Contents

1 Introduction

This document describes an example of an automated industrial installation and is intended to serve as a template for larger projects.
The aim of these training documents is not to describe the PG5 programming tool but to help you to select the right programming language with the associated editors and to structure your project effectively.
The example covered here shows how a transfer station (“pick and place” facility) can be programmed. The station presented here picks up a workpiece, rotates it as necessary, and places it on a rotary table.
The sensors are all digital and the actuators are pneumatically operated (and so can be addressed digitally).
As this type of hardware is not widely available, a second controller is integrated into the example project. This controller acts as a simulator for the hardware. For example, the simulator responds to the input for a cylinder valve with the limit switch signal, after a slight delay. This combination also allows us to simulate cable breaks and to test the behaviour of the system in “exceptional situations”.

1.1 Programming languages

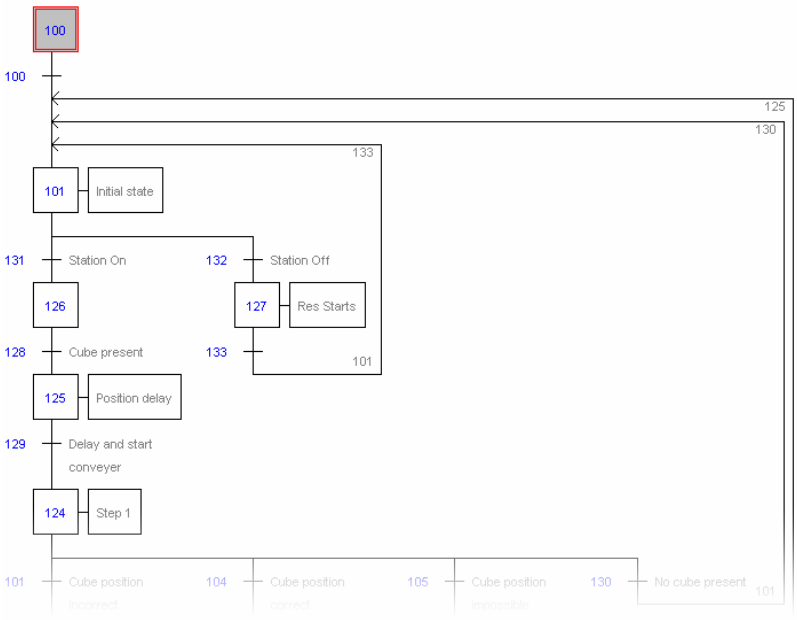
Just as nails are best driven in with a hammer, and screws fixed with an open-ended spanner or a screwdriver, PG5 contains various tools for different tasks.
The choice of the appropriate tool is driven by the tasks set, the form of documentation required, and the strengths and preferences of the programmer.
In this project, we have tried to select the most efficient programming language for each of the various tasks. This means that a relatively large number of functions are implemented in the Instruction List (IL) language.

1.1.1 Instruction list (IL editor)

The IL editor is a very efficient text-based tool, which does however assume some basic knowledge of the individual instructions. For the beginner, the online help and the “Overview of IL instructions” will be important aids in the initial phase. In them, you will find the descriptions of the instructions, their syntax and also some examples.
The efficiency of IL programming is partly due to the fact that it uses the same instruction set that is processed by the controller itself. This means that the exact code produced by the programmer is processed by the controller.

1.1.2 Graftec

GRAFTEC is a graphical tool for easy creation, administration and documentation of sequential process flows. The individual branching conditions (transitions) and actions (steps) can be set up in the IL editor or the FUPLA editor.
This structure allows simple and clear “online monitoring” of the program on the controller.

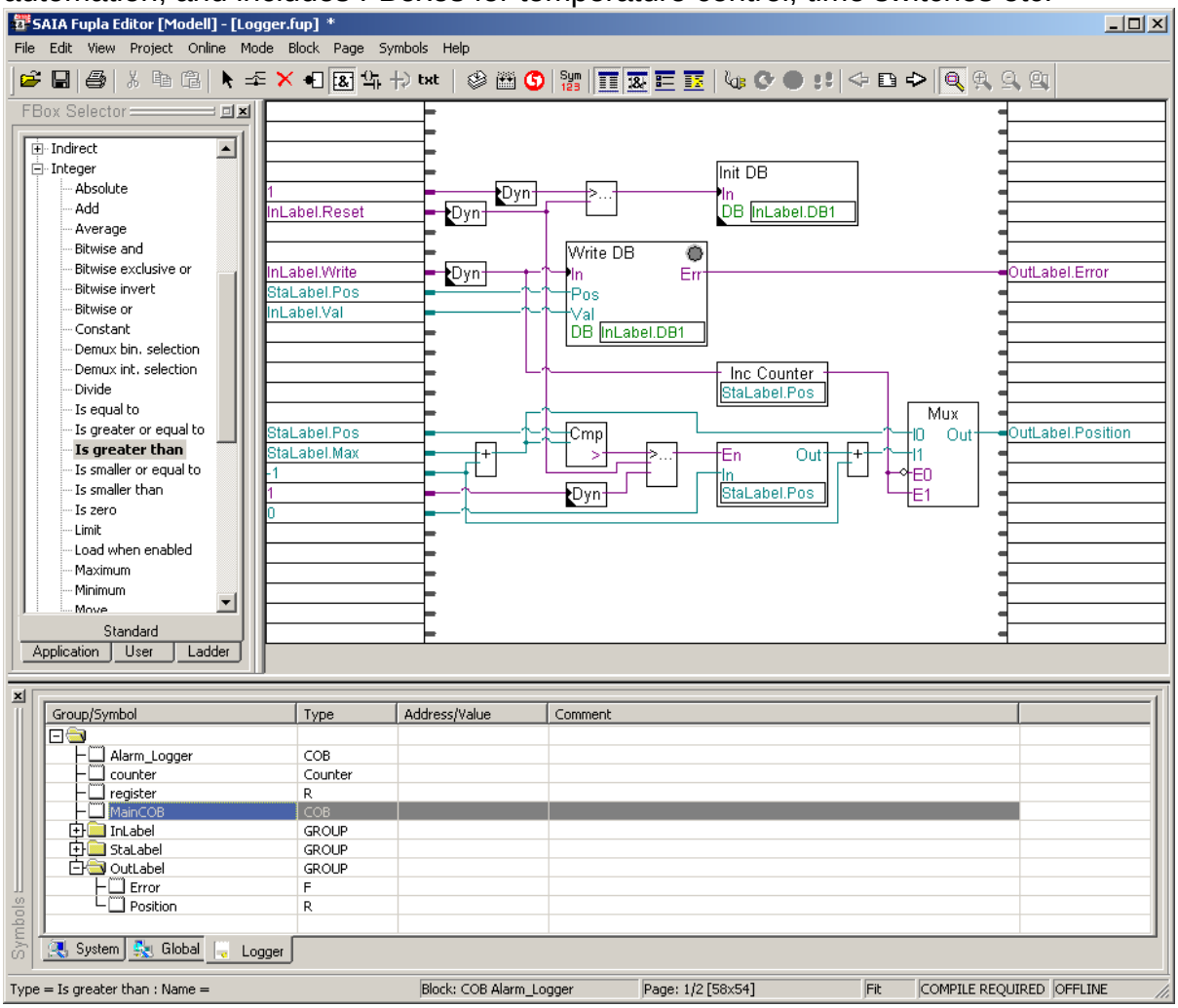


Picture 1: Graftec Screenshot

1.1.3 Function plan (Fupla editor)

Fupla in PG5 is a graphical programming tool. The range of functions extends from simple elements for contact plan programming, binary, integer and floating-point functions, to complex transformation, communication and control functions.

As Fupla programs are programmed graphically with so-called FBoxes (function blocks linked together), the corresponding page of code can be easily interpreted. Another advantage of the FBoxes is that a very complex function can be programmed with very few (possibly even just one) specific FBox(es).
Where required, special libraries can be used, e.g. the HeaVAC (heating-ventilation-air-conditioning) library. This library was developed specifically for building automation, and includes FBoxes for temperature control, time switches etc.



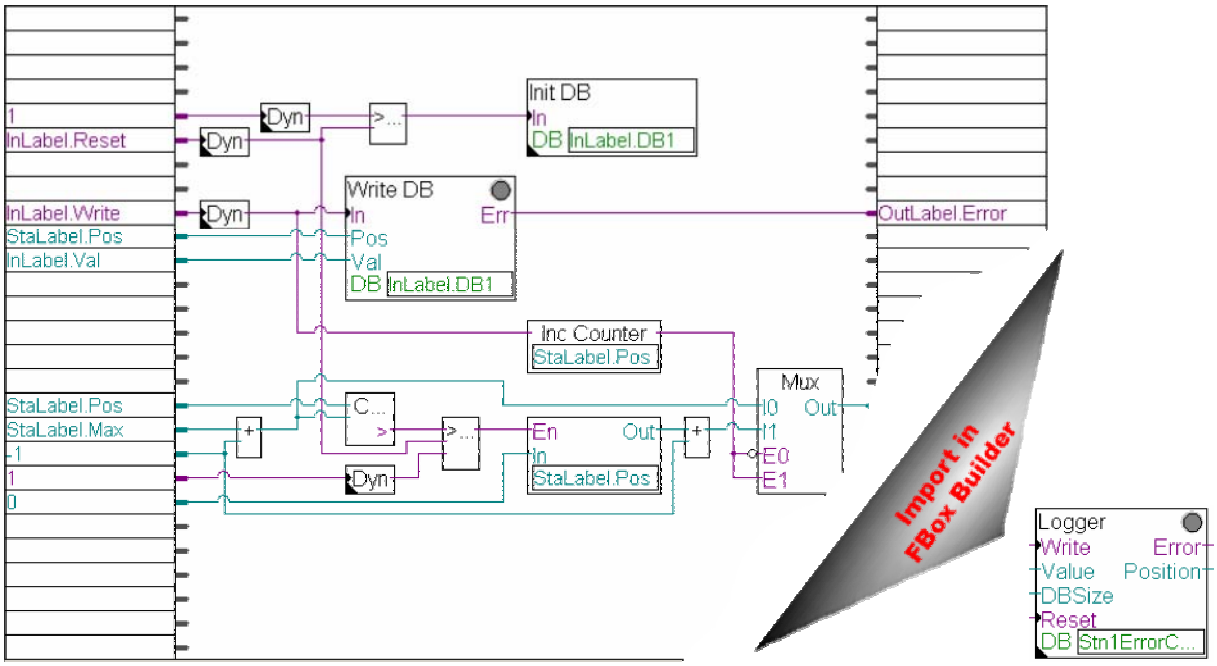
Picture 2: Fupla Screenshot

The files Modem_SMS.fup as well as Logger.fup are program files created with Fupla. The file Modem_SMS.fup is responsible for sending SMS messages while the Logger.fup does organize the alarm buffer.

1.1.4 FBox Builder

Each FBox contains (for the user hidden) IL code that realizes the requested function on the PCD.

Using the FBox Builder it is possible to compress Fupla pages to FBoxes. This allows the easy reuse of once tested program structures and avoids programming errors due to careless mistakes.
The FBoxes used in the Fupla file Logger.fup can be taken as examples for created FBoxes. The FBox with the name “Logger” contains the code shown below. Instead of copying the whole page, just the relevant FBox can be placed in new programs.



