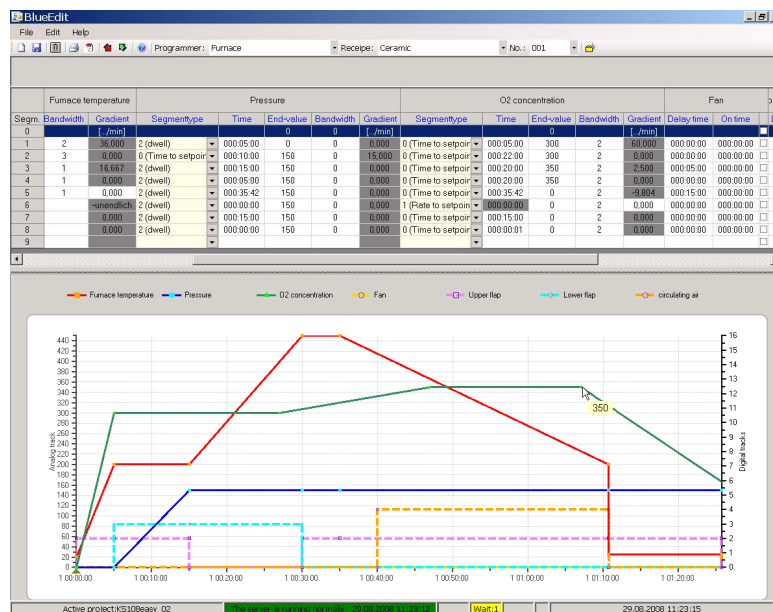


Fig. 17: Loading a recipe

## I-4.4.3 Starting the programmer

When the engineering and the related recipes were downloaded into *KS108 easy* or into the *BlueSimulation*, the programmer can be started.

If the engineering and the recipes are a match and if they use the same recipe directory for the various programmers, the first recipe is downloaded and can be started.

As the operation can be influenced extensively by the engineering, the operating principles are described in the following section.

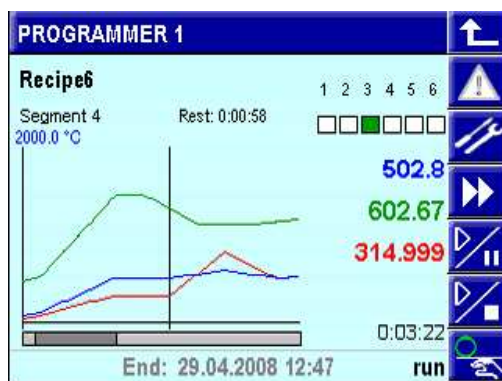





Fig. 18: Loading a recipe

### Programmer control via keys and

The programmer can be controlled via the following elements:

- the digital function block inputs,
- the interface, or
- using control keys  and .

The  key offers a choice of the following functions:

- **reset**: Selectable from any program position
- **preset**: From any program position. **preset** is used to select the review page (**preset**) to change to a different time in the program
- **segment search** : Start a search run in the segment.
- **program search** : Start a search run in the program.

The  key controls the programmer (fkey output generates a pulse when a key is activated).

For both keys, the rule saying that the states at soft-wired control inputs must be given priority over the operation. The following diagram describes the sequence of states dependent of the actions:

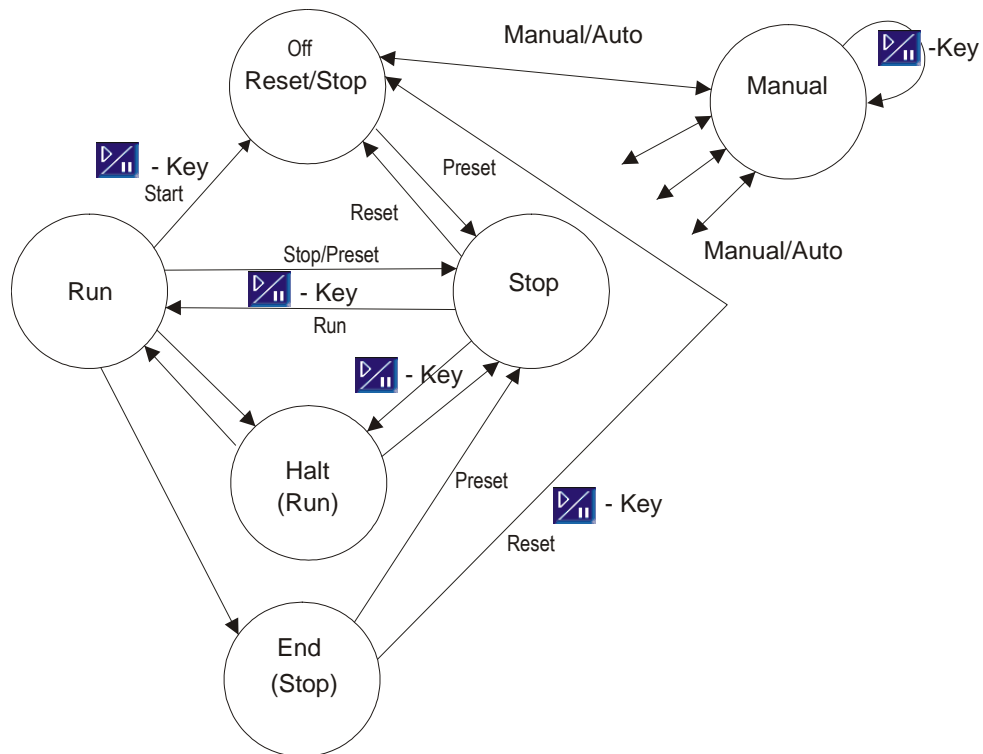


Fig. 19: Interaction between statuses of program and possible switch overs

**Programmer control: General**


- Recipes may be selected during the reset status.
- The process value is only shown on the detailed pages, if the process value input is soft-wired.
- The setpoint can be adjusted during manual operation (status "man").
- The 3 status displays are (depending on operating status):
  - Status left: man
  - Status middle: Estimated program end time
  - Status right: stop / run / reset / search / error / halt



**NOTE!**

If the inputs (function block inputs) shown in the following table are used by the engineering, this signal cannot be changed on the operating page (front-panel operation).

This concerns inputs **run**, **reset**, **preset** and **search** (as shown in the following table):

| Input fields   | Operation  | Display                                  | FB input                                     |                       |
|--|--|--|--|-----------------------|
| <b>Recipe name</b><br>on main page   | Selecting the required recipes is not possible, if the <b>RecNo</b> input is soft-wired. | Indicates the current recipe name.       | <b>RecNo</b>                                 |                       |
| <br>(Selection key) | <b>reset</b>   | Switching the programmer to segment 0.   | The programmer is switched to segment 0      | <b>reset</b>          |
|  | <b>preset</b>  | Select <b>Preset</b> via the review page | Preset page is activated                     | <b>preset, Preset</b> |
|  | <b>segment search</b>  | Starting a search run in the segment     | Programmer makes a search run in the segment | <b>search</b>         |
|  | <b>program search</b>  | Starting a search run in the program     | Programmer makes a search run in the program | <b>search</b>         |

|   |                 |  |             |
|---|-----------------|--|-------------|
|  key | Program control | Changes of the status display (bottom right) | run / reset |
|---|-----------------|--|-------------|

**Programmer operation**

**Operating structure**

The PROGRAMMER has several operating pages selectable on a main page in the menu of operating pages. If the hide input is soft-wired, no PROGRAMMER operating page is displayed.

As shown in the following diagram, there is a main page for the programmer on which a branch to the detailed pages of the tracks can be made by tapping the relevant key. Changing to the parameter page is possible from the detailed pages of the tracks. The editing page is selectable only, when digital input p\_show is soft-wired and set.

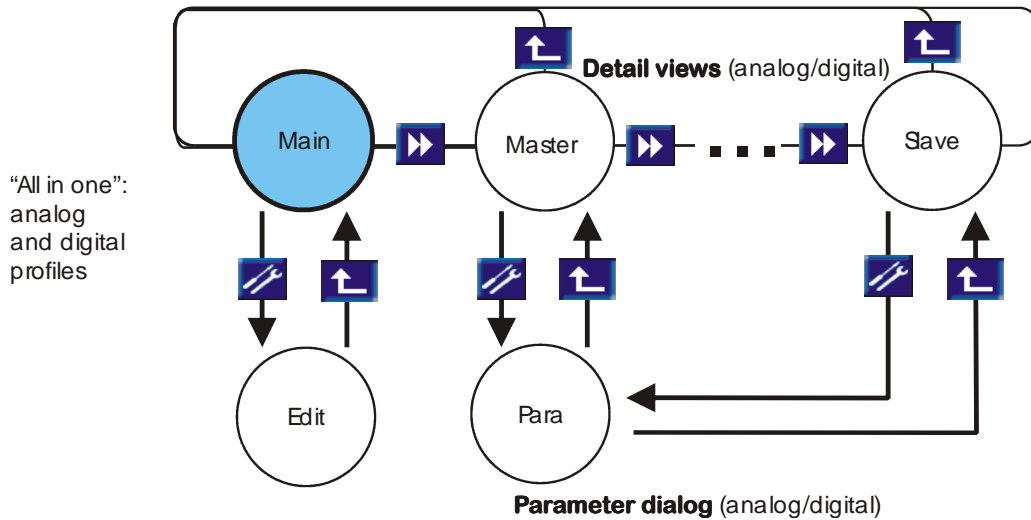


Fig. 20 Operating structure

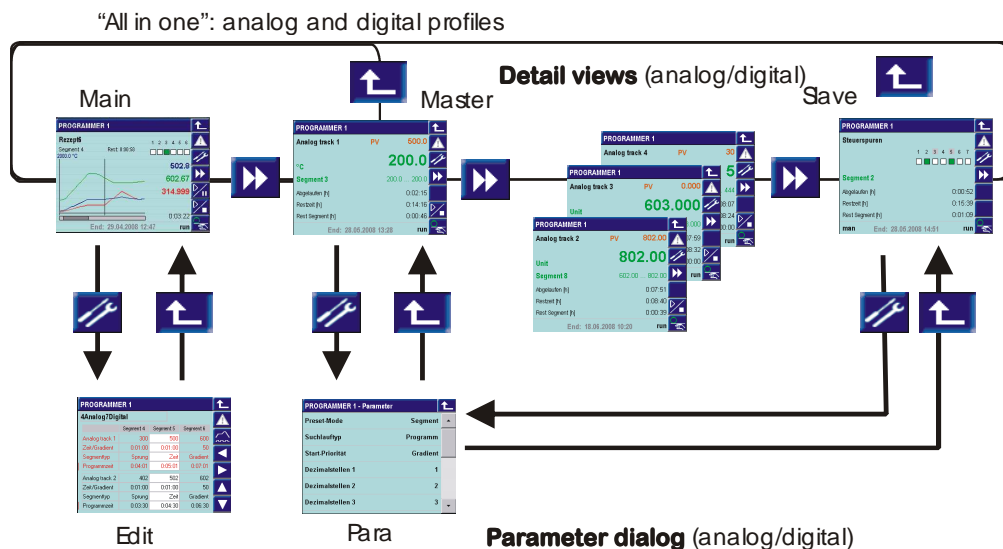


Fig. 21 Operating structure

### Main operating page



Fig. 22

Keys on the right section of the operating page:


Key (line) 1: return


Key 2: Access to the alarm page

Key 3: Access to the parameter page

Key 4: Change to the detail page

Key 5:  Programmer control

Key 6:  Programmer control selection

Key 7:  Auto / manual switch-over

### Display on left section of main page:

0: Programmer name

**1: Active recipe**

2: Active segment

3: Remaining time of active segment

**4: Current status of control tracks 1 to 6 (status)**

**5 - 8: Current setpoint of analog track (up to 4)**

9: Elapsed program time (since start)

10: Maximum setpoint (SPhi1) and unit (Unit1) of master

11: Time stamp in the program curve

12: Setpoint curve in the program

13: Bargraph of program time (bar length to overall length as displayed time section to overall program time)

14: Display of manual operation "man"

15: Estimated program end time

16: Program status: run / stop / reset / search / halt / end / error

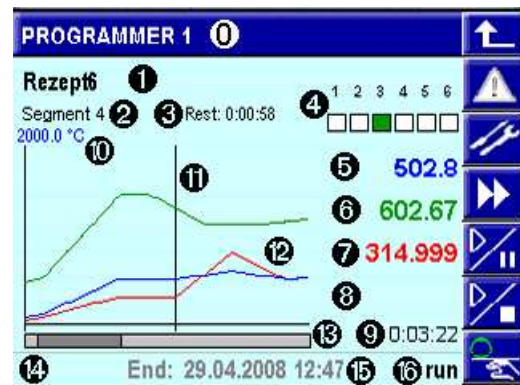


Fig. 23

The items marked in bold letters may include controls with variable values

### Display on the main page

- Up to 4 setpoints with their curves are displayed; a vertical line marks the current time in the program. The visible time range is defined in configuration **TChart**. It is always updated, and the current position with the vertical time stamp remains always in the medium range.
- The Y-axis is determined by scaling of the master track with its maximum and minimum setpoint (**SPhi1**, **SPhi1**). Depending on configuration, the slave tracks are scaled like the master track, or provided with their own scaling. The scaling of individual slave tracks is not shown on the display.
- The setpoints belonging to the current time marked on the trend curve are displayed in the same colour as the setpoint curves on the right. The current program time is shown as a value below the current setpoints.
- The bar of the bargraph indicates the visible time range in relation to the overall program time.
- Items shown with a frame can be changed by tipping with your finger:

- Recipe (only in status `reset`)
  - Setpoints (only during manual mode `man`)
  - Control tracks (only in manual mode `man`)
- An overview of the status of the first 6 control tracks is given by the "LEDs" (on / off, "LED" = Square at beginning of the line). Details and further control tracks are found on a list which can be called up by tipping on the LED field with your finger. All control tracks are listed with status and name.

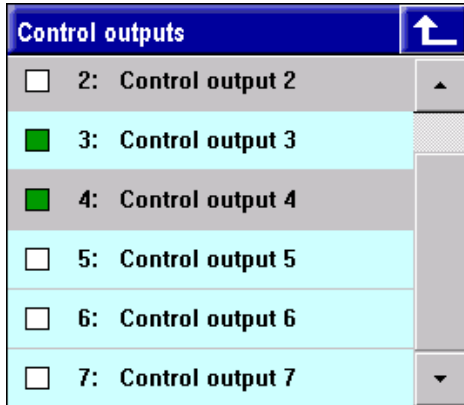



Fig. 24

**Control track operation:**

1. Symbol: white = off, green = on
2. Number of control track (1 ... max. 16)
3. Name of control track. The individual control tracks can be switched over: gray = manual mode, blue = automatic mode, if at least 1 control track is in manual mode.

- Tipping on a setpoint opens a window with display of track name, setpoint and unit.
- Switch-over to manual mode from the main page, marked by display "man" on the status line, puts all analog and digital tracks into manual mode. The variable items are marked in gray colour: setpoints are shown with a gray frame, control tracks are shown with a gray background.
  - Tipping on a setpoint opens a numeric pad for changing the setpoint.
  - Tipping on the LED field opens the list of control tracks. Change the status (off/on) of an individual control track by tipping on the corresponding icon. Tip on the name of the control track to switch it over between automatic and manual operation (gray background).

In status line "man" is displayed as long as at least 1 track is in manual operation. Tipping on  with "man" displayed all tracks are switched to automatic operation.

