

TrySim V2.9 © 2003 Cephalos GmbH

**Testen Sie Ihr SPS-Programm, bevor
sich das erste Rädchen dreht!**

Table of Contents

Foreword	0
Part I Introduction	15
1 General.....	15
2 Structure of the system.....	16
Part II Machine	19
1 System of coordinates.....	19
Origin	19
2 Editing the machine.....	20
Start up	20
Create new elements	20
Marking elements	21
Edit properties	22
OK button	22
Cancel button.....	23
Fixing elements	23
Edit window "Static element"	23
Delete elements	24
Select another view	24
Color depth	24
Resolution	25
3-D view	25
3-D control with keyboard.....	26
3-D control with mouse	26
Graphic filter	27
Properties of the 3-D window	28
Store/load a 3-D viewing angle	29
Zoom	30
Adress table	30
Element tree	31
Groups	32
Group manager.....	33
Group Editor.....	33
Positioning elements	34
Reference point	35
Simulation of the machine	36
How to find Elements	36
Structuring huge Machines	37
3 Common properties of elements.....	38
General	38
Name	39
Father	39
Child	40
Position	41
Size	41
Fixabilitiy	42

Visibility	43
Color	43
PLC Connection	44
Comment	45
Anchor	45
Script	46
4 Static elements of simulation	48
Overview	48
Sensors	49
Light barrier	49
Light sensor	49
Limit switch	50
Limit Switch Nose	51
Spy	51
Distance Sensor	52
Resolver for Transporter/Chain/Track	52
Bar Code Scanner	53
Running Wheel	54
Actuators	54
Generator	54
Continuous generator	55
Continuous dynamics / track	56
Destroyer	56
Linear mover	57
Linear mover with sensors	57
Conveyor	58
Joint	59
Hook	60
Free point	61
Chain	61
Node	62
Segment	62
Turner	62
Rider way	63
Mangle	63
Turntable	64
Arc Belt	65
Turnable Conveyor	65
Controls	65
Pushbutton	65
LED	66
Digital display	66
Data Formats For Digital Display	67
String display	68
Slide controller	68
Data formats for slide controller	69
Text display	70
Oscilloscope	70
Multiple switch	71
Master switch	71
Contact-Editor of the master switch	72
Fluids	72
General	72
Container	73

Pipe	73
Extended Pipe Properties	74
Pump	75
Valve	75
Overpressure valve	75
Check valve	76
Flange	76
Flow meter	77
Level gauge	77
Pressure sensor	78
Analyzer	78
Level switch	79
Fluidor	79
Misc.	79
Saw	79
Divider	81
Cutter	81
Melter	81
Wedge	82
Shifter	82
Dynamic converter, straighter	83
Press	84
Automatic hook	85
Sticker	85
Pawl	86
RFID antenna & data chip	86
Box	87
Board	87
Bar	88
Rider	88
Reactor	88
Peek	89
List of peekable properties	89
Poke	90
List of pokeable properties	91
Thermal elements	91
Thermal mass	91
Heating	92
Characteristic curve	92
Thermometer	93
Thermostat	93
Tools	94
Background	94
Stripe	96
Cross	96
Attractor	97
Speed trigger	98
Master List	98
5 General information about turnable elements	98
How to mark turnable elements	99
6 Drives	100
Simple frequency converter	107
Valve drive by linear mover	108

Joint drive by linear mover	109
Heating drive by linear mover	109
Servo drive for linear mover	109
Servo drive for joint	110
Direct adjustments at servo drive	111
Word motor	112
Word-drive for pump and valve	112
Absolute real drive	112
1 bit-drive for pump and valve	113
Drives of the linear mover	113
Bit motor for joint	113
Bit drives of the joint	114
Crank drive	115
7 Analog value processing.....	116
8 Dynamic elements of simulation.....	117
Normal dynamics	117
Turnable dynamics	118
Part III PLC	121
1 Introduction.....	121
2 Data memory	121
General	121
Naming inputs, outputs and memory bits	122
Naming data in datablocks	123
Instance data blocks	123
3 CPU.....	124
Introduction	124
Accumulator 1	125
Accumulator 2	125
Accu 3 und 4	125
Result of logic operation (RLO)	126
First Request	126
Global data block register	126
Instance data block register	127
Adressregister 1 and 2	127
Status register	128
4 Program.....	128
Introduction	128
Organization blocks	128
Functions	129
Function blocks	132
Static variables	133
Temporary variables	134
5 External (Soft) PLC.....	134
General	134
Safety guidelines	136
Open interface of TrySim	136
Interface, C-Code	137
Interface, Pascal-Code	138
Connection to external S7 via MPI-Datasnake	140
SoftPLC not found	141

SoftPLC not active yet	141
Connection to external S7 via MPI (Prodave)	142
PRODAVE MPI mini from Siemens	144
Interface to SoftPLC of IBH softec	144
Connection to external Allen-Bradley PLC	145
Setting of I/O configuration	146
Setting of a block of I/O configuration	147
Setting of a block of I/O configuration for MPI	147
6 Reference PLC.....	149
Data types	149
BOOL	149
BYTE	150
CHAR	150
WORD	151
INT	152
DWORD	152
DINT	153
POINTER	153
TIMER	154
DATE	154
DATE_AND_TIME	155
TIME_OF_DAY	156
ANY	156
BLOCK_FB.....	157
BLOCK_FC.....	157
BLOCK_DB	158
REAL	158
STRING	158
UDT	159
Constants	159
7 List of operations.....	160
Bit-operations	162
A	162
A(.....	163
AN	164
AN(.....	164
O	165
O(.....	166
ON	167
ON(.....	167
X	168
X(.....	169
XN	169
XN(.....	170
) (closing bracket)	170
= (assignment).....	171
S	172
R	173
SET	173
CLR	174
NOT	174
SAVE	174
FP	175

FN	175
Operations with timers	176
SD	178
Switch on delay	179
SF	179
Switch off delay	180
SP	181
Pulse	181
SE	182
Extended Pulse	183
SS	183
Retentive on delay	184
PV	185
Output DU of a timer	185
Output DE of a timer	186
Insert Timers in FBD/LAD	186
Specification of a timer	188
Entering of duration while using timers	188
Operations with counters	189
Input PV of a counter	190
Output CV of a counter	190
Output BCD_V of the timer	190
CU	190
CD	191
Set counter	192
Reset counter	192
FR	193
Load and transfer operations	193
L	193
T	194
LC	195
Operations with integers	195
+I	195
-I	196
*I	196
/I	197
MOD	198
==I	198
<>I	199
>I	199
>=I	200
<I	201
<=I	201
NEGI	202
+ (Plus)	202
Operations with double integers	202
+D	202
-D	203
*D	204
/D	205
MOD	205
==D	206
<>D	206
>D	207

>=D	208
<D	208
<=D	209
NEGD	209
+ (Plus)	209
Operations with reals	210
+R	210
-R	211
*R	211
/R	212
==R	213
<>R	214
>R	214
>=R	215
<R	216
<=R	216
ABS	217
NEGR	218
SQRT	218
SQR	218
LN	219
EXP	219
Jump operations	219
Jump labels	219
JU	220
JCN	221
JMZ	221
JM	222
JN	223
JP	224
JPZ	225
JZ	226
JNB	227
JC	227
JCB	228
JBI	229
JNBI	229
Loop	230
JL	231
Not implemented jumps	232
JO	232
JUO	232
JOS	233
Shift and rotating operations	234
SRD	234
SSD	234
RLD	234
RLDA	235
SLD	235
SLW	236
RRDA	236
RRD	237
SSI	237
SRW	237

Logical word and double word operations	238
AD	238
AW	238
OW	239
OD	239
XOD	239
XOW	240
INVI	240
INVD	240
BCD operations	241
BCD number	241
ITB	242
DTB	242
BTD	242
BTI	243
Other conversions	244
DTR	244
ITD	244
TRUNC	244
RND	245
RND+	245
RND-	245
Trigonometric operations	246
Radian angle	246
SIN	246
COS	247
TAN	247
ASIN	247
ACOS	248
ATAN	248
Misc. operations with accu 1	249
TAK	249
CAW	249
CAD	250
INC	250
DEC	251
+ (Plus)	251
Operations with accu 3 and 4	252
PUSH	252
POP	252
LEAVE	253
ENT	253
Register indirect addressing	254
Warning using AR1 or AR2	254
LAR1 or 2	255
+AR1 or 2	256
CAR1 or 2	257
CAR	257
Indirect addressing with AR1 and AR2	257
Indirect addressing	259
Operations for blocks	260
OPN	260
BEU	261
BEC	261

CALL	261
CC	262
UC	263
CDB	263
Nought / Zero operations	264
NOP 0 or 1	264
BLD	264
Master control relay (not implemented)	265
Master control relay.....	265
MCR(.....	265
)MCR	265
MCRA	265
MRCD	266
8 Differences between the TrySim-PLC to S7	266
Not implemented properties	266
Master control relay.....	267
Process interrupts.....	267
Time of day interrupts.....	267
Clock interrupt.....	267
Limited ARRAYs	268
Limited function of UDTs.....	268
Function 'FR' with timers and counters	268
JUO, JO and JOS.....	268
STRING	269
Variant implemented properties	269
AND before OR.....	269
Parameter-updating at functions	269
Out-parameter of functions	270
Data type check.....	271

Part IV Program editor

273

1 Block functions.....	273
Create new blocks	273
Block types	273
Open and save blocks	274
Delete a block	274
Export and import blocks	275
Edit OBs and FBs	275
STL	276
FBD	276
LAD	277
Transfer operand	278
Fast operand-input by the numbers-block	278
2 Operations on networks.....	279
3 Editing of a block header.....	279
4 Edit DBs.....	281
5 Transfer a block into the PLC.....	281
6 Watch the program execution	282
7 Breakpoints and single step mode.....	283
Set / delete a breakpoint in STL	284
Set / delete a breakpoint in FBD/LAD	285

Conditional breakpoints	285
Specific features in FBD/LAD	286
8 Rewire	286
9 Cross reference list	287
10 IEC-Functions.....	287
SFB 3 (TP) Time as pulse	288
SFB 4 (TON) On-Delay	288
SFB 5 (TOF) Off-Delay	288
FC 87 (LIFO) Read out the youngest value of a table	289
FC 85 (FIFO) Read out the oldest value of a table	289
FC 84 (ATT) Add a value to a FIFO/LIFO table	289
11 Interface-panel.....	290
12 Symbol table.....	290
13 System function blocks and functions.....	291
SFC 0 Set the CPU-Clock	292
SFC 1 Read out the CPU-Clock	292
SFC 20 Block move	292
SFC 21 Fill	292
SFC 22 Create DB	293
SFC 23 Delete DB	293
SFC 24 Information about DB	294
SFC 64 Read out the system time in ms	294
Part V Export and Import	297
1 Overview.....	297
2 Export to Step®7.....	297
3 Export of the symbol table	298
4 Problems with the export.....	299
5 Import from Step®7.....	300
6 Import-filter.....	301
7 Import of the symbol table	302
8 Problems with the import.....	303
9 Import from STEP®5.....	304
10 Import from STEP®5 Assignment lists.....	304
Part VI Example session	306
1 Example session.....	306
2 Creating the machine	307
3 Writing the PLC-program.....	310
4 Starting the simulation.....	313
5 Watch the program execution	313
6 Possible errors in the example.....	314
Part VII Menu items	317
1 Project.....	317

Project New	317
Project Open	317
Project Open again	317
Project Save all	317
Project Save as	318
Project Add files	318
Project comment	318
Project Export	318
Project Import	319
Project Print	319
Project Exit	320
2 Edit menu.....	320
Edit Edit	320
Edit Select all	320
Edit Select all children	320
Edit Reverse marking	321
Edit Fade Out	321
Edit Fade out children	321
Edit Fade in	322
Edit Fade in with children	322
Edit Focus	322
3 Machine	322
Machine Start	323
Machine Stop	323
Machine Slower	323
Machine Faster	324
Machine Real time	324
Machine Extended	324
Slow motion.....	324
Quick motion.....	325
Machine Library	325
Machine Groups	0
Machine Media	325
Media manager.....	326
Machine Dynamics delete normals	326
Machine Reset time	326
4 Graphic.....	326
Graphic Properties	326
5 PLC.....	327
PLC Reset PLC	327
PLC Master reset	327
PLC Blocks in AS	328
PLC CPU	328
PLC CPU properties	328
PLC Clock memory	328
PLC Clock interrupt.....	329
PLC Retentive memory	329
PLC Cycle time monitoring.....	329
PLC Time system.....	330
6 Block.....	330
Block Open	330
Block Open again	330

Block Save as	331
Block Print blocks	331
Block Block properties	331
7 View.....	332
View Toolbar	332
View Status Bar	332
View Dimension	332
View Symbolic representation	333
View Rezoom	333
View Resize window	333
View Monitor (DB-window is active)	333
8 Extras.....	334
Extras Quick Com/Script	334
Extras Quick Selected	334
9 Window.....	335
Window Cascade	335
Window Next to each other	335
Window One below the other	335
10 Help.....	335
Part VIII Options	337
1 Overview.....	337
2 Options Simulation.....	337
3 Options Directories.....	338
4 Options Machine	338
5 Options Edit machine	339
6 Options PLC editor.....	340
7 Options Toolbars.....	341
Index	342

Part



1 Introduction

1.1 General

TrySim is a development system for PLC programs that allows not only to simulate a [PLC](#), but also to reproduce the run of the [machine](#), which has to be controlled. Therefore any program created by TrySim can be tested and optimized under real conditions by using a PC and Win'98/NT/2000/XP/VISTA only.

Optional external PLC simulations or real steerings can be connected. In TrySim only the machine will be simulated.

For the professional programmer this means a considerable simplification and acceleration for the commission of a system and provides various options for the beginner to create different types of machines by writing and testing corresponding programs and without being dependent upon the workstation at his/her training center.

In preparation for a simulation of the machine, you can choose among a variety of elements, such as motors, cylinders, limit switches, light barriers etc. These elements can be arranged in a three-dimensional manner, the graphical representation corresponds to a technical diagram. In designing TrySim high priority was assigned to clarity and speed of execution but not to optical effects which do not provide any benefit. The machine can also be displayed in a three dimensional view.

The simulated PLC utilizes the instruction set of Siemens S7-400. Our program editor has little resemblance to the STEP®7 development system, because it is not our intention to replace the training at an original Siemens programming device.

If you want to run the completed program on a real PLC, you can easily transfer it to the STEP®7 system. The inverse direction, to write a program on a Siemens STEP®7 system and transfer it to TrySim, is also possible without any difficulties.

STEP®7 and S7-400 are registered trademarks of Siemens AG.

History of TrySim

TrySim has been fully developed within our company. Our job is to develop controls for special devices, such as transport installations. Although these are rather simple machines, anyone who has ever tried to control a system of 30 segments of different switches, turntables, an oven, and a scale, will confirm that many things may go wrong. Moreover, the customer asks for answers to questions as follows:

How many paletts (the exact number) will pass the machine per hour?

Will a jam caused by a 10-minutes break be disentangled to be resolved in a reasonable time intervall or will it require a substitute?

Answering questions such as these requires a considerable amount of theoretical calculations and a significant amount of time. Moreover these machines are normally mounted completely at the site of the customer and it is not before the last screw has been tightened that the real program testing can begin. Regarding this test, the programmer uses a development system which can easily increase to an amount of several million Euro. So everything must be done to shorten the duration of this final test.

Our purpose was to shorten the very expensive commissioning time and therefore we have developed TrySim, with its machine and PLC simulation, and have used it successfully. We have also found that the procedure of writing the program on the original development system first and then transferring it to the simulation program has turned out to be unsatisfactory. This is why we have also integrated program editors into the simulation system.

Of course, there is not only a need to simulate transport installations. As a result of a continuously growing range of elements (which is still expanding) the types of machines that can be simulated has been expanded, even if, at this stage, the simulation is essentially confined to mechanical machines and gadgets.

Basic considerations

For a simulation to be useful it has to fulfill the following requirements:

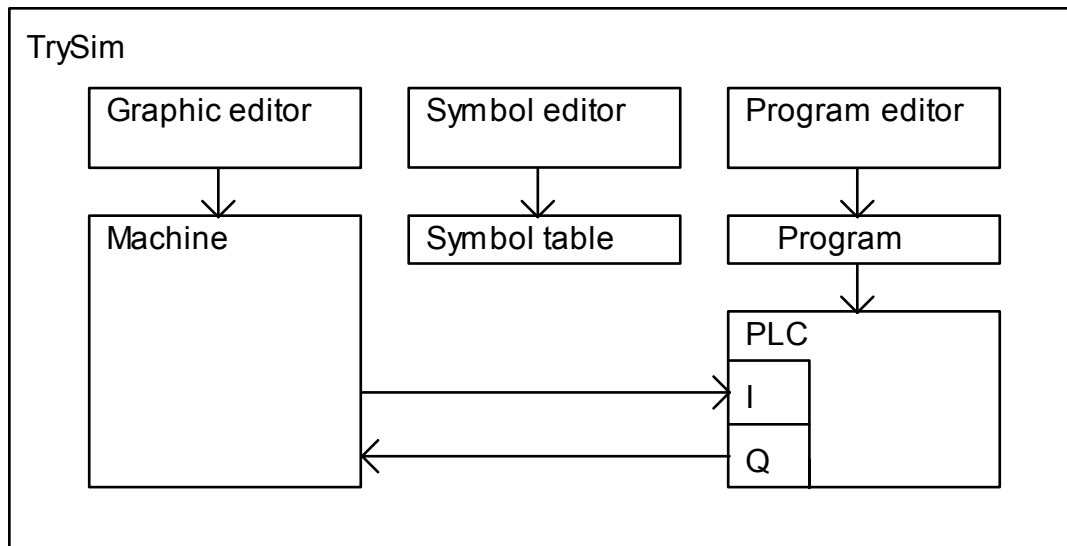
1. Creating and generating the simulation has to be easy and speedy. It is of no use saving 10 hours of commissioning time, but loosing 100 hours spent on creating the simulation.
2. It has to be realistic as far as the PLC is concerned. If, e.g., a output/motor has been set, it must be followed by an event of the corresponding limit switch at the correct time. There is no need to look realistic, as such a demand would be in contradiction to the first requirement.
3. It has to be flexible because of the immense variety of elements within mechanical engineering.
4. It requires efficient real time calculations, i.e., it must at least match the speed of the real machine, and as far as possible, should be obtainable even a higher speed.

1.2 Structure of the system

TrySim basically consists of two large functional units:

- 1.) the machine
- 2.) the PLC

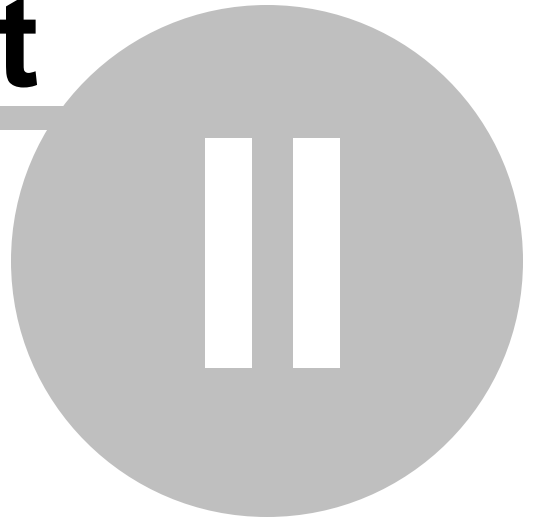
The [graphic machine editor](#) is used to construct the machine, whereas the writing of the corresponding PLC program is done with the help of the [program editor](#). Inputs and outputs of the PLC are used to build the connection between the machine and PLC. There is also a symbol editor included. For better readability, you can name PLC inputs and outputs with the [symbol table](#).



Once you start the simulation, the machine and the program run in a cyclic manner. A cycle contains the following steps:

- 1.) After reading the status of the allocated outputs, all the actuators of the machine change their position accordingly.
- 2.) After controlling the activation of the machine, its sensors set the corresponding PLC inputs.
- 3.) The [program](#) is executed once with current input data whereas the outputs are set according to your programming.

Part



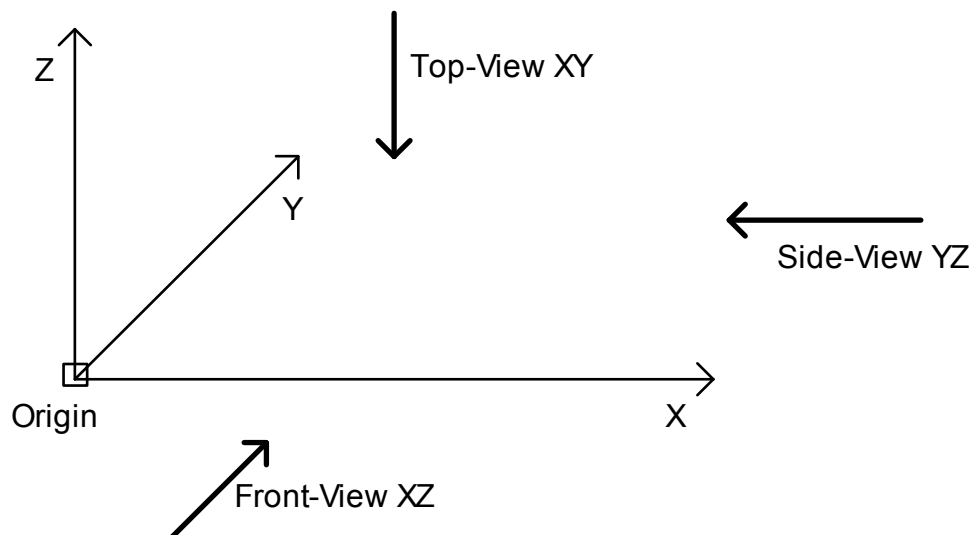
2 Machine

2.1 System of coordinates

The three space coordinates are named as x, y and z. The x- and y- direction are lying on the floor, the z-direction points to the top. You can let display the machine out of all three space directions, even out of several directions at the same time. Doing this the zero point is always at the bottom left. Experienced computer user have to get used to it because normally the zero point of the screen is at the top left. An upholding of this convention would have caused that the z-values increase from the top to the bottom and you would have to get used to it as well.

The unit of measurement in that the position of the elements is specified you can [select](#) between 0.1 mm, mm, cm, and m. Normally mm are specified.

The positions of the elements are always specified corresponding to the zero point.





2.1.1 Origin

This is an element that you will never get to see. It is existing in each machine automatically and serves as fixing point for all elements you add to the machine. The origin has got the coordinates (0,0,0). In the graphic windows it is always at the bottom left.

2.2 Editing the machine

2.2.1 Start up

Have a look at the examples: Project | open

Start / stop the simulation with the buttons  and 

[Edit elements](#) by right mouse click (when simulation is stopped)

[Watch](#): activate a PLC window and click on  (eye).

Carry out the example session! It is described in the manual (chapter 8).

Turn and shift the machine in the 3-D view with the mouse, in the online help all [navigation possibilities](#) are described.

To find a position in PLC program: click on the element with the right mouse button, click on **CR** (cross reference) then activate desired position by double click

To find additional use of an operand in PLC program: mark operand in PLC program, click double on the status bar -> the [cross reference](#) appears, activate desired position by double click.


Make use of the element tree: [Machine|Element tree](#)

Please use also the context sensitive help with the key 

When a problem occurs you can call our hotline +49 (0)4961 / 91 63 93

or send us a mail to cephalos@t-online.de.

2.2.2 Create new elements

Select **machine|new element**. (Symbol:  while graphic window is active). A selection window appears in which all elements are classified in tabsheets. After selecting the corresponding tabsheet,

drag the desired element onto a graphic window. If you want to [fix](#) the element to another one you will have to release the mouse button when the cursor is so near to the [father](#)-to-be that it is marked. After this you can shift the element to the final position.

Then you click the new element with the right mouse button and [edit its properties](#).

In the selection window a context menu is available which enables you to deselect either the names or the symbols of the elements. If you are a little familiar with TrySim the selection window can be made smaller and need not to cover the machine window.

When creating new elements the [cross](#) with which you can determine the value of the invisible coordinate is useful.

You cannot extend the list of available elements yourself. If you need an element that is not available and cannot be simulated by a combination of other elements, please contact us. You reach us under the above mentioned number or via e-mail to info@cephalos.de.

[Dynamics](#) (that is the material that is processed by the machine) are created by a [generator](#) during runtime.

2.2.3 Marking elements

To mark an element, you have to click it with the left mouse button while simulation is not running. If you want to mark several elements keep the Shift-key or the Ctrl-key pressed. You can also mark several elements by drawing a frame around the elements with the mouse.

You can shift several marked elements with the mouse.

Linear elements will be difficult to mark if they are adjusted exactly along the viewing direction. The only way out is to encircle it with the mouse or select in the menu **view|dimension** the wanted viewing direction.