Picture 3: FBox created with the FBox Builder

The optional FBox Builder can also be used to integrate self-programmed IL components (functions) into FBoxes, in order to place them graphically within a program. For this usage, an additional license (FBox Builder full version) is to be acquired.

1.2 Structuring

It is very important to structure the programs properly. This enables very complex and unwieldy functions within a controller to be implemented in small steps, which can be modified in a modular way.
The following points are vitally affected by the structure of the programs:

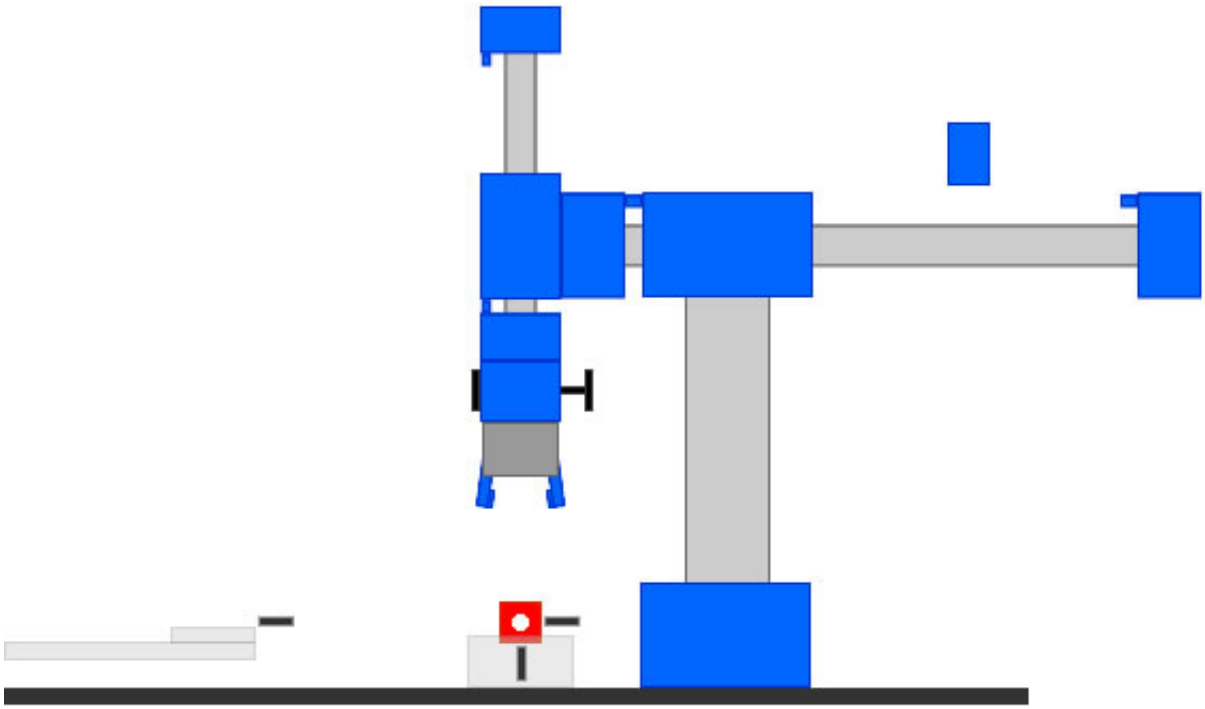
- Breakdown of the installation into small, manageable units;
- Definition of clear interfaces between the individual units;
- Improved readability, clarity and documentation of programs;
- Reduced sources of error;
- Simplified fault identification;
- Dependencies clearly defined and easy to understand;
- Reusable program code (modules).

2 The example project

The aim of this project is to describe the approach to creating an SPS project and to show possible solutions. The progression of an SPS project will be described, from the definition of the task to the commissioning of the installation. The focus is less on the details of programming than on the structuring of the project and the clear presentation of the individual sub-tasks.
In the following sections, we will always refer to the same model, as described below.

2.1 Model description

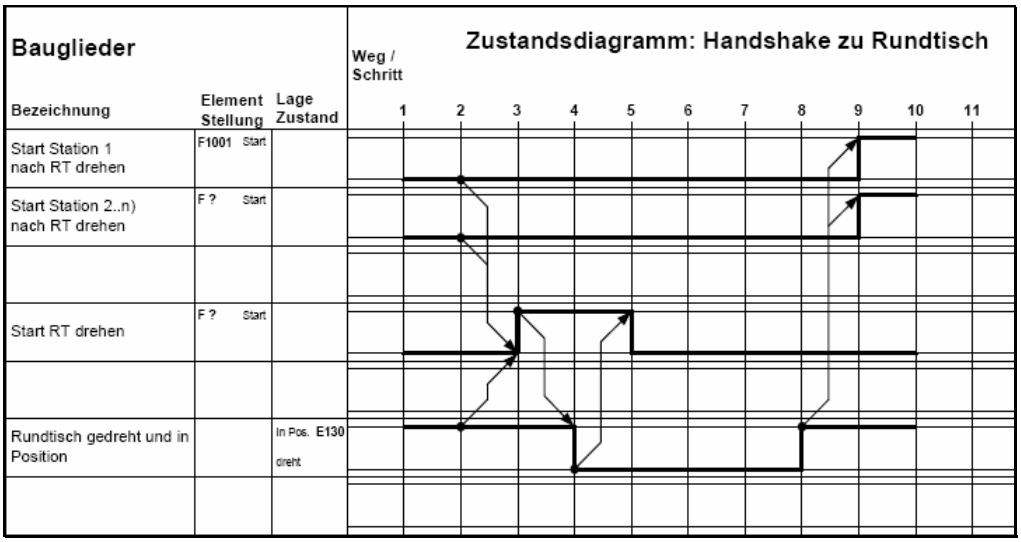
The model is a transfer station, which picks up a workpiece and transfers it to the correct position on a rotary table.
The workpiece is a cube with a hole drilled through the centre. The hole should be aligned so that it can be positioned in the retainer on the rotary table as shown in the drawing below. If the hole is positioned vertically, the cube cannot be rotated into the correct position and the workpiece will be ejected in the centre position of the horizontal cylinder.



Picture 4: View of the robot

2.1.1 Interface to other stations

The interface to the rotary table is achieved with two signals, as per the diagram below.
As the rotary table is also only virtual, the "Turn rotary table" signal is supplied via the web interface to the model station. The relevant button (Start rotary table) is located on the Operations page of the web project on the controller.



Picture 5: Transfer conditions

2.2 Installation of the project

For the installation of this project, Please refer to the accompanying “Handling_Application_Installation” document. This document describes the procedure for installing and commissioning the project.

2.3 Operation

The operating concept is taken from actual practice, and has already been tried and tested on large installations with several stations.

2.3.1 Operating modes

It should be possible to operate the station in the following modes:

- Manual (step-by-step, by keystroke)
- Cyclical (one complete cycle per keystroke)
- Automatic (start automatically when rotary table released).

2.3.2 Manual movement of cylinders

It should also be possible to move all cylinders into their different positions manually, where the mechanism allows it (crash situations should be avoided). The limit switch signals should be monitored, and appropriate error messages output where necessary.

2.3.3 Resetting the station

When a station is reset, all the associated cylinders are moved to their home position in an orderly manner. In other words, mechanical collisions are avoided.

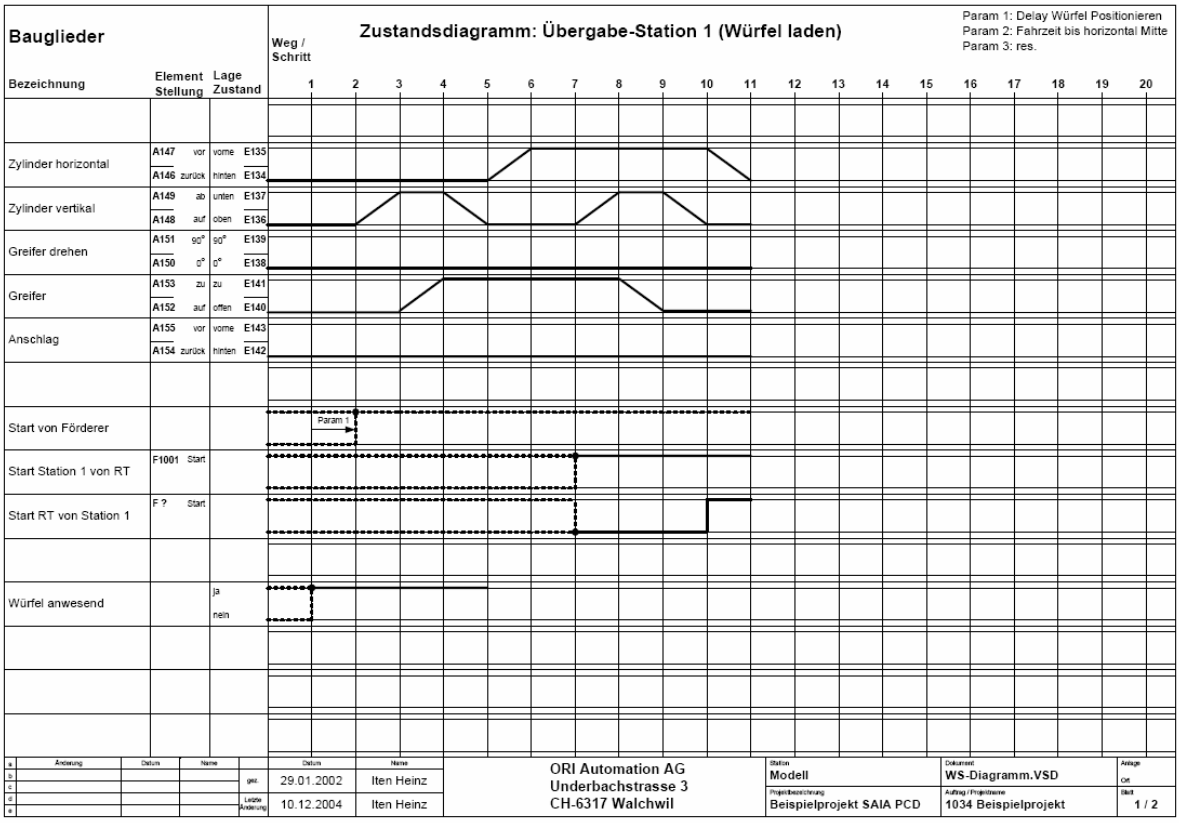
2.4 Fault monitoring

Each sensor should be monitored and a detailed error message output in the event of a fault. Where error messages are output, the processes for the station concerned should be stopped. A restart should only be allowed after the error has been acknowledged.

3 Procedure

3.1 Preparations

After gaining a broad overview of the task, we gather the information on the hardware. We base this mainly on electrical and pneumatic schemas and any available path/step diagrams.



Picture 6: State diagram

Any existing company standards and rules for control or drive design should also be incorporated at the start of the project.