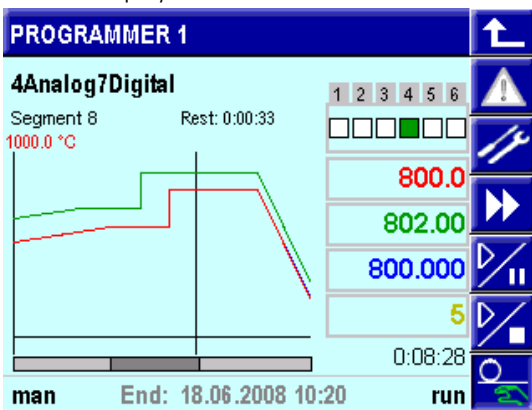





Fig. 25

Tapping  on the main page switches all tracks to manual or to automatic operation in common.

- The program can be set to a preset value. Preset to a segment or to a time in the program is possible. Preset can be activated via the profile overview page ( displays the list: "reset / preset / search", "preset" changes to the overview page). Tap the navigation keys to select the required time or segment. When leaving the page, the preset is or isn't activated ("OK" / "No"). If the preset is activated, the elapsed time is adapted and switch-over to the main operating page occurs (see operating page "Review profiles".)
- Program setting on the operating page: Direct access to the recipe parameter setting (= editing page) is enabled, when control input `p_show = „1"` is set at the programmer function block. Access is using the parameter key .

- The **run / stop / reset / preset** states are always valid for the overall PROGRAMMER, i.e. for all tracks. They can be switched over centrally on the main operating page, unless they are determined via control inputs.

### Detailed pages of the analog tracks

When manual mode is activated on the operating page of a track (key ) , only this track is switched to the manual mode. All other tracks remain in the automatic mode.



#### NOTE!

On the status line of the main operating page, "man" is displayed, if at least 1 analog or digital track is in the manual mode.

The setpoint of a track can be adjusted only, if the track is in manual mode.

A search run can be started in the current segment. Additionally, a program search run can be started on the operating page of the master track (track 1).

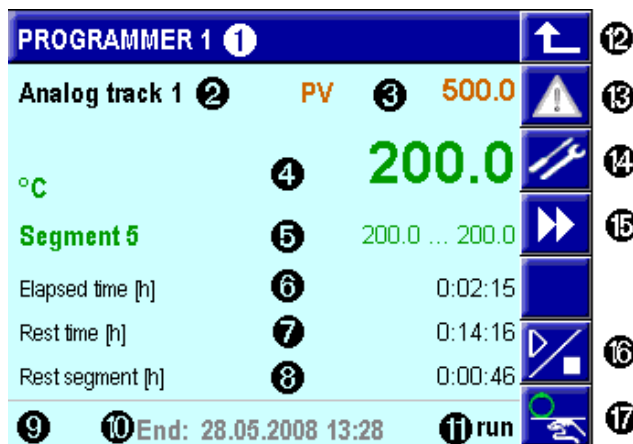



Fig. 26

- 1: PROGRAMMER name
- 2: Track name
- 3: Process value (if any) as a PV with value
- 4: Setpoint unit and **current setpoint**
- 5: Segment name/segment start and end value
- 6: Elapsed program time (master track)
- 7: Remaining program time (master track)
- 8: Remaining segment time (master track)
- 9: Display of manual mode: "man"
- 10: Estimated time for program end (master track)
- 11: Program status: stop, run, reset, search, error, halt (master track)
- 12: Return key
- 13: Key for opening the alarm page
- 14: Key for opening the parameter page
- 15: Track change key
- 16: Search run key (master track: selection dialogue)
- 17:  automatic/manual switch-over key

## Detailed pages of control tracks (digital tracks)

The screenshot shows a control interface for 'PROGRAMMER 1'. It features a 'Control outputs' section with seven LEDs (numbered 1-7), where LEDs 3 and 5 are green and others are white. Below this is 'Segment 5' with timing data: Elapsed time [h] 0:00:52, Rest time [h] 0:15:39, and Rest segment [h] 0:01:09. The status line shows 'man' (manual mode), 'End: 28.05.2008 14:51', and 'run'. A vertical toolbar on the right contains icons for return, alarm, parameter, track change, and auto/manual switch-over.

11: PROGRAMMER name  
 12: Display: Control track operating page  
 13: Display of control tracks:  
 gray marking = control track is in manual mode  
 green = control track on, white = control track off  
 14: segment name  
 15: Elapsed program time  
 16: Remaining program time  
 17: Remaining segment time  
 18: Status line (only display) with  
 19: Estimated time for program end  
 20: Program status: stop, run, reset, search, error  
 21: Return key  
 22: Key for opening the alarm page  
 23: Key for opening the parameter page  
 24: Key for track changing  
 25: auto/manual switch-over key

Fig. 27

- The control tracks are displayed as LEDs.
- The name of the segment in which the master track works instead of the current segment number is displayed.
- The time estimated for program end is displayed on the status line.
- When tipping on the LED field, the control track list is opened. Each defined control track is displayed with number, name and a symbol for the status (off/on).
- Tipping on key , the **automatic/manual** key, also opens the list of control tracks. Tip on the name of the control track to switch it over between automatic and manual operation. On the list, the line is shown with a gray background during manual mode.
- When activating the manual mode on the operating page of control tracks () , only the control track selected on the list is switched to manual operation. All other tracks remain in the automatic mode.
- The status of a track can be changed only when it is in manual mode.
- Manual mode of the control tracks is shown by a gray field around the number of the control track. During manual operation, change the off/on status of the individual control track by tipping on the corresponding icon. Additionally, "man" is displayed on the status line, when at least 1 control track is in the manual mode.

## Programmer parameters

The parameter page can be opened from the detailed pages of the tracks. On these pages, basic functions are listed, for example, the **search run type**, and can be changed. Some parameters are valid for the overall PROGRAMMER (e.g. **Preset-Mode**), others are valid for single tracks (e.g. setpoint ranges).

| PROGRAMMER 1 - Parameter |          | ↑ |
|--------------------------|----------|---|
| Preset mode              | Time     | ▲ |
| Search mode              | Segment  |   |
| Start priority           | Gradient |   |
| Decimals 1               | 1        |   |
| Decimals 2               | 2        |   |
| Decimals 3               | 3        | ▼ |

See table:  
Parameters

Fig. 28

### Editing page

All recipe parameters (analog and digital tracks) are edited on a **single** scrollable page. When opening the page, the active recipe is displayed. Tap on the recipe name to switch over to other recipes.

The upper part always shows the master track in the colour configured with **Color1**. In the lower part, another track is displayed. For track paging, tap keys  and .

The following items can be changed:

- Recipe name (i.e. file selection)
- Setpoint/control track
- Time(s) or gradient
- Bandwidth



#### **WARNING!**

**Changes on the editing page in the active recipe are effective immediately.**

Changes in the program sequence may cause potentially hazardous conditions in the process. For this reason:

- Consider the effects of the changes and take appropriate measures..

When leaving the editing page, a (memory) dialogue is opened. In this dialogue, the operator decides, if the changes are or aren't stored permanently. Changes in the current program sequence are taken into account in the further program sequence and deleted subsequently, unless they are saved.

Changes in another programm are not taken into account and are deleted, unless they are saved.

If the program is reset via the digital **reset** input or via the interface during editing on the editing page while the memory dialogue is still active, all changes are cancelled and the dialog is closed.

|                | Segment 4 | Segment 5 | Segment 6 |
|----------------|-----------|-----------|-----------|
| Analog track 1 | 300       | 500       | 600       |
| Time/Gradient  | 0:01:00   | 0:01:00   | 50        |
| Segment type   | Step      | Time      | Rate      |
| Program time   | 0:04:01   | 0:05:01   | 0:07:01   |
| Analog track 2 | 402       | 502       | 602       |
| Time/Gradient  | 0:01:00   | 0:01:00   | 50        |
| Segment type   | Step      | Time      | Rate      |
| Program time   | 0:03:30   | 0:04:30   | 0:06:30   |

Fig. 29

For description of **segment types**:  
see the Segment table.

#### Bandwidth:

If the separation between setpoint **SP** and process value **PV** exceeds the bandwidth parameter (**BW**), **halt** is activated (for description, the section Bandwidth monitoring).

- 1: PROGRAMMER name
- 2: Recipe (file name)
- 3 to 6: **Master track** (M=Master, always visible):
  - 3: Track name
  - 4: Depending on segment type:  
time/gradient
  - 5: Segment type (cannot be changed via operation)
  - 6: Change field, marked by a line. Switch-over by tipping:
    - a) Elapsed program time or
    - b) Bandwidth: Adjust the symmetric bandwidth per segment on the line.
- 7 to 10: one of the slave tracks (S=slave, 4 lines, paging with up/down keys), lines as 3 to 6:
  - 7: Track name
  - 8: depending on segment type:  
time/gradient
  - 9: Segment type (cannot be changed via operation)
  - 10: Program time/band width change field
- 11: Current segment (white background)
- 12: View and operation of 3 segments, scrolling by the up/down key
- 13: Return key
- 14: Key for opening the alarm page
- 15: Key for opening the overview page
- 16: Key for "Segment paging" right/left
- 17: Key for paging up/down slave tracks

**Digital tracks on the editing page:**

| PROGRAMMER 1     |           |           |           |
|------------------|-----------|-----------|-----------|
| 4Analog7Digital  |           |           |           |
|                  | Segment 4 | Segment 5 | Segment 6 |
| Analog track 1   | 300       | 500       | 600       |
| Time/Gradient    | 0:01:00   | 0:01:00   | 50        |
| Segment type     | Step      | Time      | Rate      |
| Bandwidth        | OFF       | 2         | 3         |
| Control output 1 | off       | on        | off       |
| Delay t-         | 0:00:00   | 0:00:00   | 0:00:00   |
| On time t+       | 0:00:00   | 0:00:00   | 0:00:00   |
| Program time     | 0:04:01   | 0:05:01   | 0:07:01   |

Fig. 30

Same picture as above, but

- 1) **Bandwidth** on the master display
- 2) **Control track** on the slave display.



**Switch-on delay t-** : delay after which the control track goes to its adjusted value.





**Switch-on time t+** : The control track remains on its adjusted value during switch-on time t+.

For a description of parameters, see section Segment time under Control tracks.

**on** = control track is on, **off** = control track is off.

**Programmer page Review profiles**

To provide a program overview, the "Review profiles" programmer page is opened. Tap  to select it on the editing page (tap  to open the editing page in the main menu). On this page, you can check the program, for example, after creation or after changes.

The page shows the curves of the analog tracks with setpoints in a graph. Tap the 4 arrow keys to scroll within the segments ( ) or per segment ( ) throughout the program time. Apart from the program time as a reference, the segments with segment names, the analog tracks with the setpoints and the first 6 digital tracks with states are displayed.

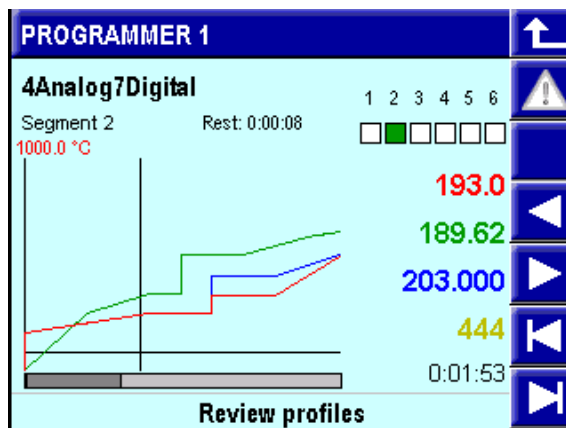




Fig. 31

The overview page is also used to make a preset for jumping to a defined program point. When tapping key  to select **preset** on the main page of the PROGRAMMER, the overview page is opened automatically.

Tip on the arrow keys to navigate to the required position in the program and on the return key to leave the page. With the following prompt if a jump to the position is required, the preset can be activated or cancelled.

Note: paging with   is done by 1 pixel at a time (the time for each pixel is a function of the 80-pixel width for the configured visible time range).

## I-5 Program management: BlueEdit

Any number of recipe files can be created using *BlueEdit*. Only those required and actually needed are downloaded into *KS108 easy*.

The programs must be available for use in the *KS108 easy*. They can be changed during operation and are also saved in this form.



### **DANGER!**

#### **Electronic devices can fail!**

As with any electronic system, a failure may cause damage to the memory.

For this reason:

- It is indispensable to maintain an external archive with back-up copies of data such as recipes and engineering.

To permit replacement of the instrument in case of failure, the recipe files just like the engineerings must be kept in an external archive outside *KS108easy*. For this reason, the recipes must be read out by the programmer and saved in an archive for the eventuality of important program changes in the instrument.

### TIPS:

Clear and meaningful naming of programmer, recipes and tracks facilitates the programmer operation considerably, i.e. it makes sense to spend a reasonable amount of time for this work.



### **NOTE!**

*For additional information on recipe management, please, refer to the BlueEdit operating manual.*

## I-6 Tutorial: A practical example of a "programmer project"

*KS108 easy* is a powerful and flexible multifunction unit. In combination with the *BlueDesign* programming environment and the *BlueEdit* program editor, applications with the most exacting programmers can be realized.

Adapting the instrument to the required application is done using the *BlueDesign* development environment. *BlueDesign* is an easy-to-operate, powerful development environment for control applications. To create an application in *BlueDesign*, select prefabricated components (e.g. controllers or filters) using a graphic editor and link them to each other. You can select the prefabricated components from a "library". Programming knowledge is not required to develop an application with *BlueDesign*.

Now, you can create recipes for the programmer using *BlueEdit*. The *BlueEdit* program editor permits creation, editing and management of recipes for existing engineering. The recipes are adapted to the selected engineering automatically. Access to the recipes in *KS108 easy* is possible at all times. Normally, the track sequences are edited graphically, however, it is also possible to use values for this purpose.

### Content

In this chapter, a simple example will teach you how to create an application with a programmer for *KS 108 easy* using the PMA library, the *BlueDesign* development environment and the *BlueEdit* program editor.



#### NOTE

Information on the fundamentals of working with *BlueDesign* is given in sections "The components of the development environment" and "Working with the development environment" of the *KS108 easy* manual.

This section contains information on the following subjects:

- Using the PMA library in *BlueDesign*
- Application parameter setting by means of *BlueDesign*
- Transfer of the application to *KS 108*
- Recipe creation using *BlueEdit*
- Transfer of the recipe to *KS 108*
- Starting the program

The time expenditure for studying the example is approx. three to four hours.

### Prerequisites

You should have the following knowledge:

- Basic knowledge of the *Microsoft Windows*™ operating system
- Knowledge of the standard programming languages to DIN EN 61131-3 (especially Function Block Diagram and Sequential Function Chart)

### Technical prerequisites

The technical prerequisites are:

- *BlueDesign* version 1.6.1.0 (or higher)
- The *BlueSimulation 108* software must be installed (see Chapter *Installing the software on PC*)
- *BlueEdit* version 1.0 (or higher)

### The components

A short overview of the *BlueDesign* development environment, the PMA library and *BlueEdit*. is given below.

#### **BlueDesign**

The *BlueDesign* development environment supports the overall development cycle of a project:

- **Developing an application:** Applications are composed of application blocks. These application blocks are selected from a library.  
An application can be structured according to various criteria. Moreover, templates can be created, so that frequently used parts of the application can be simply copied for using them again. These reusable blocks are termed macro blocks.