If there are any difficulties to mark certain elements which are in a big pile of other elements change to another view, there it might be easier. The use of the [editing tool stripes](#) is helpful to fade out disturbing elements. If you do not succeed this way you can still reach the edit mask of the element by [machine|element tree](#) or by the [address table](#).

Try **view|zoom heavy** as well, the point that is clicked afterwards will be automatically centered in the window.

If you need to mark the same elements quite often in a huge machine you will be able to define them as members of a [group](#).

While constructing a machine it will be easier in many cases to mark elements if you set the [cross](#) before adding new elements, so that the elements in all directions are already created near its final position.

You cannot mark elements in the 3-D view.

In the window '[QuickSelected](#)' all elements, marked at the moment, are listed.

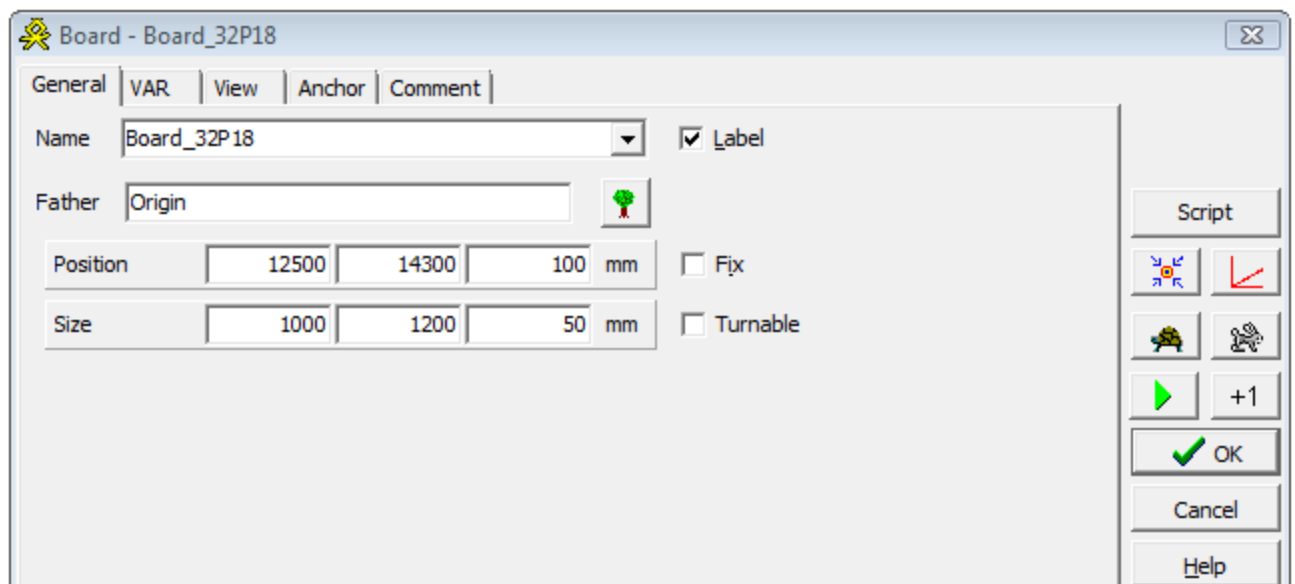
2.2.4 Edit properties

Each element is provided with specific properties which can be changed in the edit mode. Doing this you have to:

- 1.) click the element that shall be edited with the right mouse button or
- 2.) call up the whole list of all elements in the menu [machine|element tree](#), select one element and
- 3.) click the button '**edit**' (double click is possible as well) or ensure that the right element is selected in the graphic editor and click the '**enter key**'.

You can also edit the properties of several marked elements at the same time. But then only the edit fields are available which exist at all marked elements.

Example for an edit mask:



2.2.4.1 OK button

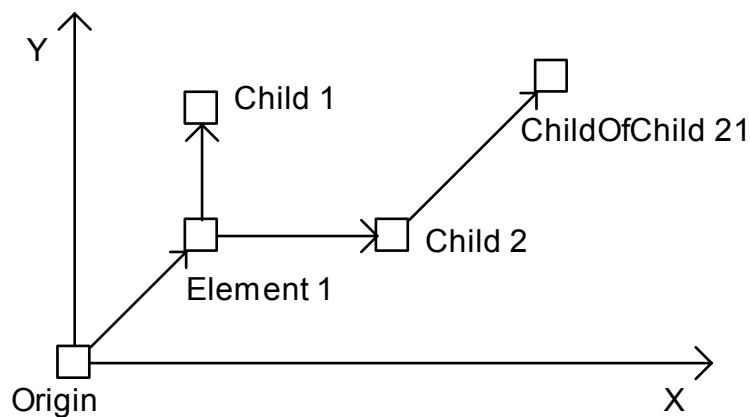
Pressing this button accepts changes and closes the property window.

2.2.4.2 Cancel button

Pressing this button rejects changes and closes the property window.

2.2.5 Fixing elements

If you build a real machine you will have to fix each part somewhere. Likewise each part in TrySim has to be fixed by specifying its father. Normally each element is fixed to the origin, it does not move then. If you want to make an element movable you will have to fix it to a moving element (e.g. [linear mover](#)). By this procedure a tree structure is build up that can be displayed in the [element tree](#). The definition of a father is essential for **structuring** huge machines.



In the picture the 'origin' is the father of 'element 1' which is the father of 'child 1' and 'child 2' again. 'Child 2' in its turn is the father of 'childofchild 21'. Suppose, e.g., 'child 2' is a linear mover, then 'childofchild 21' is able to move in relation to the origin.

You can determine for each element where at the father it shall be fixed (anchor). This is only relevant if a new machine shall be built based on an already existing one or with the help of the library. A lot of work can be avoided by the definition of the anchor if thereby the size of the elements from the template is changed. Details to this under [anchor](#).

By the help of the [sticker](#) you modify the fixation point of elements or fix elements to the material that the machine processes temporarily even during simulation runtime.

2.2.6 Edit window "Static element"

You see this edit window if you have marked several different elements and then selected 'edit'.

When more than one element is marked you can only edit these properties that exist for all marked elements and have the same function. For example, turnable and not turnable elements can not be edited at the same time, because the position of the turnable ones indicates the middle while the position of the normal ones indicates the lower left corner.

If you want to change the speed of several conveyors for example you can only choose conveyors. For easier selection there are [groups](#).

This edit mask also appears if you have marked several segments of a [chain](#). Mark just one segment and try again.

2.2.7 Delete elements

There are three ways to delete an element :

- 1.) You select it in the graphic editor with the mouse and press **del**.
- 2.) You click the element with the right mouse button as long as the context menu appears and select the item 'delete'.
- 3.) You call up the menu item [machine|element tree](#), select the desired element out of the list and press the button **delete**.

You can also delete [dynamics](#) while runtime with a [destroyer](#).

2.2.8 Select another view

Click the symbol  or select the desired viewing direction in the menu item **view|dimension**.

You will make yourself the work easier if you only change the viewing direction at a few windows. Make use of the fact that you can assign a fixed viewing direction to each of the [namable graphic-windows](#) and that you can call it up easily with the short-keys Alt+ 1..9. In **graphic|properties** you can also freeze the viewing direction of a window so that it cannot be changed by mistake anymore.

You will not be able to change the viewing direction of a window if you have fixed one or both scroll bars.

2.2.9 Color depth

If you select the 3-D representation as glass objects, the color depth should be at least 16-bit (high color).

How to modify the color depth:

- 1.) Minimize all windows
- 2.) Click anywhere on the desktop with right.
- 3.) Select **properties** out of the context menu
- 4.) Select the tabsheet **adjustments**
- 5.) Now you can adjust the desired color depth in the field **color**. The possibilities of adjustment depend on your graphic card, your monitor and the current resolution.
- 6.) In most cases Windows has to be restarted so that the new adjustments take effect.

2.2.10 Resolution

Working with TrySim you should adjust the monitor resolution as high as possible so that you can really view machine and program at the same time. We recommend at least 1280x1024.

How to change the resolution :

- 1.) Minimize all windows
- 2.) Click anywhere on the desktop with the right mouse button.
- 3.) Select **properties** out of the context menu.
- 4.) Select the tabsheet **adjustments**.
- 5.) Now you can select the desired resolution with the slide controller **screen area**. The possibilities of adjustment depend on your graphic card and your monitor.
- 6.) In most cases Windows has to be restarted so that the new adjustments take effect.

2.2.11 3-D view

For better illustration the machine can be displayed real 3-dimensional. Select **graphic|3-D view**.

You can adjust detail and viewing direction by the [keyboard](#) or by the [mouse](#).

If the machine is totally out of view you will be able to recover a standard point of view by **view|default view**.

To avoid frequent navigation between several interesting views you will be able to [store and load viewing angle](#).

By **view|lines/filled** you can select whether the elements shall be displayed as wire models or as glass objects. If you select the display as glass objects the [color depth](#) shall be at least 16-bit (high color).

During simulation runtime you should not make the 3-D window too big because otherwise the speed of execution is distinctly decreased. The line display is much faster than the filled one.

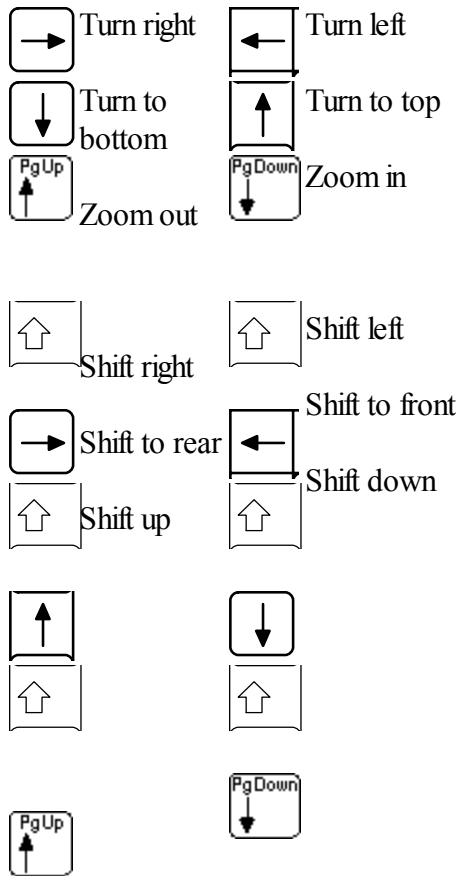
On the [edit masks](#) of the elements you will be able to adjust if a certain element is to be displayed in

the 3-D view.

There are 2 possibilities to print out the 3-D-view:

- 1.) make a screenshot
- 2.) if the 3-D view is active: [Project|print](#)

2.2.11.1 3-D control with keyboard



2.2.11.2 3-D control with mouse


If **no** mouse button is pressed:


Mouse wheel : Zoom out / zoom in
forward /
backward

If the **left** mouse button is pressed:

Mouse to right / : Turn right / left
left
Mouse up / down : Turn to top / bottom

If the **right** mouse button is pressed:
Mouse to right / : Shift to right / left
left
Mouse up / down : Shift to rear / front
Mouse wheel : Shift to bottom / top
forward /
backward

If the  - key and the **left** mouse button are pressed:
Mouse up / down : Zoom out / Zoom in

If the  - key and the **right** mouse button are pressed:
Mouse up / down : Shift up / down

By **view/default position** the machine comes in view again.

To head for a point in the machine that is interesting for you in an optimum way you act like this:

- look directly from above on the machine by moving the mouse toward you with pressed left key
- move the element with the right mouse button into the middle (where the small square flashes from time to time)
- look horizontally over the machine by pushing away the mouse with pressed left key
- move the machine with right mouse button pushed and scrolling wheel up/down until the element of your interest is in the middle
- now you find the best view by zoom and turns

You can shift the element marked in a 2-D view quickly into the middle by selecting 'focus', symbol:



2.2.12 Graphic filter

For each element it can be specified in the tabsheet 'view' on the edit mask in which of the 16 namable graphic windows it shall be displayed. The windows can be configured in **graphic|properties** so that they only take elements of one special kind right from the beginning.

Because of that it is possible, for example, to name a window 'desk' and to display only control elements on it or to reserve a window per floor in a multistorey machine.

With the button ‘opened on’ the element will be displayed in all windows that are opened at this moment.

You can also edit the filter for several marked elements at the same time. The check boxes of the windows that shall display some of the elements and some not will be displayed grey then.

2.2.13 Properties of the 3-D window

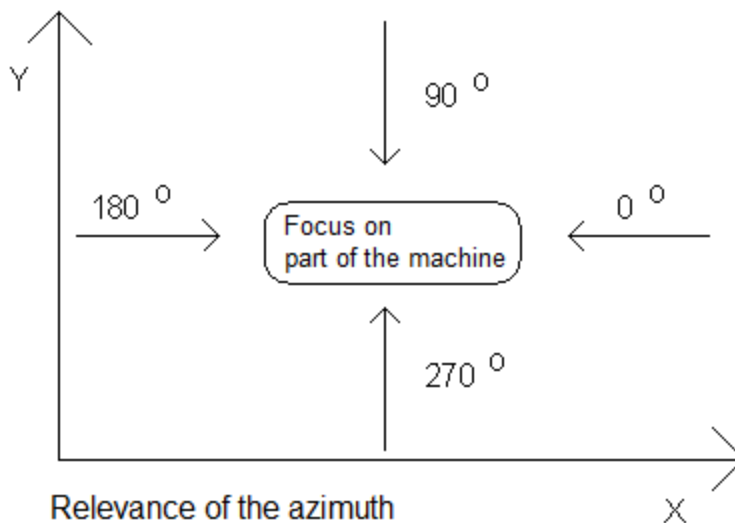
On this mask the position of the 3-D camera is described. The description happens in two steps and is not that easy to understand. Only if you cannot adjust the desired camera position with the [mouse](#) (this will sometimes be quite difficult if you have got a huge machine) you should concern yourself with this description. Try the menu item **view|default position**. The [attractor](#) gives the simple possibility to display a certain part of the machine in the 3-D window.

Focus: With this you specify whereat the camera shall look. An attractor does not do anything else than to set this point to its own position.

Distance and position of the camera: If you have set the focus correctly you will have to specify the distance of the camera to this point. After that the machine part that is interesting for you should at least be visible in the 3-D window. The adjustment of the direction out of which the machine part shall be viewed will than be adjusted easiest with the mouse (keep left key pressed).

The position of the camera will be specified by two angles (which are easier to specify with the mouse like already said) next to the distance of the focus:

- 1.) Elongation: With this it is specified how far the camera looks from the top. If the elongation is 0 degree the camera looks exactly horizontally onto the focus. If the elongation was 90 degree (but it can be adjusted maximally 80 degree) the camera would look exactly from the top onto the focus. This value can be changed with the mouse by moving to the top or bottom with pressed left key.
- 2.) Azimuth: With this it is specified whether the camera shall look from north, west, south or east onto the focus. If the camera looks against the x-axis the elongation will be 0. If it looks against the y-axis the elongation will be 90 degree and so on: if the camera looks into the direction of the x-axis the elongation will be 180 degree and if it looks into the direction of the y-axis the elongation will be 270 degree. The following picture illustrates this assignment.



This value can be changed with the mouse by moving right or left with pressed left key.

2.2.14 Store/load a 3-D viewing angle

Storing an interesting 3-D viewing angle you select the menu item **view|3-D viewing angle** while the 3-D window is active. Click the button 'new' immediately. Then you will be asked to enter a name for the viewing angle. The button will not be selectable anymore if you have already selected a viewing angle out of the list.

Recalling a viewing angle you select it out of the list with a double click or you mark it and click 'ok'. You can also browse the list by the arrow keys, the 3-D view is updated then immediately. If you have found the desired viewing angle you will have to confirm with 'return'.

It is not possible to change a viewing angle that is already stored. The size of the window and the position is not stored.

You can also activate the stored viewing angles by the PLC. To do this load the number of the entry that you want to activate into accu 1 and call up the FUNC 110. If you want to store new viewing angles or delete them you will have to adjust the numbers in the PLC-program. You should not call up the FUNC 110 cyclically but only if a really new viewing angle shall be displayed because it is very time-consuming.


The FUNC 111 works just the same but in addition the time (in 1/10 sec) that shall be used to change from one view to the next one has to be displayed in accu 2. This display relates to the real time and not to the virtual simulation time.

You can also specify changing or moving viewing angles by the [attractors](#).

2.2.15 Zoom

This function is not available in the 3-D view.

To do so click the symbols  and  or select the desired operation out of the menu 'view'.

The more accurate the area that is interesting for you is in the middle of the window while zooming the more often you can click  without the area leaving the window.

The menu item **view|zoom heavy** is very useful to zoom in a special detail quickly. After selecting the menu item the mouse pointer becomes a magnifying glass with which you click the area that shall be enlarged.

With **view|make very small** you undo this enlargement.

You will not be able to change the zoom anymore if you have [fixed](#) one or both scrollbars.

2.2.16 Address table

Machine|Address table

In the address table all PLC-addresses that are used by the machine are united. Please do not mix up the address table with the [symbol table](#), with which you can assign symbols to the PLC-addresses.

You can change the element names and the addresses. In [extras|options|PLC-editor|rewire](#) it is adjustable whether the addresses in the PLC-program shall be changed automatically.

The table can be sorted according to elements, addresses and element-types. Doing this click the corresponding column heading. If you click left it will be sorted automatically in ascending order, if you click right you will be able to decide whether the order shall be ascending or descending.

Out of the address table you will be able to reach the edit mask of an element directly if you select 'edit' out of the [context menu](#).

If you want to check the use of an operand in the PLC you will have to click right and select 'cross reference' out of the context menu. The [cross reference list](#) will then be opened at the right position and from there you will get to the desired program position with a double click.

If you want to edit the symbol or the comment to the operand you will have to click right and select 'symbol table' out of the context menu.

You reach the address table by the little button 'addr' on each [interface panel](#) as well.

You print out the active table with **project|print out**.

In the column for the operand 'E', 'A' and 'M' as well as the point can also be entered above the [number block](#).

2.2.17 Element tree

You reach this display of all elements that are existing in the machine by **machine|element tree** or by the tree symbol on each edit mask. The element tree is very useful for working with huge machines and you should use it quite often.

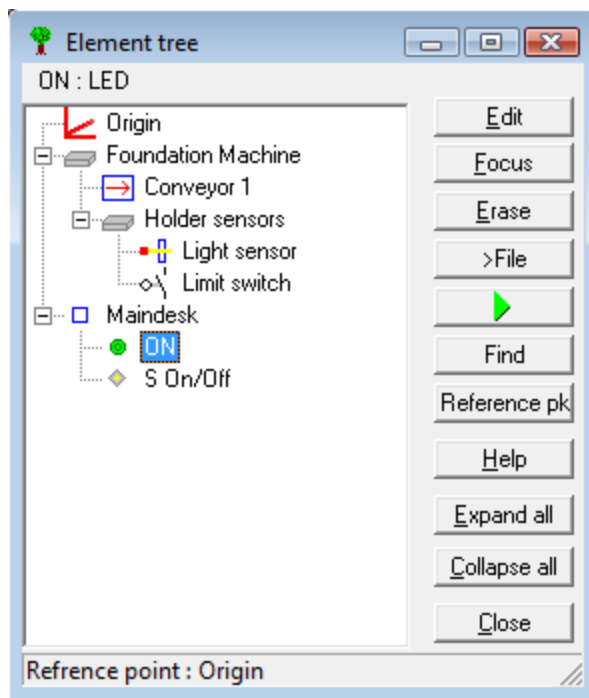
In the element tree the father-child-relations are displayed. So you are able to recognize quickly to which a certain element is [fixed](#). If you drag an element with the left mouse button and drop it on another one it will be fixed to that one; that means it will get a new father.

In the element tree elements can be renamed as well. Doing this click the name of an already marked element or use the function key F2.

With the button 'focus' the current graphic window is shifted so that the marked element is displayed in the middle. This function is also available in **process|display in middle of picture** and it also works for the 3-D graphic.

The element tree cannot be printed out (but see: [screen shot](#)), but it can be saved as a file ">file".

With the button '[>reference point](#)' you can place the beginning of the coordinate system on the current element. This one has then the position (0,0,0) and all the other position are referred to it. This saves a lot of mental arithmetic in case of huge machines. By clicking on the origin it will be the reference point again.



Remark: It is not recognizable in the element tree that all elements have got the origin as (grand-) father. This is not a mistake but it is made by us because of the saving of space. If we had displayed the origin correctly a whole tree plane would have been wasted, what is not necessary having always such a little screen.

2.2.18 Groups

Groups are useful for managing bigger machines. You can unite any selection of elements to a named group. All elements of a group can be marked quickly and/or edited. The membership of a group has got no functional meaning - a group just consists of a listing of all elements that are member, nothing else.

Typical applications are:

- 1.) Creating a multistorey machine the upper storeys cover the lower ones. For each storey a group is created, then it is easy to switch invisible all storeys excepting the one that is worked on at the moment. For this purpose [stripes](#) or [graphic filters](#) are useful as well.
- 2.) The minimum necessary speed shall be determined in a machine with lots of similar drives. All drives are united to a group, then a little different speeds can be adjusted to ascertain the effect on the throughput.

For many ranges of use of groups like 2.) it is necessary that the members are as similar as possible. One single LED in a group of conveyors makes the editing of their drives impossible because it is not

provided with such one.

The management of the groups happens in two steps:

- 1.) In the [group manager](#) groups can be created and deleted. Here are also the control elements to select all elements of a group at the same time and/or to edit.
- 2.) In the [group editor](#) it is defined which elements a group shall contain.

Groups cannot be printed out.

2.2.18.1 Group manager

You reach the group manager by **machine|groups**.

Elements | Select:

Select a group out of the list. If you click the button 'select' all elements of the group will be marked in the graphic windows and the group manager is closed.

Elements | Edit:

Select a group out of the list. If you click the button 'edit' an element edit mask will be called up on which only such properties can be edited that are existing in all elements of the group. The more similar the elements of a group are the more properties can be changed by you. If you want to change the speed of drives, for example, only elements must exist in the corresponding group that actually have got a drive.

Group | New:

After entering the name of the new group the [group editor](#) is called up. If elements were marked in the graphic windows these will be added automatically to the new group.

Group | Edit:

With this the [group editor](#) is called up to add elements to a group or to take them off.

Group | Erase:

The group is deleted without further inquiry. The elements that are in the group are not deleted naturally.

2.2.18.2 Group Editor

You reach the group editor indirectly by **machine|groups | click on edit under the heading group**

Button ">" :

The element that is marked in the element tree on the left side is added to the group.
Corresponding keyboard control: ENTER

Button “>>“ :

The element that is marked in the element tree on the left side and all its children and children's children are added to the group. If the origin is marked **all** elements apart from the origin itself will be added to the group.

Corresponding keyboard control: SHIFT ENTER

Button “Selected >“ :

All elements that are marked in the graphic windows are added to the group.

Button “<<“ :

All elements that are marked on the right resp. left side are taken off the group. On the right side a multiple selection is possible as well.

Corresponding keyboard control: DEL

Button “<<<“ :

All elements that are marked on the right resp. left side are taken off the group with their children. On the right side a multiple selection is possible as well.

Corresponding keyboard control: SHIFT DEL

Further functions :

- A context menu is available in the left window as well as in the right one.
- Elements can be edited, on the left side only by the context menu, on the right side also with ENTER or by double clicking.

You can only add semi dynamics to a group by marking them in a graphic window before you call up the group editor and then you have to click the button ‘selected >>’ because they are not listed in the element tree.

2.2.19 Positioning elements

Positioning elements with the mouse is the simplest way. In larger machines this is sufficient only for rough positioning. In positioning the mouse please mind the [raster](#).

Indicating the coordinates on the edit mask is more exact.

Position	<input type="text" value="2500"/>	<input type="text" value="7510"/>	<input type="text" value="1926"/>	mm
Size	<input type="text" value="188"/>	<input type="text" value="94"/>	<input type="text" value="283"/>	mm

It is useful that the [reference point](#) can be changed. 

You can move an element about an defined amount in entering the displacement path with a prefixed + into a positioning field:

Position	+25
----------	-----

Attention!

If you want to move about a negative value you have to enter e.g. '+-25'. The reason for this is that a negative number can stand for an absolute value and is interpreted in TrySim like that. Especially concerning a variable reference point this is a frequent occurrence. Only because of the + sign (that is normally not entered) TrySim recognizes that you meant a shift:

Position	+ -25
----------	-------

Moving several elements

Before you move several elements you should check if it isn't more advantageous to fix them to a common [father](#). Then you only have to move this one. With the growth of the machine you can't avoid this way of [structuring](#) anyway.

2.2.20 Reference point

Each element in TrySim has got a position, identified by three coordinates (X,Y,Z). Internal these values are saved and processed in micrometers, the edit mask often just shows mm.

Internal the position is always measured with the distance to the 'origin'. This element that you can only see in the [element tree](#) has got per definition the coordinates:

X = 0.000 mm
Y = 0.000 mm
Z = 0.000 mm

The numbers quickly turn confusing for a human user in case of huge machines. Who on earth can see at first glance that the distance of two boxes with the positions X=650021 and X=649001 is quite exact 1 m?