3.1.1 Breakdown of the installation into stations

First, the whole installation should be broken down into stations. By this, we mean units that can perform a closed sequence of operations. Examples are:

- Rotary table (RT)
- Equipment handling
- Stacking mechanism
- Labelling station
- Marking cell
- Winding or unwinding mechanism
- Press
- Robot cell

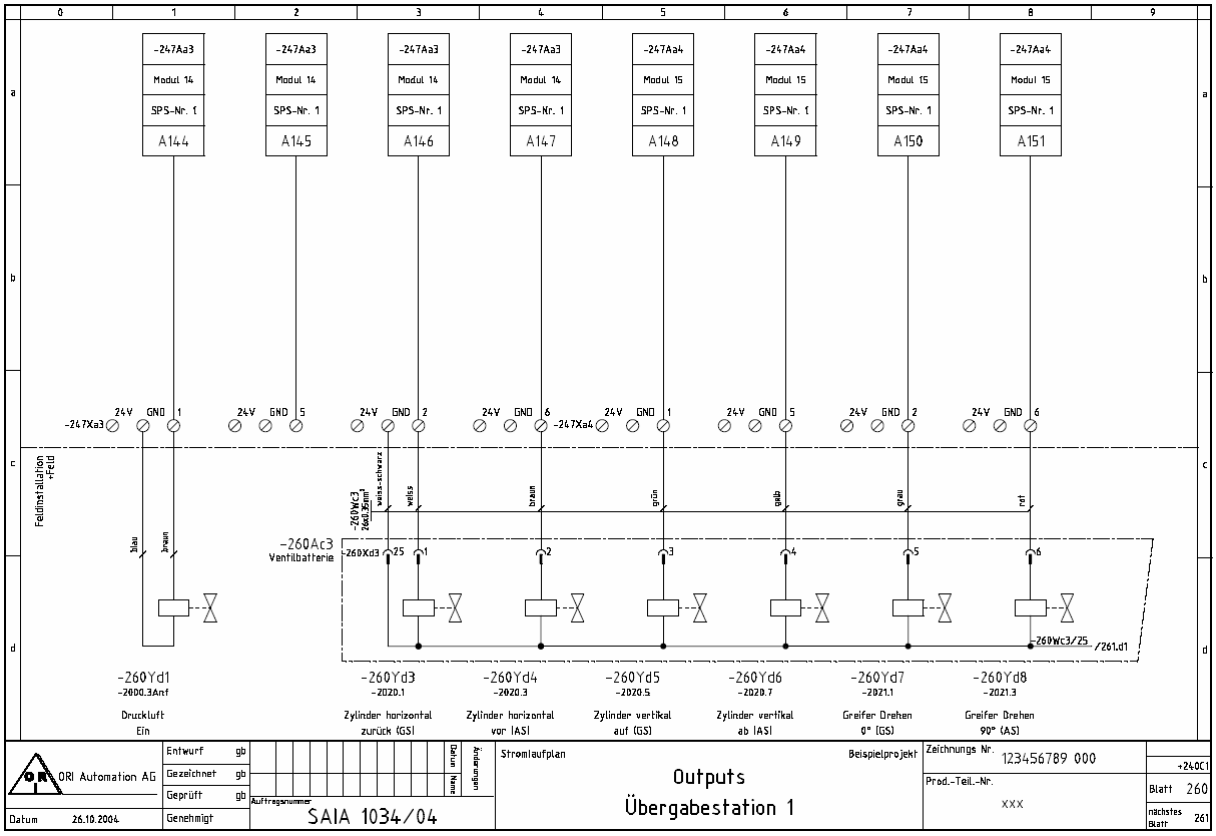
☞ It should be possible to operate all these stations according to the same operating concept. The cylinders and associated limit switches required for processing are assigned to the stations.

To break the project into sub-tasks, the PG5 development environment can be used alongside tools like Excel and Word. This part of the project planning process is crucial, and often determines the subsequent complexity of the project. This breakdown defines the interfaces between the individual software modules. It also has a bearing on the specification of data structures.

3.1.2 Production of input and output lists

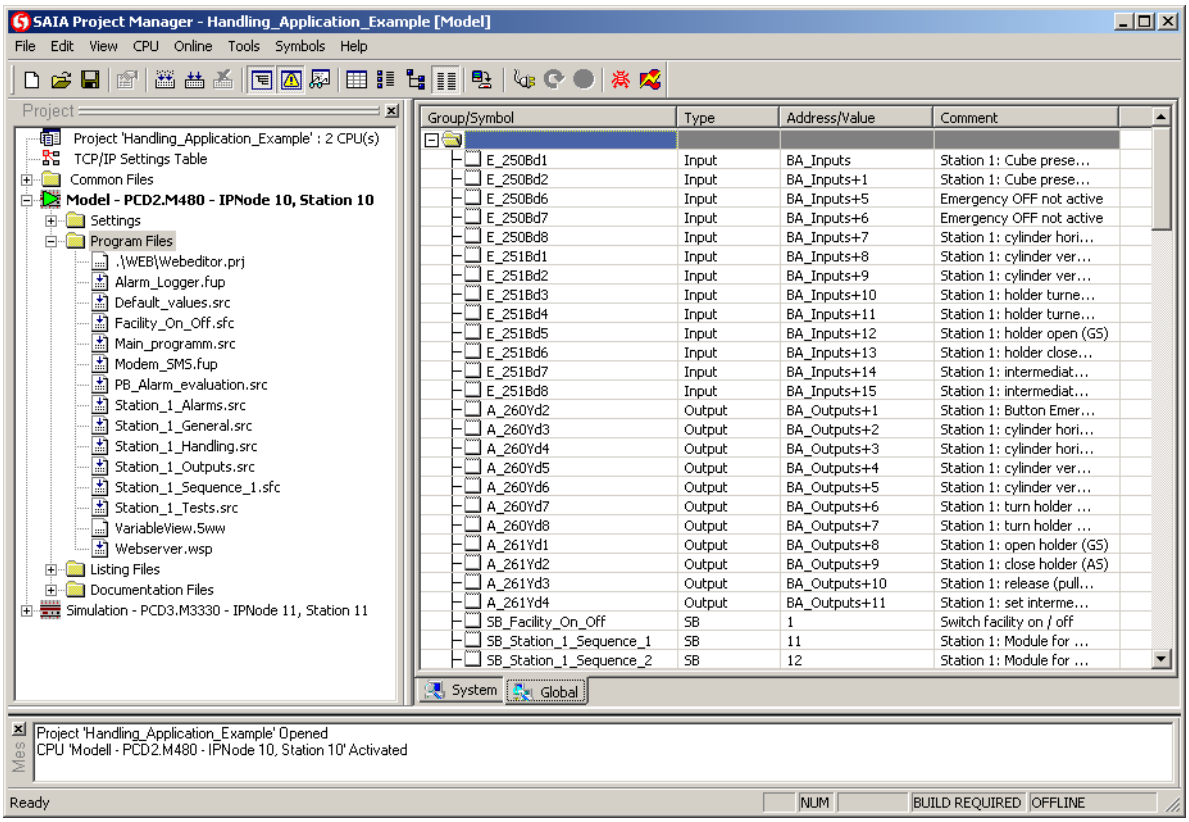
As the input and output assignments are also required for programming, the input and output lists should be entered into the programming tool at the outset if possible. For this, self-explanatory symbols should be defined as far as possible, with meaningful comments. This will make them easier to use later at the programming stage.

The comments and element names can often be taken more or less directly from the CAD system.



Picture 7: CAD schema

The end-customer will gain the greatest benefit if the names of the SPS symbols, the electrical equipment codes and the plain-text labels on the control panel are identical.



Picture 8: PG5 Project Manager

From the point of view of the programmer and the program documentation, the symbol name should also be as self-explanatory as possible. Taking the simulator as an example, the symbol list then looks something like this:

Group/Symbol	Type	Address/Value	Comment
Würfel_anwesend_hinten	Output	16	ES-Sim.: Würfel anwesend von hinten (Bohrung nicht horizontal in Richtung RT)
Würfel_anwesend_unten	Output	17	ES-Sim.: Würfel anwesend von unten (Bohrung nicht vertikal)
RT_positioniert	Output	18	ES-Sim.: Rundtisch: gedreht und in Position
NOT_Aus_inaktiv	Output	21	NOT-Aus OK (nicht betätigt)
Horizontal_hinten	Output	22	ES-Simulation: Zylinder horizontal hinten (GS)
Horizontal_vorne	Output	23	ES-Simulation: Zylinder horizontal vorne (AS)
Vertikal_oben	Output	24	ES-Simulation: Zylinder vertikal oben (GS)
Vertikal_unten	Output	25	ES-Simulation: Zylinder vertikal unten (AS)
Greifer_gedreht_0_grad	Output	26	ES-Simulation: Greifer gedreht auf 0 Grad (GS)
Greifer_gedreht_90_grad	Output	27	ES-Simulation: Greifer gedreht auf 90 Grad (AS)
Greifer_offen	Output	28	ES-Simulation: Greifer geöffnet (GS)
Greifer_zu	Output	29	ES-Simulation: Greifer geschlossen (AS)
Anschlag_eingezogen	Output	30	ES-Simulation: Zwischenanschlag gelöst (GS)
Anschlag_ausgefahren	Output	31	ES-Simulation: Zwischenanschlag gestellt (AS)
B_Not_Aus_Taste_Statio...	Input	1	Taste NOT-Aus bei Station 1 OK
Y_horizontal_zurück	Input	2	Ventilstellung: Zylinder horizontal zurück (GS)
Y_horizontal_vor	Input	3	Ventilstellung: Zylinder horizontal vor (AS)
Y_vertikal_auf	Input	4	Ventilstellung: Zylinder vertikall auf (GS)
Y_vertikal_ab	Input	5	Ventilstellung: Zylinder vertikal ab (AS)
Y_drehen_0	Input	6	Ventilstellung: Greifer Drehen zu 0 Grad (GS)
Y_drehen_90	Input	7	Ventilstellung: Greifer Drehen zu 90 Grad (AS)
Y_greifen_auf	Input	8	Ventilstellung: Greifer öffnen (GS)
Y_greifen_zu	Input	9	Ventilstellung: Greifer schliessen (AS)
Y_Anschlag_lösen	Input	10	Ventilstellung: Zwischenanschlag horizontal lösen (GS)
Y_Anschalg_stellen	Input	11	Ventilstellung: Zwischenanschlag horizontal stellen (AS)

Picture 9: PG5 symbol table

3.2 Programming the PLC

Before we can start programming, we need to structure the software. As manufacturers of production equipment generally operate in one specific area, the tasks and the type of installation are often very similar, and they can often be implemented with one and the same basic structure.

☞ Experience shows that new requirements and functions necessitate maintenance and possibly enhancement of the software. This demands tidy management of the various development phases.

3.2.1 Structuring the software

The structuring of the software automatically defines interfaces between the individual software modules. The aim of a good structure is to break the tasks down into small, manageable modules, producing the simplest and clearest interfaces.

☞ It is important to note that an excessively detailed structure and breakdown will not make the programs easier to understand.

Experience shows that the following tasks can be bundled into separate modules without resulting in over-complex interfaces.

3.2.2 Breakdown of modules

In this example, the following modules have been generated. Some of the functions can be produced with different tools. With these files, the choice of tool is mainly influenced by the personal preferences of the programmer.