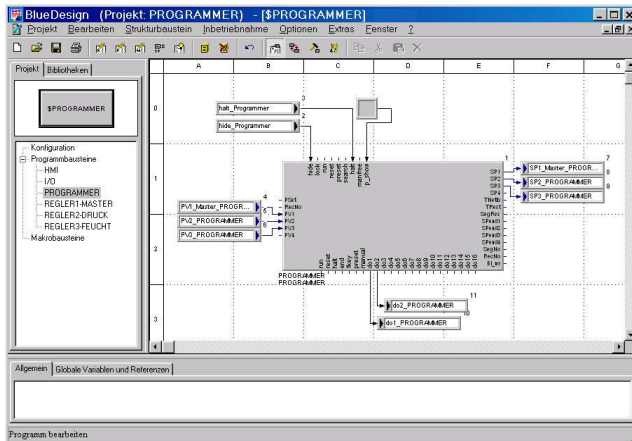


Fig. 32: Programmer project: Example for developing an application

- **Creating an operator interface:** A graphic editor is used to create the operator interface. This editor provides functions to place controls, displays and symbols on the worksheet and to configure them. The appearance of these items is largely identical to the one of the operator interface used on *KS 108* later.

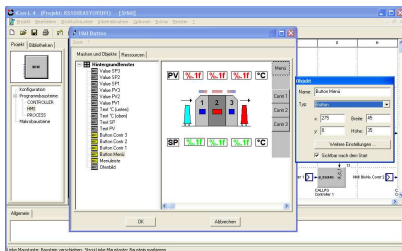


Fig. 33: Programmer project: Example for creation of an operator interface

In our example, we use mainly standard PMA operating pages. These are created automatically by using function blocks with operating pages (programmers, controllers, visualization library).



**NOTE!**

A description of the operator interface is not provided in this example. For this purpose, refer to the tutorial "A practical example" from Chapter II Development environment in the *KS108 easy manual*.

- **Parameter setting:** The out-of-the-box functions of the PMA library are ready for use. To adapt them to your individual requirements, however, they must be configured. This is done using parameters which can be simply entered or selected in *BlueDesign*. For numerous parameters, value lists offering a choice of possible options are displayed. Moreover, the entries are checked. If a value is inadmissible, it is corrected into the next permissible value.

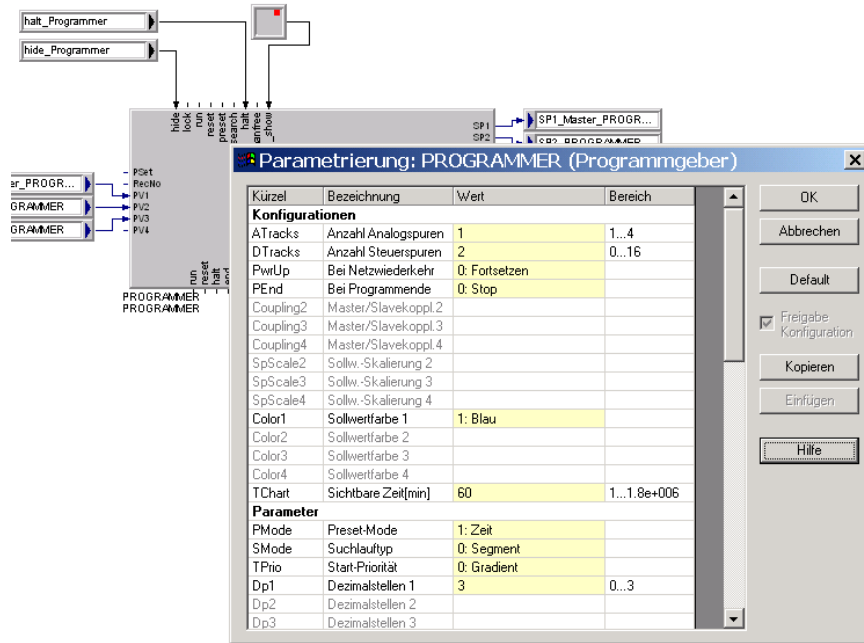


Fig. 34: Programmer project: Parameter dialogue

In the parameter dialogue, the on-line help is offered. It provides detailed information on each function block.

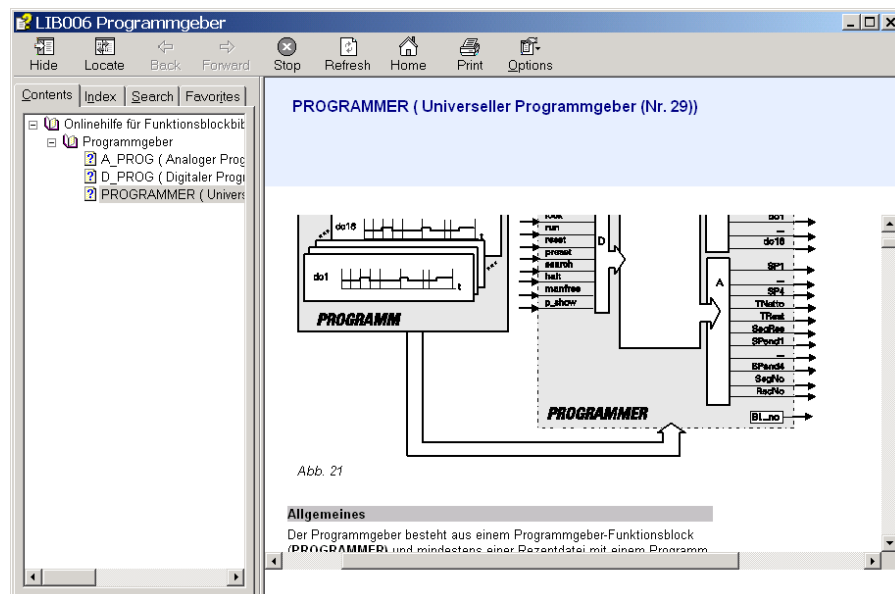


Fig. 35: Programmer project: Parameter dialogue

- **Testing an application:** There are two possibilities to localize application errors: When you position the mouse pointer on a connecting line, the relevant values are displayed (no. 2 in Fig. 36). Alternatively, special blocks from the PMA library can be used.

As trouble shooting in applications is also called "Debugging", these are termed "Debug blocks". You can use these blocks to display information on the program during runtime (1 and 3 in Fig. 36). They are displayed exclusively in the development environment (i.e. not on the instrument).

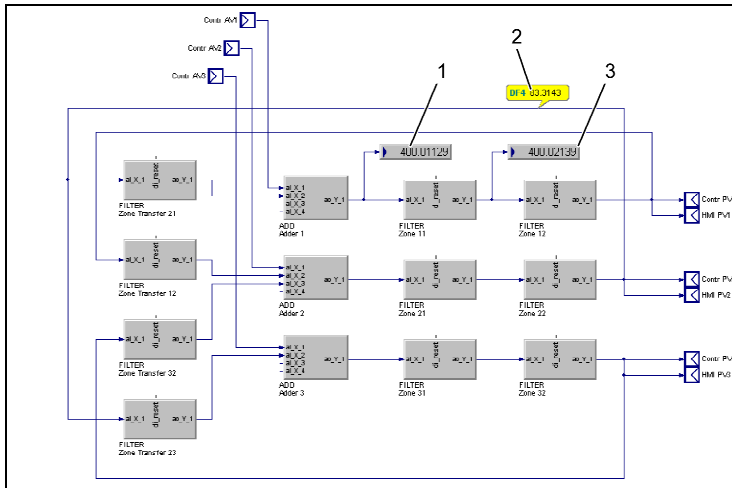


Fig. 36 Programmer project: Debugging example

**PMA library**

The PMA library comprises a large number of function blocks covering all functions normally used for operation of a process. These include among others:

- Mathematic functions
- Logic functions
- Alarm and limit value functions
- Controllers
- Programmers

Moreover, visualization for numerous library functions is made available to you automatically, e.g. for the controller termed "Control" that will be used in our example:

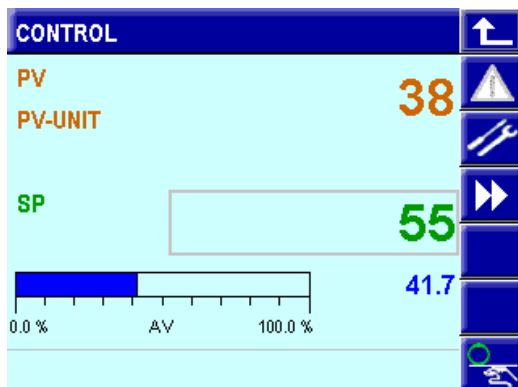


Fig. 37: Programmer project - Control, with examples

or for the universal programmer called PROGRAMMER

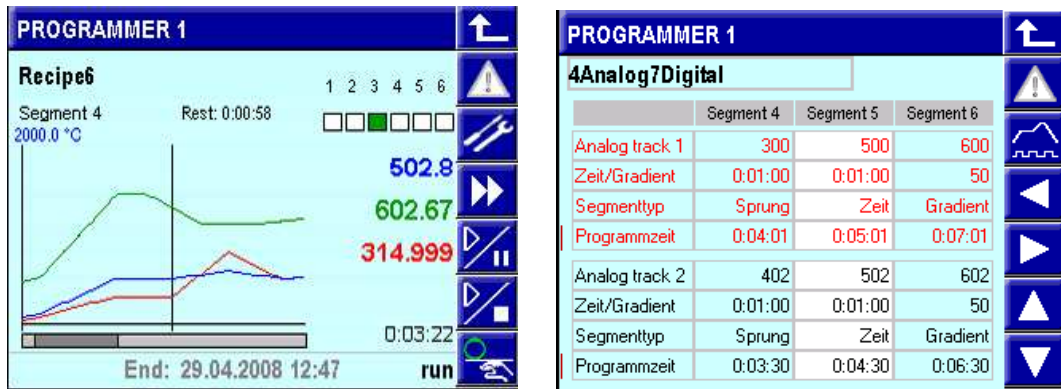


Fig. 38: Programmer project - programmer, with examples

The PMA library facilitates the program development in two respects:

- **Speed:** It speeds up the application design, because applications can be built up almost completely using functions of the library and the user interface is created to a large extent automatically.
- **Quality:** The PMA library offers optimized functions which can be considered as a "black box". Thus potential programming errors are avoided.

### BlueEdit

The *BlueEdit* programmer offers all functions required for recipe creation, operation and management:

- Creation of recipes for the engineering
- Reading recipes from KS108 easy, loading recipes into KS108 easy
- Recipe management

The recipes are displayed graphically for the largest part. For precise information, the track values are displayed simultaneously:

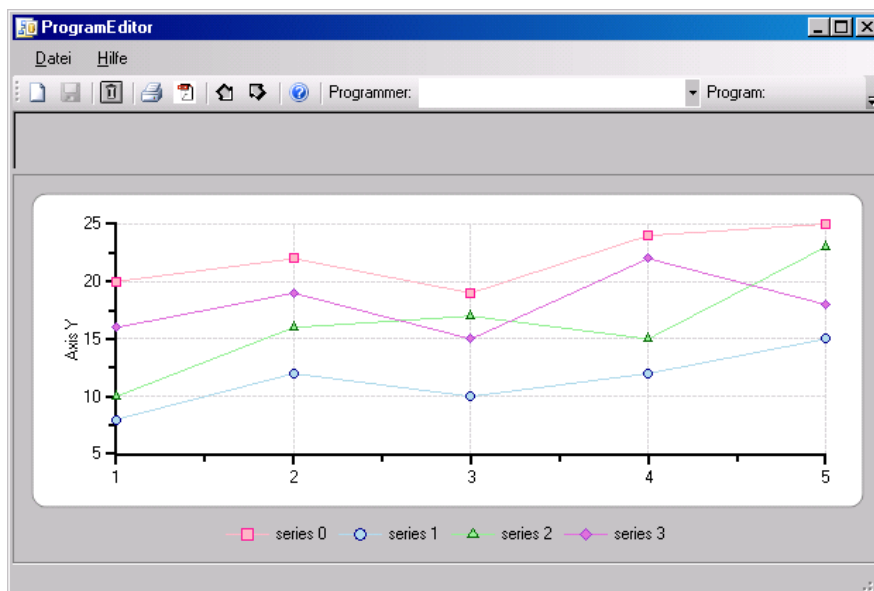


Fig. 39: Programmer project: Program editor

*BlueEdit* offers the following advantages for program design:

- **Speed:** Recipe creation is speeded up, because the graphic display provides at-a-glance information on the curves at all times.
- **Reliability:** Due to the clarity of recipe management in *BlueEdit*, recipes can be found easily. You can set up your own recipe "library".

### The example

The application example to be created must meet the following requirements: It must permit control and simulation of an oven. For checking the heat resistance, a thermal profile is required. The current process temperature must be displayed on the controller operating page. The programmer operating pages must permit the entry of small program changes, e.g. of setpoint and runtime of two program steps. The access to the operation can be limited using codes to be entered at various levels. The main operating page of the application is the programmer:

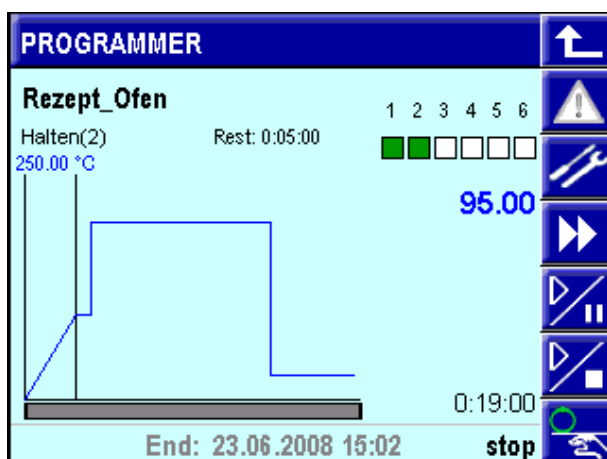


Fig. 40: Programmer project: Oven, main programmer operating page

### Description

The following characteristics for control of the oven are required:

- Process values, setpoint and correcting variable are displayed on the controller operating page.
- The setpoint and duration of the program for checking can be adapted.
- Access to the operation is enabled by 3 access authority levels, otherwise, the access is disabled completely.
- The setpoint may be within 0 and 400.

### Simulation

The behaviour of the oven must be simulated. Simulation is based on the following assumption:

- Setpoint changes are realized by the oven with a delay (2nd order).

## I-6.1 Step 1: Creating a new project

At first, you must create a new project:

1. **Creating a project:** To create a project, proceed as follows:
  - Click menu item "Project/New".
  - Give the project a name (in this example: "PROGRAMMER EXAMPLE").