Therefore there is the possibility in TrySim to determine a reference point ever-changing. This definition has nothing to do with the internal storage and has no influence on the simulation. All displayed data is converted so that you can read it more easily.

If you choose the box_1 as reference point in the example above then it has got the X position '0'. This is easy to remember.

For the box_2 then the X position '1020' is shown, about 1 m.





Sometimes the end of the element is more interesting than the beginning. For that reason, you can

switch from 'bottom left' to 'top right' with a symbol in the working rail. This is very useful if you have to solve a positioning problem like: 'This light barrier is 50 mm before the end of the conveyor'.

2.2.21 Simulation of the machine


The machine is constructed by the graphic editor. You select the needed elements out of a list, place them in the graphic window and edit their properties.

Then you start the simulation with the symbol . While the simulation is running (green light on the bottom left) you cannot edit. The stopping of the machine with the symbol  is equivalent to the switching to the edit mode. You can stop a simulation run for editing as many times as you like. After restarting the simulation will be resumed at the point where it is interrupted.

2.2.22 How to find Elements

With increasing size of the machine it becomes more and more difficult to find a specific element quickly. That is why in TrySim there is a vast number of mechanisms to mark elements, to call up their edit mask or to check their PLC-connection. This may confuse you in the beginning but after a while you will appreciate that you can identify an element even with incomplete information.

If you only know where the element you are looking for is used in the PLC-program, you will have to click the operand with the right mouse button and select [address table](#). You reach the edit mask of the element directly by the context menu. Please note that the marks will change in the graphic windows if you change the line in the table with the mouse or the keyboard. Please note as well that you can sort the table by element name, address and element type.

If you only know the rough position of the element you will have to enlarge this area by **view|zoom heavy**. If the element cannot be identified yet because it is covered by many other elements use the [stripe](#) to fade out all the other areas. Keep also in mind that the display of the searched element could be hidden with a  [graphic filter](#).

If you only know where an element is fixed to or which other element is fixed to it you will have to select the [element tree](#). Using elements without PLC-connection which are made invisible in all directions in addition and which are not in a group, this is the only possibility to get in contact again.

If you only know the [group](#) the element belongs to you will have to open the group editor and click on the element. Please note that the element will be marked in the graphic window then and if the element tree is open even there. By the context menu you get to the edit mask of the element directly.

Elements, that move a lot in the machine, but that have to be viewed with heavy zoom, can be provided with an [attractor](#). So you can focus very quickly to the appropriate area. Because the attractors can be activated and deactivated, e.g. by a [multiple switch](#), you can switch over to different attractors.

2.2.23 Structuring huge Machines

If you create a huge machine it will be indispensable that you structure them well, otherwise the work will become unbearable. The principle of structuring in TrySim is the same that you use for the management of data on the hard disk of your computer: matching data is kept in a folder and if there is too many in it, sub-folders will be created if necessary.

There are no folders in TrySim, but [boxes](#) are used instead. These are like a foundation to that all elements of one functional group are fixed. In the edit mask of the box you can select 'foundation', it will then be colored light grey (this can be changed later again) and also appears separately in the element tree.

If too many elements have to be fixed to a foundation it will be advisable, especially if some of them form independent functional groups, to create a new light grey box again for these elements to which they are fixed.

With your first machines you will realize the necessity of configuration only afterwards. In this case act like this: Create a box for the selected functional group, select 'foundation' in the edit mask (thereby it is colored light grey) and position it at the logical reference point for the functional group (just to think about it, where this one is, leads often to valuable help at later starting up). If the functional group represents a stackable construct, for example segments of conveyor, it will be quite useful to make the foundation as long as the stacking length - then, other segments that are created either by 'copy and paste' or out of the [library](#) can already be roughly positioned by the mouse. The fine positioning will then be done by the input of coordinates on the edit mask of the foundation. In other cases it is better to make the foundation very small. It is really worth thinking about position and size of the 'foundation'. For foundations that do not only have the name symbolically it is, for example, absolutely recommendable to set the z-coordinate always to '0'.

After having created and configured the 'foundation' fix the matching elements to it. If your (unstructured) machine is not too big yet, this is easiest done in the element tree: just drag the element that is to be fixed with the left mouse key and drop it on the new foundation. You will save work if you unite the smaller functional groups first so that these can be assigned to the next higher one as a whole. If your machine is already so big that the element 'tree' (in this case rather a pole) does not fit a screen anymore (the TrySim tree does not scroll automatically yet) it will be easier to mark the matching elements in a graphic window and to [change their father](#) then. Creating huge machines please look at the element tree occasionally and act as long as it is still possible with less effort!

You will get to know the advantages of a well structured machine at the latest if you [search for an element](#) or if you want to configure a graphic window so that it displays exactly what you want to see at the moment. For the last one especially the functions that are existing in the [menu edit](#) are quite useful. For example, the function ‘fade out children’, applied to a ‘foundation’, lets disappear all elements belonging to it. But the foundation itself stays visible and so it can be marked quite easily and with ‘fade in with children’ the function group will become visible as a whole again.

The following subjects will be interesting as well, if it concerns the editing of huge machines:

[Groups](#)

[Namable graphic windows](#)

[Graphic filter](#)

[Focus window on a moving part](#)

[Reference point](#)

2.3 Common properties of elements

2.3.1 General

Nearly all elements have got following properties:

[Name](#)

[Father](#)

[Position](#)

[Size](#)

[Fixability](#)

[Visibility](#)

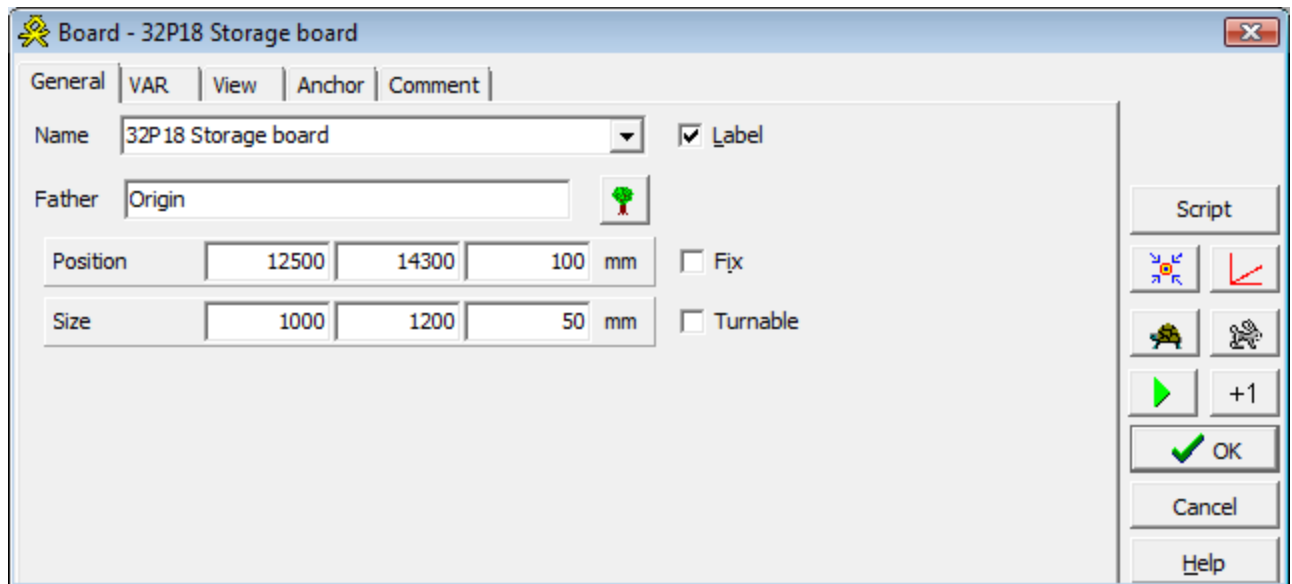
[Color](#)

[PLC-connection](#)

[Comment](#)

[Fastening point \(anchor\)](#)

[Script](#)



2.3.2 Name

Each element has got a name. The names do not have to be unique. At first a new element gets a name from the system that is made up by the element type and the ID. The name of the element is written in the graphic next to the element. You can change the position of the name by clicking the name with the left mouse button and keeping the button pressed. Then you drag the name and drop it at the desired position.

In machines with many elements names are often more disturbing than useful. Consequently you can suppress the display of a name by deselecting the check box 'label' in the edit mask.

The name of the element will then appear in the hint, that will appear if you point at the element with the mouse in the edit mode and wait about one second. Even if you do not want the name to be displayed you shall give a clear and unique name to each element so that the element in the [element tree](#), in the [address table](#) or at forming of [groups](#) can be made out easily.

Push buttons and light emitting indicators do also have got a name². This name (if you did enter one) is shown in the graphic window. However, the element tree and the address table still display the regular name. This proved to be useful for creating desks on which a plain text should be seen, while the actual name can be occupied with the device tag.

2.3.3 Father

All elements except the origin have got a father. The father is any other element, normally the origin. If the father is moved during editing or runtime the element will be moved as well. If the father is

deleted the element will also be deleted.

By assigning several elements to a common father you can structure the machine which makes the editing easier, especially if you have got a huge number of elements. It is essential to look up in the element tree often and to tidy up in case of disorder.

You can specify the father of an element as follows:

- 1.) While creating the element you specify it by putting the new element on the father (position the mouse cursor so that the future father is marked).
- 2.) You call up [machine|element tree](#), drag the element and drop it on the new father. This method is very useful to sort things out in a machine in which too many elements still have the origin as father.
- 3.) Click on the element with the right mouse button a little longer. Select 'change father' out of the context menu that appears then. Click on the new father with the left mouse button. Please note to click not until the word 'father' is underlined at the mouse cursor.

Since version 3.0 you can not change the father of an element by selecting one out of the edit mask. This has proved to be a difficult method for huge machines which led to operation errors.

If you have selected several elements for editing and change the father only these fathers of the elements will be changed whose old father is not in the selection. By this it is avoided that an already built up structure of father-child-connections gets destroyed.

If you create a new element and place it with the mouse so close to an already existing one so that it shows the black marking squares this one will automatically be selected as father of the new element. If you want to add new control elements, for example, to a box that is called 'desk' you will have to put down the element shortly in the near of the border of the box (only there boxes are markable) and only then shift it to its final position. So you save the additional specification of the father.

Fathers are not just there to move the children but to structure the machine logically as well. Assign all elements that belong to a machine part to one common father, if possible. This simplifies the editing and makes the finding of elements in the element tree easier.

2.3.3.1 Child

Child is an element that is moved as well at each moving of the hotspot of the father. Please note: deleting the father means deleting the children as well.

If you want to know which children an element has got it is best to look at the tree that you can reach by the menu item [machine|element tree](#).

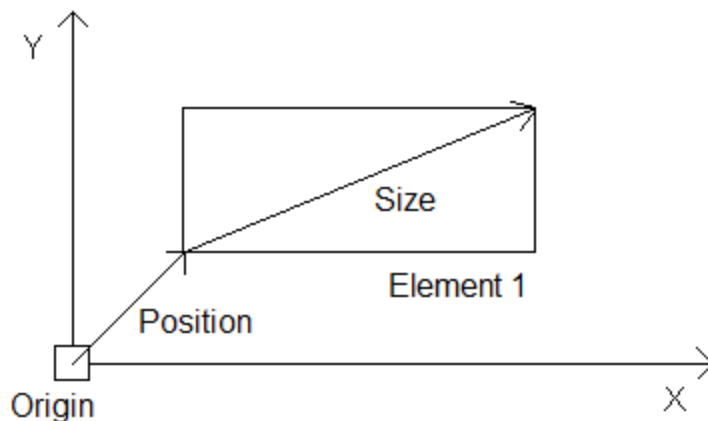
2.3.4 Position

All rectangular shaped elements have got a position (the line-shaped ones like linear mover and light barrier are an exception). The position indicates the corner that is the nearest to the origin. Normally it is indicated in absolute coordinates concerning the origin, but you can change this with the [cross](#).

You can change the position of an element in the edit mode by entering the new coordinates in the edit mask, or by dragging the element with the left mouse button and dropping it at the desired place. If the element has got children these will follow the movement.

You can also shift several already marked elements at the same time with the mouse. But then there must not be any fixed element in the selection whose father is not in the selection as well.

The position will also be changed implicitly by changing the size with the mouse if the corner that is selected for the size changing is the position corner or if it is fixed to this one directly.



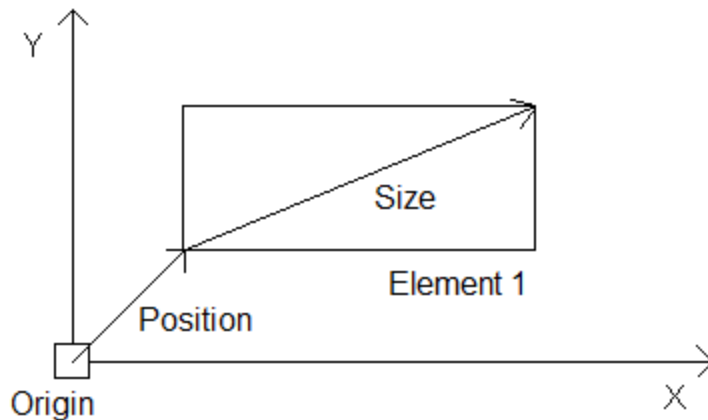
The picture above is only two-dimensional for reasons of clearness.

2.3.5 Size

All square shaped elements have got a size. The size is the vector from the corner that is specified as [position](#) to the diagonally opposite corner.

You change the size either by entering the coordinates in the edit mask or with the mouse. Changing with the mouse you have to mark the element first by clicking the left button and position the mouse cursor so near to one of the black marking squares so that the mouse cursor changes into a double arrow. After that you drag the corner and drop the element at the desired size.

Changing the size with the mouse the position of the element can change.



The picture above is only two-dimensional for reasons of clearness.

You can also change the size of static elements by the PLC-program with the help of the [POKEs](#).

2.3.6 Fixability

If you have specified the final position of an element you will be able to protect the element from being relocated by the mouse by selecting the check box 'fix'. This fixing is only valid for the immediate relocation of the element. If you relocate the (not fixed) father the element will follow naturally. You are at liberty to change the position of the element by entering new coordinates.

You should fix elements as early as possible, because otherwise elements can easily be moved by mistake. Therefore the context menu of each element contains the items 'fix children' (originally it ought to be 'fix with children and children's children') and 'unfix children'. By 'fix children' the element itself as well as all elements attached to it are fixed. By 'unfix children' only the fixing of the children directly fixed to the element is canceled. The element itself and the children's children stay fixed. We did this asymmetry intentionally and not by mistake.


The fixing does not concern the size changing with the mouse because this is hardly possible by mistake. For the same reason fixed elements can be moved via arrow keys (since version 4.0)

Please note that you can also [fix the part of a window](#) but this has got nothing to do with the fixing of single elements.

2.3.7 Visibility

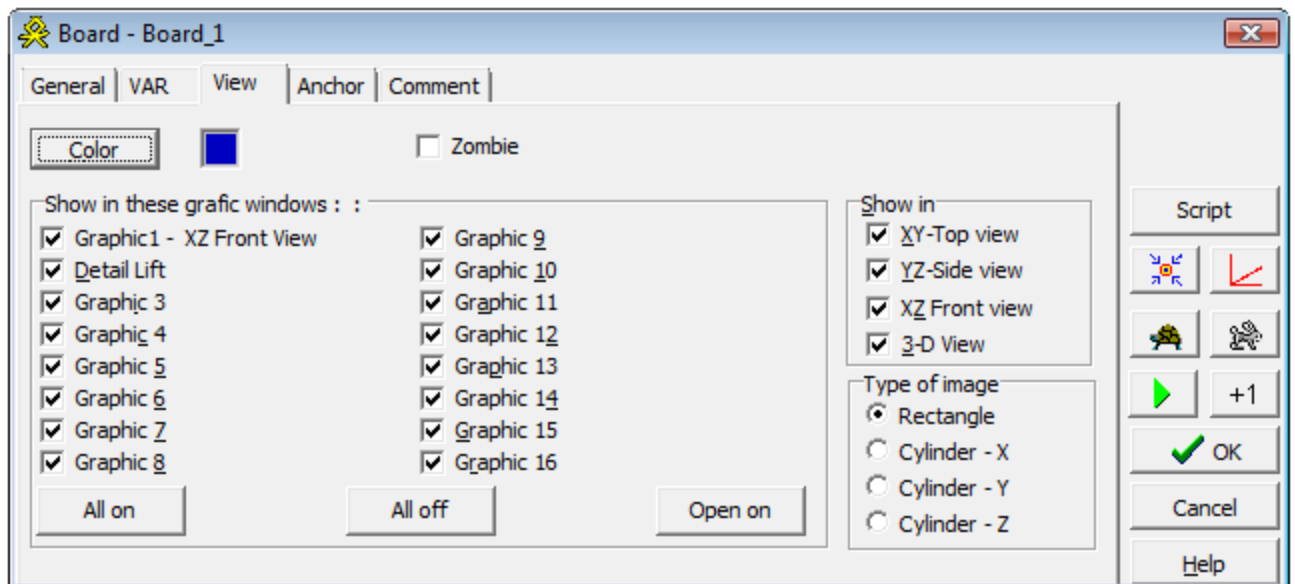
In order to display a clear three-dimensional machine you can specify for each element out of which direction it is visible. Normally the elements are visible out of all space directions.

You can inactivate the visibility by deselecting the check boxes 'display in XY, ZY, XZ and 3-D view'.

In addition you can specify for each element by the  [graphic-filter](#) in which of the 16 namable graphic windows it shall be displayed. The windows can be configured by **graphic|properties** so that they only take elements of one special kind right from the beginning.

Using the [editing tool stripe](#) you can specify temporarily while editing which of the parts of your machine you want to see.

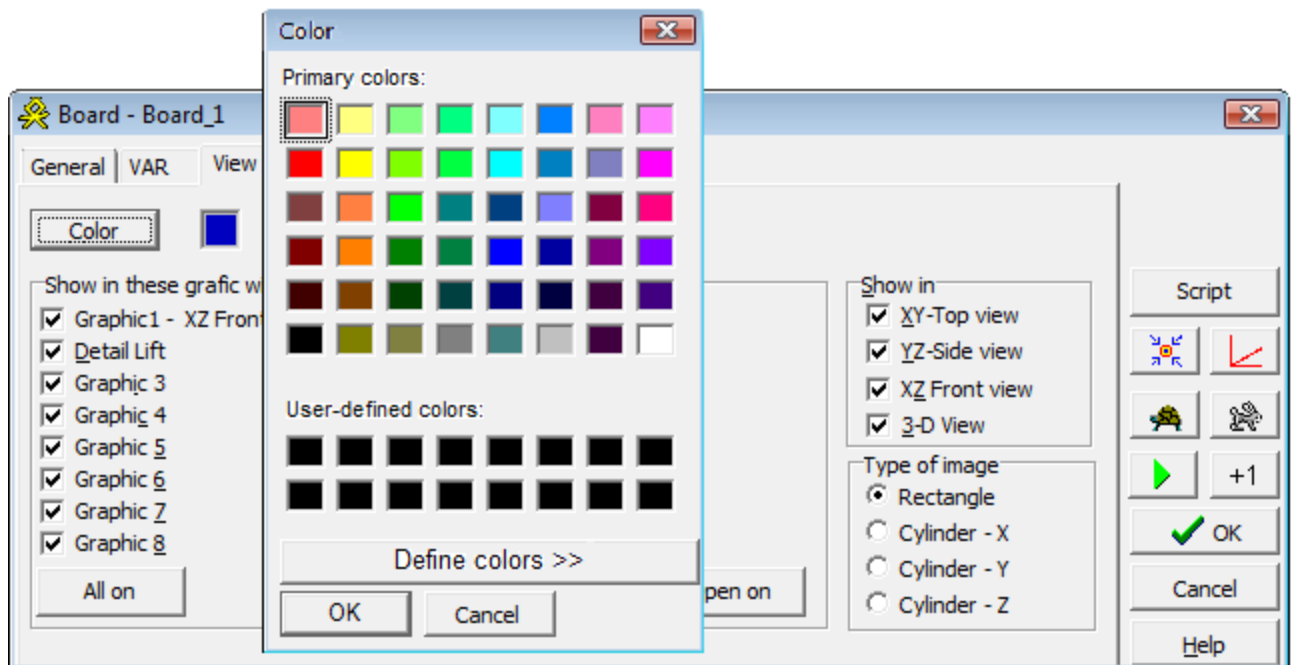
In this context please note the possibility to form named [groups](#).



2.3.8 Color

You can select the display color of each element freely. Normally each element gets a color dependent on its type. You cannot select the color 'white'.

The colors of the LEDs are restricted to the primary colors.



2.3.9 PLC Connection

*_***


Most elements of TrySim have to be connected to one bit or several bits of the PLC. Therefore an [interface panel](#) exists within the edit mask, in which you may input the operand. If you have made a symbol table you might enter the symbol as well.

You may connect all the elements to each bit of the areas inputs, outputs, markers and data bits.

Whether the bit is read or written depends only on the function of the element. Reading elements, e. g., indicators, will evaluate the state of the bit after the end of the PLC cycle. Writing elements, e. g., limit switches, will set the appropriate bit at the beginning of the PLC cycle.

If you have set up a [symbol table](#), the comment to the symbol is also displayed on the edit mask. There you can modify the comment, the modification will be taken over into the symbol table. The symbol is displayed as well, but you cannot modify it because of reasons of consistency. To do this you have to open the symbol table.

You will have all the connections from the machine to the PLC clearly displayed within the [address table](#).

If you want to check the use of the PLC connection in the PLC program, click the button  to open the [cross reference](#).

2.3.10 Comment

To each element you can enter a comment. Note all characteristics here that have to be considered during the installation or creating a user documentation.

To call up the comment click the tabsheet 'Comment' on the edit mask of the element. If you have marked several elements the comment will not be able to be edited.

The element comment can be printed out in Quick Com | Script.

2.3.11 Anchor

The adjustability of the fastening point is not essential when first built and has no influence on the simulation.

If you want to change the size of an element it can be useful to determine for the children on which point they are fixed exactly.

For example, if a light barrier is fixed to the right end of a conveyor (not to the position corner) it is cumbersome if you have to carry along the light barrier after a resizing of the conveyor.

For each of the directions in space three values can be defined:

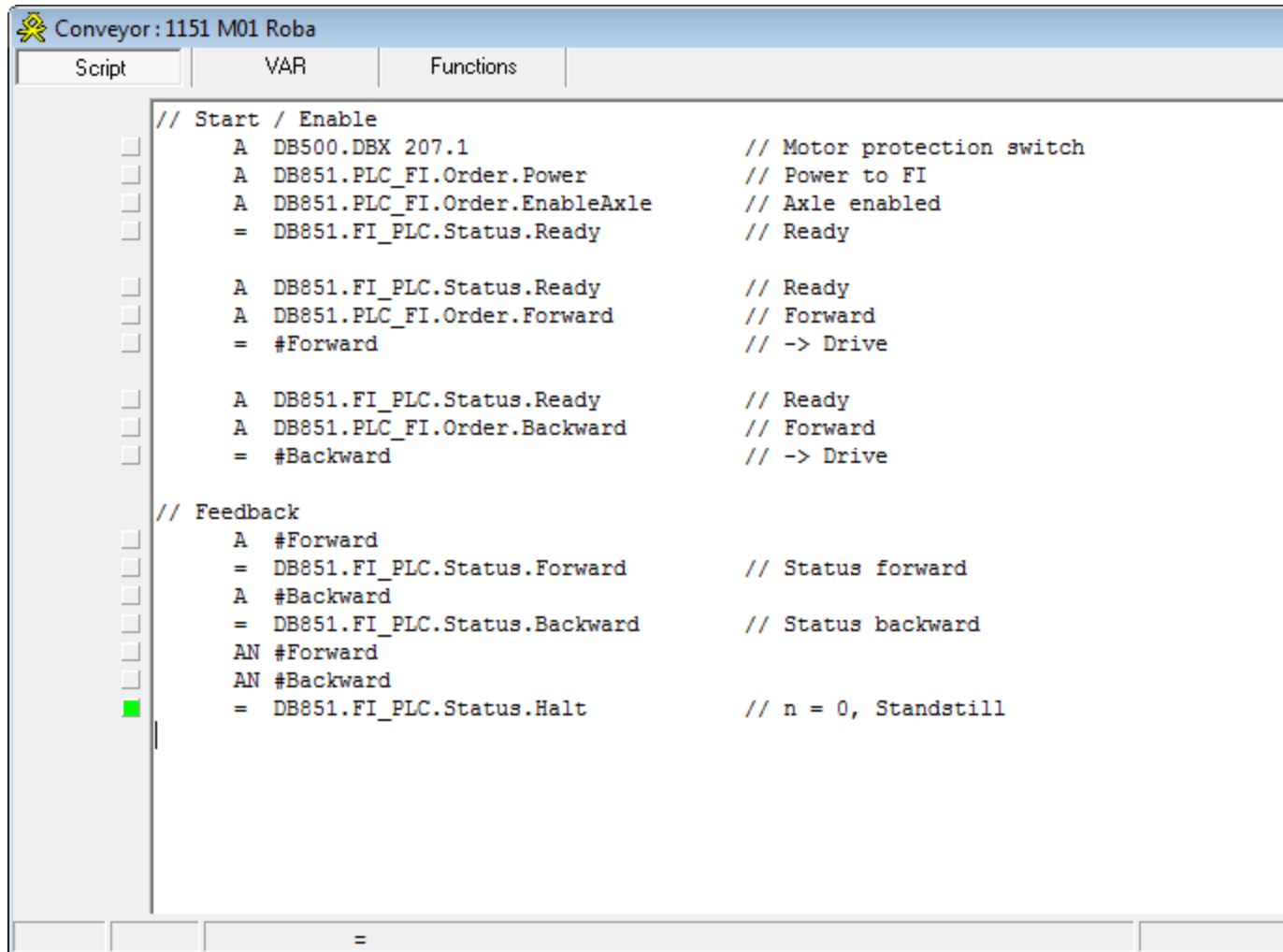
X : left (standard), middle, right

Y : front (standard) middle, behind

Z : bottom (standard) middle, top

In the main menu Extras|Quick anchor you can watch the fastening point promptly. Changes have to be confirmed with the button '!'.
!