Task	Tool	Description
Main program (Main_Program.src)	IL	General functions such as blinker flag etc.
Starting the installation (Facility_On_Off.sfc)	Graftec	Starting and stopping the whole installation. This includes e.g. switching on compressed air - waiting for pressure to build up – switching on and monitoring hydraulic assembly, conveyor belts and other motors in sequence – orderly shut-down of installation etc.
General operation (not used)	IL	Special operational functions for the whole installation, e.g. editing the system clock on the OP
Analogue inputs (not used)	IL or Fupla	Cyclical reading and scaling of analogue inputs
Analogue outputs (not used)	IL or Fupla	Cyclical conversion of control points, and output to analogue outputs
Alarm aggregation (not used)	IL or Fupla	Collating alarms from all stations
Alarm monitoring (PB_Alarm_evaluation.src)	IL or Fupla	Set alarm monitoring with time delay according to bits for individual stations
Alarm logging on SPS (Alarm_Logger.fup)	Fupla or IL	Saving alarms with timestamps

Alarms via SMS (Modem_SMS.fup)	Fupla	Sending alarm messages to operator via e.g. SMS
Functions 1..n	Various	Various functions that can be called with different parameters from different stations

3.2.3 Modules for stations 1 .. n

Task	Tool	Description
Station_1_General	LIST	Calling the various functions of this station as described below. • Process day and total counters • Check home position • Check critical states, e.g. release of rotary table from station, release of workpiece from station
Station_1_Handling	LIST	Operation of station, setting of operating mode, switching condition for station sequences
Station_1_Sequence_1	Graftec	1st processing sequence for station
Station_1_Sequence_n	Graftec	Further processing sequences
Station_1_Outputs	LIST	Control station outputs based on operating mode and progress of work
Station_1_Alarms	LIST	Set alarm monitoring with possible time delay for all operating modes

If you look more closely at the table above, you may notice that the outputs and alarms are handled by separate modules and cover all operating modes.

You may wonder:

? Aren't the interfaces between operation, the sequences, the output connections and the alarms then much too complex?

⇨ No:
Operation from the OP or terminal only sets control bits (flags), not outputs. The decision as to whether outputs can be addressed is always governed by the SPS.
The sequences set status bits (flags) in the SPS which reflect the desired status of the output and do not address the output directly.
In the "Station_x_Outputs" module, the outputs are not only set on the basis of the desired status from the sequences. It can also be used to prevent mechanical collisions during manual cylinder movements or an orderly reset.

⇨ The alarms are triggered independently of the sequence. Alarms are generally incorrect sensor signals in specific actor states, and they can be monitored independently of the operating mode.
In other words, where a vertical module (e.g. cylinder) is moved upwards, the system should release the lower limit switch after a delay, and address

the upper limit switch. If this has not happened after a certain time, an appropriate alarm display should be output (as it is clear that the mechanical movement of the cylinder has been impeded).

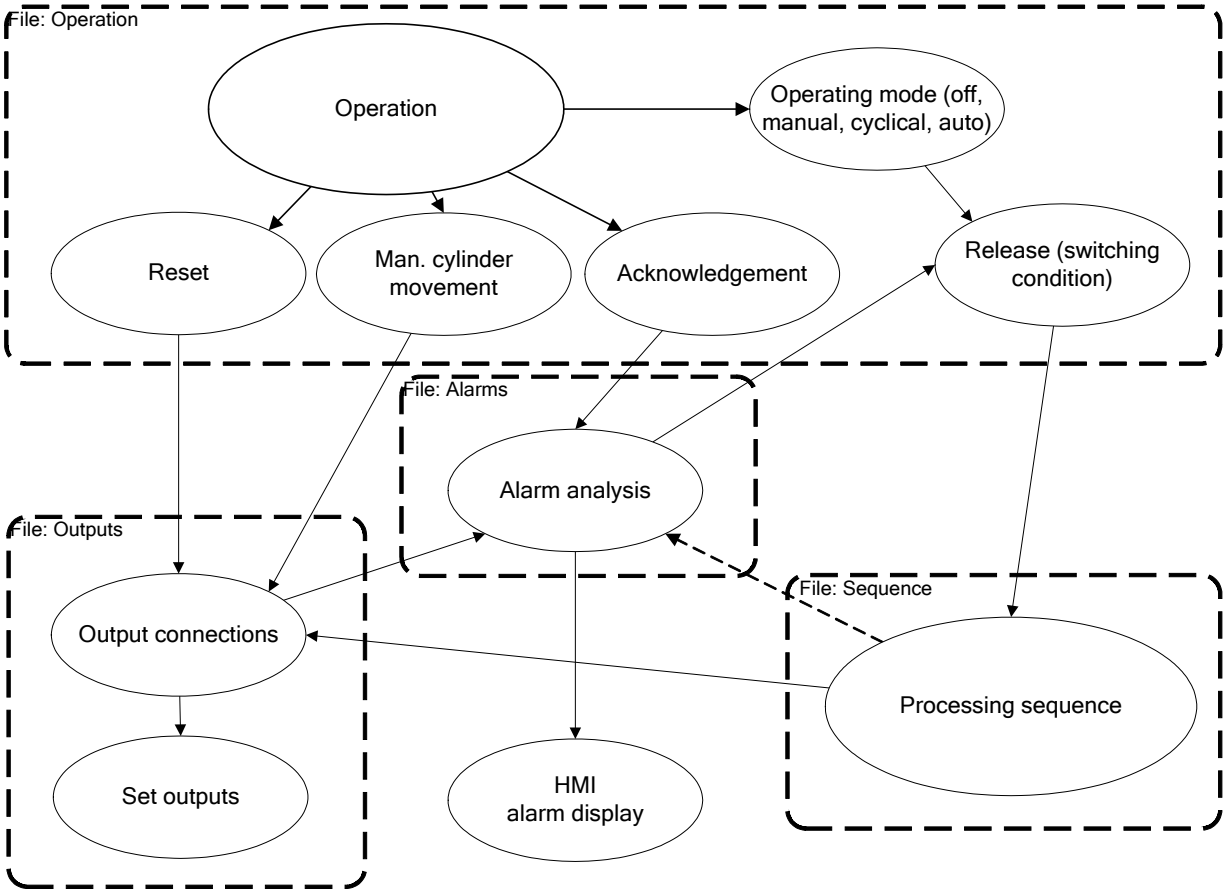
3.2.4 Advantages of this type of structuring

- The tasks are clearly assigned to certain modules and so handled in one place.
- Anyone who knows this structure can find his way around the program more quickly.
- Enhancements and corrections do not have to be made in several places (sources of error are minimized).
- There is only one processing sequence for all operating modes.
- On commissioning, each operating mode does not have to be tested separately.
- Troubleshooting is simplified if the outputs are set in one place only.
- The security of critical situations such as turning the rotary table can be increased by additional querying of the “RT basic setting” flag by all the stations involved. These are set up cyclically, independent of the sequence. This also helps to detect loss of pressure or a limit switch left on after the correct termination of the sequence.

3.2.5 Disadvantages of this type of structuring

- For very small simple tasks, the minimum effort is greater.

3.2.6 Information flow within a station



Picture 10: Information flow

The following information can be inferred from the flow diagram shown above.

- The outputs are not set directly either by HMI or the sequence.
- Alarm analysis is driven mainly by the output connections, and only in special cases by the sequence.
- In the processing sequence, the release signal (continue) and the state of the installation (sensors) are accessed, not the individual operating modes.

3.3 Structuring of the example project

The following structure is suggested for the example project:

3.3.1 Main program

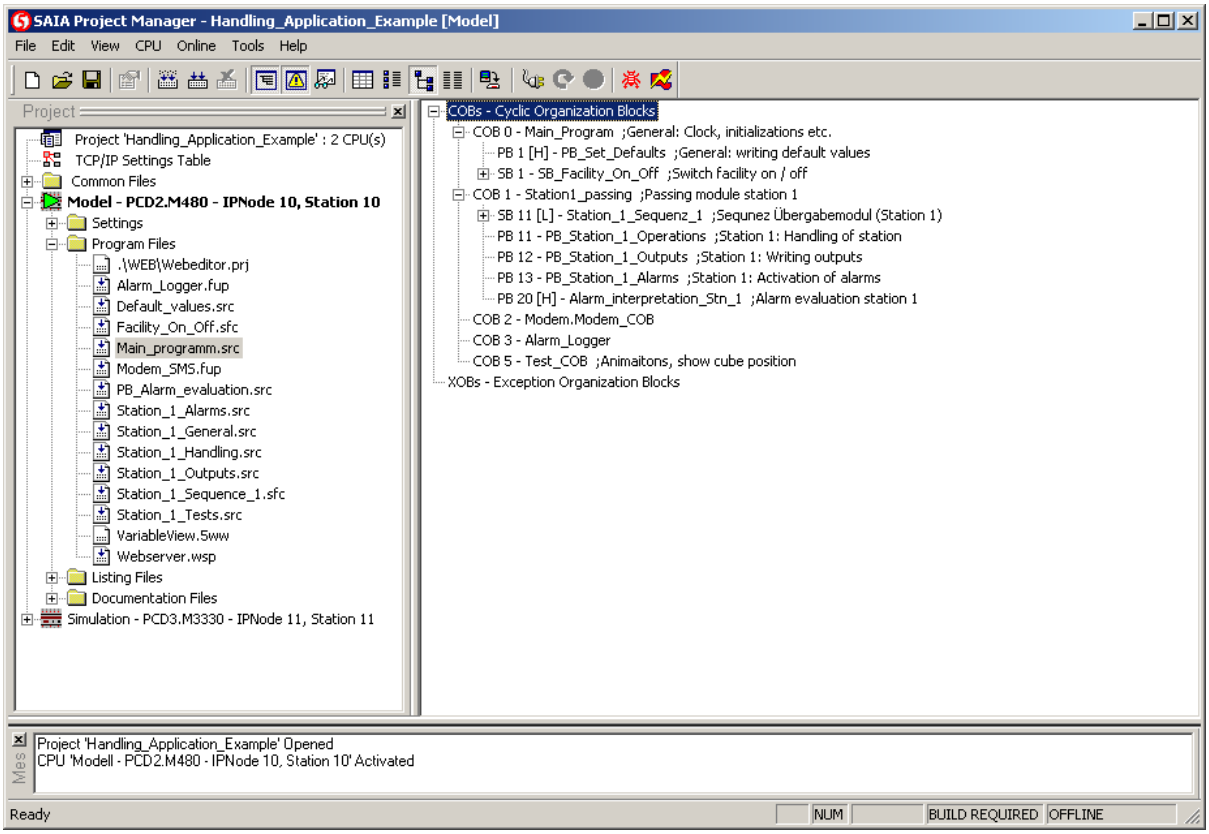
The main program (COB0) handles all general tasks. These include:

- Loading default values
- Setting timers
- Starting the installation

3.3.2 Transfer module – station 1

The whole transfer module is handled by COB1. This includes the following functions, held in separate files.

- General, with monitoring, basic settings, calls to various station modules
- Operation of the station
- Raising alarms on the station
- Output connections for the station
- Processing sequence for the transfer module



Picture 11: PG5 Project Manager with program structure

For test purposes, the image numbers for the animated model and the position of the cube are set up in COB5.