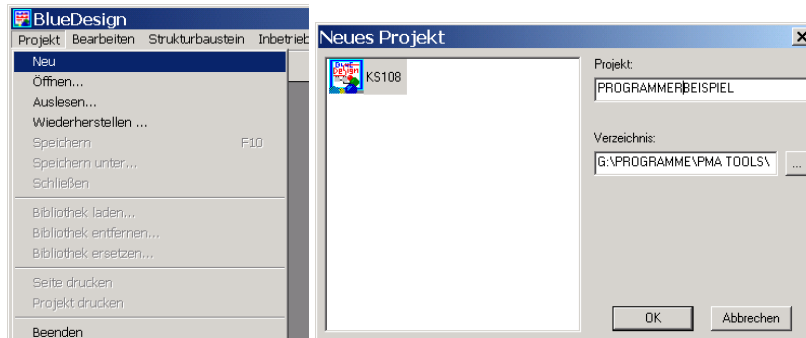


Fig. 41: Programmer project: BlueDesign dialog window "New Project"

2. **De-activating the quick start wizard:** Now *BlueDesign* may start a quick start wizard. In this wizard, you can choose which action you want to carry out next. For the example, we don't need the wizard.
  - To de-activate this option, tick checkbox "Show quickstart next time".
  - Then click button "Program" in the quick start window.



Fig. 42: Programmer project: Quick start wizard

3. **Creating a new program block:** To create a (new) program block, enter the name "PROCESS" into the input field and click button "OK". In the structure tree on the left, there is now a program block with the name "PROCESS".

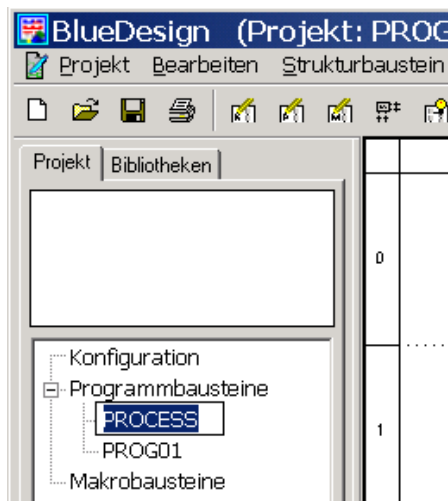


Fig. 43: Programmer project: BlueDesign dialog window "Program block", project overview

4. **Deleting the PROG01 program block:** The automatically generated program block "TASK01" is not required any more. Mark the block using the mouse and select command "Delete" in the context menu.

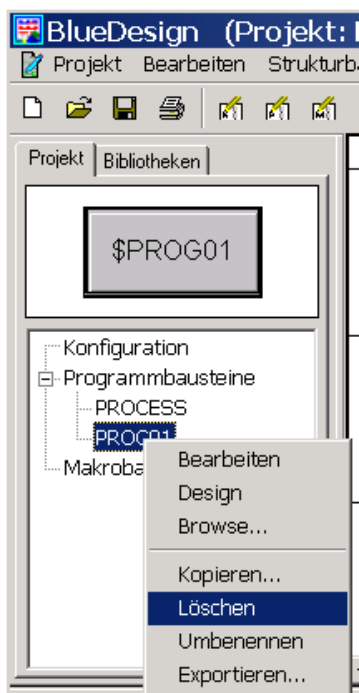


Fig. 44: Programmer project: BlueDesign dialogue "Delete program block"

### Digression: How to build up projects in *BlueDesign*

As shown in the figure below, tab "Project" provides three main categories for subdivision of your project.

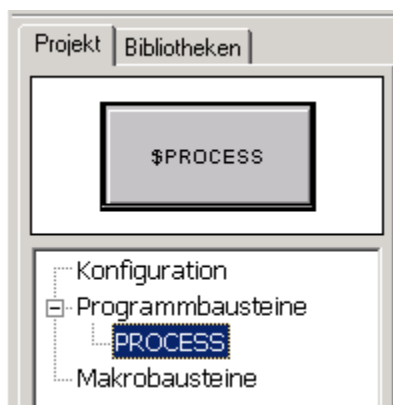


Fig. 45: Programmer project – Project structure in BlueDesign

- **Program blocks:** Program blocks are the basic units making up your project. Program blocks consist of function blocks (for example controllers, filters, totalizers, etc.), links between function blocks as well as inputs and outputs. An example of such a function block is given in Fig. 36. Your application may consist of max. 15 program blocks.
- **Configuration:** In the "Configuration", you can make the settings for interaction of your program blocks. Each program block is executed as a separate "task" and is given its own computing time by the run-time environment. For this, the run-time environment invokes the program blocks cyclically (as shown below). However, determination that (some) program blocks are invoked at longer intervals is also possible via the "cycle time", i.e. they are sometimes skipped.

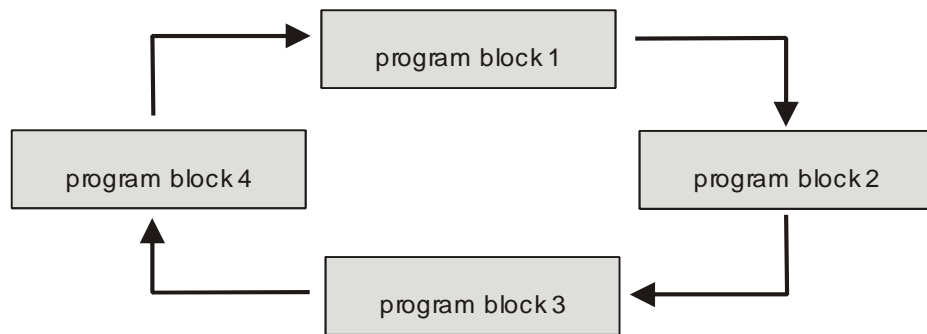


Fig. 46: Program tasks are invoked cyclically

Moreover, the order according to which the program tasks are invoked ("priority") can be determined. These settings can be made during "Configuration".

- **Macro blocks:** When identical combinations of tasks are used in your program blocks, macro blocks can facilitate working. By means of macro blocks, you can create a template that may be used like any other blocks in your projects.

This explanation of the project structuring process may seem abstract and implausible. However, the whole purpose of the project structure will become transparent in the further course of the example.

## I-6.2 Step 2: Creating a controller



### NOTE

*In this chapter, the properties of the components (controllers, filters, etc.) are explained only to the extent required for the example. More detailed information on these components is given in Chapter "Function library".*

The oven must be controlled using a controller. For this purpose, a controller from the PMA library is used. Proceed as follows:

1. **Creating a program block:** Create a new program block and give it the name "CONTROLLER". Proceed as described in "Step 1". The result should look as shown below:

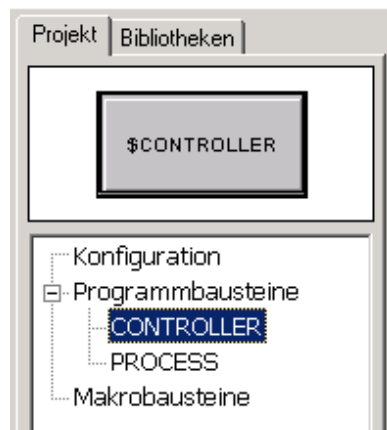


Fig. 47: Programmer project: New program block "CONTROLLER"

2. **Selecting tab "Libraries"** : Now, open tab "Libraries" and select the required block in the next step.

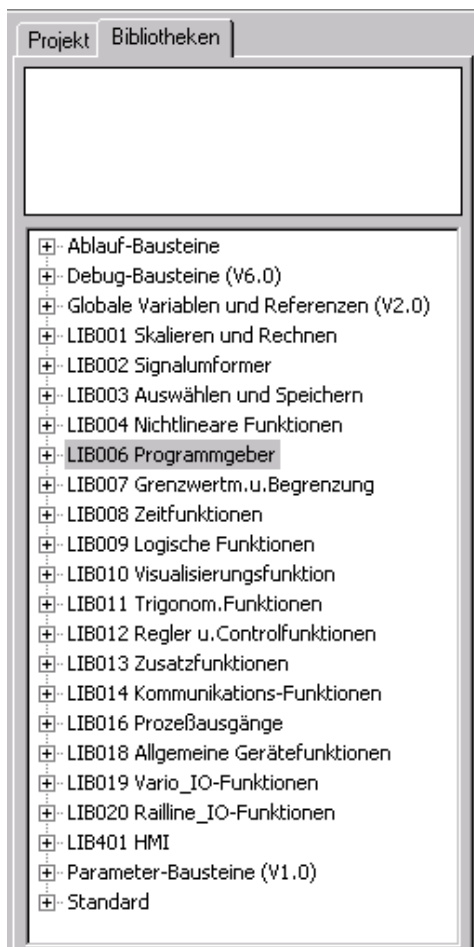


Fig. 48: Programmer project: The Libraries tab

A list of the available libraries is displayed. Libraries with characters "LIB" at the beginning are part of the PMA library. The remaining libraries provide general blocks such as input and output fields, or debug and comment fields.

3. **Selecting a controller**: The controller we need is in library "LIB012 Controllers". Click symbol  to open the library. Now, select controller "CONTROL".



Fig. 49: Programmer project: Selecting controller "CONTROL"

The selected controller is displayed in the upper part of the tab window.

4. **Using the controller:** Click the controller symbol displayed in the upper part of the tab window. Keep the left mouse key pressed down and draw the block right into the worksheet. Then release the left mouse key.

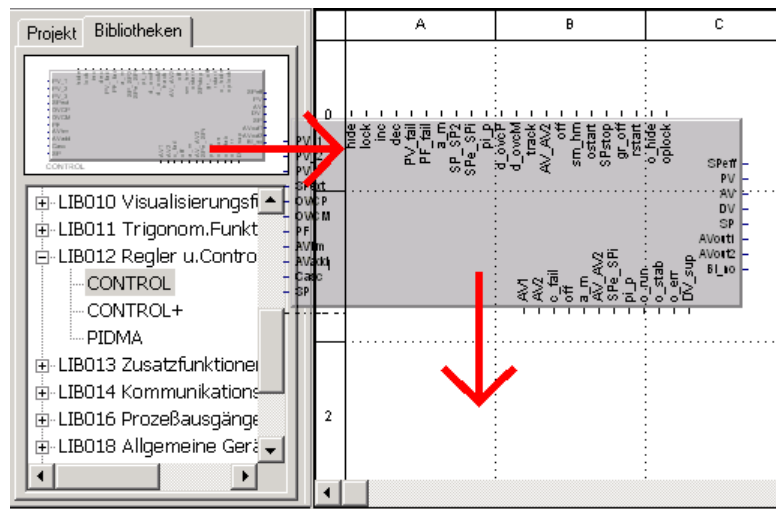


Fig. 50: Programmer project: Using controller "CONTROL"

The result should look as shown below:

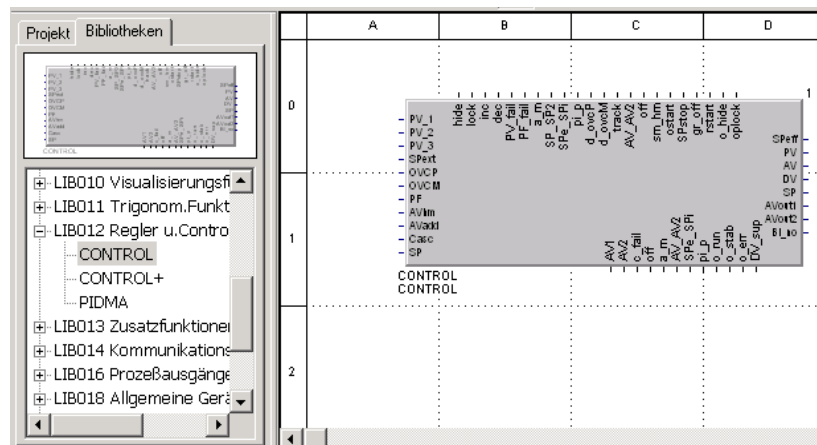


Fig. 51: Programmer project: Using controller "CONTROL"

5. **Defining a name:** Select the controller with the right mouse key. In the context menu, click command "Parameter Dialog ...". Scroll down in this dialogue window, until you find entry "Titel". Enter the name "Control" into column "Value" (shown with a yellow background). For "Unit\_PV", enter the unit "°C" into the column "Value" (shown with a yellow background). Click button "OK" to save your entry.

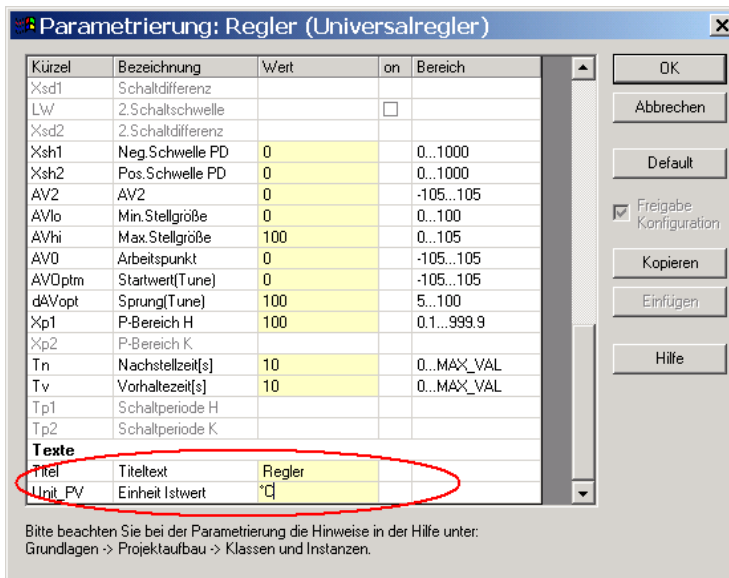


Fig. 52: Programmer project: Parameter setting window

- Determining inputs and outputs:** Now, connect the controller input and the controller output. As the actual application logic of our project must be located in program block "PROCESS", a possibility to transmit an input value or to save the output value is required. For this purpose, we use the blocks "Input" resp. "Output" from library "Standard".

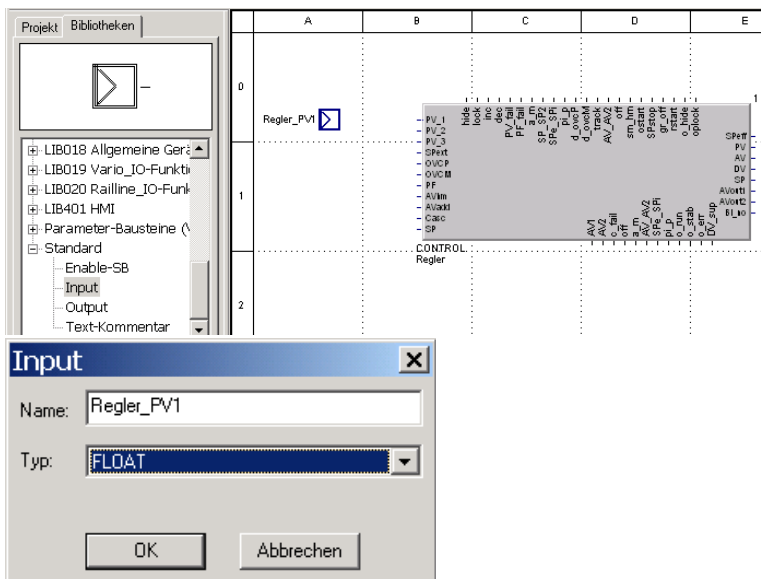


Fig. 53: Programmer project: Using inputs

- Select block "Input" and draw the symbol into the worksheet. Window "Input" opens.
- Assign the name "Controller\_PV1" to the block.
- Select data type "FLOAT". Click button "OK" to save your entry.
- Create an input with the name "Control\_SPExt".
- Then create an output. For this purpose, use the Output block. Give the output the name "Controller\_AVout1" .