2.3.12 Script



In the script a short piece of AWL-Code can be written for each element to modify the properties or to reproduce hardware outside the PLC. As an example we take a conveyor that can not work if a safeguarding case happened in the motor circuit-breaker.

The script would be:

```

A "Q motor" // motor circuit-breaker (input)
A "O motor" // output motor start
=#bool0

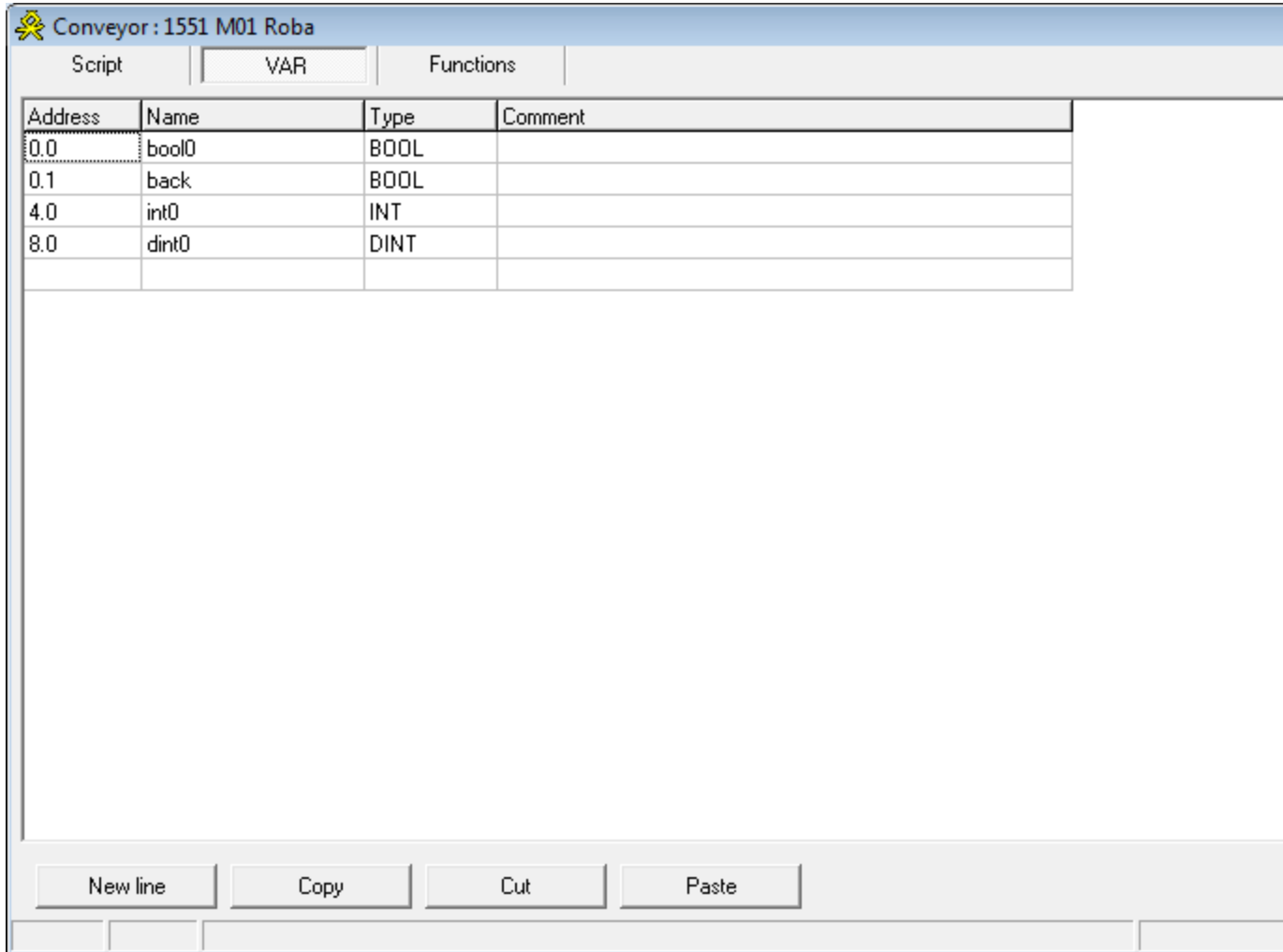
```

Now #bool0 has taken the place in the drive mask that was 'O motor' before.

The #bool0 is a bit of the instance data block that is assigned to each element individually. All static

variables of this IDB are only known inside the element and keep their value from one simulation step to the next, unless they are in an [interface panel](#) of a writing element.

The declaration of the static variables can be displayed and edited with the button **VAR**.



There you can declare new variables (and also change the name of the predefined #bool0, #int0, #dint0). The data types BOOL, BYTE, INT, WORD, DINT, DWORD and REAL are permitted. You can choose all addresses up to 20.0. The address allocation is not controlled that strictly in TrySim as in Fbs, even overlapping addresses can be entered. Overlapping addresses can be useful if e.g. you want to access Words bit by bit. However, the free address allocation requires also more attention in generating.

At the left margin the values of the operands are displayed. Please mind that not the result of logic operation (RLO) and not the accu contents are shown. You can change these by a click on the LED or the number.

You can see the advantage of these scripts especially when reuse machines for the next project. Almost everything that was placed in OB2 and OB3 is now directly in the element and is adopted

with it. Furthermore we made more transparent the things influencing the behavior of an element outside the PLC program additionally.

The script is also useful if you don't want to write a 'real' PLC program but need a controlled simulation only for demonstration purposes.

You can access the static variables of the superior element e.g. with #father.bool0 (only symbolically). This has been proved useful for communication of elements among each other, since no absolute addresses (e.g. marker or data blocks) have to be assigned anymore.

We prepare an access in a freer way (then probably via e.g. #friend.bool0). An easy access to internal data, on which you only could access with peek and poke elements so far, will also be possible with the syntax e.g. #element.position_x.

Suggestions from practice for the script are very welcome.

2.4 Static elements of simulation

2.4.1 Overview

Naming elements 'static' does not mean that they are not able to move but that they are unable to be created or deleted during simulation runtime.

In the help of the elements it is marked in the first line how difficult this element is to use. It means:

* : easy, to handle by beginners as well

** : middling

*** : difficult, only recommended for experienced users

**** : very difficult, these elements are quite often still in the test-stage.

Sensors

[Limit switch](#) [Light barrier](#) [Spy](#) [Bar code scanner](#) [Resolver](#) [Distance sensor](#)
[Thermometer](#) [Thermostat](#)

Actuators

[Conveyor](#) [Free point](#) [Generator](#) [Linear mover](#) [Chain](#) [Hook](#) [Destroyer](#) [Joint](#)
[Turner](#) [Arc belt](#) [Turnable conveyor](#) [Turntable](#) [Heating](#) [Mangle](#) [Continuous generator](#)

Controls

[LED](#) [Digital display](#) [Push button/Switch](#) [Slide controller](#) [Text display](#) [String display](#)
[Oscilloscope](#) [Multiple switch](#) [Master switch](#)

Fluids

[General](#) [Container](#) [Pipe](#) [Pump](#) [Valve](#) [Flange](#) [Level switch](#) [Level gauge](#) [Flow meter](#) [Pressure sensor](#) [Check valve](#) [Analyzer](#) [Reactor](#) [Fluidor](#)

Misc.

[Box](#) [Board](#) [Bar](#) [Shifter](#) [Melter](#) [Divider](#) [Saw](#) [Wedge](#) [Press](#) [Pawl](#)
[Automatic hook](#) [Sticker](#) [Dyn converter](#) [Straighter](#) [Thermal mass](#) [RFID antenna & data chip](#)

Utilities (also tabsheet Misc.)

[Stripes](#) [Attractor](#) [Cross](#) [Poke](#) [Peek](#) [Speed trigger](#) [Background](#)

2.4.2 Sensors

2.4.2.1 Light barrier

*

Light barriers within TrySim are only covered by [dynamics](#). As there might be a lot of these elements within the machine, the checking whether the light barriers might be covered by one of them, can cost a great deal of computing time. Therefore the light barriers are standardized in such a way that they are only sensitive in one point of the middle of the ray. If you need sensitivity along the whole length you can increase the number of the watched dots within the field 'Reality degree'.

In TrySim designations like 'limit switch' and 'light barrier' should not be taken literally. Use limit switches every time you need to detect elements of the machine and light barriers every time to detect dynamics.

Light barriers can be configured as NO or NC contacts. When a '1' has been reported to the PLC, the little square will be colored red at one end. You can also configure a light barrier as a switch, then it will change its state with every actuation. In the example 'wedge' a light barrier is used in this fashion.

The most frequent mistake when using light barriers is that during the creation the light barrier lays on the ground initially (what cannot be seen within the XY-top-view) and so the dynamics pass over it. The [editing tool cross](#) helps to avoid this problem.

To change the orientation of the light barrier with the mouse you have to point with the mouse cursor near to one end of it and wait until the mouse cursor changes to a cross.

The light barrier can also detect the track that is created by a [continuous generator](#).

2.4.2.2 Light sensor

*

This is a further development of the [light barrier](#). Though the light sensor is always aligned by TrySim

along the X, Y or Z axis. Then it surely captures all [dynamics](#) crossing its ray.

As long as your sensors only have to capture right/left, forward/backward, up/down the light sensor is easier to use. But if you absolutely have to look obliquely you should better use a light barrier.

The light sensor has proved helpful, for example, if you want to capture very thin dynamics, because the light barrier is standardized in such a way that it is only sensitive in one point of the middle of the ray. If these thin dynamics (e.g. board) are located outside the only sensitive point of a light barrier can't be recognized suddenly. This problem can't arise in using light sensors.

In huge machine (>500 I/Os) with many dynamics the exchange of light barriers against light sensors can make a substantial contribution to reducing computation time. However, please contact us prior to such modifications. www.trysim.de

Unfortunately, the simplified calculation rule of the light sensor does not work in recording [turnable dynamics](#). That is why you can enter the number of points at the light sensor that are supervised along the ray. This number you can adjust within 1-19 has only an impact on registration of turnable dynamics. Please ignore this parameter if you don't have any turnable dynamics in your machine.

ATTENTION! On the other hand, if the light sensor itself is turned it will not work anymore. In this case please use the light barrier.

2.4.2.3 Limit switch

*

Limit switches are detecting the position of a certain static element called [master](#). This element has to be selected inside the edit mask of the limit switch. In most cases, it will be a box, but it can also be any other movable element. Do not choose a linear mover as master, however, but one of the boxes which are moved by it. (There are also [linear mover](#) with integrated sensors).

In contrast to real limit switches, those of TrySim can have any size you like. Long-shaped limit switches can thus take over the function of a security bar, for example.

The sensitive area corresponds to the size of the limit switch. You cannot change the size with the help of the mouse. To do that you have to enter the edit mask.

If your limit switch has to be covered by more than just one element, please use the [limit switch nose](#)

.

With the field 'type' you are able to configure a limit switch as 'NO contact' or 'NC contact'. The also existing option 'Switch', by which the limit switch changes its status corresponding to every covering, is only useful for the simulation of special switches like the cross bar switch. When reporting a '1' to the PLC, the limit switch is colored red.

In order to detect dynamics, please use the [light barrier](#).

If the sensor is covered by dynamics as well as by a part of the machine you have to use a light barrier as well as a limit switch. Do not put them on the same address, but program them in the OB2 :

A light barrier

S limit switch

Please note that it states: 'S limit switch'. The assignment '= limit switch' does not work (why not?).

Limit switches can be made turnable. For this just choose within the edit mask the check box 'turnable'. Please note that currently the turnable limit switches just detect the master if either one edge of the limit switch is inside the master or one edge of the master is inside the limit switch. Please read the [general information about turnable elements](#).

2.4.2.4 Limit Switch Nose

**

Usually a [limit switch](#) detects exactly one element. Sometimes however one needs a limit switch that can detect many elements. Therefore we designed the nose: it activates every limit switch with which it overlaps; it has not to be configured as master.

The [nodes](#) of a [chain](#) limited the nose property a bit: they activate any limit switch that has the chain as master.

Noses can be made turnable. Choose within the edit mask the check box 'turnable'. Please read also [general information about turnable elements](#).

2.4.2.5 Spy

**

You can use the spy in order to detect the position of movable elements. In reality you would therefore use rotational encoder, absolute encoder, potentiometer etc. In the last analysis all these elements have the task to deposit the position of another element somewhere in the PLC. The spy does exactly the same.

The spy detects the distance within the three space directions from his own position up to the position of the master. You must enter a DINT of the PLC for every space direction. If you need only one coordinate, you simply give the others a MD with a high number you do not use otherwise, e.g., MD 17000.

As in reality position detectors have different resolutions, you must decide inside the field 'scaling' how many units of the PLC-value have to be changed, if the master is relocated by about 1 mm.

In order to adjust the zero point of the spy, please proceed as follows: Bring the detected element to a position for which you know the value to be reported to the PLC. In case this element is hanging on a linear mover, you can shift it by the slide controller on its edit mask. Then you enter the value

which is to be sent to the PLC into the fields 'display'. That is all.

If you want to detect the position of the spy itself, e.g., if you have fixed it directly to the moving element for reasons of clearness, you need to specify the origin as master. In this case you should enter a negative value for the resolution to avoid the spy counting reversely.

For detecting the position of a linear mover, it is more suitable to use the [linear mover with sensors](#).

2.4.2.6 Distance Sensor

**

This element is used to measure the distance to [dynamics](#). This sensor detects just dynamics.

The distance is reported to a word within the PLC in the units chosen by you on the edit mask (default are 1 imp/mm). In the field 'points' you have to specify how many points on its length the sensor checks to find a dynamic at all. Please adjust this value to the size of your dynamics. If it is too low, it is possible that the sensor ignores a piece. If it is too high, the sensor consumes more CPU power than necessary.

With the check box 'max value if nothing found' you can adjust whether the sensor shall report a '0' or the maximum value to the PLC if it does not find a dynamic at all.

To change the orientation of a distance sensor you have to point with the mouse cursor near to the end of it and wait until the mouse cursor changes to a cross.

If you only need a contact that switches at a special distance, just program in OB2 for example:

```
L IW // enter the correct address here
512 // enter the switch distance here
L 500 // according to desired functionality <I as well
>I // enter the correct input here
= I 4.6
```

2.4.2.7 Resolver for Transporter/Chain/Track

**

This element is used to figure out how far a [conveyor](#), a [chain](#) or a [mangle](#) has transported something or how many material has come out of a [continuous generator](#). In order to do that, the conveyor resp. the chain resp. the generator must be specified as the master of the resolver and a word needs to be defined in the PLC. The resolver counts forward and backward, it just does incremental changes on the PLC word. It means you can change the value in the PLC program at any time, and the resolver will continue counting starting with the new value. You can define the resolution similar to the other position detectors. We have not designated a mechanism of synchronization, because you can influence the reported value in the PLC program.

You can only connect this resolver to the above-mentioned elements. If you connect it to a diagonal

conveyor, please note that the movement measured is too small by a factor of square root 2 (about 1.4). But you can correct this easily by adapting the resolution.

If the generator is connected to a continuous generator it will only get the movement of the track if this one covers its below left edge.

If you do not need a number, but just an indicating input, use the lowest bit. If you need two impulse tracks, which are offset by 90°, use the bit1 for track A and the bit1 XOR Bit0 for track B. for example like this:

```
A    Bit1
=    TrackA
X    Bit1
X    Bit0
=    TrackB
```

Of course you have to set the resolution so low that the value changes not more than 1 from cycle to cycle. If necessary you have to adapt the [simulation rate](#) or you have to use another pair of related bits with higher bit numbers.

2.4.2.8 Bar Code Scanner

**

The bar code scanner reads the bar code of dynamics set through the [generator](#) when being created. This value is written to a word in the PLC. If the bar code scanner does not interleave with a dynamic, '0' will be written.

In case of common dynamics the reader is considered to be covered if it overlaps anywhere with the dynamic. For all turnables it still applies that the position edge (on the bottom left) has to be inside the dynamic.

You can alter the size of the bar code scanner only by entering the new values in the edit mask. If the bar code scanner is covered by more than one dynamic, it will not be determined which value it will read.

The bar code scanner can be configured to be a bar code writer as well. It supplies each dynamic which covers it with the code that is in the PLC-Word.

If this value is zero, nothing will be written!

The bar code scanner can also detect and change the color of the dynamics. The numbers in PLC-Word are since version V2.9.60 a compressed RGB format that can be seen best with the digital display in hex-word format.

```
Red      : 0 0 0 F
Green    : 0 0 F 0
Blue     : 0 F 0 0
Black    : 0 0 0 1 !
```

```

--- #element funktions ---
#element.dyn_barcode_w      (read/write)
#element.dyn_color_w       (read/write)
#element.dyn_color         (read/write), dword, normal RGB format
---

```

2.4.2.9 Running Wheel

With this element you can detect the movement of [dynamics](#), but it also acquires the movement of a [track](#) which is created by a [continuous generator](#). On the edit mask you can input several data formats and the resolution.

For the application it is important that the position corner (on the display screen it is always on the bottom left) is inside the dynamic.

The output format 'impulses' and the Zero-impulse are not implemented at the moment.
The running wheel cannot distinguish between changing directions of movement: it always counts forward.

2.4.3 Actuators

2.4.3.1 Generator

*

You need a generator to create the [material](#) (dynamic elements, dynamics) that is processed by the machine. It is connected to a bit of the PLC. With every rising slope of this bit a dynamic is generated.

If the bit is always on, new dynamics are generated in regular intervals, which are specified by the field 'repeat rate'.

The dynamics are always cuboid-shaped but they can have every size you like. You enter the size of the dynamics in the mask that is called up by the button 'dynamics'. The easiest way is to click 'as generator' because then you are able to adapt the size of the created dynamics by dragging the margins of the generator even subsequently.

The size of the dynamics is randomly enlarged or reduced by the values that you can find in the fields below the size in order to simulate the adversities of the real life better.

If the dynamics should always have exactly the same size, you have to enter 0 there. Please note in this context the [random numbers](#) as well.

To identify dynamics you can also specify a word within the PLC. Each time a dynamic is created it is marked with the value which is currently in this corresponding word. Using the [bar code scanner](#) you can pick out this value at any position in the machine, and you can make the processing dependent on that, e.g., dynamics with even numbers to the right side, the ones with odd numbers to

the left.

You have the opportunity to create [turnable dynamics](#) as well. The only for this kind of dynamics possible options Center points, Edge points and X-Y-turns are explained in the section dealing with the turnable dynamics.

Normally the dynamics are gravity dependent in a simplified way, that means if they do not stand on a base they will fall with a constant speed. You can adjust this speed. Normally the default value 400 mm/s leads to good results. But if you put dynamics on movable bases which move downwards faster you should adjust this value. Eventually the 'thickness of the platform' has to be raised in **View|Options|Machine**, otherwise dynamics will fall through conveyors and the like. If you deselect the gravity dependence completely you will have to make sure that the dynamic leaves the generator with the help of a hook or a shifter for instance.

If you make the dynamics fillable they are like a [container](#).

In the standard adjustment the created dynamics are visible in every graphic and from every direction. You can change this behaviour by the [properties of the graphic windows](#). But you can also activate the check box 'visible like generator'. Then the visibility of the dynamics follows the same rules as the ones for the generator, that means the directions out of which the dynamic is to be seen and the [graphic filter](#) are taken over by the generator. Please note that the taking over happens during the creation of the dynamic. The generator does not have any access to an already created dynamic.

Please note that a new dynamic is created not until the previous one has left the generator. If necessary you have to reduce the size of the generator.

Often beginners' mistake: A generator is placed in the X-Y-top view and so it does not seem to create any dynamics. TrySim simulates the world three-dimensional! If a generator which stands on the floor of a hall creates a dynamic, that one will stand on the floor of the hall as well and there is no reason to move in any direction. The new dynamic is not even visible, unless you look at it exactly and you realize that the color of the generator has changed from violet to black.

If you want the dynamics to fall out of the generator, as you can see in example 1, you will have to place the generator above the floor of the hall. This is only possible in the Z-Y side view or in the X-Z front view. And only in these views you can notice the dropping of a dynamic out of a generator.

2.4.3.2 Continuous generator

With this you create a continuous track of material. The speed of creation is specified by the drive, for this generator there are the same drives as for the conveyor. On the edit mask you select the transport direction (+X, -X, +Y, -Y, +Z, -Z). The cross-section of the track is specified by the measurement of the generator.

With the help of the [saw](#) you can cut off the pieces of the track. These pieces will then become

normal dynamics whose properties you can adjust on the edit mask of the continuous generator in 'dynamics'. You can only cut crosswise to the track, it is not possible to cut along its direction of movement.

So that a track is created the bit has to be set 'active' or the corresponding check box 'always active' on the edit mask has to be clicked. If the continuous generator is not active anymore the current track will be cut off and turned into a normal dynamic. If the generator is activated again a new track will be created.

You can detect the existence of the track with the light barrier. You can register its movements with the [resolver for conveyor and chain](#). This one will then work like a running wheel: only if the track covers the resolver (exacter: its edge on the bottom left) the impulses will be counted. The generator has to be determined as master of the resolver. Please note that the runner will stop counting if the track is cut off and changed into a dynamic (e.g. by a saw or by deactivating of the generator).

This element is still at the experimental stage. Suggestions out of practice for improvement are welcome.

2.4.3.2.1 Continuous dynamics / track

These are created by a [continuous generator](#). Please read there first.

You can record the movement of a track with the [running wheel](#) as well. This has got the advantage over the recording with a generator, that the movement will still be recorded if the track is cut off by switching off the generator or by a [saw](#).

2.4.3.3 Destroyer

*

The material that is created by one or several [generators](#) will leave the machine anytime in reality, either it will be taken away or it will be processed by other machines. For the machine that is simulated by TrySim the material is not important anymore and that is why it should be 'destroyed', otherwise it will take computer capacity unnecessarily.

Basically the destroyer is a [conveyor](#), but it destroys every dynamic as soon as all edges stands on it or hang in the air. According to our experience, only a few dynamics will succeed in reaching the destroyer in the beginning, so that after a while there will be dynamics all over the machine. Then you can either eliminate the dynamics all in one step entering the menu instruction **Machine|Dynamics** or you can eliminate single dynamics by marking them with the mouse and the key **Del**.

[Turnable dynamics](#) are destroyed when at least 3 points stand on the destroyer. Eventually centre- and edge-points are included as well.

For free movable elements which always stay in machine but which have got the properties of

dynamics, there are the storable dynamics. These will not be destroyed by the destroyer.

2.4.3.4 Linear mover

See [linear mover with sensors](#).

2.4.3.5 Linear mover with sensors

*_**

You need these elements to set your machine in motion. In reality they correspond to all those components that are just carrying out a straight movement. These might be hydraulic/pneumatic cylinder, rack and pinion drives, piezo-electrical actuators, wagons running on rails and all kinds of things like this.

In TrySim, a linear mover is a line that can be located anywhere in space along which the movement is executed. If the desired movement does not follow a straight line, you either have to interlink two or more linear movers or, if this is not possible, use a [joint](#), the [free point](#), or the [chain](#).

The linear mover has a marked spot, which is called hot spot. This spot can be moved along the linear mover, however not beyond its limits.

In order to [fix](#) other elements to the hot spot, you have to define the linear mover as their father. All these elements will then move just like the hot spot.

The speed or the position of the hot spot is preset by the [drive](#). So the drive corresponds to a motor, a valve or a control electronic. To select the drive you have to click it within the edit mask from the list 'drive type'. By the button 'drive' you call up the edit mask of the drive.