3.3.3 Choice of tools

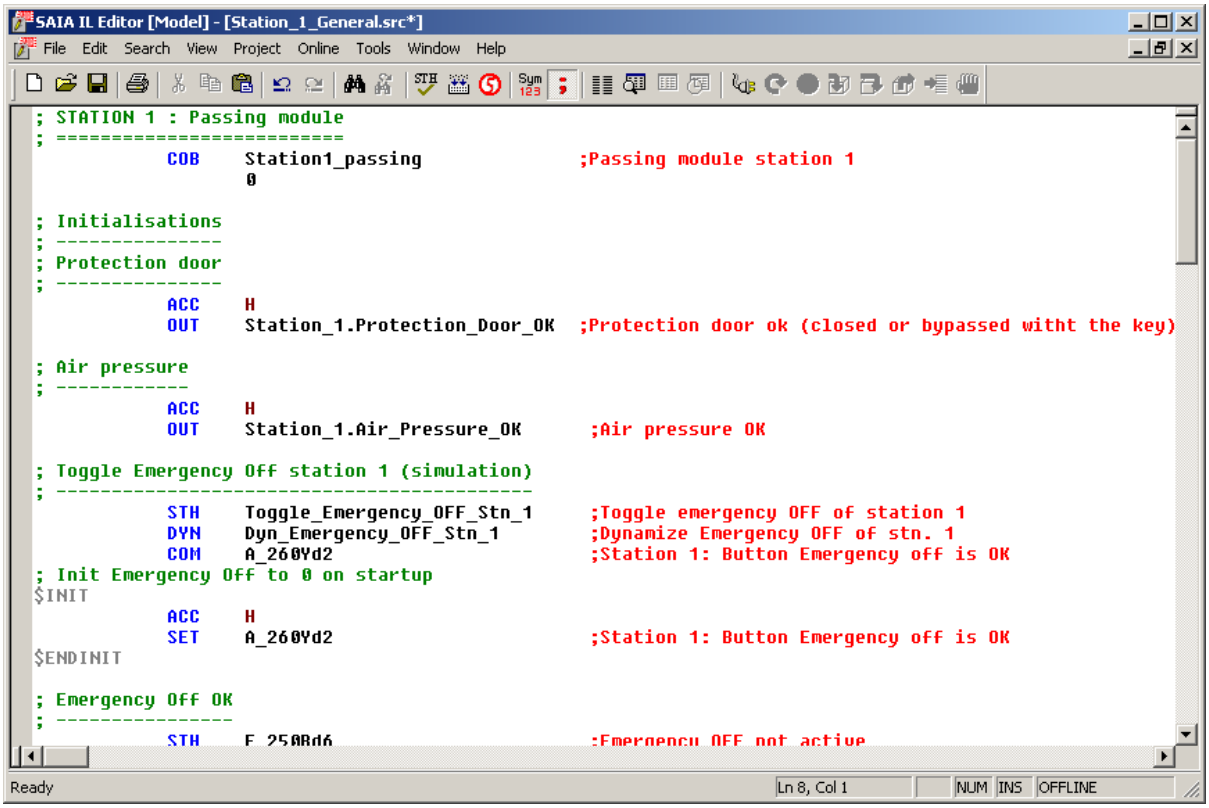
Experience has shown that the choice of tools is influenced by various factors. For example, the customer may often prefer a particular type of documentation, or the programming knowledge of the support staff may be taken into account. The programmer will have a preference for one tool or another, or he may focus on efficiency.

In our example project, we will deploy the best and most-used tools for the tasks.

3.3.4 Coding: general

This file executes the following tasks:

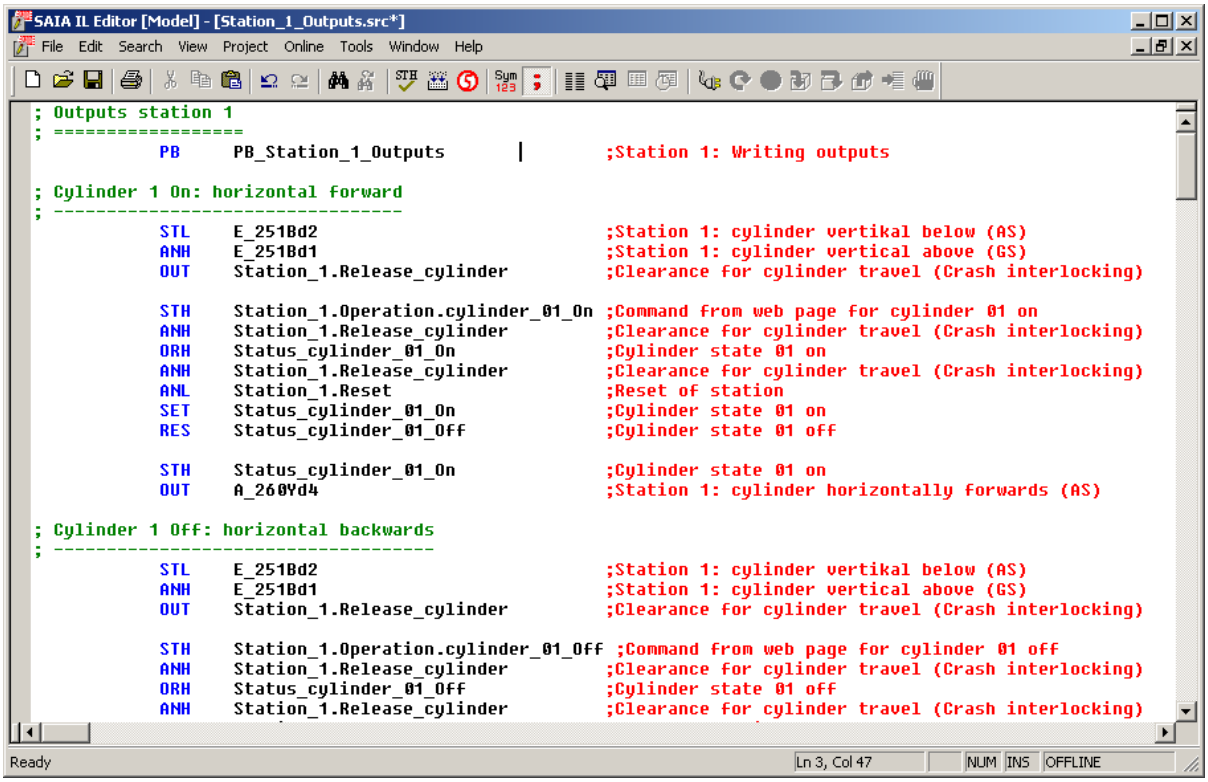
- Various initialization tasks on the station
- Various monitoring tasks (guard doors, air pressure etc.)
- Checking the home position for the station
- Reading the enabling instructions (turning the rotary table, centring the workpiece holder etc.)
- Calling the sequences
- Calling various program blocks (operation, outputs, alarms etc.)
- Alarm analysis



Picture 12: Saia IL Editor (SEdit)

☞ Important: So as not to have to reserve separate status flags for the cylinder circuits, the same flags are used for all stations. The cylinder states for 16 cylinders are held in a register for each station, loaded before use, and save again afterwards. This procedure allows a kind of multi-instance programming.

3.3.5 Coding: output connections



Picture 13: SEdit

In the output connections, all the outputs assigned to the station are switched on or off. As an example, we will look at the first cylinder on our transfer station.

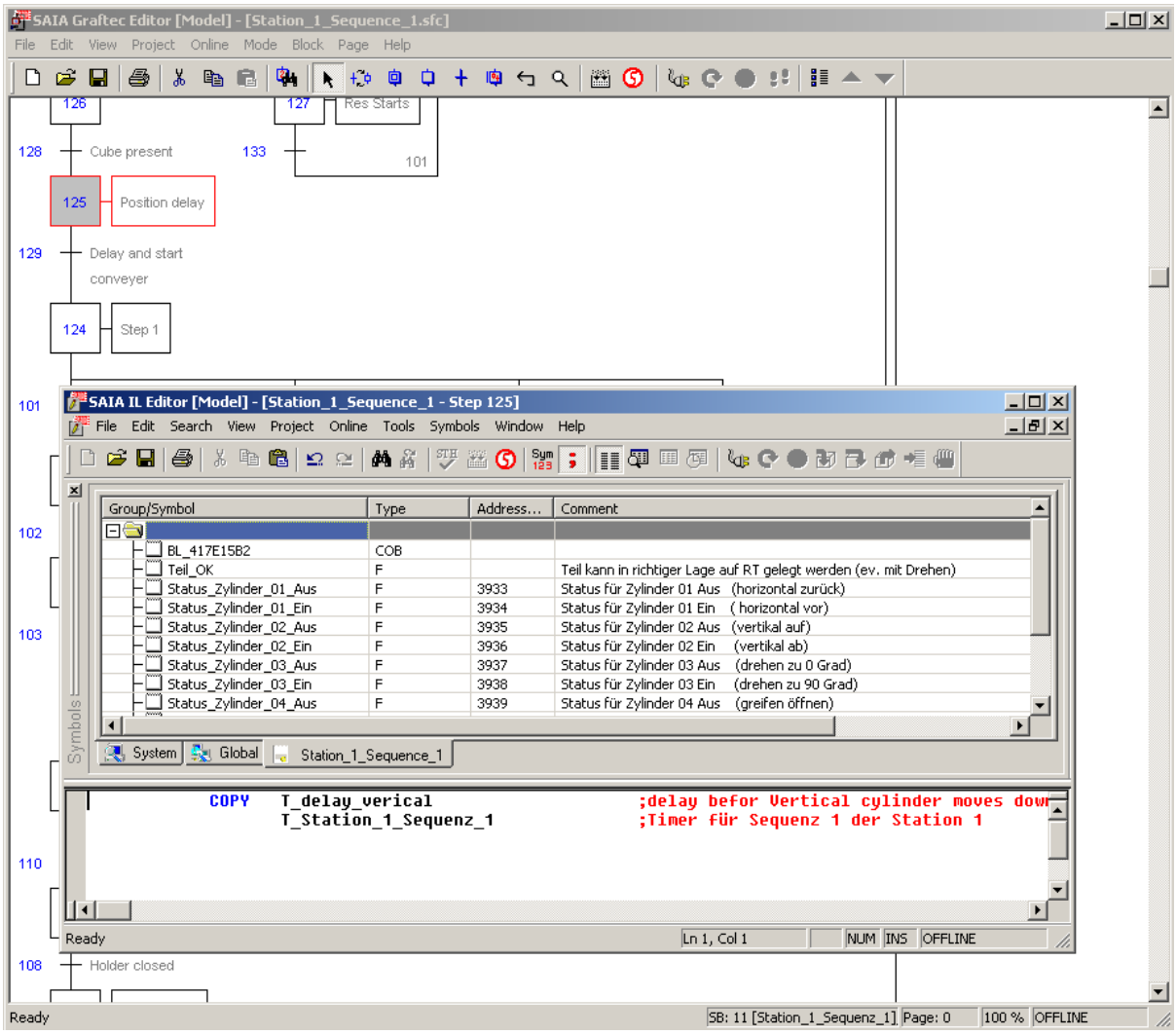
In out case, a bipolar valve is set.
In other words, The valve is moved into an “On” or “Off” position with a coil for each. The notes show such valves as monostable (e.g. 5/2-way valve) and bistable (5/3-way valve with centre position) The control of the output connections des not change. (Bistable valves are used e.g. where you want to interrupt the movement via the hardware with a light curtain.)

Where unipolar valves are used, there is only one coil for the “On” setting. This is switched on (SET instruction) when "Status_Zylinder_01_Ein” is set, and switched off (RES instruction) when “Status_Zylinder_01_Aus” is set.