- The result should look as shown below:

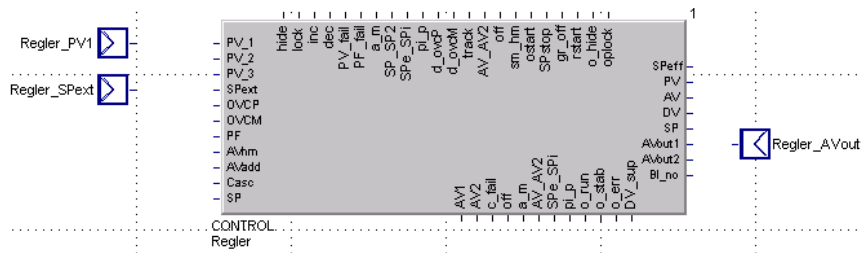


Fig. 54: Programmer project: Creating inputs and outputs



**NOTE!**

The data type "Bit/Bytes" is offered. For the inputs and outputs, however, we need the data type "Float". Always make sure to select the appropriate data type. Otherwise, it is not possible to make the connection between e.g. the input and the block.

**7. Connecting inputs and outputs:** Now, connect the input and output with the controller.

- For this, position the mouse pointer on the connecting element of the input. The mouse pointer changes into a pen (as shown below).
- Click with the left mouse key. The mouse pointer changes into a cross.
- Now, draw the mouse pointer from the input block to the "ai\_PV\_1" input of the controller. Then click on the target point of the connection to create the line.

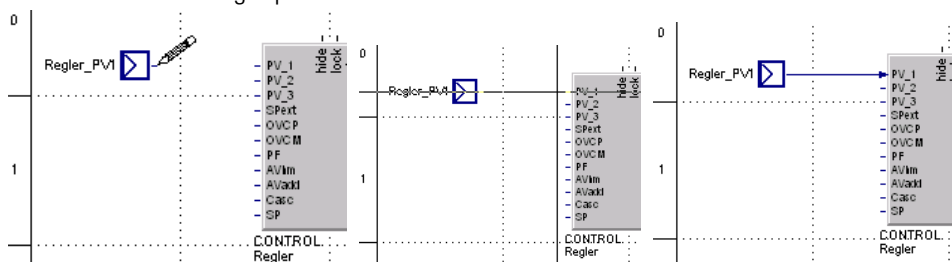
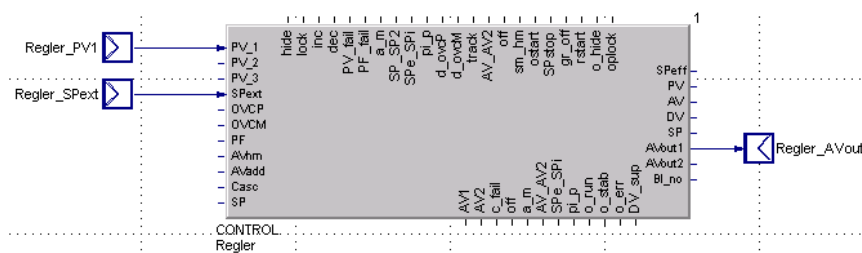


Fig. 55: Programmer project: Drawing a connecting line.

- Connect the controller output "ao\_AVout1" to a block of the "Output" type. The result should look as shown below:



Connecting inputs and outputs

Fig. 56: Programmer project:

**8. Saving the changes:** Save your changes. For this purpose, click button "Save project".

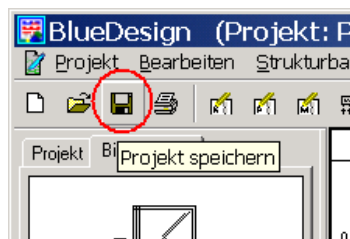


Fig. 57: Programmer project: Save project

## I-6.3 Step 3: Creating a programmer



### NOTE!

In this chapter, the properties of the components used (programmer, etc.) are described only to the extent required for the purpose of the example. Detailed information on these components is given in Chapter "Function library".

We want to make the oven go through a temperature profile. For the purpose of the temperature profile, we use a programmer from the PMA library.

Proceed as follows:

1. **Creating a program block:** Create a new program block with the name "PROGRAMMER". For this, proceed as described in "Step 1". The result should be:



Fig. 58: Programmer project: New program block "PROGRAMMER"

2. **Selecting tab "Libraries":** Go to tab "Libraries" to select the required blocks in the next step. Now you see the available libraries in the lower part of the window.
3. **Selecting the programmer:** The required programmer is in library "LIB006 Programmgeber". Click symbol to open the library. Select the programmer "PROGRAMMER".

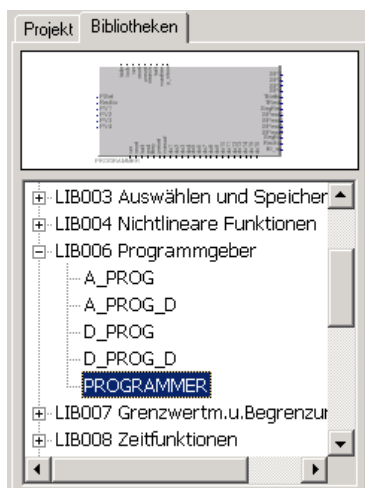


Fig. 59: Programmer project: Selecting programmer "PROGRAMMER"

The selected block is displayed in the upper part of the tab window.

4. **Using the programmer:** Draw the block right onto the worksheet.

The result should look as shown below:

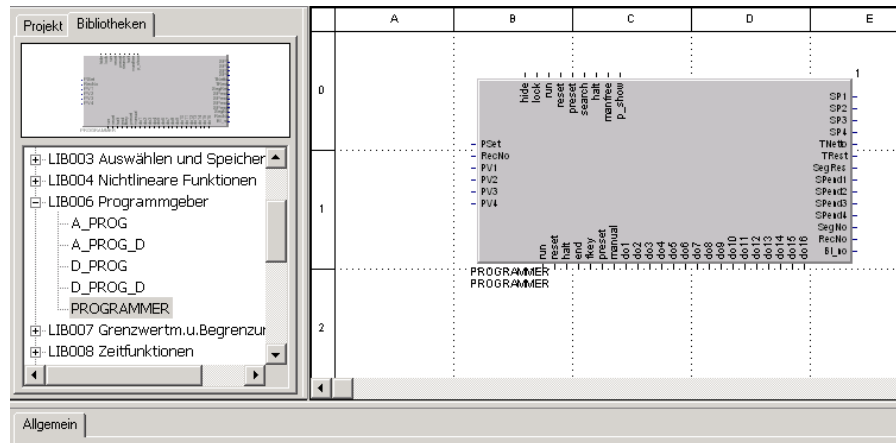


Fig. 60: Programmer project: Working with "PROGRAMMER" blocks

5. **Setting inputs:** For our example, you want to be able to change the program. For this purpose, you need the standard operating page (editing page) of the programmer. To activate the editing page, input "p\_show" must get a positive signal. For this, use function block "Parameter (Bit)" from library "Parameter blocks".
  - Select block "Parameter (Bit)" and draw the symbol onto the worksheet. Window "initial setting" is opened.

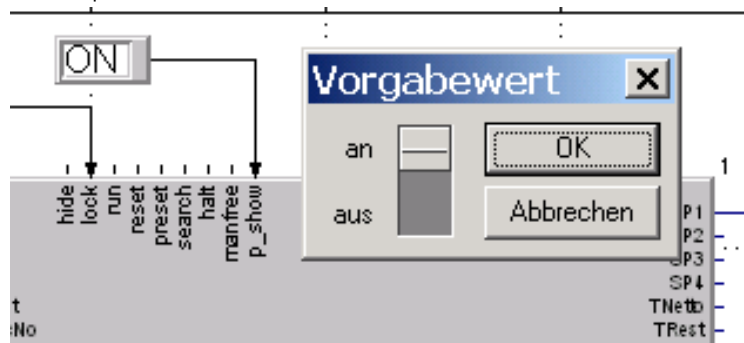


Fig. 61: Programmer project: Using block "Parameter (Bit)"

- Set the switch to position "on". Click "OK" to save your entry.
  - Connect the "Parameter (Bit)" to input "p\_show".
6. **Determining the program inputs and outputs:** Now, connect the programmer inputs and output. This means that you have to make the connections: to our process and to the controller. As with the controller, use blocks "Input" resp."Output" from the "Standard" library for this purpose.
    - Select block "Input" and draw the symbol onto the worksheet. Window "Input" is displayed. Give the block the name "Programmer\_PV1".
    - Select data type "FLOAT". Then click button "OK" to save your entry.
    - Create a FLOAT output using block Output. Give the output the name "Programmer\_SP1".
    - Create two BIT outputs: one for control of a ventilator with the name "do1\_ventilator", and another one for control of the door lock "do2\_door lock".
    - Create another BIT output for the password operating page with the name "Programmer END". We will use it to change to the operating page of the access privilege at the end of the program.



#### NOTE

The offered data type is "Bit/Bytes". However, we need data type "Float" for the PV and SP inputs and outputs. Make sure to select the appropriate data type. Otherwise, it is not possible to connect e.g. the input and the block.

7. **Connecting inputs and outputs:** Now, connect the input and output to the programmer as follows:

- Input "Programmer\_PV1" to input "PV1".
- Output "Programmer\_SP1" to output "SP1".
- Output "do1 ventilator" to output "do1".
- Output "do2 door lock" to output "do2".
- Output "Programmer END" to output "end".

The result should look as shown below:

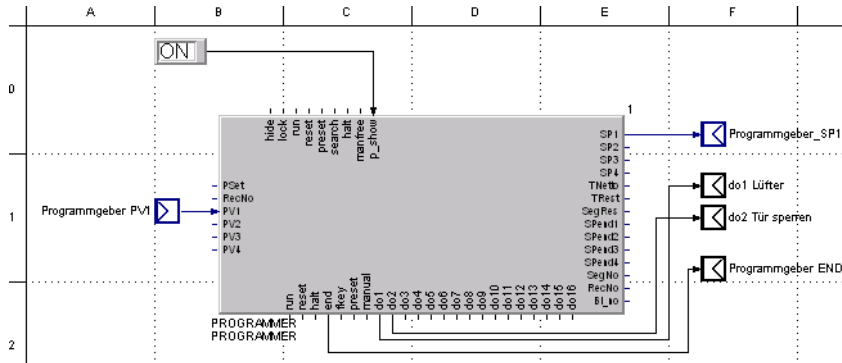


Fig. 62: Programmer project: Creating inputs and outputs

8. **Saving changes:** Click button "Save project" to save your changes.

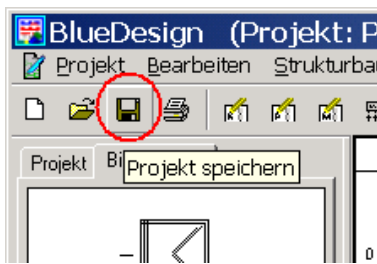


Fig. 63: Programmer project: Save project

## I-6.4 Step 4: Creating the password management



**NOTE**

In this chapter, the properties of the components used (function block PASSWORD, etc.) are described only to the extent required for the purpose of the example. Detailed information on these components is given in Chapter "Function library".

We want the operation of the oven to permit a number of changes and operator interventions for various users via an access management function. For access management, function block PASSWORD from the PMA library must be used.

Proceed as follows:

1. **Creating a program block:** Create a new program block and give it the name "PASS". Proceed as described in "Step 1".
2. **Selecting the "Libraries" tab:** Go to tab "Libraries" to select the required block in the next step. Now you can see the list of the available libraries.
3. **Selecting the access management:** The required PASSWORD function block is found in library "LIB013 Supplementary functions". Click symbol to open the library. Select programmer "PASSWORD" and draw it onto the worksheet.

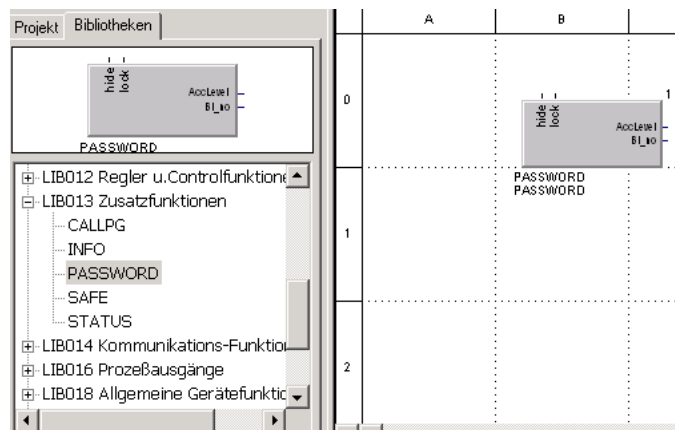


Fig. 64: Programmer project: Selecting "PASSWORD"

#### 4. Setting inputs:

For the event that the programmer arrives at the program end, we want to make the password available. Only with extended access privilege, it is possible to select a recipe and to change the program via the editing page and parameters and configurations become available.

The PASSWORD function block signals the access level via output "AccLevel". When the programmer is in the end position this output is used to display the PASSWORD operating page. For this purpose, function block "CALLPG" is provided.

- Select block "CALLPG" from library "LIB013 Supplementary functions" and draw the symbol onto the worksheet. Connect output "BI\_no" of the PASSWORD with input "BlockNo" of block CALLPG.
- For switching over, select block "Alarm" from library "LIB007 Limits and limit values" and draw the symbol onto the worksheet. Connect output "AccLevel1" of the PASSWORD to input "X\_1" of block ALARM.
- Select block "AND" from library "LIB009 Logical functions" and draw the symbol onto the worksheet. Connect output "alarm" of the ALARM block to input "d\_1" of the AND block. Connect the output "z\_1" of the AND block with input "d\_1" of the CALLPG block.

Now the program should look roughly as shown below:

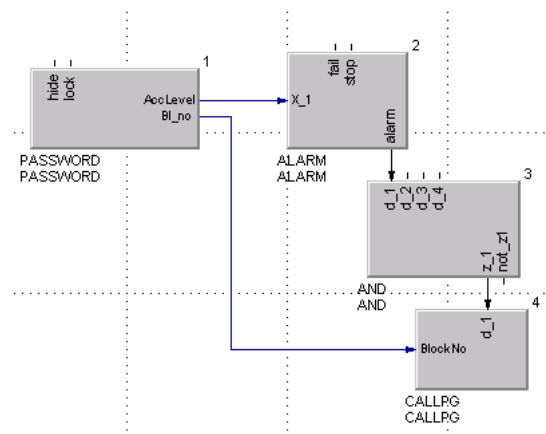


Fig. 65: Programmer project: Purposeful display of an operating page

#### 5. Determining the program inputs and outputs: Now an input must be connected. As with the controller, we use the "Input" block from library "Standard" for this purpose.

- Set block "Input" and assign the name "Programmer END" to the block.
- Select data type "BIT". Then click button "OK" to save your entry.

The result should look as shown below:

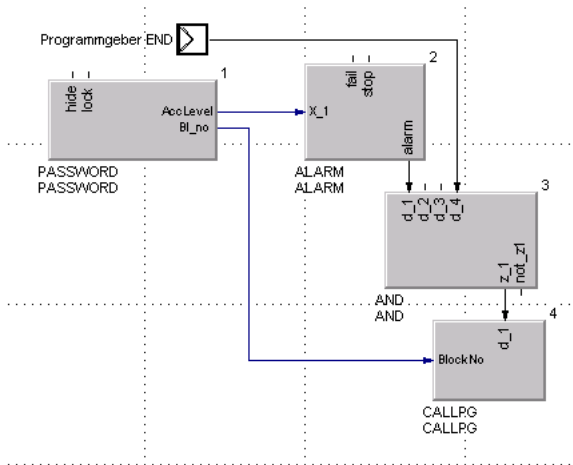


Fig. 66: Programmer project: Creating an input

6. **Saving changes:** Save your changes. For this, click button "Save project".