You find the possible drives of the linear mover further below in this chapter.

On the edit mask the position of the hot spot is shown as a number or by a slide controller. During the construction time the position can be changed in order to adjust limit switches and things alike.

There is also a [linear mover with integrated sensors and position indicator](#).

To change the orientation of the linear mover with the mouse you have to point with the mouse cursor on one end of it and wait until the mouse cursor changes to a cross.

You can also use a linear mover to control a [joint](#), a [valve](#) or a [heating](#).

The [linear mover](#) is equipped with integrated limit switches and a position detection. You can configure the sensors in the tabsheets 'limit switch' and 'position' at the edit mask of the linear mover.

The procedure to synchronize the position detection is as follows:

- 1.) Move the hot spot to a position where you know the value that the position detection should display. To do that you can move among other things the slide controller 'hot spot' on the linear

mover form.

2.) Open the mask of the detection with the tabsheet 'position'. You can either enter the offset as way or put the current desired donor value into the field 'display'.

Please note the difference between this position detection and the one that is done by the [spy](#): If you use linear movers which work oblique in space, the output will be the distance along the linear mover, whereas the spy will provide three spatial coordinates separately.

The integrated limit switches will only work if the hot spot of the linear mover is absolutely at the stop. If the hot spot does not stop there (this will often happen if the linear mover is [moved by a joint](#)) this will lead to errors. In such cases you should use a normal linear mover and normal limit switches. You can adjust these so that they respond in every case.

The position determination provides the value as DINT. We did so because the value range of the INTs (-32000 to + 32000) is not enough in many cases. If you need just one INT in the program you just have to load the last two bytes of the DINT. Example: You state ED 1200 for the Sensor. This ED consists of the bytes 1200, 1201, 1202 and 1203. If need just one INT, because your positions are guaranteed smaller than 32.000 units, write in the PLC program: L EW 1202. Please note that the bytes 1200 and 1201 may no longer be used, since the position determination will write on them in every simulation cycle.

Apropos of nothing: It is wise always to use DINTs. Modern PLCs contain storage space without end and you hardly ever have to consider whether the value range is big enough.

Please read the help to the [resolver for conveyor](#)

See also:

[Common properties](#)

[Static elements of simulation](#)

Tabsheet: Actuators

2.4.3.6 Conveyor

*

[Dynamics](#) can be transported by the conveyor. Like the linear mover they have a [drive](#). When a dynamic is standing on the conveyor with all four corners, it will be moved according to the speed which has been selected for the drive. In case one or more corners are not standing on the conveyor, the dynamic will accordingly be moved slower and will be colored red in order to indicate its unnatural condition. Please note that several conveyors which are one behind the other need to border on each other precisely.

By fastening a conveyor to a linear mover that is oriented vertically, you can construct a lift for pallets. If you connect it to a linear mover that lays on the floor you will get a conveyor wagon.

Only movements in the XY-plane are possible, and there are 8 directions only. The conveyor

transports the dynamics with the speed that is specified by the drive. The direction of transport is indicated by the arrow within the edit mask, the direction can be changed by clicking.

The diagonal conveyor transports the dynamics not along the X- or Y-direction, but in a diagonal direction.

2.4.3.7 Joint

You can use the joint to simulate turnings around an axis which is orientated in any direction. It has the form of a linear mover. You specify the turning axis by specifying the position of the line in space. All elements (also their children) which you [fix](#) to this joint are turned around this axis.

The elements which shall be turned need to be quite small, because most elements cannot be turned itself in the current version yet. An exception are the elements which have got the form of a line, for example light barrier, linear mover and bar and also the box, the hooks, the limit switches, the [turnable conveyor](#) and the turnable dynamics.

There are several [drives](#) for the joint:

- 1.) [Bit drive](#): This drive will be connected to one or several bits of the PLC, and it makes the adjustment of the speed of rotation in both directions and up to two speeds possible. Also the acceleration and deceleration ramps can be adjusted, as well as possible limitations if the joint shall not go round continuously.
- 2.) [Servo drive](#): Creating the simulation you specify some turning angles here which will be covered during runtime of the simulation. Therefore you only have to give the PLC the number of the turning angle.
- 3.) [Linear mover as drive](#): With this you can translate a linear motion into a rotary motion. A rattle function is possible as well.
- 4.) [Crank drive](#): With this you couple the current joint to another one that already exists. The transmission ratio is adjustable.
- 5.) [Absolute real drive](#): Direct specification of turning angle. The same function is better fulfilled by a servo drive with direct specification.
- 6.) [Simple Frq converter](#): This drive corresponds with a frequency controlled engine or a proportional valve of a hydraulic engine. It has to be connected to a word (E, A-, M- or data-) of the PLC.
Next to the address you have to specify at which value of the word the maximum speed shall be reached. If the word shall head for an analog output in the real PLC that shall give +/- 10V to a frequency converter you will have to enter 27.648 here. For this drive you can specify slopes and

limitations of the turning angle as well.

There are two slide controllers on the edit mask of the joint:

1.) The slide controller 'without' loosen the axle nut. So you turn the axis with all the fixed elements. You do not cause any constructive changes of the machine in doing this. This turning possibility has only been created to monitor the movement of the elements which are fixed to the joint easily and to adjust limit switches, e.g. while editing. As soon as you restart the simulation the elements which are fixed to the joint will go back to their original position.

2.) The slide controller 'with' loosen the axle nut will cause constructive changes of the machine though. Adjusting this controller means the same as turning the elements which are fixed to the axis after loosening the axle nut.

With the page-up/down-buttons you cause a changing of the turning angle of about 45° each, an accurate adjustment is possible with the help of the arrow right/left-buttons.

With a [peek](#) you can load the current angle of the joint in a Word of the PLC (unit 0.1 degree).

The examples 'Robby1' and 'Robby2' demonstrate the use of joints.

```
--- #element.funktions ---  
#element.angle_act    (read, REAL, grad).  
---
```

2.4.3.8 Hook

*

With the hook you can catch dynamics like you do it with a crane. You should not take the name literally. Use the hook every time you have to transport dynamics that are not on a base.

Whether the hook is active or not is controlled by an output bit of the PLC. In case you activate the bit, the hook will catch all the dynamics with which it overlaps anywhere.

Please note that the hook can also be activated externally by an [on-/off-pawl](#).

Hooks are very suitable as ejector and stopper as well. Because the simulation of such elements is extremely difficult (you need to take into consideration weights, coefficients of friction, stabilities etc.), you need to help along in this case and switch on and off the hook in the right moment. If a PLC program code is also necessary for this, you shall arrange this one in the [script](#) (eventually also in [OB 2](#) or [OB 3](#)). In simple cases a [shifter](#) can be of good use as well.

Hooks can be made turnable. Select the check box 'turnable' on the edit mask.

Please read also the [general information about turnable elements](#) further below in this chapter.

2.4.3.9 Free point

This element is provided in case a machine would carry out a movement which is not composed of simple linear movements, e.g. a circle movement. Because it is easier to write we call it free point. You can define each of the three coordinates of the free point by the PLC. Therefore you have to write a function block, which calculates the required movement.

For each of the three space directions the free point provides a DInt, in which you have to deposit the position of the chosen units for the system (standard: mm) on part of the PLC.

If you calculate the position not as DInt, but as INT, you have to take care that the value will not become negative. Otherwise you have to convert it to the DInt-format by the instruction ITD.

2.4.3.10 Chain

The chain is a closed polygon consisting of as many [segments](#) as you like. On this polygon any number of [nodes](#) move in circles with a speed specified by the [drive](#) of the chain. You can adjust the position of each node separately or you can automatically distribute them regularly.

When creating a chain it consists of 4 segments at first. You create new segments by dividing existing ones. You do this by marking a segment and calling the edit mask of the chain by clicking right. Then you select the button 'divide segment' in the tabsheet 'parameter'. You can shift and turn the separated segments like a [linear mover](#). To turn a segment you have to position the mouse cursor near to the end of the segment so that the cursor changes into a cross. This is sometimes a little bit tricky, especially with segments that pass exactly along an axis.

The nodes will be created automatically if you [fix](#) an element to the chain. The position of the new node is chosen as near as possible to the position of the new fixed element. You can shift the node with the help of the mouse to the desired position. Around edges it is difficult to shift nodes. In these cases you can open the edit mask of the node and correct its position on the chain with the slide controller 'offset'. Before fixing the nodes you should [fix](#) the chain to avoid an unintended movement of the segments.

If you want to detect the movement of the chain, please use the [resolver for conveyor/chain](#).

In connection with chains the [on/off pawl](#) and the [limit switch nose](#) are quite useful. If you define a chain as [master](#) of a [limit switch](#), it will be activated by every node with which it overlaps.

The samples 'formula_1' and 'tree saw' demonstrate the use of a chain.

In this version the things which are hanging on the nodes are not turned in curves.

The editing of a chain is still in need of improvement.

2.4.3.10.1 Node

These elements always appear as part of a [chain](#). They serve as fixing points for other elements in your simulation. They are created automatically when you fix an element to a chain. You can move a node along the chain with the mouse or you can readjust the slide controller 'offset' on the edit mask. For a real chain this corresponds to the unscrewing of a hook and the replacement at another position. In contrast to this moving the slide controller at the edit mask of the chain corresponds to start the drive of the chain.

If you want to detect individual nodes with a [limit switch](#), you have to specify the particular node as master of the limit switch. But if you want all nodes to cover a limit switch you will have to specify the chain as master.

2.4.3.10.2 Segment

A [chain](#) consists of segments. The segments can be shifted or turned with the mouse or by the edit mask like other line shaped elements as well. While shifting a segment the previous and following segment is automatically adjusted in that way that the chain is closed again.

New segments are created by dividing an existing one. To do this select the segment which needs to be divided, open the edit mask with a right mouse click and choose the button 'divide segment' in the tabsheet 'parameter'.

If you delete a segment, the ends of the previous and next segment are connected so that they meet in the middle of the deleted segment.

Deleting the last but one segment (then the chain only consists of two segments lying on top of each other) is the same as deleting the whole chain.

To reach the edit mask of a segment, select it, open the edit mask of the chain and choose the button 'edit segment' in the tabsheet 'parameter'.

The [nodes](#) sitting on a segment do not belong to it but to the chain of overriding importance instead.

You can set the visibility for each segment of a chain separately, but this setting will be overwritten if the visibility of the whole chain is changed.

If you have made an element invisible in all directions you can only process it by calling it over the [element tree](#).

2.4.3.11 Turner

**

You should not use this element anymore, the [joint](#) is much more variable.

The turner moves elements which are fixed to a circular arch in the XY-plane with the speed which is specified by the drive. You should only fix one element to the turner directly.

2.4.3.12 Rider way

The rider way is a further development of the chain. It is simpler but harder to insert. Beginners shouldn't use the rider way.

You need three elements for the run of a rider way:

1.) Segments of the rider way intrinsically. Each of these segments has got an own drive and moves all elements attached to it with the speed that is given of the drive in the direction of the segment. But you should only fix 'riders' to the rider way.

In contrast to the chain each element of the suspension railway is completely independent. The connection between the segments has to be created by them explicitly by transitions. The rider way does not have an end like the linear mover. All riders fixed to it moved towards the railway until they have found a new father.

2.) Rider. These are elements which differ slightly from the box. You should use them as hangers for hooks or similar. A rider always moves with the speed of the rider way whose child it is. And here transitions come into play: they can change the father of a rider. So a rider on its round trip can be moved by different segments of the rider ways.

3.) Transitions. A transition works as follows: in every cycle it checks: Am I overlapping with a rider? If it has found such a rider it asks: Am I overlapping with the end or the start of a rider way? If there is such a rider way the rider is re-hanged from the old rider way to the new one, that means the father of the rider is changed from the old to the new railway.

Important while placing an transition is that it overlaps with the new railway, but not with the old one.

You can realize switches, for example, by fixing two rider way segments to a linear mover. Which segments is in the transition at the moment decides from which segment the rider is incurred.

Tips to work well with the rider ways:

A transition is only usable for one direction. If the rider shall run over a transition point in both directions you need two transitions. But you have to take good care that the rider never overlaps with both transitions.

2.4.3.13 Mangle

The name of this element might irritate you presumably. It is like quite often in TrySim: we just have to give it a name. If elements with the same function are used in your company they definitely will not be called 'mangle'. Maybe in America but in Germany such a machine will be called 'Doppel-Walzen-Linear-Förderer' or 'Walzen-Ausstößer'.

The mangle is made of two rollers which soak up each dynamic that is contacted by the lower roller and transport it as long as it does not contact the lower roller anymore.

The speed of conveyance is specified by the drive: for the mangle the same drives are available as for the conveyor. The mangle will only soak up dynamics if the upper roller is sunken. You have to select on the edit mask of the mangle which one is the upper roller. Here you can only select boxes. If the box that is called 'upper roller' overlaps with the mangle this one will transport the dynamics, not otherwise. If no element is specified as upper roller the dynamics will always be transported.

With the option 'stay flying after leaving' the dynamic maintains its speed in the XY-plane, after it does not overlaps anymore with the mangle. The dynamic is stopped when it lands on a [conveyor](#) or bumps into a [shifter](#). On other dynamics or a [board](#) it slides further.

The option 'liftable' is for mangles that are ejection-height adjustable. Without this option the dynamics are transported straight ahead which makes it easier positioning them. With the option the dynamics are raised to the upper edge. This can be useful but requires more carefulness while generating the simulation.

At present the mangle can only transport in the X- or Y-direction, that means neither diagonally nor vertically.

2.4.3.14 Turntable

This is an element that only works with [turnable dynamics](#). It causes an active turn of the dynamics which stand on it. The turn speed is specified by the [drive](#).

The displayed speed in mm/s means the speed for dynamics which are on the outer periphery of the turntable.

If you do not deposit the dynamics onto the turntable with a [hook](#) or an [automatic hook](#), you must use a vertically orientated [linear mover](#) to lift the turntable from below through the [conveyor](#), on which the dynamic is standing. So you can make use of the fact that the elements in TrySim can go through each other.

The turntable itself cannot be turned.

The turntable is displayed as an octagon but internally it is cuboid-shaped like all the other elements. So it will also move dynamics which are in the cut off and not visible edges.

The sample 'turns' demonstrates the use of this element.

Other possibilities to turn dynamics result from the use of a [joint](#) in connection with a [hook](#), a turnable [board](#) or a [turnable conveyor](#).

2.4.3.15 Arc Belt

This is an element that only works with [turnable dynamics](#). You specify the direction of movement on the edit mask. The check box 'rev.' reverses the direction.

Certainly the arc belt moves the edges of all turnable dynamics exactly in an arc (you can try this with a very small dynamic) but because of the unavoidable sliding while moving from and to linear conveyors the dynamics will not be right-angled anymore. By adjusting the speed (higher) this effect can be compensated extensively.

You can correct a remained turn of the dynamics with the [straighter](#).

If the belt is meant to transport less than 90°, you can arrange a [turnable conveyor](#) to the desired angle of rotation a little higher. Then the dynamics will be transported on that one and will not finish the 90°.

The arc belt is not matured completely yet. Mainly the representation as a cuboid is quite disturbing in this case, especially if you take belts which have a big radius. The arc belt itself cannot be turned.

The sample 'turns' demonstrates the use of this element.

2.4.3.16 Turnable Conveyor

This is a [conveyor](#) that you can turn in all directions. It only functions well with [turnable dynamics](#).

You can [fix](#) the turnable conveyor to a [joint](#) in order to obtain a turntable, or if you orientate the joint in the XY-plane, you will get a switch which moves up and down. You can also turn the conveyor in the editing mode by clicking the conveyor with the right mouse button for quite a long time and choosing 'turn' out of the context menu afterwards.

You can change the size of the conveyor with the mouse only if it is in its original position.

The turnable conveyor needs lots of computing power.

The samples 'sample1' and 'turns' demonstrate the use of this element.

2.4.4 Controls

2.4.4.1 Pushbutton

*

With the help of push buttons and switches you have the possibility to manipulate the process during simulation runtime. Their role is exactly the same as that of real push buttons/switches with the exception that you always have to connect real push buttons to an input of the PLC, whereas with TrySim you are capable to manipulate also outputs and markers directly. For example, if you like to test a motor immediately during the construction of the machine you only have to create a push

button and by doing so you will be able to switch the output to which the motor is connected.

Switch/NO contact/NC contact

In fact these three elements are one and the same. However, the selection list will show it in three versions, in order to avoid a subsequent configuration.

The switch changes the state of a bit each time you click it.

The NO contact sets the state of the bit to '1' as long as you click it, otherwise it sets the state of the bit to '0'. Do not click it too briefly, otherwise the NO contact may not recognize the click.

The NC contact sets the state of the bit to '0' as long as you click it, otherwise it sets the state of the bit to '1'. NO contacts are quite good to use to simulate a backup case.

You can change the elements at any time by changing the field 'type' accordingly within the edit mask.

Please note that it is not possible to use a switch/NO contact/NC contact in order to overwrite a bit which is also written by the PLC program. If this element does not work as expected you are able to check with the help of the [cross reference list](#) whether the controlled bit is modified in the PLC program. The easiest way to call the CR is by clicking the button 'CR' on the edit mask. In the CR all the places where the bit is modified are marked by a red point.

2.4.4.2 LED

*

The LEDs function exactly as real LEDs, but they do not burn out that fast. You can interface LEDs not only with outputs but also with inputs and markers and place them everywhere in the machine.

This is quite useful to display hidden information more clearly than in the status-variable-list.

With the check box 'original' on the edit mask you will be able to determine if the indicator keeps its size when zoom is changed.

Indicator

This special design of the LED flashes with a constant frequency of 1 Hz. Use it if the flashing of an LED that is controlled by the PLC is to be seen indistinctly because of the variable speed of the simulation within TrySim. The indicator changes its status every 0.5 sec either if the bit, that has been connected to it, is '1' or if the status of the bit has changed during the last 0.5 sec. In case that the bit is '0' and there has been no change of the status during the last 0.5 sec, the indicator is switched off. See also: [clock memory](#).

2.4.4.3 Digital display

*

With this feature you can display the content of every byte, word or dword of the PLC in various [formats](#).

The digital display serves as a digital input at the same time: By a double click at runtime you activate the edit mode and enter the desired number.

Another possibility to display analog values is to use the [slide controller](#) or the [oscilloscope](#).

If you want to change the size of the value (byte, word or dword) that is to be displayed so you will have to select the desired size first and then enter the new operand. It is quite annoying, but we have not had the time to make it more comfortable yet.

2.4.4.3.1 Data Formats For Digital Display

If you do not pursue an object you shall prefer the use of INT, DINT or REAL.

BIT, BOOL: the state of the bit is displayed as 0 (off, wrong) or 1 (on, true). In general an indicator light is more useful to display a bit.

Unsigned BYTE (1Byte): The range is from 0 to 255.

Useful PLC-commands: [INC](#) and [DEC](#)

Signed BYTE (1Byte): The range is from -128 to +128. In FBD, LAD and STL this data type has got no correspondence.

Hex-BYTE (1Byte): The range is from 00 to FF, permitted are the numbers from 0 to 9 as well as the letters from A to F. If you only make use of numbers it will be possible to use this type as two-digit BCD as well.

BYTE as bit pattern (1Byte): The range is from 0000 0000 to 1111 1111. Only the numbers 0 and 1 are permitted. You can enter space characters and ' _ ', but these are not displayed afterwards.

WORD (2Byte): The range is from 0 to 65.535.

INT (2Byte): The range is from -32.768 to +32.767.

Useful PLC-commands: [+I](#), [-I](#), [*I](#) and [/I](#)

Hex WORD (2Byte): The range is from 0000 to FFFF. Permitted are the numbers from 0 to 9 as well as letters from A to F. If you only make use of numbers you can use this type as four-digit BCD without a sign.

Three-digit BCD with sign (2Byte): The range is from -999 to +999. The number will be interpreted as negative if the four bits with the highest value equal 1.

Useful PLC-commands: [BTI](#) and [ITB](#). You cannot make a calculation with BCD-numbers.

WORD as bit pattern (2Byte): The range is from 0000 0000 0000 0000 to 1111 1111 1111 1111. Only the numbers 0 and 1 are permitted. While entering you can use space characters and ' _ ', but these are not displayed afterwards.

DWORD (4Byte): The range is from 0 to 4.294.967.295.

DINT (4Byte): The range is from -2.147.483.648 to 2.147.483.647.

Useful PLC-commands: [+D](#), [-D](#), [*D](#) and [/D](#)

Hex-DWORD: The range is from 0000 0000 to FFFF FFFF. The numbers from 0 to 9 as well as the letters from A to F are permitted. While entering you can use space characters and ' _ ', but these are not displayed afterwards.

7-digit BCD with sign (4Byte): The range is from -9.999.999 to +9.999.999. The number will be interpreted as negative if the four bits with the highest value equal 1.

Useful PLC-commands: [BTD](#) and [DTB](#). You cannot make a calculation with BCD-numbers.

REAL (4Bytes): floating point numbers

Useful PLC-commands: [+R](#), [-R](#), [*R](#), [/R](#), [DTR](#) and [RND](#)

2.4.4.4 String display

**

With the string display you can output a line of the data type [STRING](#) of the PLC. Determine the first byte of the string as PLC-byte.

Do not mix up the string display with the [text display](#) which has got a completely different function. Also the normal [digital display](#) has nothing to do with the string display.

We generated the display just for the data type STRING that is hardly ever used in simple S7 programs. Of course you can use it for the general presentation of strings, but therefore it is inevitable to have caught up on this data type at least shortly.

2.4.4.5 Slide controller

*

This is an input device used by mouse, whose value is written into any byte, word or dword variable of the PLC in different [formats](#).

You can also control the slide controller with the keyboard after you have activated it by clicking: With the page up/page down keys you achieve an adjustment by one of the displayed graduation marks. A high precision adjustment is possible by the arrow right/left keys.

The minimum position will be below if the slide controller is vertical, if it is horizontal it will be on the left side.

The slide controller can be used as an analog display as well. Just transfer a value into the connected PLC-variable.

Another possibility to give analog values is the [digital display](#), which can be used as digital input as well.

If you want to change the size of the input value (byte, word or dword), just select the desired size first and enter the new operand afterwards. This is quite annoying, but we have not had the time to make this more comfortable yet.

2.4.4.5.1 Data formats for slide controller

If you do not pursue an object you shall prefer the use of INT, DINT or REAL.

Unsigned BYTE (1Byte) : The range is from 0 to 255.

Useful PLC-commands: [INC](#) and [DEC](#)

Signed BYTE (1Byte): The range is from -128 to +128. In FBD, LAD and STL this data type does not have any equivalent

Two-digit BCD (1Byte): The position of the slide controller is translated into a two-digit BCD number from 0 - 99. If you want to display this value with the digital display you will be able to use the format hex-byte.

WORD (2Byte): The range is from 0 to 65.535.

INT (2Byte): The range is from -32.768 to +32.767.

Useful PLC-commands: [+I](#), [-I](#), [*I](#) and [/I](#)

Three-digit BCD with sign (2Byte): The range is from -999 to +999. The number will be interpreted as negative if the four highest bits equal 1.

Useful PLC-commands: [BTI](#) and [ITB](#). You cannot make a calculation with BCD-numbers.

DWORD (4Byte): The range is from 0 to 4.294.967.295.

DINT (4Byte): The range is from -2.147.483.648 to +2.147.483.647.

Useful PLC-commands: [+D](#), [-D](#), [*D](#) and [/D](#)

Seven-digit BCD with sign (4Byte): The range is from -9.999.999 to +9.999.999. The number will be interpreted as negative if the four highest valued bits equal 1.

Useful PLC-commands: [BTD](#) and [DTB](#). You cannot make a calculation with BCD-numbers.

REAL (4Bytes): floating point numbers

Useful PLC-commands: [+R](#), [-R](#), [*R](#), [/R](#), [DTR](#) and [RND](#)

2.4.4.6 Text display

**

The text display corresponds to a simple operator panel, with which you can display error messages.

The edit mask contains a table for entering error memory bits and corresponding messages. If the memory bit has been registered within the symbol table as message the comment will be proposed, however, it can be changed. During runtime the text display appears as a single-lined field, on which the latest error is displayed. By clicking the field a larger mask appears and all the currently present messages are shown. A button on this mask serves as 'confirm key'. You have to assign a PLC-bit to this button first.

Please do not mix up the text display with the [string display](#) which has got a completely different function.

2.4.4.7 Oscilloscope

*

This is an element to display the temporal development of bits and analog values (16 bit). For displaying analog values, you have to adjust the min and max value appropriately. If the min value is less than zero the PLC word is interpreted as integer and the max value is limited to 32.767.

If you want to display DINTs or REALs you have to recalculate the values into words in the PLC program. For that purpose you can use the FC51 resp. FC52 in the directory [IECFuncs](#). We recommend to place this kind of calculations that is needed for simulation purposes only in the OB 2 or OB 3.