For each manual movement, i.e. “forward” and “back”, a separate crash blocking mechanism can be implemented. In our case, the cylinder is only released for either movement when the vertical cylinder is up (E251Bd2=low, E251Bd1=high). This crash blocking mechanism is also used in the reset case.
When programming processing sequences, the status bits are manipulated directly, i.e. the programmer is responsible for preventing collisions. On the one hand, it is possible to program a collision into the processing sequence, but this will be corrected in the manual operation module after successful commissioning, at the latest. On the other hand, special operations like switching off pressure or two-way pressurising of cylinders can be achieved.

☞ Caution: The same flag is used to release the cylinder for all output connections within a station.

The operation of individual cylinders (manual movement of cylinders) is handled via the flags “Station_1.Bedienung.Zylinder_01_Ein” or “Station_1.Bedienung.Zylinder_01_Aus”. In our case, these are manipulated with the operation created in the web editor. The flags are set for as long as the corresponding keys are pressed.



Picture 14: Graftec Editor with SEdit and Symbol Editor (SSE)

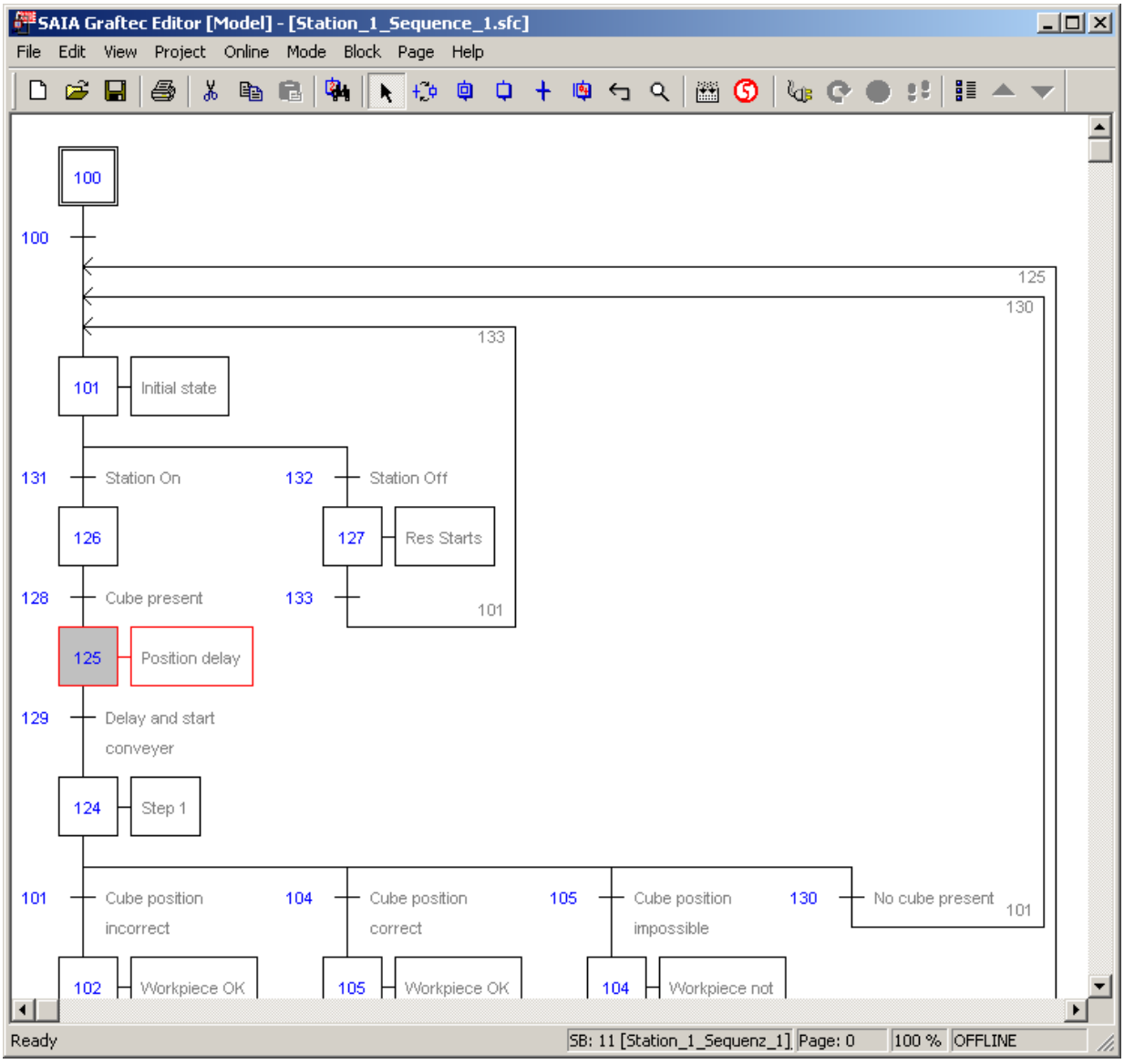
From the processing sequence, the movements are triggered by manipulating the flags ”Status_Zylinder_01_Ein” and “Status_Zylinder_01_Aus”. In our case, the global symbols have also been defined locally and expanded with information on the movement. This makes the code clearer and possible oversights are easier to detect.

3.3.6 Coding: operation

Operation is mainly handled via flags. A group of identically named operating flags is defined for each station.
When a control panel or the web editor is used, these flags are set directly whenever the relevant control elements are activated. If operation is via mechanical keys and switches, the flags have to be set and reset before operation using the digital inputs.

3.3.7 Coding: sequence

We program the processing sequences with GRAFTEC, a graphical tool for easy creation, administration and documentation of sequential process flows.



Picture 15: Graftec Editor

The programming of processing flows begins with the graphical construction of the workflow structure. The starting point is the home position of the station, because a reset of the station also remains active until the station returns to its home position.

We only call this sequential block (SB) where there is no reset. In the rest case, the SB is returned to its initial setup to restart the sequence.

```
; Reset Graftec-sequence
            STH     Station_1.Reset                 ;Reset of station
            RSB     H       SB_Station_1_Sequence_1 ;Station 1: Module for passing forward (sequence 1)
                    100
            CSB     L SB_Station_1_Sequence_1       ;Station 1: Module for passing forward (sequence 1)
```

Picture 16: Reset and call of Graftec sequence

3.3.8 Coding: alarm analysis

Before the alarms from a station can be analysed, the individual error bits must first be set. This is done in the module “Station_1_Störungen.src”.

```
; Error message 10: Intermediate stop not set
            STH     A_261Yd4                    ;Station 1: set intermediate stop horizontal (AS)
            ANL     E_251Bd8                    ;Station 1: intermediate stop set (AS)
            ORH     A_261Yd4                    ;Station 1: set intermediate stop horizontal (AS)
            ANH     E_251Bd7                    ;Station 1: intermediate stop open (GS)
            OUT     Temp_element_F0+10          ;Temporary element 0 [to 64]; type flag

; Error message 11: Emergency OFF active (-E250Bd6)
            STL     E_250Bd6                    ;Emergency OFF not active
            OUT     Temp_element_F0+11          ;Temporary element 0 [to 64]; type flag
```

Picture 17: Setting active alarms

When an input signal does not reflect the expected state, a temporary error bit is set. This will be the case e.g. where a cylinder movement is started, until its end-position is reached. 32 of these temporary flags are placed together at the end of the module in the “aktuelle_Störungen” register.

Alarm analysis is called once a second or by acknowledging the alarms.

```
; Alarm check
            STH     F Pulse_1_Sek               ;Clock; high for 1 program cycle every second
            ORH     Station_1.Acknowledge_Alarm ;Acknowledge alarm
            CPB     H Alarm_interpretation_Stn_1 ;Alarm evaluation station 1
```

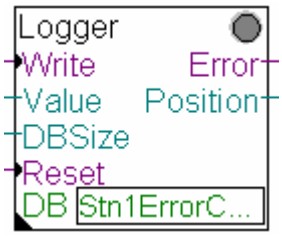
Picture 18: Calling alarm analysis

The bit pattern in the “aktuelle_Störungen” register is regularly checked for changes. At every change to the bit pattern, the configurable delay for alarm displays in restarted. Only when this delay time has elapsed and an error bit has been set is the relevant alarm displayed and the station set to status “Störung_aktiv”. Based on the error bit set in the “angezeigte_Störungen” register, a plain-text display can now be output.
The status “Störung_aktiv” for the station is retained after the alarm has been cleared until it has been acknowledged, to prevent any unintentional continuation of the sequence.

☞ Advantage: This sequence-independent alarm analysis is generally applicable and will always result in the correct alarm display, even in manual cylinder movement.

3.3.9 Coding: Storing alarms

Each time an alarm occurs, this event is stored by the program module Alarm_Logger.fup. This is done by specially for this project created FBoxes with the name Logger.
Thanks to these FBoxes, this rather complicated task is quick and easily realized:



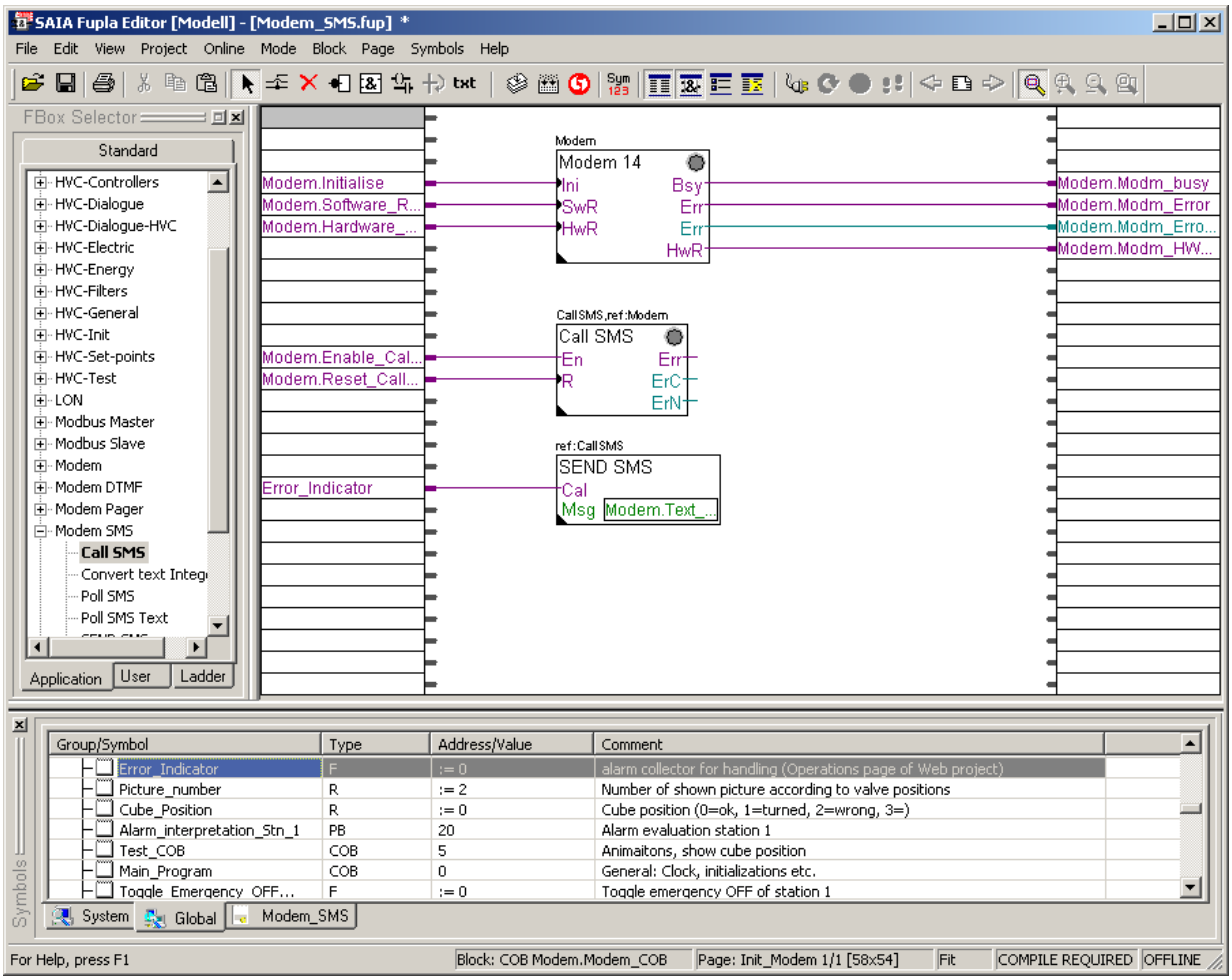
- A rising edge at the input “Write” causes the value present at input “Value” to be written into the in the field “DB” specified data block (DB). The value is appended to the existing values.
- The value present at input “DBSize” specifies the amount of values to be written into the DB.
- A rising edge at input “Reset” resets the buffer.
- The position of the last entered value is written to the FBox output “Position”.