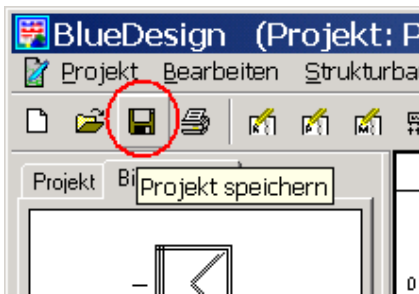


Fig. 67: Programmer project: Save changes

## I-6.5 Step 5: Creating the simulation

As mentioned above, the program example is intended to simulate the behaviour of the oven. The behaviour of the oven is simulated using a filter from the PMA library. Task of the filter is to provide the input variable with a delay (to be defined) as an output variable. The delay will be determined by a parameter (for further information, refer to section "I-6.7 Step 7: Determining parameters").



### NOTE

For additional information on these components, refer to Chapter "Function library".

We want to use 2 control tracks: A ventilator must run dependent on the program and the door must be blocked while the program is running.

For changing the program, operators must have the corresponding access privilege.

1. **Selecting the PROCESS program block:** Open tab "Project" and *double-click* to select program block "PROCESS" .

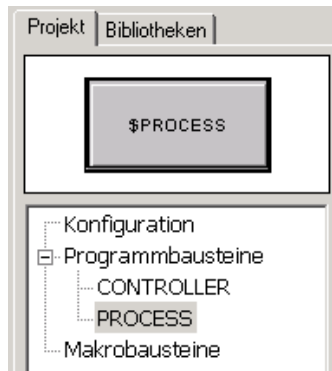


Fig. 68: Programmer project: Selecting program block "PROCESS"

- Adding a filter:** We want to simulate the behaviour of an oven chamber. To simulate an oven chamber, we use two filters connected in series. Task of the filters is to smoothen the (often quite important) changes of the output variable so that the filter output value is similar to the behaviour of an oven chamber. The following graph is intended as an illustration (the upper curve shows the output value, the curve in the middle shows the value after passing the first filter and the bottom curve shows the final value after passing the second filter):

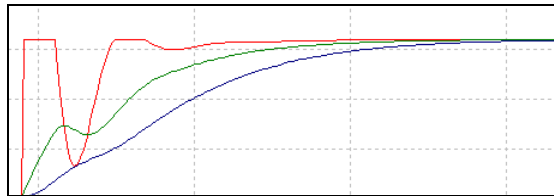


Fig. 69: Programmer project: Filter curves

To add the filters, proceed as follows:

- Go to tab "Libraries" and open library "LIB008 Timing functions".
- Click block "Filter" and draw two filter blocks onto your worksheet (as described above).
- Assign the name "Zone 1" to the first filter and the name "Zone 2" to the second filter. For this, click the filter and select command "Parameter Dialog ..." in the context menu. Enter the name into Column "Titel" of this dialogue.
- Connect the two filters as shown below.

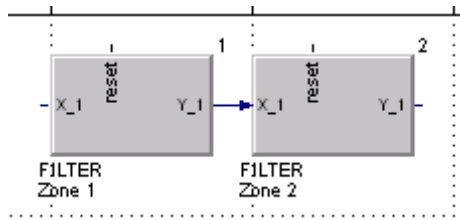


Fig. 70: Programmer project: Two series-connected filters

- Adding a scaling block:** a scaling block must be added to the simulation for the following reasons: Conversion of the current correcting variable provided by the controller into a temperature value is required. The correcting variable is specified in per cent (values between 0 and 100 are possible), the oven temperature is specified in degrees (values between 0 and 400 are possible). For these facts, the correcting variable is multiplied by factor 4 in the scaling block. You will probably wonder where this information comes from, i.e. where the scaling block knows, for example, that the input variable must be multiplied by factor 4. These values are defined as parameters later (also refer to section "Defining parameters").

Add a scaling block to the program, connect it as shown in the following figure and assign a name to it. Note: the scaling block can be found in library "LIB001 Scaling and computing"; the block name is "SCAL".

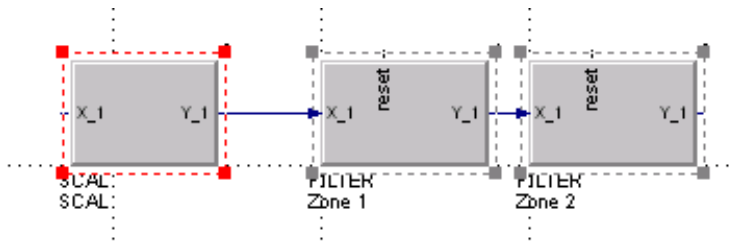


Fig. 71: Programmer project: Scaling block and filters

- 4. Preparing the connection to the controller:** The controller correcting variable must be used as an input variable for simulation. To make the data available, a block of the "INPUT" type is used again. Select a block of the "INPUT" type from library "STANDARD". Assign the name "Controller AVout1" to it and select data type "FLOAT". Connect the block to input "X\_1" of the scaling block.

To provide the data to the controller and to the programmer, an "OUTPUT" type block from the "STANDARD" library must be used. Assign the name Controller PV1 to the block and select the "FLOAT" data type. Connect the block to the "Y\_1" output of the last filter "Zone 2".

The result should be look roughly as shown below:

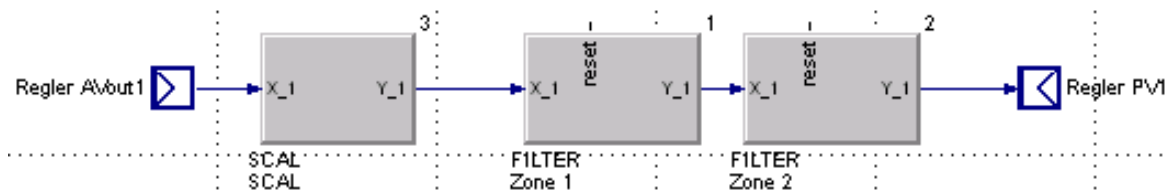


Fig. 72: Programmer project: Program with input



**NOTE**

If an error message "Error! Invalid connection" is displayed when making an attempt to connect two blocks, you have probably selected a wrong data type for an output. In this case, delete the relevant output and create a new one. To delete a block, position the mouse cursor on the block and select command "Delete" in the context menu.

- 5. Additional connection :** In a real project, inputs and outputs are connected via an I/O system. In our example, we simulate the ventilator and the door lock using an LED.

Insert 2 LEDs from library "Debug blocks".

Insert a BIT input "do1 ventilator" and connect it to the first LED.

Insert a BIT input "do2 lock door" and connect it to the second LED.

The result should look roughly as shown below:

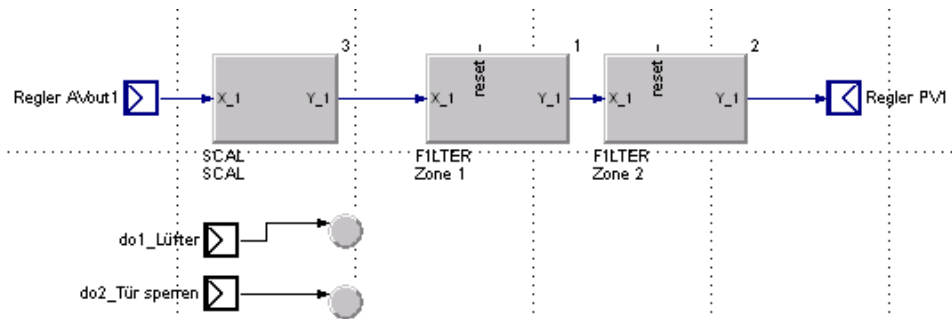


Fig. 73:

Programmer project: Program with input

**NOTE**

If an error message "Error! Invalid connection" is displayed when making an attempt to connect two blocks, you have probably selected a wrong data type for an output. In this case, delete the relevant output and create a new one. To delete a block, position the mouse cursor on the block and select command "Delete" in the context menu.

6. **Saving changes:** Save your changes. For this purpose, click button "Save project".

## I-6.6 Step 6: Determining interfaces, connecting program blocks

So far, we have created the program blocks "CONTROLLER", "PROCESS", "PROGRAMMER" and "PASS". Each of these program blocks has an input and an output interface. However, these inputs and outputs still cannot be accessed from "outside". Moreover, the program blocks in our project are never called up. For this reason, the following steps are required at this point:

1. **Creating the main program:** The "Configuration" is the uppermost level of a project. All program blocks must be provided at this level.
  - Double-click node "Configuration" on tab "Projekt" to display the "Configuration" worksheet.
  - Click node "CONTROLLER".
  - Draw the "CONTROLLER" block onto the configuration worksheet.
  - Proceed identically with program blocks "PROCESS", "PROGRAMMER" and "PASS".

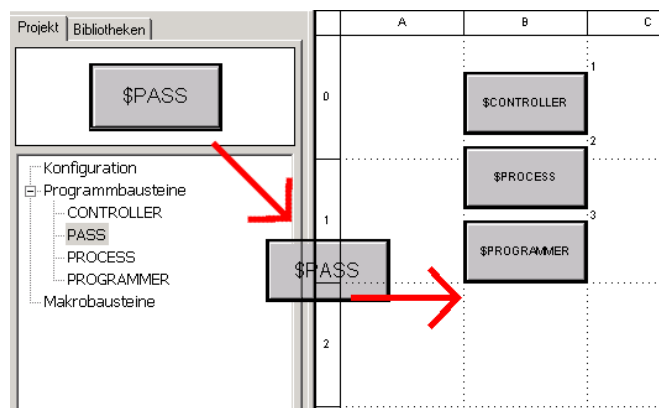


Fig. 74: Programmer project: Creating the "Configuration"

**NOTE!**

Program blocks are executed only if they are found in the "Configuration". Configuration without program blocks doesn't make sense, because the program would lack functionality (a program without "tasks").

The two program blocks don't have inputs and outputs (interfaces). The inputs and outputs must still be assigned to make them visible at the higher level. For this, proceed as follows:

2. **Selecting the "PROCESS" program block:** Double-click program block "PROCESS" in the tree structure to select it.
3. **Executing the Design context menu command:** Click command "Design" in the context menu.

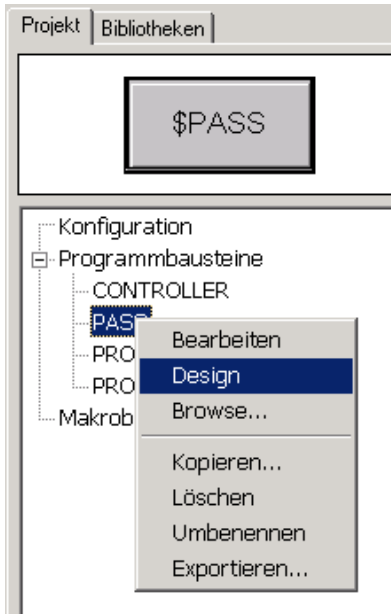


Fig. 75: Programmer project: Selecting the "Design" command

Now a worksheet for input and output assignment is displayed in the right section of the window.

- Assigning inputs:** First, the inputs must be assigned. Position the mouse pointer above input "Controller AVout1" in column "Inputs". A hand symbol appears as a mouse pointer (see below). Click with the left mouse key. Keep the mouse key pressed and draw the input to the left border of the program block symbol. Then release the left mouse key. You may also change the design into the normal PMA view showing the name of the function block at the bottom left and the inputs/outputs with designations inside. Position the digital inputs "do1\_Ventilator" and "do2\_lock door" at the top.

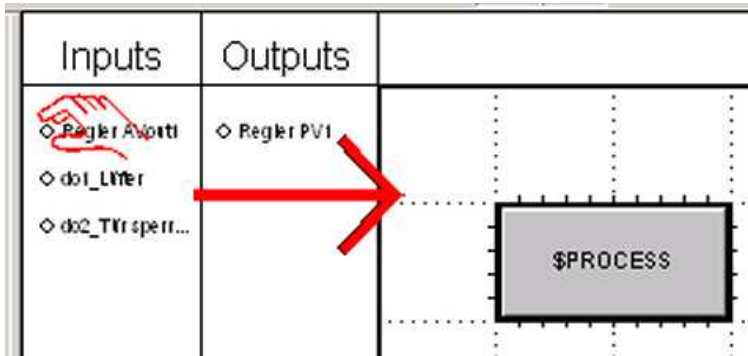


Fig. 76: Programmer project: Assigning inputs

Then position the outputs on the right side of the dialogue window. The result should look as shown below:

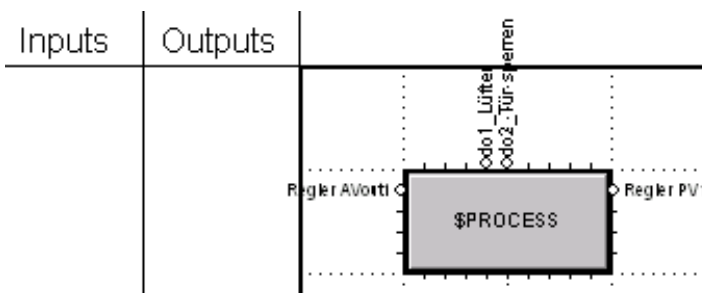


Fig. 77: Programmer project: Inputs assigned

- Adapting the labels:** As standard, the input and output names are displayed outside the program block. However, names can be displayed also inside the blocks to increase the clarity. With the name at the

bottom left, the clarity is improved.

For this, click button "Connector labels inside or outside". For the name, click "Name bottom left, outside".

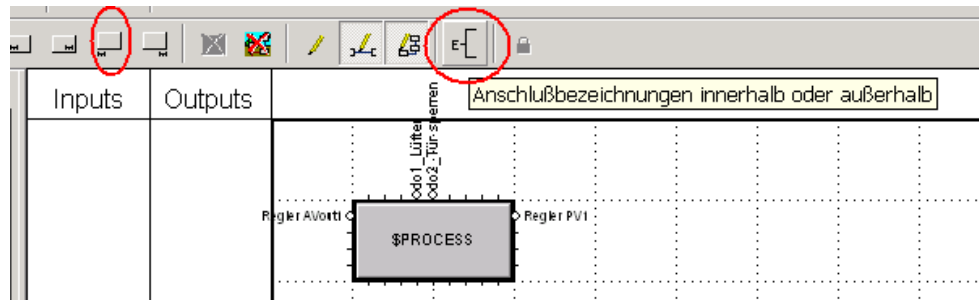


Fig. 78: Programmer project: Adapting the layout

6. **Adapting the block size:** The appearance of the program block symbol is unacceptable. For this reason, position the mouse pointer in the bottom right corner of the block. The mouse pointer changes into a double arrow. Keep the left mouse key pressed down to enlarge the symbol (as shown below).