With the editing field 'Time/Div', you can stretch or compress the display in temporal direction. Usually it is not necessary to adjust this value. Use this setting together with the changing of the size of the oscilloscope to get a representation which is appropriate to your application.

With the editing field 'recording rate' you can modify the recording rate of data. A time, shorter than the [simulation rate](#), has no effect. The higher you adjust this value, the smaller is the temporal resolution for short events, but the longer is the recording period.

Please note the difference to the setting Time/Div, with this just the representation of the already recorded values is influenced on the monitor.

There are no further trigger possibilities or other adjustments, but have a look at the [speed trigger](#).

To delete a line in the table use the key combination 'Ctrl + Del'.

2.4.4.8 Multiple switch

This element you have to use at least as a pair. All multiple switches that are [fixed](#) to each other form a unit. Only one of these switches can be activated at a time. That means that by activating one of them by clicking during the runtime, all other active switches are deactivated automatically (with NC contacts activated means that the bit is '0').

It is recommended to leave the contact type of all multiple switches at 'NO'. And it still keeps manageable even if all get the contact type 'NC'.

You can select every possible combination and you can even choose the type 'switch', although we cannot think of a useful application.

With the edit mask, you can configure the behaviour of the contacts :

- Instant: the signals are changed from one PLC cycle to the other
- Via Zero: for at least one cycle all switches are deactivated
- Overlapped: only if the new switch is activated first the old one will be deactivated.

We recommend to configure all multiple switches of one unit to the same contact behaviour. Which consequences the configuration of the behaviour of the contacts have got you can see best by using an [oscilloscope](#), select a Time/Div of 0.2 and an extremely slow motion shot.

The best way to check with which multiple switches a determined one is connected is to open the [element tree](#). Please note that only those switches are considered to be fixed to each other, that have the same (grand-) father which is also a multiple switch. For clarity you should put all multiple switches which belong to one unit directly onto this distinguished switch (you recognize it easily because its father is not a multiple switch) in the element tree. This seems to be more complicated than it is actually.

Through the use of multiple switches, you should really note the behaviour of the switches of your actually used switch. Ask yourself questions like these: Does it have overlapping contacts? Which intermediate states can occur? Please check out first how your program reacts when switching. Potentially this is a serious source of error.

2.4.4.9 Master switch

**

The master switch looks like a [slide controller](#) with up to 16 positions. On the edit mask you can specify up to 16 bits, and to each position you can assign one or more bits to be activated by. When configuring a master switch for the first time, you have to specify the number of positions and also the bits to which they are connected. Then you click the button 'contacts': a matrix of size 'count of positions' x 'count of bits' is displayed. Click the bits here which are to be activated at each position.

If the master switch is directed vertically, the position '1' will be at the bottom, if it is directed horizontally, the position '1' will be at the left side.

If you have entered a new bit into the table, please note that you have to leave the new row with the cursor (e.g., with the 'arrow down' key), so that the operand is accepted.

If you want to manipulate the position with the PLC program you have to use a [word-poke](#).

2.4.4.9.1 Contact-Editor of the master switch

**

On this mask you determine which bits will be switched on in which switch position of the [master switch](#).

Before you fill in this mask, you should determine in the edit mask of the master switch, how many switch positions it shall have and which bits shall be connected to it. After this the mask will have the right number of fields.

If you have entered a new bit into the table please note that you need to leave the new row with the cursor (e.g., with the 'arrow down' key), so that the operand can be transferred.

2.4.5 Fluids

2.4.5.1 General

**

From version 2.3 on there are elements to simulate fluids. We tried to model the behaviour of the fluids as realistically as possible, but we had to make severe simplifications. It was our aim to enable a PLC programmer to simulate programs with containers, pipes, pumps, and valves. It was not our aim to create a tool that helps designing such machines.

The fluids in TrySim consists of a mixture up to 16 different [media](#). Each medium has a name, a density and a viscosity. The resulting properties of the mixture are calculated by an averaging that is weighted by share of the several media. If this simplification is too big you will have the possibility to transform media into each other by the [reactor](#) and to lead to the wanted properties doing this.

The best way to construct your machine is to start with a [container](#). On its edit mask you click the check box 'source', then it is always filled, no matter how much fluid you take. After that you install further containers and connect them to each other by [pumps](#), [valves](#) and [pipes](#). For easier use the pumps and valves are already connected to a straight pipe. Only if you want to go around corners you will have to put a pipe explicitly.

The ends of these elements are either connected to a container or they are connected to each other by a [flange](#). You can connect several pipes to a flange as well. You have to consider the geometrical proportions. That means if you want to take fluid out off a container that is half-full by a pump, the pump will have to be connected to the lower half of the container, otherwise it will only pump air.

The PLC gets the necessary information for operating the machine by the [level gauge](#), the [level](#)

[switch](#), the [flow meter](#), the [pressure sensor](#) and the [analyzer](#).

The fluidor transform dynamics in fluids, dissolves them, so to say.

2.4.5.2 Container

**

The containers in TrySim are cuboid-shaped. Their capacity is defined by the geometric dimensions. In each container is a mixture of different media whose amount can be detected with the [level gauge](#) and with the [level switch](#). The composition can be detected using the [analyzer](#). You can register the pressure in a determined height by the [pressure sensor](#).

On the edit mask the level is displayed in percent by a slide controller and you can modify it there as well. If you need an inexhaustible source, e.g. a tap, click the check box 'source'. Then the container will always be filled, no matter how much fluid you take out of it. If you need a drain you will have to click the check box 'sink'. Then the container is always empty, no matter how much fluid you pump into it.

The mixture that is currently in the container is displayed on the edit mask by four percent numbers. If you want to see the whole mixture that can consist of up to 16 different media click the button 'all media'. The order in which the shares shall be displayed you can specify for each container separately. Doing this you select the wanted medium for each place by clicking the arrow that points down next to the field of the media description.

You create a new medium by either clicking 'new medium' out of the selection list or by using the button [media manager](#) and clicking 'New'.

If you edit several containers at the same time you cannot change the order of the media and the composition of the media.

To see the level of the fluid, you have to use the XZ-front view or YZ-side view. It is a frequently asked question to the hotline: 'Why can I not see the level of the fluid?'

2.4.5.3 Pipe

**

A pipe consists of any number of pipe segments that are connected to each other. If you have created a pipe it will consist of only one segment first. New segments are created by choosing 'divide segment' on the edit mask. The segments can be shifted with the mouse by grabbing them in the middle, or the orientation of a segment can be modified by you positioning the mouse cursor near their ends so that it changes to a cross.

You should adjust the nominal diameter of the pipe on the mask [extended pipe properties](#).

The ends of the pipe can be connected to a [container](#) or to a [flange](#). This is simply done by positioning the end inside the container or flange. You do not need to be very precise, because when starting the simulation ends that are not connected are detected by the system, and after confirmation they are connected to the nearest point. But, consider the three dimensionality of the TrySim

machine!

The [cross](#) and the [stripe](#) are useful tools constructing the machine.

A [flow meter](#) and an [analyzer](#) can be connected to a pipe.

Pipes and all other [pipe-like elements](#) can also be used to fill fillable dynamics. In this case you should deactivate the option 'warn if not connected' on the edit mask 'fluids'.

If a pipe end is not connected the fluid that leaks there will disappear. At least one end of a pipe must be connected to a container or a flange, otherwise the behaviour of the pipe is not processed at simulation runtime.

2.4.5.3.1 Extended Pipe Properties

**

Diameter

The elements [pipe](#), [pump](#), [valve](#) and [check valve](#) have a length that is defined by their geometrical dimensions and a nominal diameter that you can adjust on the mask. These two values are used together with the viscosity, the density and the speed of flow to calculate the flow resistance. In fact it is a rough estimate only, since it depends on many more factors and to record all of them would go beyond the scope of TrySim. We tried to do without the calculation of the flow resistance but it turned out that a sufficient realistic behaviour of the fluids cannot be obtained without it. Therefore it is **important** to **adjust** the **nominal diameter** of all above mentioned elements at least order of magnitude wise to your machine. If you, for example, connect two containers with 1 l volume each to a pipe 100 mm in nominal diameter you will not be able to expect reasonable results. In **View|Options|Machine** you can specify a default value which will be used for new elements.

Number of sections

In TrySim each pipe is treated as if it is composed of several (at least 2) sections. For most applications it will be sufficient to maintain the default of two sections, but if you want to simulate a siphon* for example, you will have to raise this number. Another necessity to raise the number of sections will be if it is important for your use that the pipes have got a limited volume and that the mixture of the fluid along the pipe can be changed. But do not select the number higher than necessary because this is payable by the simulation speed.

Warn if not connected

Normally there will be a warning when simulation is started and a pipe end which is not connected to a container or flange is found.

But if you want to fill dynamics that reach the place of filling during the simulation in general this control will be quite annoying. By deselecting this check box the control is switched off.

* What is meant here is, for example, the procedure to let water out of an aquarium by a hose after you have sucked in shortly.

2.4.5.4 Pump

Pumps in TrySim are described by their maximum pressure and the flow. The maximum pressure is specified on the edit mask while the flow is defined by the [drive](#). Up to now there are two drives: the bit pump and the word pump. The drive 'linear mover' that appears in the selection list is actually meant for the [valve](#) but it also functions with the pump and a pump with variable power and motor potentiometer can be simulated.

A pump is automatically connected to a (not bendable) pipe whose properties can be edited by the button [extended](#). You should at least adjust the nominal diameter of the pipe to the actual conditions.

How to connect the ends of the pipe and which sensors can be connected to it can be read in the description of the [pipe](#).

If you define a negative power or if you define a negative set point at the word pump by the PLC the direction of conveyance is inverted.

The pump always pumps the amount specified by the drive as far as it is possible with the adjusted maximum pressure. If the resistance of the following elements is too big only that much fluid is transported as it is possible with the maximum pressure. You can modify the maximum pressure with the help of a [poke](#) by the PLC program. Doing this enter the pressure in kPa (ca 0,01 bar) in the word of the poke, for a max. pressure of 8 bar you have to poke the value 800.

2.4.5.5 Valve

**

A valve limits the flow to a definite value. This value is specified as with the [pump](#) by the drive. In reality many valves or shifters are controlled by jet propulsion. Corresponding to that you can select a [linear mover](#) as drive (that has to be created before). Depending on the position of the hot spot of the linear mover the valve is opened from 0% to 100%.

You should adjust the nominal diameter of the valve in [extended pipe properties](#) corresponding to the actual conditions.

Please note that with the valve type described in place, pneumatic or hydraulic ones, used to the control of drives, are not meant.

2.4.5.6 Overpressure valve

**

This element is acting as a [pipe](#), not capable of folding, that becomes permeable at a pressure adjustable on the edit mask.

You can adjust in the edit mask:

1.) max pressure

The valve tries to limit the pressure between both ends approximately to this value. The overpressure valve behaves in both directions the same.

2.) flow

Here you have to enter with which flow the valve has to expect. Enter rather too much than too little. The TrySim kernel just needs an idea of how much fluid will flow off by the overpressure valve (there are TrySim users who calculate in cubic meter and other who calculate in cubic millimetre).

3.) characteristic line

Hereby you can adjust how exactly the pressure limits wished by you shall work. Valid for standard applications:

1 - 5 Imprecise pressure, but soft in the simulation

6 - 14

See also:

[Fluids](#)

[Valve](#)

[Pump](#)

[Static elements of simulation](#)

2.4.5.7 Check valve

**

This element behaves like a not bendable [pipe](#) that is pervious in one direction only.

You should adjust the diameter in [extended pipe properties](#) corresponding to the actual conditions.

See also:

[Fluids](#)

[Valve](#)

[Pump](#)

[Static elements of simulation](#)

Tabsheet: Fluids

2.4.5.8 Flange

**

The flange is used to connect two or more pipe ends. It will not matter if these are ends of a normal pipe or of a [pipe-like element](#). You connect a pipe to a flange by simply positioning the end inside the flange. You do not need to be very precise since on start of simulation not connected ends are detected by the system and after confirmation they are connected to the nearest flange.

At the internal calculation of the flows through a flange this one is seen as an expansion container that can buffer a certain volume. The value that is marked quite vague as 'factor' on the edit mask determines how 'hard' this expansion container is. Normally you can let this value be on '1'. For some use (especially if it is worked with high pressures) the value has to be raised though to reach a fast reacting to the operating of valves and pumps. But if you make it too big the simulation becomes unstable, all flows and pressures will show a chaotic behaviour then. Naturally we would like to calculate an appropriate value for this factor out of the data of the pipes, pumps and valves that are connected to the flange but we did not manage this yet and that is why you have to adjust this value yourself, should this occasion arise.

See also:

[Fluids](#)

[Container](#)

[Static elements of simulation](#)

Tabsheet: Fluids

2.4.5.9 Flow meter

**

This element has to be assigned to a [pipe](#) or a pipe like element by positioning it on the line. You do not need to be very precise because by starting the simulation all flow meters are checked and should the occasion arise shifted to the next pipe after further inquiry.

The flow meter can measure the flow per time and the integrated flow. If it measures the integrated flow the PLC-Word will only be modified incremental, that means you can overwrite the value with the PLC, e.g., to perform a reset.

If you use [flanges](#) you will have to raise the there described factor eventually to be able to execute a satisfactory flow measurement.

See also:

[Fluids](#)

[Level gauge](#)

[Static elements of simulation](#)

Tabsheet: Fluids

2.4.5.10 Level gauge

**

The level gauge has to be positioned inside a [container](#). It writes the level in the adjustable units and resolution on the edit mask in a word of the PLC

If you select the unit kg the [density](#) of the mixture is also considered, that means the sensor behaves

like a balance.

You can detect the level of fillable dynamics with this element. In this case you should switch off the option 'Warn, if not connected'.

See also:

[Level switch](#)

[Fluids](#)

[Static elements of simulation](#)

TabSheet: Fluids

2.4.5.11 Pressure sensor

**

With the pressure sensor you can measure the pressure in a container or in a pipe like element.

If the pressure sensor is assigned to a [container](#) it will measure the pressure on the height of its lower plane.

If the pressure sensor lies on a [pipe](#) the pressure will be measured by linear interpolation between the pressure at the beginning and the end of the pipe. If it is about a bended pipe in z-direction and the gravity pressure plays a significant part this will be a rough simplification and the results should be viewed with caution.

If the pressure sensor lies on a [pump](#), a [valve](#) or a [check valve](#) the pressure at the nearest end will be reported to the PLC.

If you use [flanges](#) you will have to raise the there described factor eventually to be able to execute a satisfactory flow measurement.

See also:

[Fluids](#)

[Level sensor](#)

[Static elements of simulation](#)

Registerkarte: Fluids

2.4.5.12 Analyzer

**

With the analyzer you can detect the percentage of a [medium](#) in the mixture in a [container](#), a [pipe](#) or a [pipe like element](#).

Which medium is to be detected can be adjusted on its edit mask. The resolution is 1% by default but can be modified as well.

It is not important where the analyzer is positioned inside a container, that means it is not necessary that it is inside the fluid. If you want to analyze the fluid in a pipe it will have to cross the analyzer. If an analyzer is neither assigned to a container nor to a pipe, it will be moved after confirmation to the nearest adequate element when the simulation is started.

You can analyze the fluid in fillable dynamics with this element as well. In this case you should deactivate the option 'Warning, if not assigned'.

See also:

[Fluids](#)

[Static elements of simulation](#)

2.4.5.13 Level switch

**

The level switch is positioned inside a container. It is activated when the fluid reaches its lower edge.

You can use this element to detect the level in fillable dynamics as well. In this case you should deactivate the option 'Warn, if not connected'.

See also:

[Fluids](#)

[Level gauge](#)

[Flow meter](#)

[Analyzer](#)

[Static elements of simulation](#)

2.4.5.14 Fluidor

**

The fluidor transforms dynamics in fluids. You have to position it in a [container](#).

See also:

[Fluids](#)

[Static elements of simulation](#)

2.4.6 Misc.

2.4.6.1 Saw

With the saw you can do two things:

- 1.) Cutting off of a piece of the track that is created by the [continuous generator](#).
- 2.) Cutting off of a piece of a dynamic (that is not turnable).

Cutting off of a track

You can only cut crosswise to the track. The cut off piece will become a dynamic which properties can be adjusted on the edit mask of the continuous generator in 'dynamics'.

So that the saw has got any effect to the track two prerequisites have to be met: 1.) it has to overlap with the track and 2.) the PLC-bit of the saw has to be on '1' (or the check box 'always 1' has to be clicked). Just if both prerequisites come true the saw will be called 'active' in the following.

To do cuttings you have to move the saw through the track. The point with that the saw cuts actually is exactly in its middle. A cut is started as soon as the saw gets active for the first time. The piece is cut off when the first and the last point of intersection lay on opposite sides of the track. Internally seen a cut is completely done not until the saw is not active anymore. The piece can already be cut off at that time but a new cut is only started after the saw has been inactive at least for a short time. If the saw gets inactive before the cut has been done it will be turned down that means that you cannot stop and continue the started cut later on.

You can cut a track with two orientations. You can cut a track that goes eastwards for example from the top to the bottom or from south to north. By the direction the track is cut completely by the saw for the first time you adjust in TrySim with which orientation it is cut.

To cut from the top to the bottom you have to make the saw a little wider as the track. To cut from south to north it is enough to make the saw a little thicker than the track.

Even if you do a cut that is not straight the created dynamic as well as the rest of the track will be rectangular.

Cutting off of a piece of a dynamic

Essentially this is the same as cutting off a track. Merely there is a little different rule for the direction of cutting and orientation, because there is no generator with a dynamic, that gives a direction. This rule will be the easiest to describe if you shape the saw like a bar and imagine it is the blade of a compass saw: On the side which gets a first cut that extends the whole width the cut starts. It stops not until the saw blade comes out the opposite side. The orientation of the cut happens automatically then.

This element is still at the experimental stage. Suggestions out of practice for improvement are welcome.

To divide a dynamic (not a track) into equally pieces you can use the [divider](#) as well.

See also:

[Common properties](#)

[Continuous generator](#)

[Static elements of simulation](#)

Tabsheet: Misc.

2.4.6.2 Divider

If the bit of the divider has got a rising slope it will cut all dynamics it overlaps with in the number of smaller dynamics specified in the edit mask. You can give the amount of the cuts per direction separately.

The restriction to the rising slope is necessary, because the divider would otherwise divide the newly created dynamics (at least the ones with which it still overlaps after the dividing) into quarks.

This operation can be undone with the [melter](#), for example. You can use the [wedge](#) to separate the parts.

If you use the divider with [turnable dynamics](#) please keep in mind that this dynamics cannot stand on each other yet.

The sample 'tree saw' demonstrates the use of a divider.

The symbol of the divider has changed with version V2.9.8 because the old symbol goes better with the new element [saw](#), with that pieces of any width can be cut off of a dynamic.

See also:

[Common properties](#)

[Generator](#)

[Static elements of simulation](#)

Tabsheet: Misc.

2.4.6.3 Cutter

You can specify, from which of the six sides of the dynamic shall be cut off.

See also:

[Common properties](#)

[Static elements of simulation](#)

Tabsheet: Misc.

2.4.6.4 Melter

When the bit of the melter is '1' it combines all dynamics it overlaps with to one single dynamic. The block will be made so big that it contains all melted dynamics. That means that the created block can be bigger than the sum of all dynamics, if these have not been positioned in the cuboid-shaped.

This operation cannot be undone, because it is internally executed so that one of the dynamics gets blown up to the new size while the others are deleted. But you can cut the new dynamic with the [divider](#) into parts.

The sample 'stacker' demonstrates the use of a melter.

See also:

[Common properties](#)

[Generator](#)

[Destroyer](#)

[Static elements of simulation](#)

TabSheet: Misc.

2.4.6.5 Wedge

**

The wedge separates two dynamics standing close to each other. To display its wedge effect you have to stretch it oblongly.

The wedge is useful after the application of a [divider](#) or a [saw](#).

If you want to make a gate out of wedges, the distance between two wedges has to be bigger than the things you want to divide. If the things get stuck, you should zoom in on the corresponding position. The limited resolution of monitors can deceive!

The samples 'tree saw' and 'switch' demonstrate the use of wedges.

The wedge is in need of improvement, of course, it hardly functions with the [turnable dynamics](#).

See also:

[Shifter](#)

[Static elements of simulation](#)

[Common properties](#)

TabSheet: Misc.

2.4.6.6 Shifter

**

With the help of this element you can shift and stop dynamics. It is a passive element without connection to the PLC and therefore easier to handle as the [hook](#) that is more flexible on the other way.

The shifter moves dynamics in the XY-plane only. You can take advantage of this feature by pulling

it up or down.

In some cases, a [wedge](#) is more suitable to manipulate dynamics in a passive way.

The samples ‘assembly way’, ‘stacker’ and ‘sorting way’ demonstrate the use of shifter and hook.

The shifter functions with [turnable dynamics](#) as well. Please note that these elements are mainly defined by their edges, so take care that the shifter hits as many edges as possible, eventually you have to activate the centre-points and the edge-points or you should restrict the turnability of the dynamics to the XY-plane.

See also:

[Common properties](#)

[Static elements of simulation](#)

Tabsheet: Misc.

2.4.6.7 Dynamic converter, straighter

The new [turnable dynamics](#) in version 2.1 of TrySim have not got most of the new properties of the [normal dynamics](#) yet. Consequently they need a great deal of computing time. That is why we have created this element to transform one type into another dependent on which properties of the dynamics are needed at the moment. The converter can also change the behaviour of the dynamics.

Change turnable into normal

You can specify the direction of the conversions on the edit mask. All dynamics which contain the lower left edge of the converter are converted. If a dynamic which is turnable and not standing right-angled is to be converted the normal dynamic will be created which can be got by the smallest turning. To avoid the unnatural jumping of turnable dynamics which are in a very sloping position you can enter a maximum angle on the edit mask. If the dynamic is more sloping than this angle it will not be converted.

Change normal into turnable

If you convert normal dynamics into turnable ones you will be able to specify, just like the creating by a [generator](#), whether the plane and/or the edge points of the turnable dynamics shall be considered as well or whether the turnings in the XY--plane shall only be possible.

Straightening

In reality sliding turned dynamics (e.g., on an [arc belt](#)) will be straightened right-angled to the transport direction by guidances after the turning. The reproduction of such guidances is very costly concerning the computing time effort as well as the effort of creating the simulation. Because the dynamic converter has already got the ability to change a dynamic which is in a sloping position into a dynamic which is standing right-angled you can also use it to straighten turnable dynamics which are a little bit sloping into just the right-angled position.

The example ‘turns’ demonstrates this use.

Change properties

These options are originally created for acceleration of machines with very many dynamics. If you are transporting 500 dynamics on a cooling section for example, with which nothing happens there except that they are transported then it is a waste of computing time to control in every simulation step whether they are covered by the existing light barriers in the machine. Nearly the same is with hooks which have to check whether they overlap with one of these dynamics. Even the drawing of so many elements needs much time, especially in the 3-D view. Lots of computing time is needed for the collision monitoring of the dynamics among each other, because each of the dynamics has to check whether it does not collide with any other one. If you have got lots of dynamics you can switch following properties off by the help of the dyn-converter and switch them on with another dyn-converter:

Visibility: If a dynamic is made invisible it will not be displayed in any window anymore. If you have lost track of things how many of the now invisible dynamics are still in the machine you will be able to display them again with **Machine|Dynamic|Make All Visible**. But this operation cannot be cancelled.

Collision monitoring: With the option 'make inactive' you deselect the ability of the dynamics to shift others or to stand on others.

Deactivate hook: If this option is selected the dynamic will not be able to be grabbed by hooks and automatic hooks anymore.

Deactivate sensors: By this the dynamic is made invisible for light barriers, distance sensors and bar code scanners.

See also:

[How to select turnable elements](#)

[Common properties](#)

[Static elements of simulation](#)

Tabsheet: Misc.

2.4.6.8 Press

**

The press is used to make dynamics smaller. It works always in the direction of its smallest extension, that means you have to change its shape so that it becomes flat. It can be used as plane as well.

Do not let a dynamic fall down onto a press, because otherwise it will be pressed so much that you will not be able to see it anymore.

The sample 'assembly way' demonstrates the use of a press.

See also:

[Divider](#)

[Saw](#)

[Static elements of simulation](#)

[Common properties](#)

Tabsheet: Misc.

2.4.6.9 Automatic hook

**

This special form known in crane technique does not need any external triggering to catch and release a dynamic. It will automatically latch and unlatch. If it sinks on a dynamic it will latch. When the dynamic is deposited after the transport it will unlatch. The function works like a ball pen: the first pressing makes the cartridge come out, the second makes it disappear.

A bit with the function 'rope is loose' will report to the PLC if the next upward going causes the shifting of the state of catching.

Please note that the bit does not mean whether a dynamic hangs on a hook. It just means that the rope is loose, that means that not even the hook is hanging at the rope.

And anyway there is another bit with the function 'Load hangs on the hook'.