The values written into he DBs (one DB for the timestamp, one for the date and one for the error code) will be visualized by the web interface of the PCD.

The FBox described is not present by default in the PG5! The relevant library must be installed first!
To do so, please execute the file “FBox_Installer.exe” from the PG5 Project tree folder „Common Files“.

3.3.10 Coding: Sending alarm messages

The program module Modem_SMS.fup is responsible for sending an SMS to a mobile every time an alarm occurs on the application.



Picture 19: Screenshot Fupla program Modem_SMS.fup

For this program module to work, the following prerequisites are to be fulfilled:

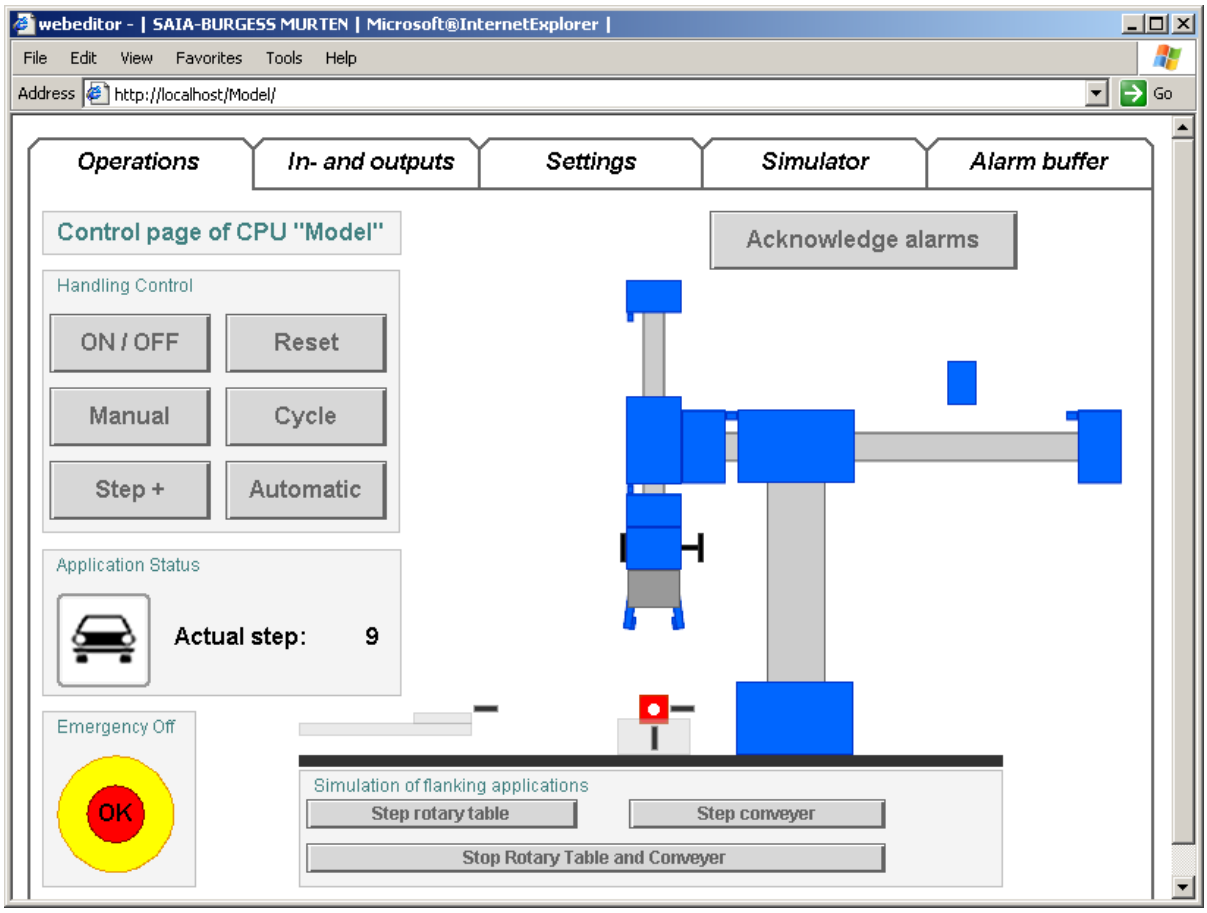
- An analogue modem module PCD2.T813 must be present, connected to the TTL plug (port 1) over slot 2 of the PCD2.M480.
- An analogue phone line must be available and connected to the modem.
- The number of your provider's SMS service center must be entered correctly in the FBox "Call SMS"
- The button "Enable SMS" on page "Settings" of the web interface must be activated.

Additional information regarding the SMS functionality as well as online connections to the PCD with modems can be found in the example project "Sending and receiving SMS" on the support site:
http://www.sbc-support.ch/GettingStarted/examples/examples.htm.

Operation with Web Editor

The PCD2.M480 control used has an integrated web server. With this, it is simple and economical to implement a user or configuration interface.
An (optional) Saia S-Web Editor is integrated into the PG5 programming tool; this especially simplifies connection to the PLC program via the symbol tables. The functions of the web editor are outside the scope of this document; please refer to the relevant manuals.
The WEB Builder tool is used to group the HTML pages created in S-WEB Editor and other files into PCD-compatible data blocks.

For instructions of the operation of this project, Please refer to the accompanying "Handling_Application_Installation" document. This document describes the procedure for installing and commissioning this project, and the use of the web pages on the stations.

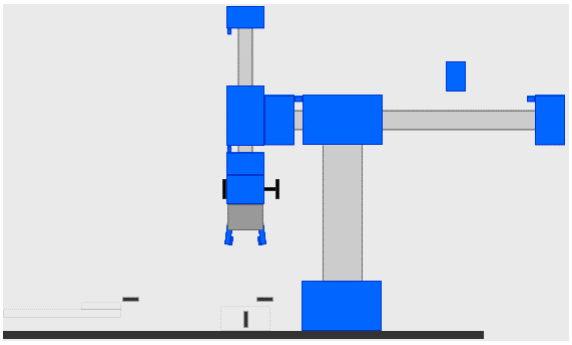


Picture 20: Main page of web interface

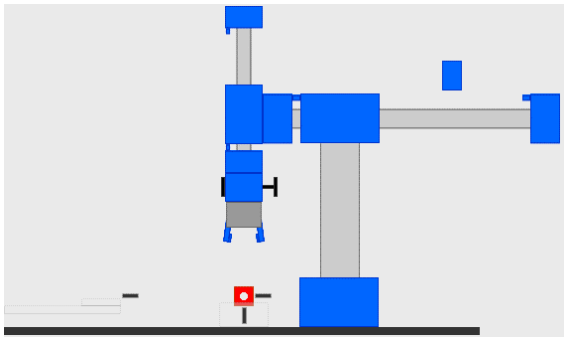
3.4 Movement sequences

In the following sections, the movement sequence of the stations is presented in diagrammatic form. Depending on the position of the cube, the movement sequence of the transfer station may vary.

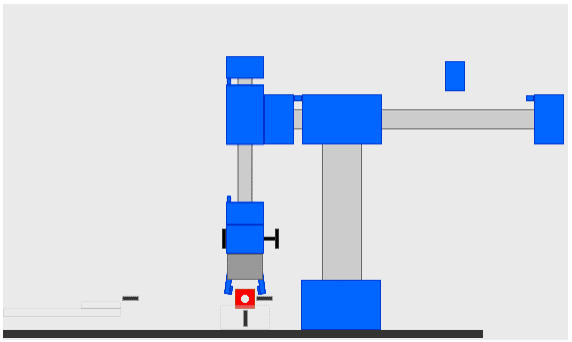
3.4.1 Cube in correct position



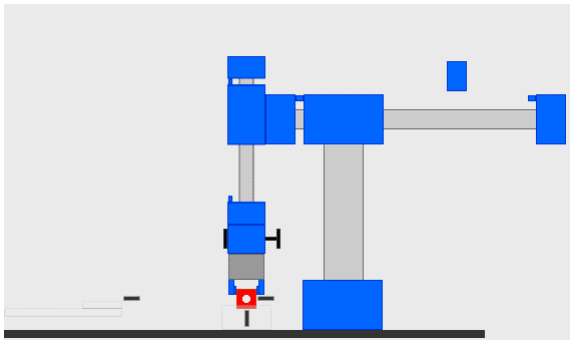
Home position



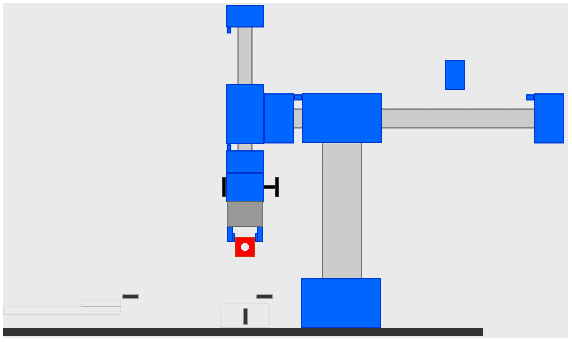
Home position with cube (position OK)