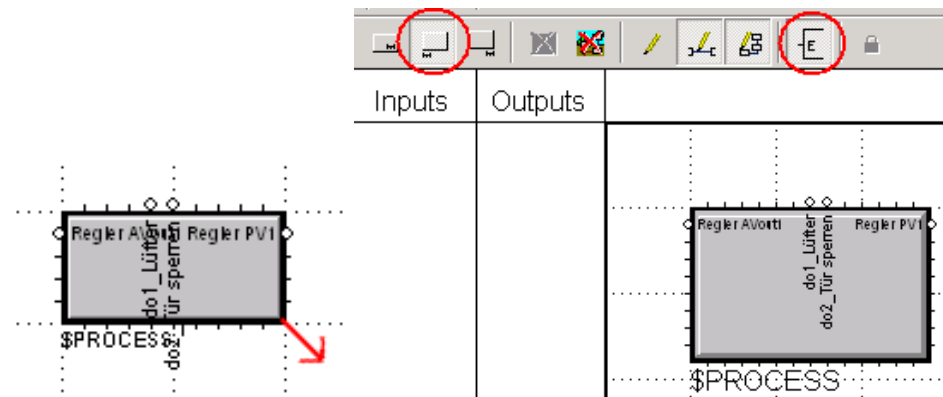


Fig. 79: Programmer project: Adapting the block size

7. **Configuring the "CONTROL" block:** Proceed analogously with the "CONTROL" program block. The appearance of the result should be:

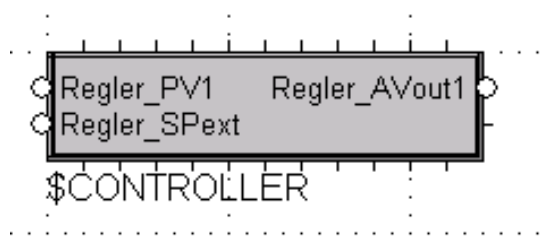


Fig. 80: Programmer project: Configuring program block "CONTROL"

8. **Configuring the "PROGRAMMER" program block:** Proceed analogously with the "PROGRAMMER" program block. The appearance of the result should be as shown below:

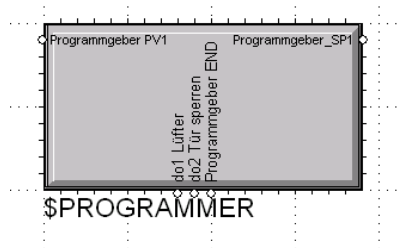


Fig. 81: Programmer project: Configuring program block "PROGRAMMER"

9. **Configuring the "PASS" program block:** Proceed analogously with the "PASS" program block. The appearance of the result should be as shown below:

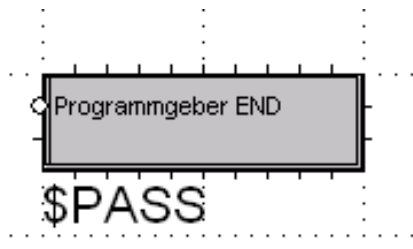


Fig. 82: Programmer project: Configuring program block "PASS"

10. **Connecting program blocks:** So far, the program blocks stand side by side without connections. The controller doesn't know the simulation, and vice versa. The programmer and the access privilege (PASS) must be included as well. For this reason, we have to connect the program blocks in the next step. At first, connect output "Control\_AVout1" of the controller with the input "Control\_AVout1" of the simulation. The "Control\_PV1" simulation output must be connected to the controller input "Control\_PV1" and to the "Controller\_PV1" programmer input.

This is done as follows:

- On tab "Project", double-click on item "Configuration" in the tree structure.
- Connect the two program blocks as shown below.

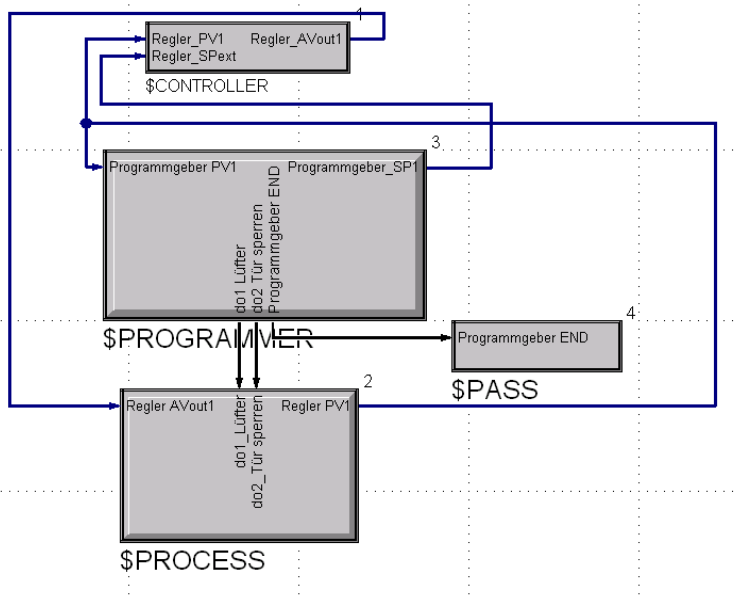


Fig. 83: Programmer project: Connecting program blocks

11. **Saving changes:** Save your changes. For this, click button "Save project".

## I-6.7 Step 7: Determining parameters

Now the PMA library components used for the example must be configured. For configuration, parameters are used to determine the behaviour and properties of the blocks.

### Digression: Parameter types

*BlueDesign* offers two basic methods to work with parameters. One possible method was already explained: The names of blocks or the data type of inputs and outputs were defined by parameters.

This was done in the *BlueDesign* editing mode. In this mode, program blocks, macro blocks and connections are handled and parameters can be entered. However, there is a problem: Multiple program blocks of the same type can be used in a program. Example: For control of two ovens each with three chambers in a project, the result might look as shown below:

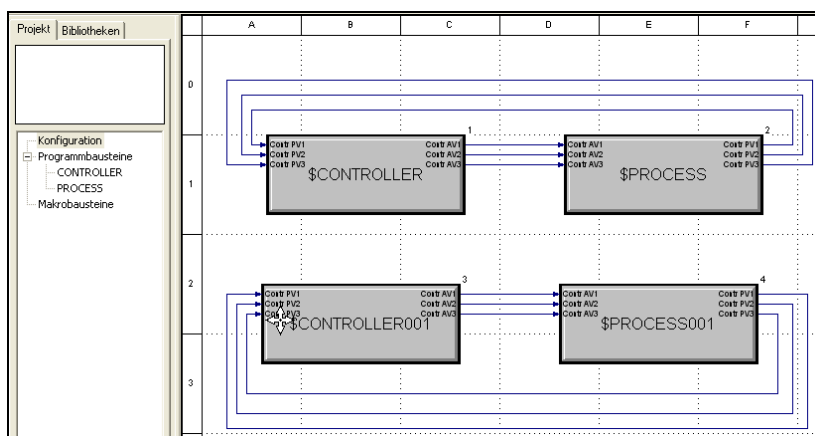


Fig. 84: Programmer project: Multiple use of program blocks

In the example shown above, two program blocks of types "CONTROL" and "PROCESS" are used. Now the maximum oven temperature and various simulation properties, etc. are determined. If we would determine the parameters in the editing mode, the parameters of each copy of the same program block type would be identical. This is not reasonable. After all our two ovens may have different properties (e.g. maximum temperature) or a different behaviour.

*BlueDesign* offers a simple solution to this problem: Apart from parameter editing in the editing mode, the parameters can be edited also in the "Run mode". In this mode, parameters for each copy of a block or program block can be defined individually.

Note on terminology: a template is also termed a "class" and a copy an "Instance".

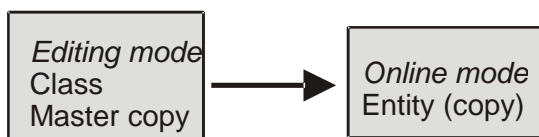



Fig. 85: Editing mode and online mode

Parameters entered during the online mode relate exclusively to the instance, *never* to the class ("template"). This can be compared to the following situation: If you copy a page from a book, the remarks you make on the copy aren't found in the original.

**Summary:**

- **Editing mode:** Parameters entered in the editing mode are valid for all instances ("copies") of a block. These parameters can be overwritten in the Run mode.  
*Purpose:* These parameters should be used only to determine general properties such as data types.
- **Run mode:** Parameters entered during the Run mode are valid only for the relevant instance ("copy") of a block. They overwrite the parameters entered into the corresponding instance ("Copy") during the editing mode.  
*Purpose:* Determination of the (virtual) parameters for a block.

 **NOTE!**  
As a basic rule: Enter the parameters during the Run mode. Use the editing mode for parameter input only, if you are sure that a parameter must be applicable to all instances of a block.

**Entering parameters**

To determine parameters, proceed as follows:

1. **Starting the Run mode:** Start the Run mode using menu command "Run/Enter".

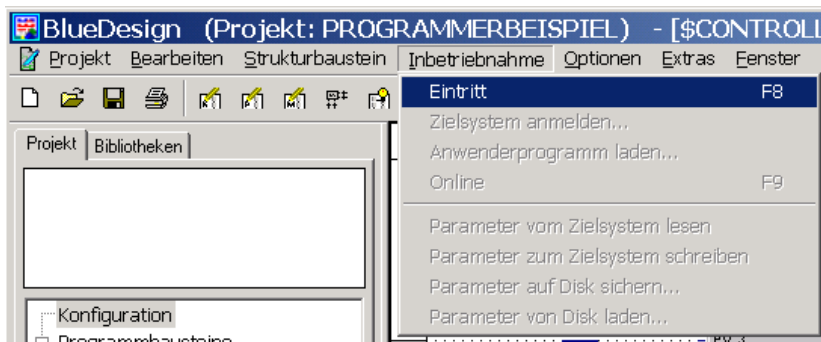


Fig. 86: Programmer project: Starting the Run mode

In the tree structure on the left side, an overview of the project instances is shown. Click item "\$Controller".

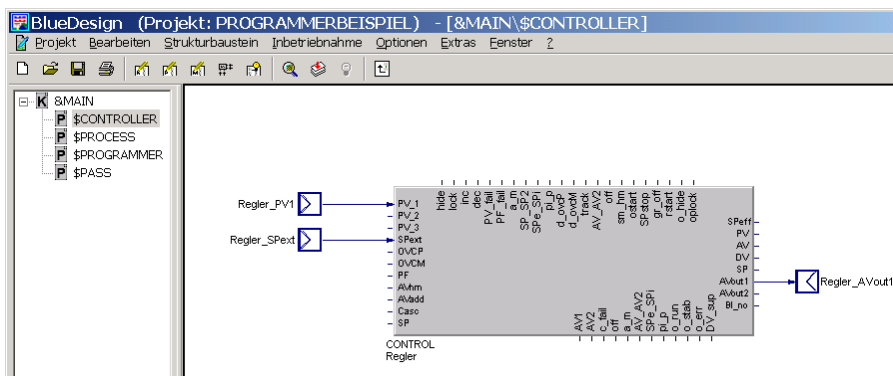


Fig. 87: Programmer project: Run mode

2. **Configuring the "Controller":** Position the mouse pointer on controller "Controller". Select command "Parameter Dialog ..." of the context menu. We want to determine the setpoint using a program, i.e. as an external setpoint. Therefore, assign value "1: Setpoint/cascade" to parameter "SPfunc". The maximum temperature of the oven chamber must be 400 °C, so assign value "400" to parameter "SPhi". As further parameters, we have changed the **title** "Controller" and entered the unit "°C" under **Unit\_PV**. Click button "OK" to save the entry.

**NOTE**

Additional information on the parameter meaning is given in Chapter "Function block reference".

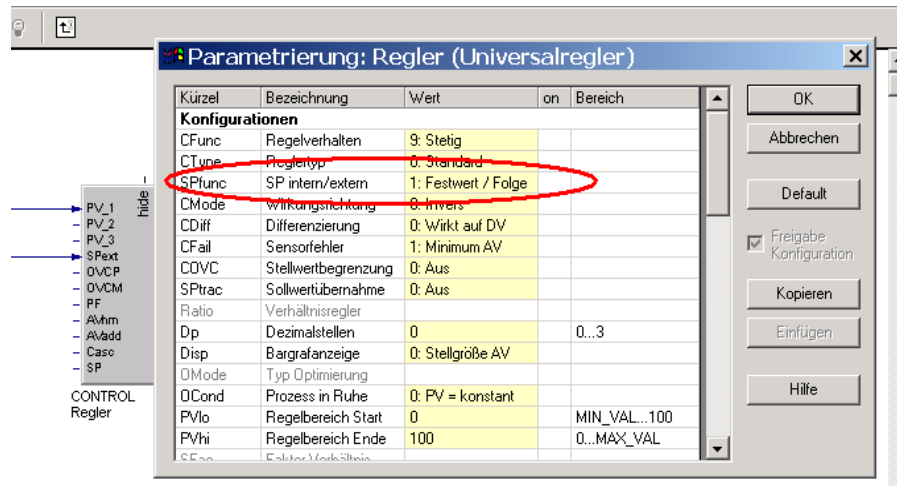


Fig. 88: Programmer project: Configuring the setpoint for the controller

3. **Changing over to program block "PROCESS":** Now, click the program block name in the tree structure to change over to program block "\$PROCESS".
4. **Configuring filters:** Two filters are used to simulate the behaviour of the oven. To simulate a more or less realistic behaviour of the oven, the filter input variable must be applied to the output with a delay of 10 seconds.

For this purpose, assign the value of 10 to the time constant of "T" of both filters.

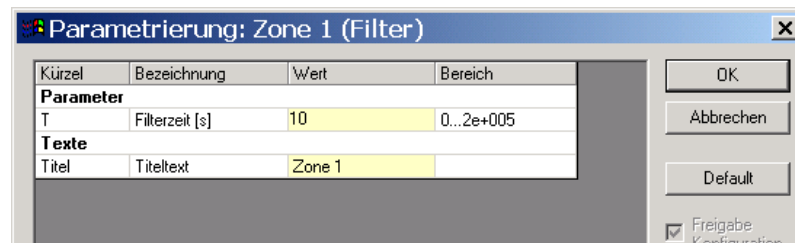


Fig. 89: Programmer project: Configuring the time constant for a filter

5. **"Scaling block:** The scaling block is required to convert the current controller correcting variable into a temperature value. The correcting variable is specified in per cent, the oven temperature is specified in degrees Celsius (maximum 400). To show these conditions, the correcting variable is multiplied by factor 4 in the scaling block.

Enter factor "4" for input "X1" of parameter "A1".

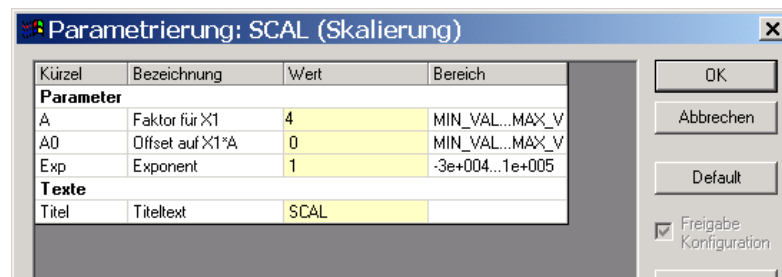


Fig. 90: Programmer project: Configuring a scaling block

6. **Changing over to program block "PROGRAMMER":** Click on the program block name in the tree structure to change over to program block "\$PROGRAMMER".