See also:

[Common properties](#)

[Static elements of simulation](#)

[Hook](#)

Tabsheet: Misc.

2.4.6.10 Sticker

If activated the sticker will stick at the first element that meets it. On the edit mask you will be able to adjust if it sticks at dynamics, semi dynamics or statics. Every combination is possible, too. If the PLC bit to which the sticker is connected to is turned off you will be able to choose between two possibilities: 1.) The sticker remains at its current position. 2.) It jumps back to the position it was picked off. If the sticker sticks at a dynamic that is going to be deleted it will jump back to its starting position in any case.

In combination with an [attractor](#) it is, e.g., possible to track the way of a dynamic through the machine. This is demonstrated in 'sample 1' as well.

You can use the [pawl](#) to activate or deactivate a sticker as well.

See also:

[Common properties](#)

[Static elements of simulation](#)

TabSheet: Misc.

2.4.6.11 Pawl

With the help of this element you can switch on or off a [hook](#) or a [sticker](#).

Usually a hook is directly controlled by a PLC bit. However, if there are many hooks (e.g., at a [chain](#)) that have no own identity but are only distinguished by their [position](#) it is very difficult to identify the appropriate bit. The pawl solves this problem by copying its own state to every hook it overlaps with.

Of course the bits of the hooks must be all different and unused that means that they must not be headed for by the PLC. It is most convenient to shift them to high addresses, e.g., A °10000.0, --.1, --.2 and so on. Also the bits of the hooks have to be all different, otherwise all hooks will be switched on/off at the same time.

The sample 'tree saw' demonstrates the use of pawls. The hooks will be activated by the first pawl if the light barrier is covered (I 0.1, click 'addr' within the edit mask of the pawl to check this out.). The second pawl deactivates the hooks.

See also:

[Common properties](#)

[Static elements of simulation](#)

TabSheet: Misc.

2.4.6.12 RFID antenna & data chip

**

These two elements are used together.

The data chip is passive, it is intended as a memory for data up to 10 000 bytes. This data is also stored with the machine.

The antenna communicates with every data chip with which it overlaps.

See also:

[Statistic elements of simulation](#)

TabSheet: Misc.

2.4.6.13 Box

*

This element is the simplest one. It consists of a square, whose size and color can be defined by you. Like all the other elements the box has got a father. The father is any other element the box is connected to. If the father is moved, the box will follow him.

Boxes chiefly serve to structure a machine graphically and logically. For example, you can create a box and name it 'desk'. If you define this desk as father for all the push buttons and indicators, you will be able to shift all these elements by only one move of the mouse later.

Boxes are able to interleave each other. Boxes are able to activate [limit switches](#). This requires their selection as master within the edit mask of the limit switch.

If you need a storage surface for [dynamics](#), please use a [board](#) .

Boxes can be made turnable. Therefore select the check box 'turnable' within the edit mask. Please read the [general information about turnable elements](#).

Boxes cannot be moved by [conveyors](#) or [hooks](#), this is only possible by dynamics which are created by a [generator](#).

See also:

[Static elements of simulation](#)

Tabsheet: Misc.

2.4.6.14 Board

*

The board is similar to the [box](#) in every respect, indeed you may deposit [dynamics](#) on it. Therefore it will be sufficient if only one corner of the dynamic stands on the board. This certainly does not look like reality but with regard to the simulation it is of great use. When a board has been created, it actually has got the form of a plate, but that should not prevent you from forming a block or a bar out of it.

Note that at least one **corner** of the dynamic is deposited on the plate, otherwise the dynamic will fall through the plate or it will be put over the dynamic.

The plate can be made turnable. To do this select the check box 'turnable' on the edit mask. Please read the [general information about turnable elements](#).

[Common properties](#)

[Static elements of simulation](#)

Tabsheet: Misc.

2.4.6.15 Bar

**

With the bar you can connect two elements. One of the elements is the father, the other one you have to assign as master.

Normally the middle parts of the two elements are connected. But you can choose that the corners which are nearest to the origin (bottom left) get connected to each other by the deactivating of the check box 'middle'.

The bar has got no influence on the simulation, it is only for better representation. So it is not possible to pull or push an element with the help of the bar. The length of the bar fits automatically to the distance of the two elements.

See also:

[Common properties](#)

[Static elements of simulation](#)

Tabsheet: Misc.

2.4.6.16 Rider

See [rider way](#).

2.4.6.17 Reactor

The reactor can be used to convert media inside a container into other media. On the edit mask you specify one or two source products and one or two end products. The percentage of the every first medium of the reaction can be adjusted by you, the percentage of the other medium is added automatically to 100%.

The speed of the reaction is specified in milli liter per second, liter per minute or cubic meter per hour. If you select 'concentration dependent' the speed will be decreased if the source products are not present in sufficient concentration. Example:

You have selected medium 1 with 60% and medium 2 with 40% as source product as well as a reaction speed of 100 l/min. Assume in the container is the percentage of medium 1 30% and the one of medium 2 is only 10%. Then the reaction speed will only be: $100 \text{ l/min} * 30/60 * 10/40 = 12.5 \text{ l/min}$.

If 'Concentration dependent' is not selected the reaction will run with the selected speed as long as both source media exist in sufficient amounts.

You can adjust whether the reaction will have to take place in any case or only if the reactor is actually in the fluid.

A volume modification that is combined with the reaction can be considered as well. E.g., if 6 units of A and 4 units of B result in 9 units of C you will have to enter 90% in the corresponding field.

You can carry out a reaction with these elements in fillable dynamics as well. In this case you should deactivate the option 'Warning, if not connected'.

See also:

[Fluids](#)

[Static elements of simulation](#)

2.4.6.18 Peek

These are special elements which do not have any correspondence in reality, but serve to make TrySim more flexible. Peeks have the opposite function as [pokes](#) and are handled the same. You can let copy properties of the elements like position and size into a PLC-word to analyse them in the PLC-program or at other places in the machine.

To use a peek you have to act like this:

- Create a peek (tabsheet: Misc.)
- [Fix](#) it onto that element whose properties you are interested in.
- Select the data type of the interesting size on the edit mask of the peek.
- Select the desired property on the edit mask of the peek in 'value'.
-

[List of Peekable Properties](#)

Tabsheet: Misc.

2.4.6.18.1 List of peekable properties

The list is being extended at the moment (as well as immediately by inquiry!)

Bit-Peek

All elements with drive: drive runs (speed \leq 0)

Word-Peek (unsigned 16 bit number)

All elements: Position X,Y,Z and size X,Y,Z
or from X,Y,Z and to X,Y,Z

Joint: current angle (the unit is 0,1 degree)

Servo drive : position (in the preset units)

2.4.6.19 Poke

ATTENTION! If your projects do not have to be loaded by older TrySim versions (this is mainly the case in schools) please use the element that is just called 'poke'. The elements which are called 'poke word' and 'poke dword' are still existing for a transitional period because of the compatibility. They will not appear in the element list in a later version. If projects still contain these elements then they will be transformed automatically in the new element while loading.

These are special elements that have no correspondence in reality, but are used to make TrySim more flexible.


With the help of a poke you can modify several properties of the elements by the PLC. In the PLC program you give a word / double word and the value will be passed to the father of the poke element as if you enter it on the edit mask. Which field you want to poke is defined by the selection list 'Field'.

The poke elements pass a value to their father if it (the value) has been **changed** by the PLC. To program this in the PLC more easily you can click the check box 'reset to poke to:' and specify an appropriate reset value. After each transfer into the word / dword the poke will now be executed and the value of the word /dword will be set back to the reset value. You can select this value absolutely arbitrarily, but please note that it is outside the range in that you want to poke. If you choose this option it will be better to make sure that a value is only transferred in the word / dword if you really want to change the property of the element, because each transfer into the word / dword means a changing now, (apart from transferring the reset value, naturally). You must not use this option if the project is to be opened with TrySim versions below V2.9.10.

Please note that you must not alter the units of measuring (mm, cm, etc.) anymore while using pokes (or you have to adjust the poked value in the PLC program correspondingly).

The poked values are given nearly unchecked to the element. That means that the use of pokes needs an appropriate care, otherwise you will get unexpected results.

Most of the properties are expected as unsigned 16-bit-number (word-poke), not many (e.g., the hot spot of chains) are expected as unsigned 32-bit-number (Dword-Poke), so you have to select the element correspondingly. [Click here to get a list of pokeable properties.](#)

If you want to know for a special element whether some of its properties are poked just open its edit mask and click the symbol . Now you can see whether pokes are one of the children of this element.

The sample 'formula1' demonstrates the use of pokes.

See also:

[Common properties](#)

[Static elements of simulation](#)

Tabsheet: Misc.

Remark: The name of this element comes from a command of BASIC.

2.4.6.19.1 List of pokeable properties

The list is being extended at the moment (as well as immediately by inquiry!)

Word poke (unsigned 16 bit value)

All elements: position X,Y,Z and size X,Y,Z
resp. From X,Y,Z and To X,Y,Z

Generator: size of dynamics X,Y,Z

Master switch: position

Servo drive: max. speed and slope

Bit drive of the joint: speeds and slopes

Reset button of the thermal mass (1 = Reset)

DWord poke (unsigned 32 bit value)

Chain : hot spot

Node : hot spot offset

[Return](#)

2.4.7 Thermal elements

2.4.7.1 Thermal mass

The thermal mass consists internally of a rectangular arrangement of cubes. Inside these cubes the temperature is considered as constant. You can adjust the number of division on the edit mask. The heat flow between the several cubes takes place with the 'inner conductivity' that has to be adjusted as well. You can supply power to the thermal mass by a [heating](#) or you can cool it by specifying a negative heating power. The [thermometer](#) is used to detect the temperature.

The model which is used for the thermal mass is the simplest one that gives quite realistic results. It just considers the heat capacity and the heat conduction. As the energy transport mainly happens by

convection in reality which would be quite costly to simulate, an effective value has to be specified for the conductivity that can only be determined by trial. It has, like so often in TrySim, never been our aim to recreate physical processes in detail but to reduce them to the essential part that is important for the PLC programmer.

With the button reset you set the thermal mass back to the environment temperature on the edit mask. You can also give the reset a push by the PLC with the help of a [word poke](#). If the value 1 is poked the thermal mass will be set back. If you select 'reset to poke' on the edit mask of the poke and you keep the reset value of zero you will only have to set the LSBit of the word to '1' for the reset, e.g. from MW 1200 the bit M 1201.0.

See also:

[Static elements of simulation](#)

TabSheet: Thermal elements

2.4.7.2 Heating

**

With the heating you can supply energy to or take energy from a [thermal mass](#) at a specific place. (Remark: in V2.6 the temperature of the [fluids](#) is still constant.)

The heating has got a [drive](#) that defines the heating power. A negative heating power means a cooling. Currently there are three drives for the heating: the bit heating that you can only switch on and off, the bit word heating whose heating power can also be specified by the PLC and the drive by a linear mover. With this one you can simulate a mechanical working cooling shutter, for example, or it is possible to recreate the loss of energy of a heating when you open the doors.

The heating power of the drive is weight with the efficiency that can be modified dependent on the temperature with the help of a [characteristic curve](#). By this it can be considered that a 1000° C hot flame cannot supply energy to a workpiece that has got the same temperature.

See also:

[Thermometer](#)

[Static elements of simulation](#)

TabSheet: Thermal elements

2.4.7.3 Characteristic curve

This element is needed to convert one quantity into another. Currently it is only used for the efficiency of the [heating](#) dependent on the temperature but we are going to extend it to other applications.

You create a characteristic curve either just like a normal element or you select 'New Characteristic Curve' out of the list of an element which can conduct characteristic curves. You can also call the

edit mask of the characteristic curve by double clicking the characteristic curve field of such an element.

When creating the characteristic curve, you have to enter the number of the desired x-axis-points and also the XMin- and XMax-values. Then you select the points which need to be edited with the slide controller above the graphic and enter the corresponding values into the field 'Y-'. If you confirm the input with 'Enter', the slide controller will jump onto the next x-point automatically.

This element is not ready yet, but it will do most of its job at least.

If the edit mask cannot be opened the data gsw32.exe, gswdll32.dll and graph32.ocx will not be existing or they will not be registered. Unfortunately we cannot supply you with these data by this version.

[Static elements of simulation](#)

TabSheet: Thermal elements

2.4.7.4 Thermometer

**

The thermometer is used to detect the temperature at a certain position inside a [thermal mass](#). (In V2.6 the temperature of the [fluids](#) is still constant.)

You can select the units between two possibilities on the edit mask.

If the thermometer is not placed inside a thermal mass it will not be able to give any values, which means it does not influence the connected PLC-word.

If you just need to check a single temperature you will be able to use the [thermostat](#) instead.

See also:

[Heating](#)

[Static elements of simulation](#)

TabSheet: Thermal elements

2.4.7.5 Thermostat

**

With the thermostat you can detect the temperature at a specific place inside a [thermal mass](#) like you do it with the [thermometer](#).

But the thermostat just gives a binary value 'temperature exceeded'.

You have to adjust the temperature on the edit mask as well as the hysteresis.

The thermostat will switch on if the specified temperature is exceeded. It will switch off if the temperature falls below this limit minus the hysteresis.

Normally the thermostat is set on 'NC - contact', that means that it will give '0' if the limit of the temperature is exceeded. But you can change it into 'NO - contact'.

If the thermostat is not inside a thermal mass it will not give any value, that means that it does not influence the connected PLC-bit.

See also:

[Heating](#)

[Static elements of simulation](#)

Tabsheet: Thermal elements

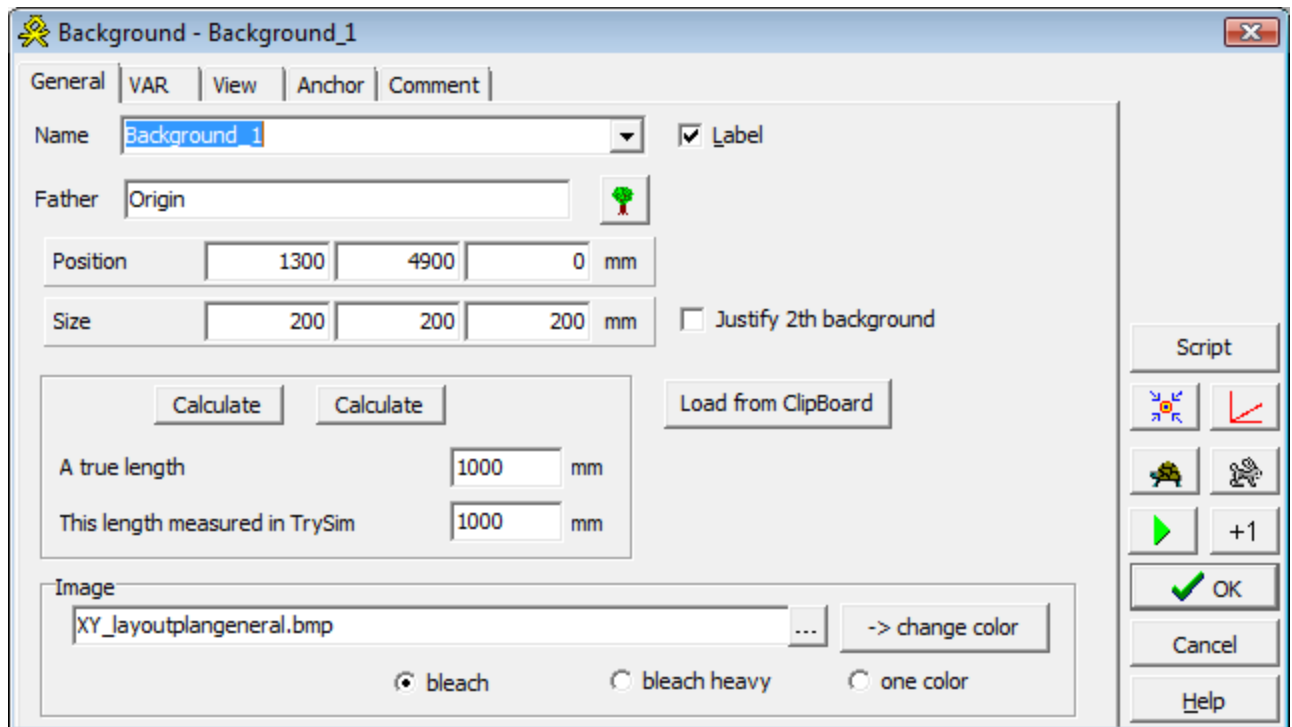
2.4.8 Tools

2.4.8.1 Background

**

You can underlay the background of the **2-D** windows with any graphic. This can be helpful if you take the layout plan in case of a bigger machine.

You generate the graphic with a graphics software of your choice. We have provided in TrySim just the modification of the color as an editing. But you have to know which width and length your graphic has in the **real machine**. Exactly these values (in mm) you enter in the edit mask. You can also use the calculation aid explained below. You can copy the externally created graphic as *.bmp file in the project directory. If you have placed the graphic into the clipboard, for example with the snapshot function in the adobe reader, you can insert it by means of 'load from clipboard' directly into TrySim.



Calculation aid for size:

- You will know the real size of your machine approximately. Firstly enter these values for the two corresponding coordinates and close the mask.
- Then search a component as large as possible whose size you know exactly, for example by dimensions, on the picture.
- For this component you determine a length as it currently measured in TrySim. The easiest way is to place a temporary box above. The size of this box for example is shown in the status bar of TrySim.
- open the edit mask again and enter the real length and the just measured length.
- Then click on that 'calculate button' which matches the measured direction.
- Now this size is adjusted first, thereafter the other one so that the width/length proportion correspond to the proportion of the bitmap.
- You can adjust the size of the machine (white area) via Extras|Options|Machine.

It is not possible to modify the background with the mouse intentionally. To change size and position you reach the edit mask via the [element tree](#) or the 'ENTER' key while the background is still marked.

Important! If there are problems with the calculating time during the simulation you should fade out the backgrounds first.

See also:

[Common Properties](#)

[Static elements of simulation](#)

TabSheet: Tools

2.4.8.2 Stripe

**

While editing you can fade in and fade out parts of the machine with the help of the stripe. Only the elements that are inside the stripe are displayed. But you can also invert the stripe, then only the elements are displayed which are outside.

When you create a stripe it is inactive at first. After positioning and sizing, it must be activated. Open the edit mask and activate the check box 'active' or double click the border of the stripe while the simulation is stopped.

You can restrict the effectiveness of the stripe to certain directions of view.

If you have several stripes all elements that are at least inside one stripe are displayed. Several stripes are always activated and deactivated at the same time, that means if you activate a stripe all the others will be active as well automatically.

The stripes are designed as editing tools only and they are not optimized for run time. If you want to simulate a big machine, you should delete them, when your machine is finished.

When you create new elements while a stripe is active, you should position a [cross](#) inside the stripe. Otherwise the new elements will lie in the invisible part of the machine.

Stripes are comparativeless easy to handle, but if you construct a very large machine you should think of making use of [groups](#) and [graphic filters](#).

See also:

[Visibility](#)

[Common properties](#)

[Static elements of simulation](#)

TabSheet: Tools

2.4.8.3 Cross

*

The editing of the machine is only possible in two-dimensional windows. If you place a new element there you cannot make out along the line of vision where the element will end up.

When creating new elements, you can determine the value of the invisible coordinate with the help of the cross. In this way you avoid that the new elements lie on the floor resp. stick onto the wall.

Before constructing a new part of your machine, you should position the cross at least in two different views in the near to the building site.

There can only be one cross at a time. When you create a new one, an existing cross will be deleted automatically. If you have fixed elements to the current cross (what is not recommended) these will be fixed to the father of the cross (usually the origin).

If you plan to create new elements inside a [stripe](#), you shall use the cross in every case.

Even without the cross you can roughly determine the invisible coordinate if you [assign the new element to a father while creating](#).

The cross can also be used as reference point to enter the coordinates of other elements relatively to a place in the machine. To do this click the check box 'as reference point'.

In opposite to all the other (not turnable) elements the 'position' of the cross does not call the corner on the bottom left but the centre (there where the lines cross).

See also:

[Common properties](#)

[Static elements of simulation](#)

TabSheet: Tools

2.4.8.4 Attractor

The attractor has its name because it attracts the views onto it. The position of an attractor will be displayed in the middle of a graphic window, even if the attractor moves.

Because of that it is possible to adjust a heavy zoom and to have the right detail in a critical moment nevertheless. In 3-D graphic the attractor can be used to make a kind of movie of the function of the machine with little effort.

In **View|Attractor** you have to release each window separately that shall look out for attractors. If all windows followed the attractors no sensible working would be possible anymore.

Attractors can be connected to a PLC bit that activates them. Several attractors can be used to switch between different details. If you have more than one attractor you will have to take care that only one is activated at a time, otherwise it is indefinite which attractor is accepted in the end. If you plan to activate attractors with the help of buttons, you should use the [multiple switch](#). Do not forget to deactivate the check box 'always 1'.

An attractor can only be activated in the views that are selected on the edit mask in 'View in'.

With the help of a [sticker](#) you can attach an attractor to a dynamic and follow its way through the machine. The 'sample 1' demonstrates this use.

See also:

[Editing tool stripe](#)

[Store and load 3D viewpoints](#)

[Common properties](#)

[Static elements of simulation](#)

TabSheet: Tools

2.4.8.5 Speed trigger

**

With the help of this element you can modify the simulation speed automatically. By specifying a bit to the PLC you can activate slow motion resp. quick motion.

There are four trigger possibilities:

- 1.) Rising edge. Here the speed will be set to the selected value once at the rising edge of the bit.
- 2.) Falling edge: similar to 1.)
- 3.) 'Always on 1': Here the speed will be set to the specified value at a rising edge. If the speed is not changed till the falling edge (by another speed trigger or by the rabbit/turtle), the old speed will be set again when the signal disappears.
- 4.) 'Always 0': like 3.)

You can also select that the simulation is to be stopped completely as soon as the trigger conditions occur.

The sample 'osci' demonstrates the use of a speed trigger.

See also:

[Slow motion](#)

[Breakpoints](#)

[Common properties](#)

[Static elements of simulation](#)

TabSheet: Tools

2.4.9 Master List

Some elements need besides the father another element (master) to fulfill its function.

See also:

[Limit switch](#)

[Spy](#)

[Bar](#)

2.5 General information about turnable elements

Most of the elements in TrySim are not turnable yet, but we work on it to make them turnable bit by

bit. The inclusion of turns makes the editing and monitoring much more difficult. You should only use turns if they are absolutely necessary and if you have already gained experiences with TrySim. As the turns need a lot of computing time, a fast computer is necessary.

In the editing mode you can turn a turnable element by clicking it for quite a long time with the right mouse button. Then the context menu will appear and you can select the point 'Turn'. The small window which appears has three slide controllers for turning around every axis. With the button 'standard' you can recreate the unturned position.

You will only be able to change the size of a turnable element with the mouse, if it is straightened rectangular. The reason for this is that otherwise there is no clear assignment of a two dimensional mouse movement to a three dimensional size changing.

In contrast to the not turnable elements at which the 'position' specifies the edge at the bottom left, the center of the element is displayed on the edit mask at the turnable ones because this is the only point that stays unchanged while turning.

Some elements are turnable right from the beginning, these are:
[Light barrier](#), [linear mover](#), [joint](#), [turnable conveyor](#)

Some elements can be made turnable on the edit mask with the help of the check box 'turnable'. These are:

[Box](#), [board](#), [limit switch](#), [limit switch nose](#), [hook](#)

Caution! If you have made one of these elements turnable once, you are not able to change it into a non-turnable again.