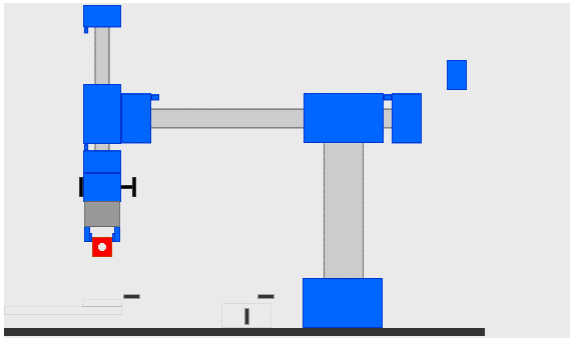
Vertical down



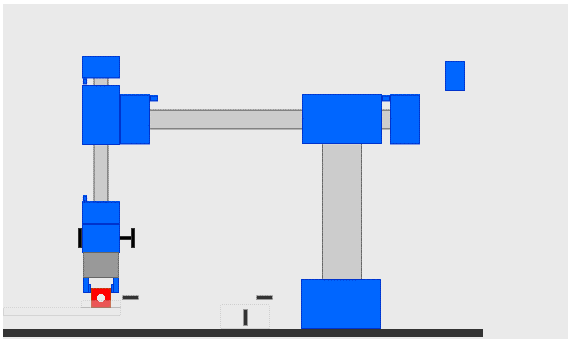
Pick up cube



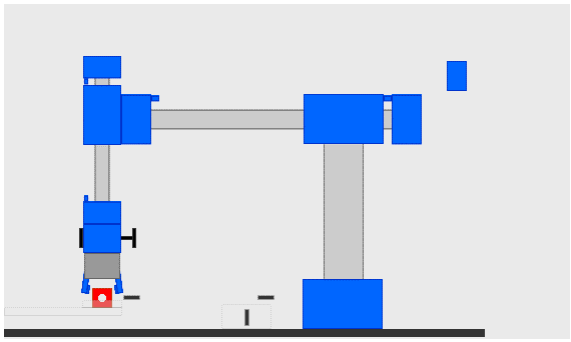
Raise cube



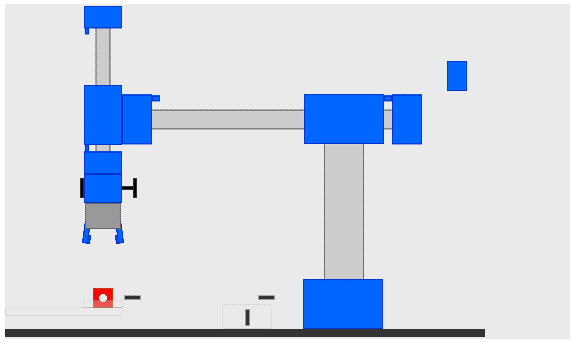
Horizontal forward



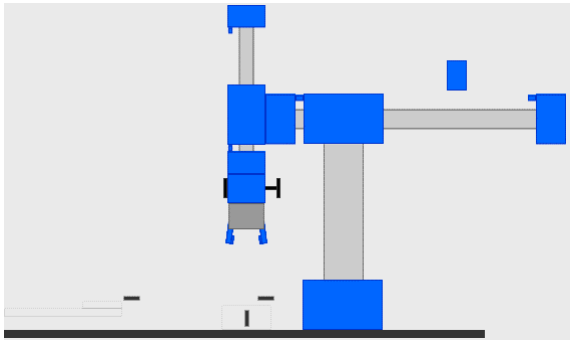
Cube on rotary table



Open holder



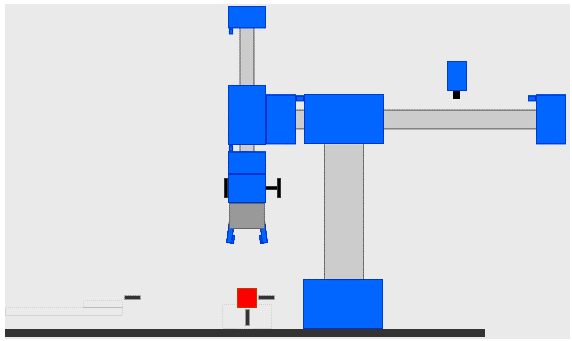
Raise holder



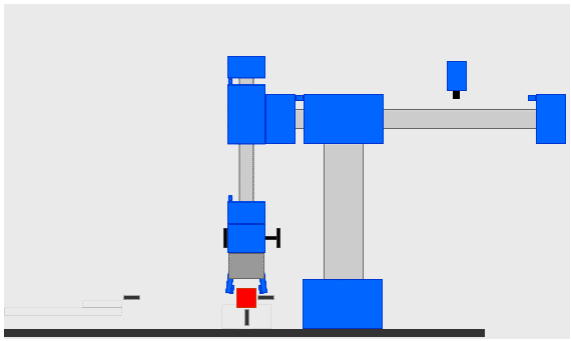
Starting position

3.4.2 Cube in incorrect position

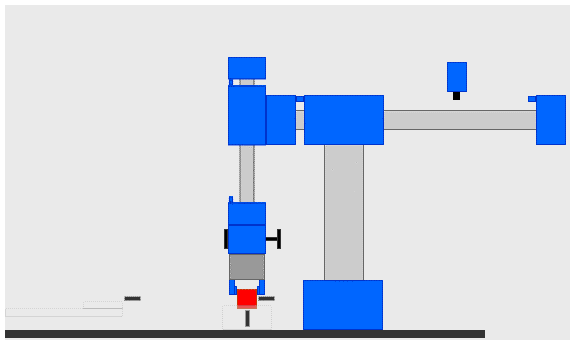
The cube cannot be moved into the correct position. It is removed and discarded in the eject position.



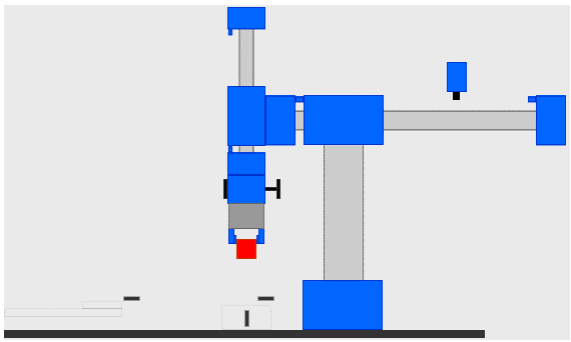
Set stop



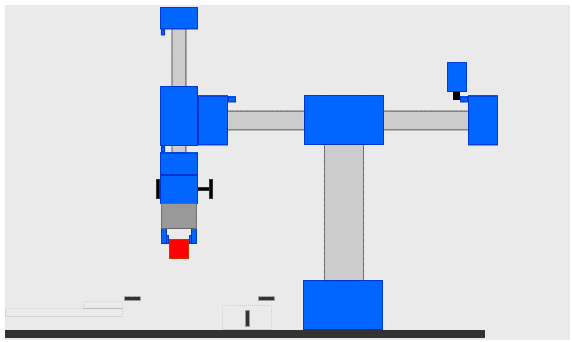
Vertical down (cube incorrect)



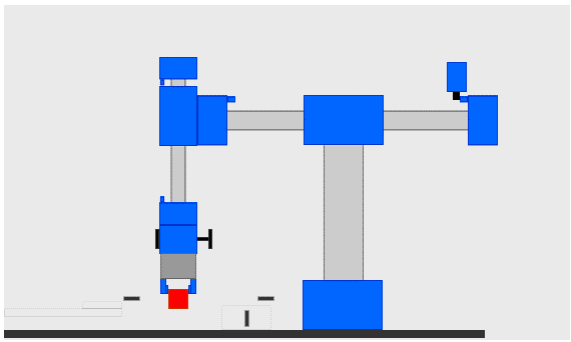
Pick up cube - incorrect



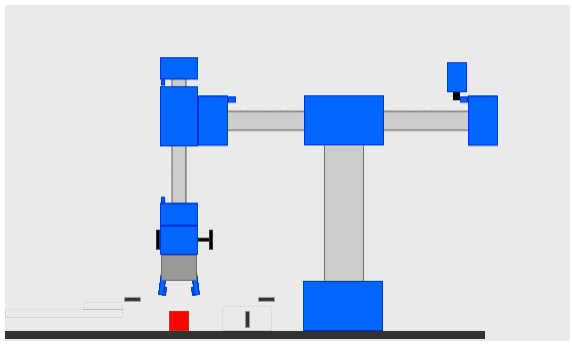
Vertical up (cube incorrect)



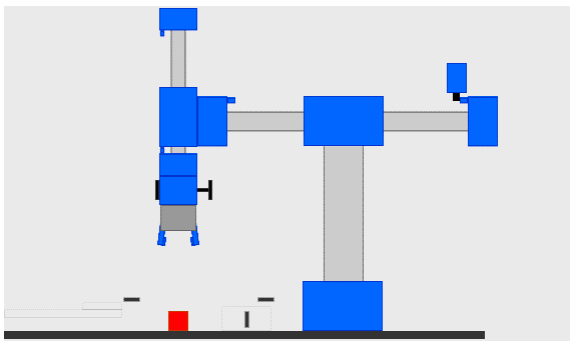
Move to centre



Centre lower



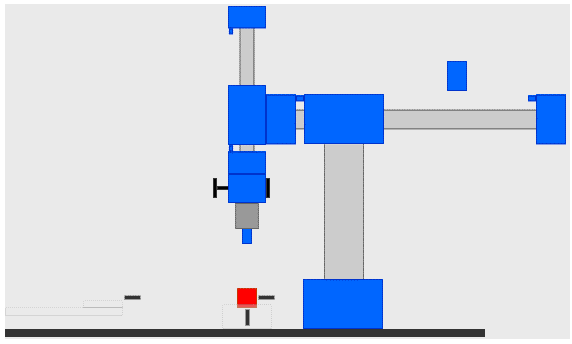
Centre open holder



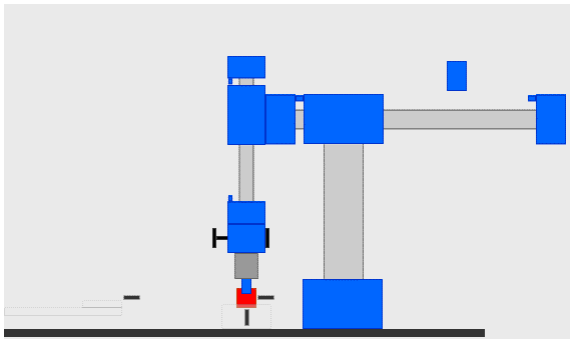
Centre raise

3.4.3 Cube in rotated position

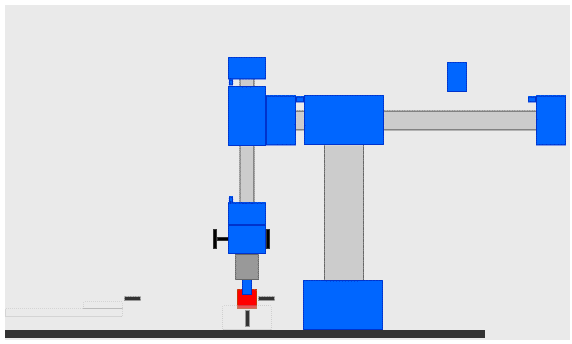
The holder most be turned to pick up the cube, to place it in the correct position on the rotary table.



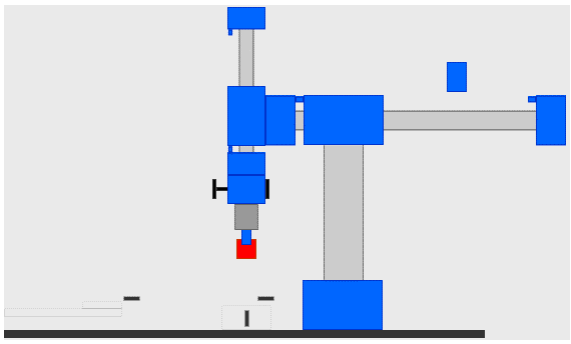
Turn holder



Lower rotated



Pick up rotated



Raise rotated

From this position, the sequence continues as for the correct position of the cube from “Raise cube” onwards.