7. **Configuring a PROGRAMMER:** To determine a temperature profile, the programmer PROGRAMMER is used. Some parameters for the correct function and some additional parameters to increase the operating convenience must be determined:  
Specify the number of the analog tracks as "1" and the number of the control tracks as "2".  
The recipe directory as indicated is left unchanged: "PROGRAMMER"

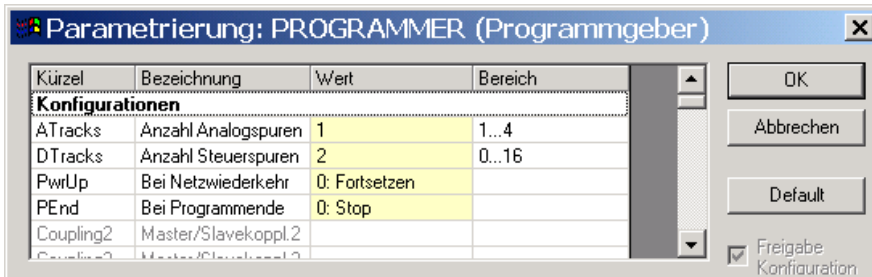


Fig. 91: Programmer project: Configuring a time constant for a filter

Define a name "Temperature" for the analogue tracks and "Ventilator" and "Door lock" for the control tracks.

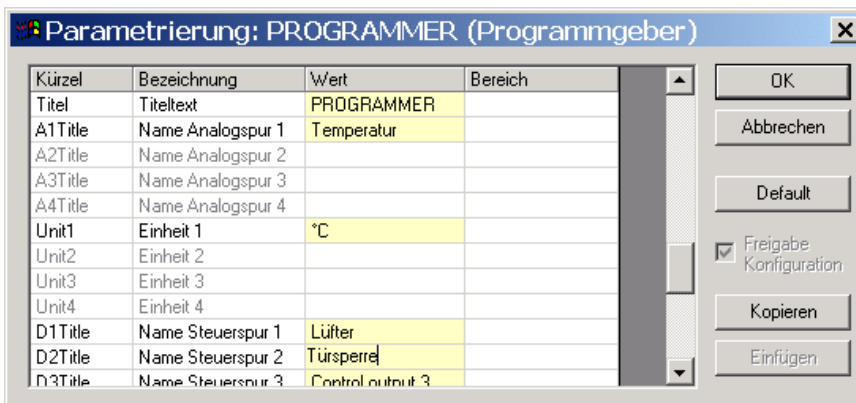


Fig. 92: Programmer project: Configuring a time constant for a filter

8. **Changing over to program block "PASS":** Click the program block name in the tree structure to change to program block "\$PASS".
9. **Configuring the access privilege:** The access to the application must be limited using different privileges. For this, use function block PASSWORD. The password operating page must be opened when exiting the program.  
We use 3 of the 4 possible levels: e.g. operator, supervisor and engineer. Enter these names for levels 1 to 3 and set the passwords, for example, "11", "22" and "33":

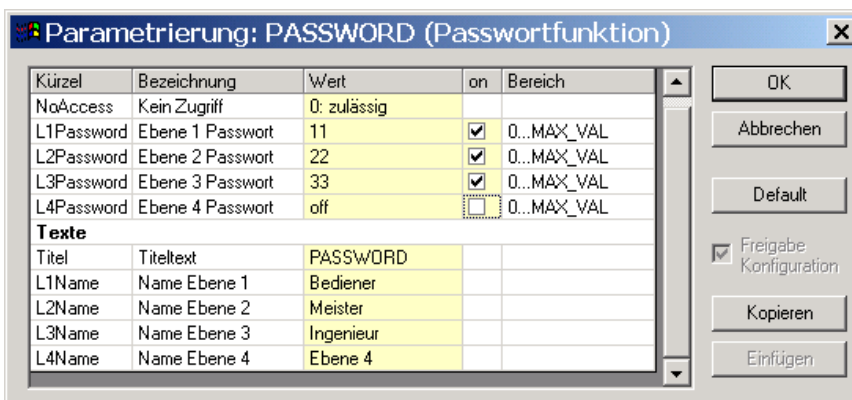


Fig. 93: Programmer project: Configuring the access privilege using function block PASSWORD

The operating page is opened via function blocks CALLPG, ALARM and AND. For this purpose, set the minimum alarm limit to "LimL" = 2. The PASSWORD operating page is invoked only with access level 2 (Supervisor).

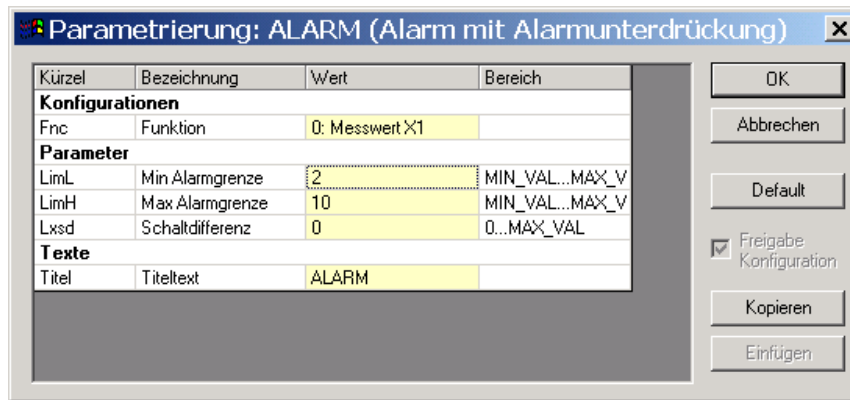


Fig. 94: Programmer project: Configuring a time constant for a filter

10. **Saving changes:** Click button "Save project" to save your changes.

## I-6.8 Step 8: Generate a symbol file

In the previous steps, we have created an application using BlueDesign. For the program, we need a file providing the required data for recipe creation. This file is realized using a simple export function: Select item "Symbol file" in menu "Tools":

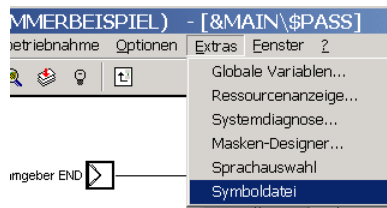


Fig. 95: Programmer project: Configuring a time constant for a filter

A window to specify the storage path for the file is opened. Click save to confirm:

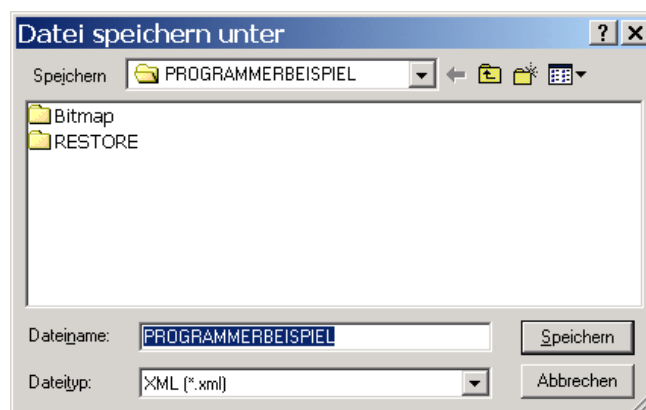


Fig. 96: Programmer project: Configuring a time constant for a filter

Now all preparations in BlueDesign for the application with a programmer were made.

## I-6.9 Step 9: Creating a recipe

In the previous steps, we have created an application for control and simulation of the oven. We have also created the programmer and set up an access privilege. Only the recipe is still lacking.



### NOTE!

Detailed information you find in the manual of BlueEdit 9499-040-91211.

1. **Step** Open the configurator of BlueEdit BlueEdit-Config from PMA Tools.
2. **Step** Create a new project "Tutorial programmer".
3. **Step** Blend in the toolbox <View><Toolbox> and drag an interface (Modbus on TCP/IP) from the toolbox and drop it onto the spreadsheet.
4. **Step** Click on the symbol of the interface (symbol is colored black) and drag a device (KS108 easy) from toolbox and drop it to the spreadsheet.
5. **Step** Clicking on the device brings in a table on the right to fill in the interface settings (IP address).
6. **Step** Drag a programmer from the toolbox and drop it onto the spreadsheet.
7. **Step** Search for the name of the symbol file (\*.xml). Displayed is a list of the available program blocks of the engineering with the comprised function blocks.
8. **Step** Select the programmers (PROGRAMMER; double click on block name) from the XML file and confirm with "Execute". A dialog window opens for controlling the imported parameters of the programmer. Close the window with "Execute".
9. **Step** Save the project and close the BlueEdit configurator.
10. **Step** Open the program editor of BlueEdit from PMA Tools. The just created project is opened automatically.
11. **Step** Create a new recipe (<File><New>) with recipe name and number.
12. **Step** Fill in the settings of the tracks in the displayed table (segment name, segment type, segment time / gradient, segment end value and bandwidth) and store the recipe. For the example see figure below:

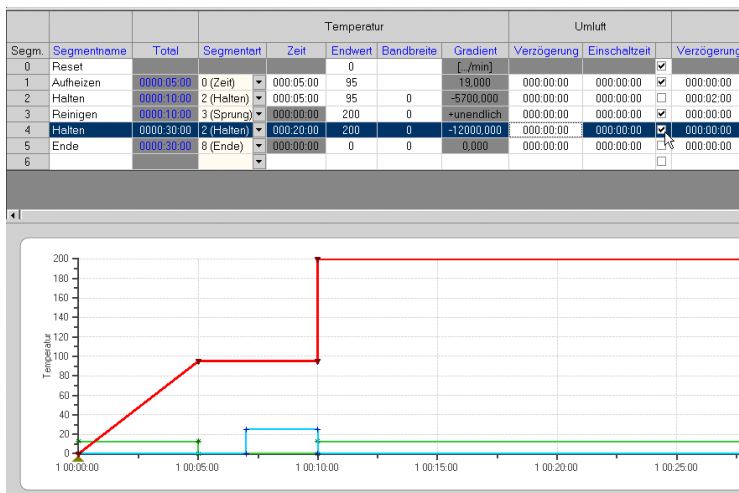


Fig. 97: recipe of example

13. **Step** Send the recipe to KS 108 easy or the simulation. (For the example you send the recipe after loading the engineering into the simulation, see next step.)

## I-6.10 Step 10: An application test

Now it's time for testing the application. For this purpose, the *KS 108* simulation software will be used.

1. **Starting BlueSimulation:** Start the BlueSimulation application.
2. **Starting the Run mode:** Put your BlueDesign project in the Run mode. For this purpose, use the menu command "Run/Enter".

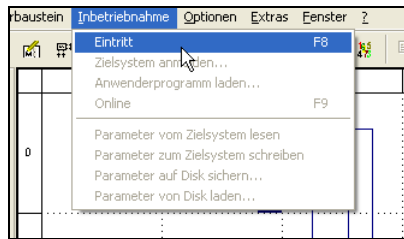


Fig. 98: Programmer project: Starting the Run mode

3. **Starting dialogue "Logon to Target System":** Now you have to transfer the application to the simulating software. For this, use menu command "Run/Download ...". Dialogue window "Logon to Target System" is displayed.

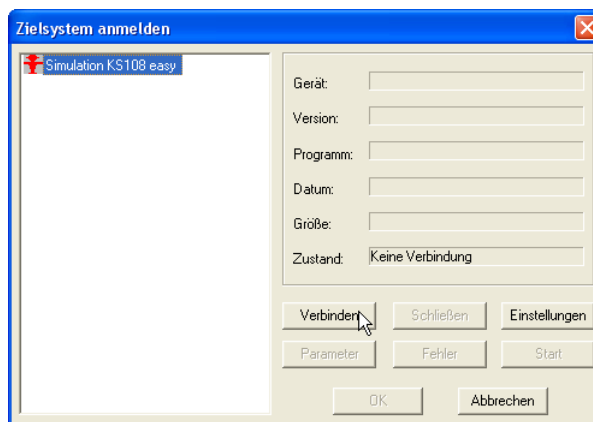


Fig. 99: Programmer project: Dialogue "Logon to Target system"

4. **Connecting to the target system:** If several target systems are displayed in the left part of the dialogue window, select entry "Simulation KS108 easy". Then click button "Connect".  
 Note: Additional settings are not required for working with the simulator.  
 After establishing the connection to the target system, information relating to the application is displayed in the dialogue window (name and size of the application, state, etc.).

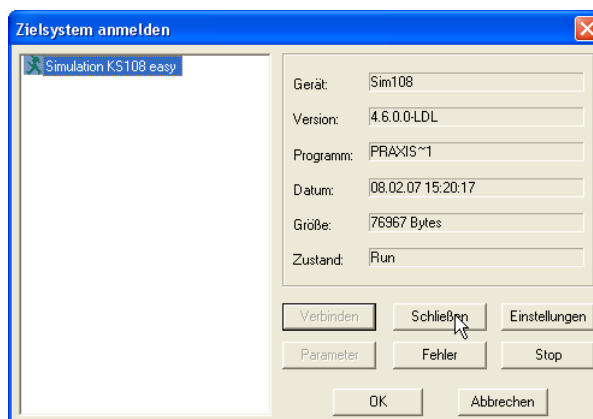


Fig. 100: Programmer project: Dialogue "Target System" (2)

5. **Closing the dialogue:** Click button "OK" to finish the dialogue. The application is transferred. The transmission status messages are displayed on tab "General".

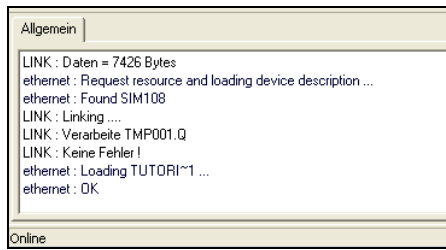


Fig. 101: Programmer project: Status messages

- 6. Testing the application:** Now you can test the application using the "BlueSimulation" software.

# II Index

|                         |                        |                           |
|-------------------------|------------------------|---------------------------|
| <b>B</b>                |                        |                           |
| Bargraph .....          | 33                     |                           |
| <b>C</b>                |                        |                           |
| Control track .....     | 25, 34, 36, 39         |                           |
| Controller .....        | 52, 60, 62, 64, 66     |                           |
| <b>D</b>                |                        |                           |
| Display .....           | 31, 33, 35, 36         |                           |
| <b>F</b>                |                        |                           |
| Filter .....            | 59                     |                           |
| Function .....          | 41, 49, 54, 56, 58, 67 |                           |
| <b>H</b>                |                        |                           |
| HMI .....               | 13                     |                           |
| <b>I</b>                |                        |                           |
| installation .....      | 6, 7                   |                           |
| Installation .....      | 6                      |                           |
| IP address .....        | 70                     |                           |
| <b>L</b>                |                        |                           |
| LINUX .....             | 8                      |                           |
| Logical functions ..... | 57                     |                           |
| <b>M</b>                |                        |                           |
| Macro block .....       | 49                     |                           |
|                         |                        | Main operating page ..... |
|                         |                        | Manual .....              |
|                         |                        | Manual mode .....         |
|                         |                        | <b>P</b>                  |
|                         |                        | Program block .....       |
|                         |                        | Programming .....         |
|                         |                        | <b>R</b>                  |
|                         |                        | Recipe .....              |
|                         |                        | Recipe name .....         |
|                         |                        | Recipes .....             |
|                         |                        | <b>S</b>                  |
|                         |                        | Scaling .....             |
|                         |                        | Segment .....             |
|                         |                        | Segment types .....       |
|                         |                        | Standard .....            |
|                         |                        | <b>T</b>                  |
|                         |                        | Target system .....       |
|                         |                        | <b>V</b>                  |
|                         |                        | Vario .....               |