[Marking of turnable elements](#)
[Static elements of simulation](#)

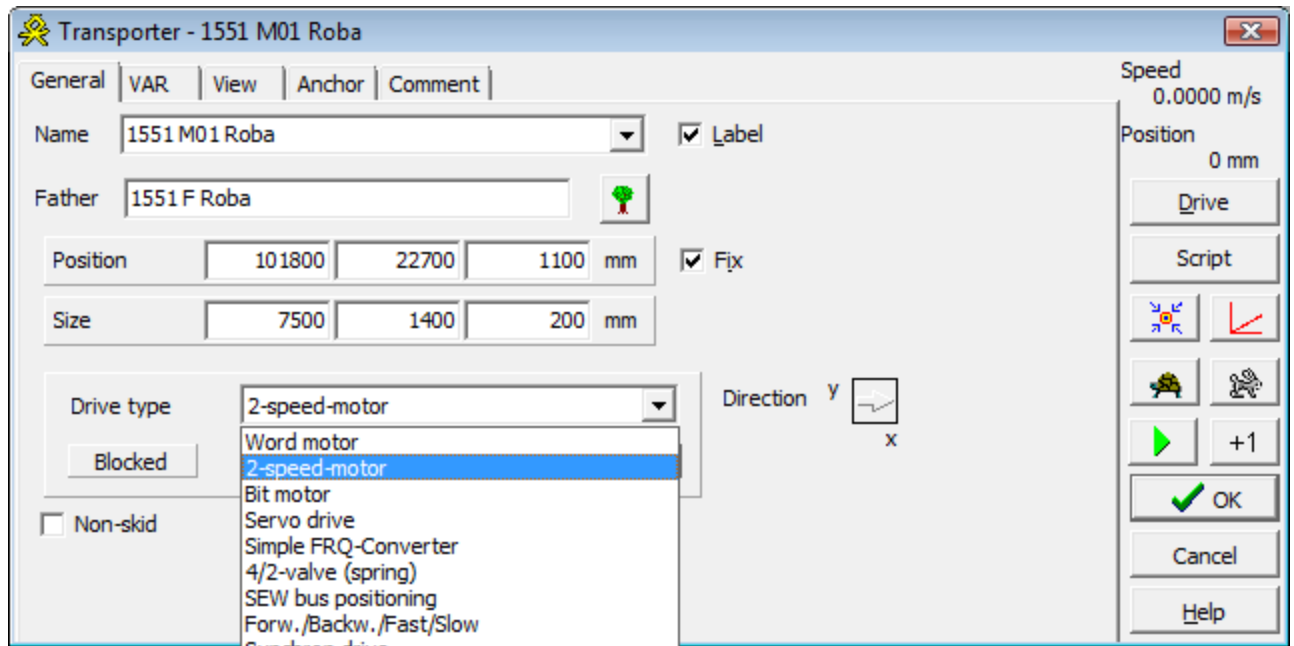
2.5.1 How to mark turnable elements

To mark elements that have already been turned you have to click the (invisible) border of the rectangular that encircles the element. The easiest way to do this is to click narrowly inside a corner of the element.

See also:
[Select elements](#)

2.6 Drives

*



A drive exists never alone but is always a part of another element such as [linear movers](#), mover or [conveyor](#). A drive translates outputs of the PLC into speeds or positions.

There are several kinds of drives:

- **Bit motor**

This drive can go forward and backward on the same speed. You can either enter one or two PLC outputs. You can also select forward/backward with pulse response. In this case the drive behaves like a 5/2-way valve with pulse response.

- **Bit motor with two speeds**

Here you can enter another bit 'fast' and corresponding to that another speed.

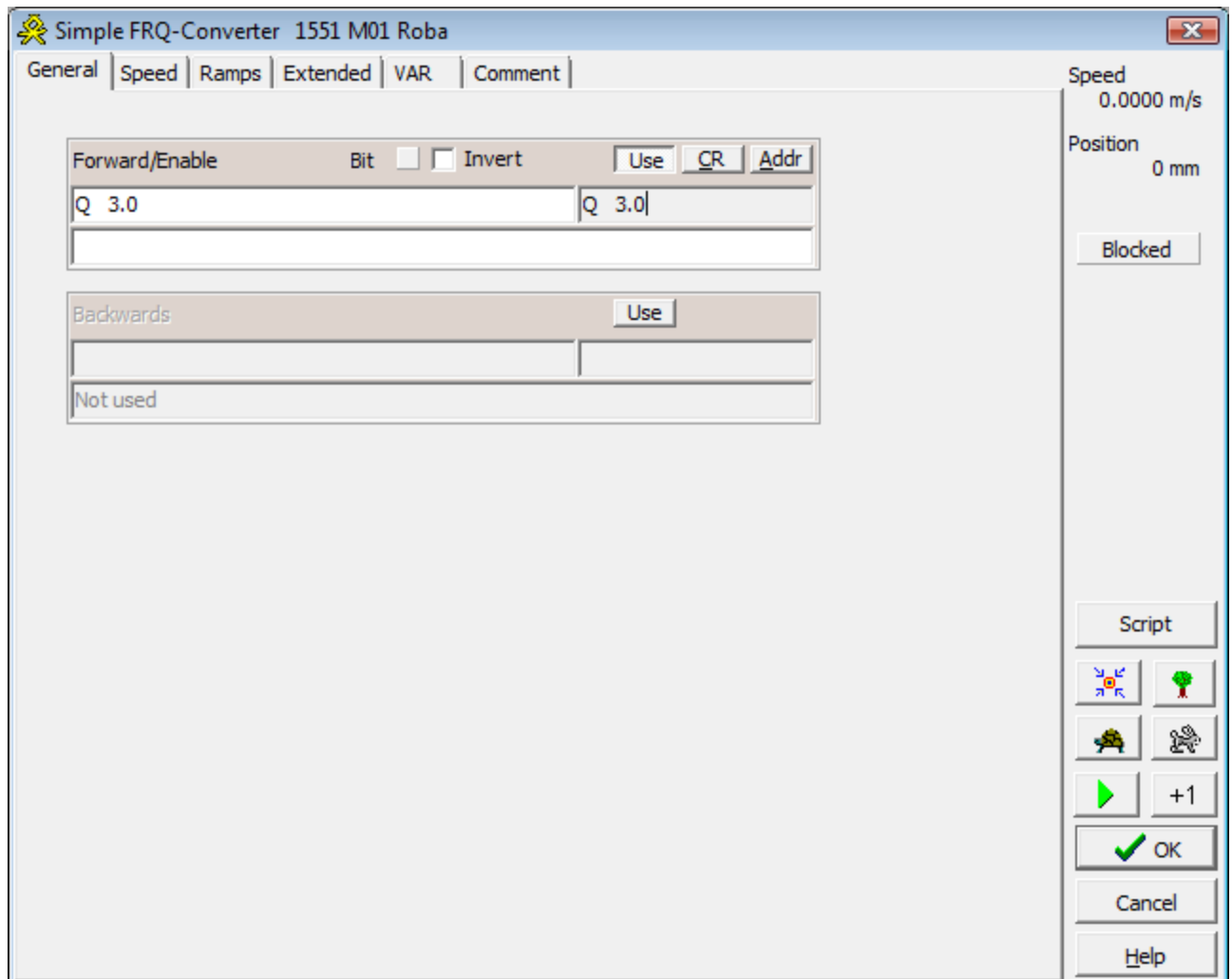
- **4/2-way valve (spring returned)**

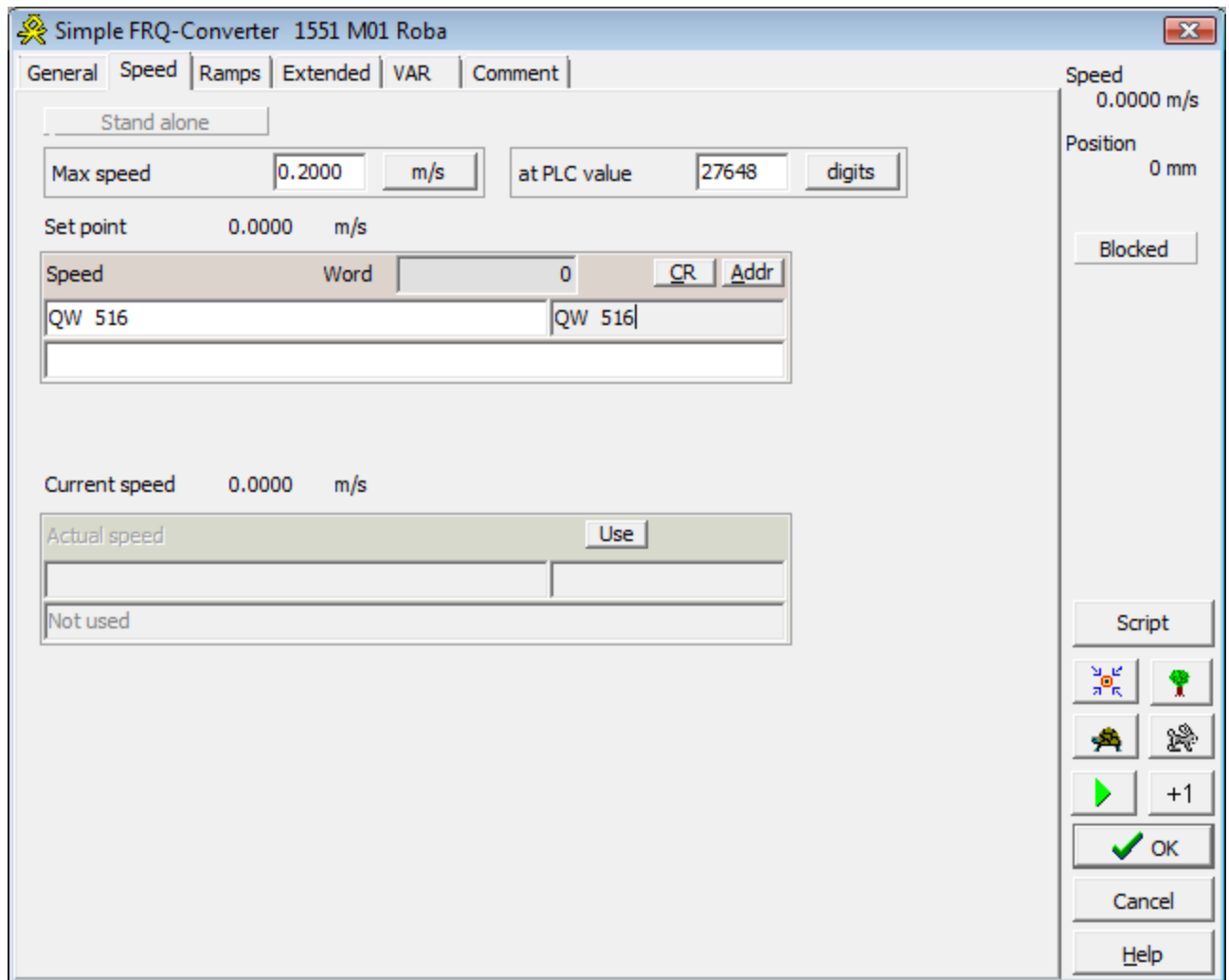
Here you only have to specify a PLC output, if it is specified the drive will go forward, if not the drive will go backward.

- **Forward/Backward - Fast/Slow**

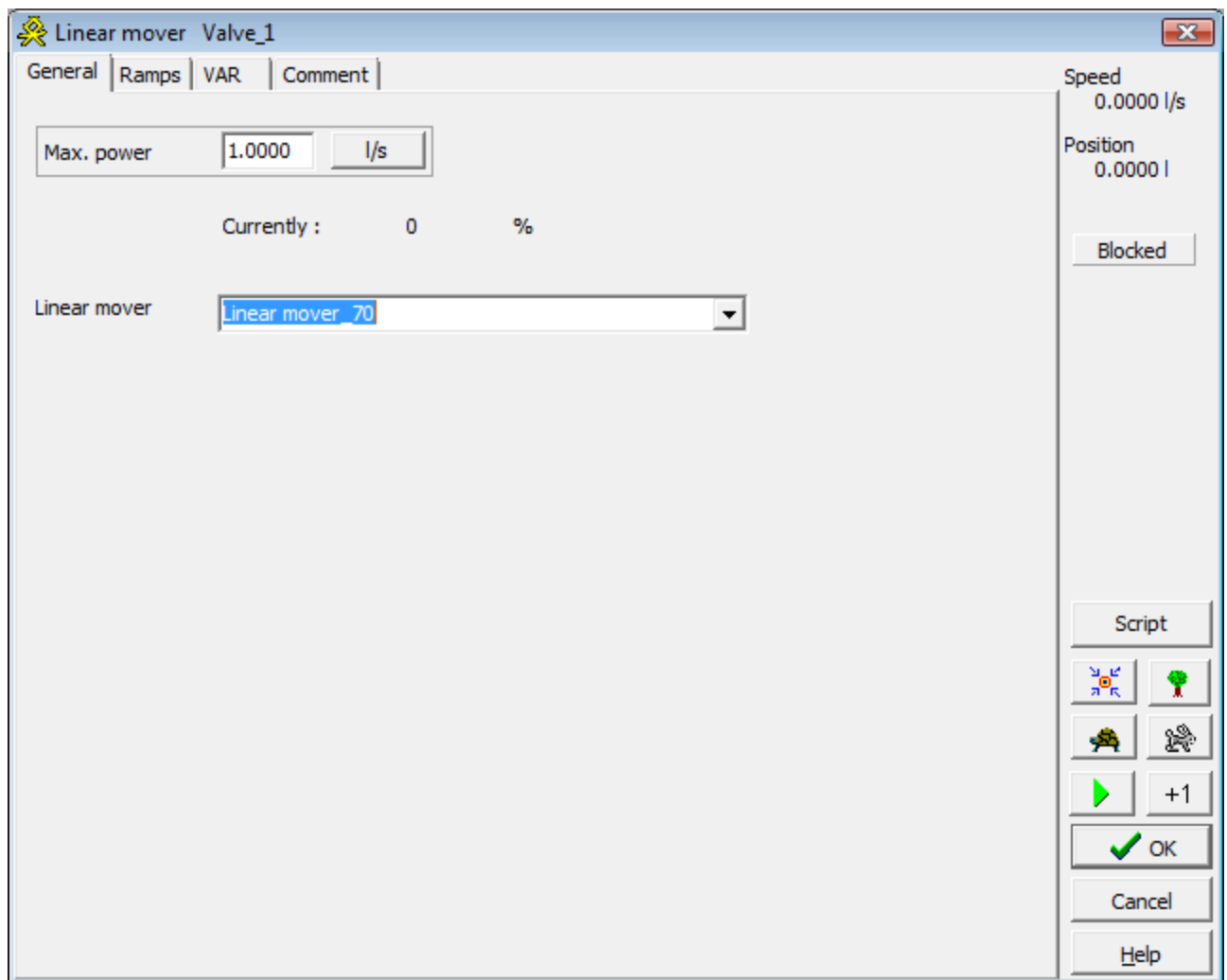
Here direction and speed are specified by four bits.

- [Simple frequency converter](#) with variable speed

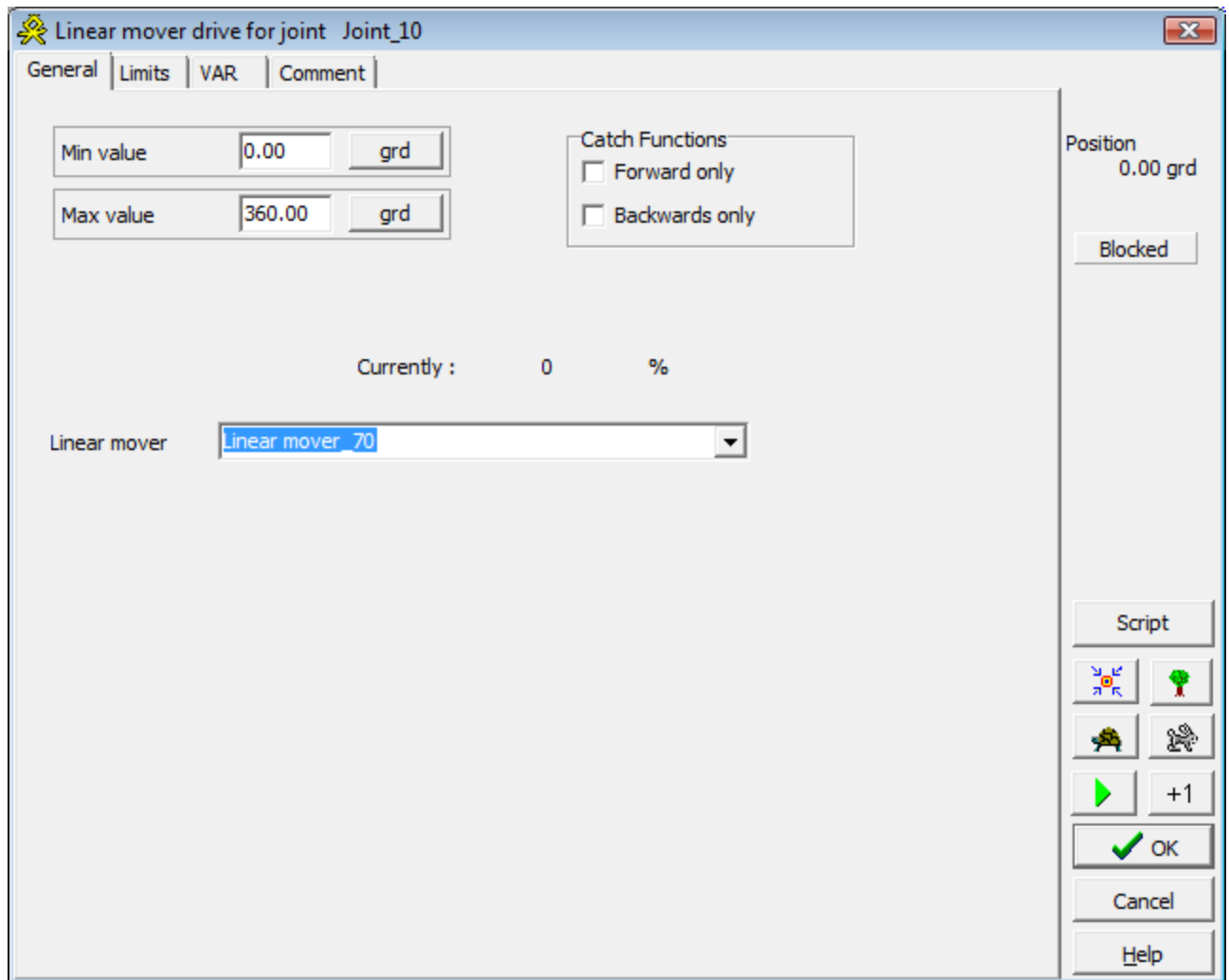




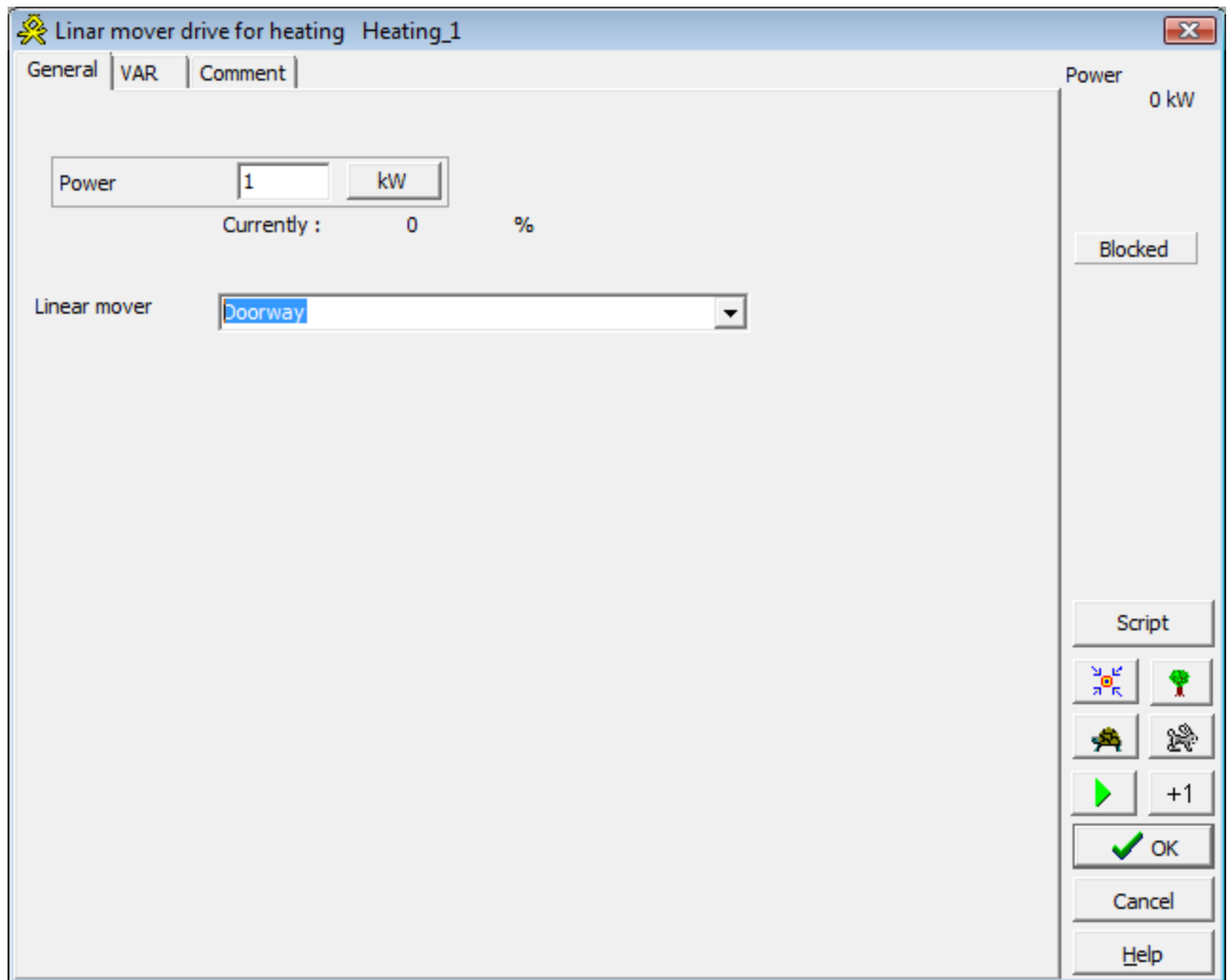
- [Valve drive by linear mover](#)



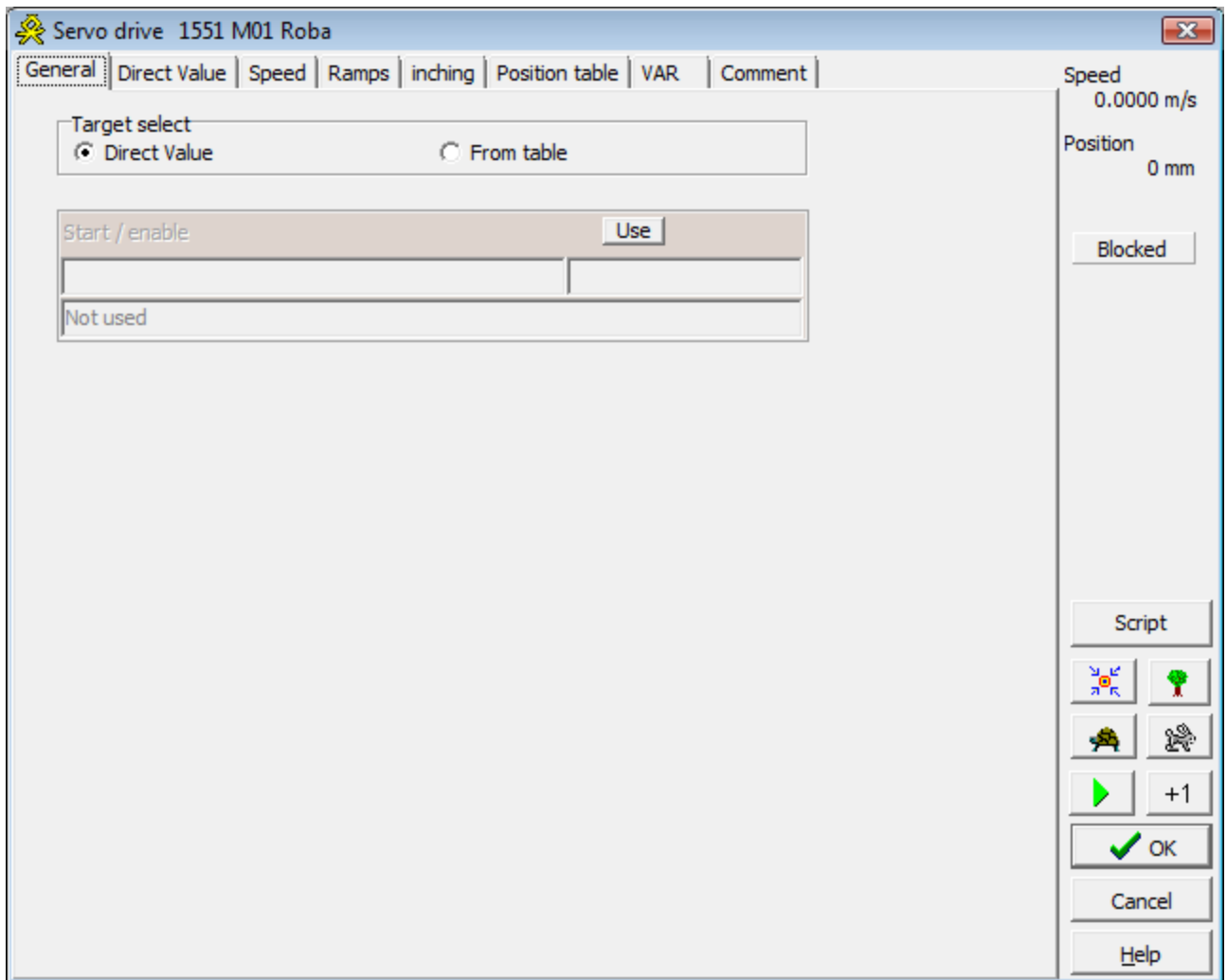
- [Joint drive by linear mover \(lever\)](#)



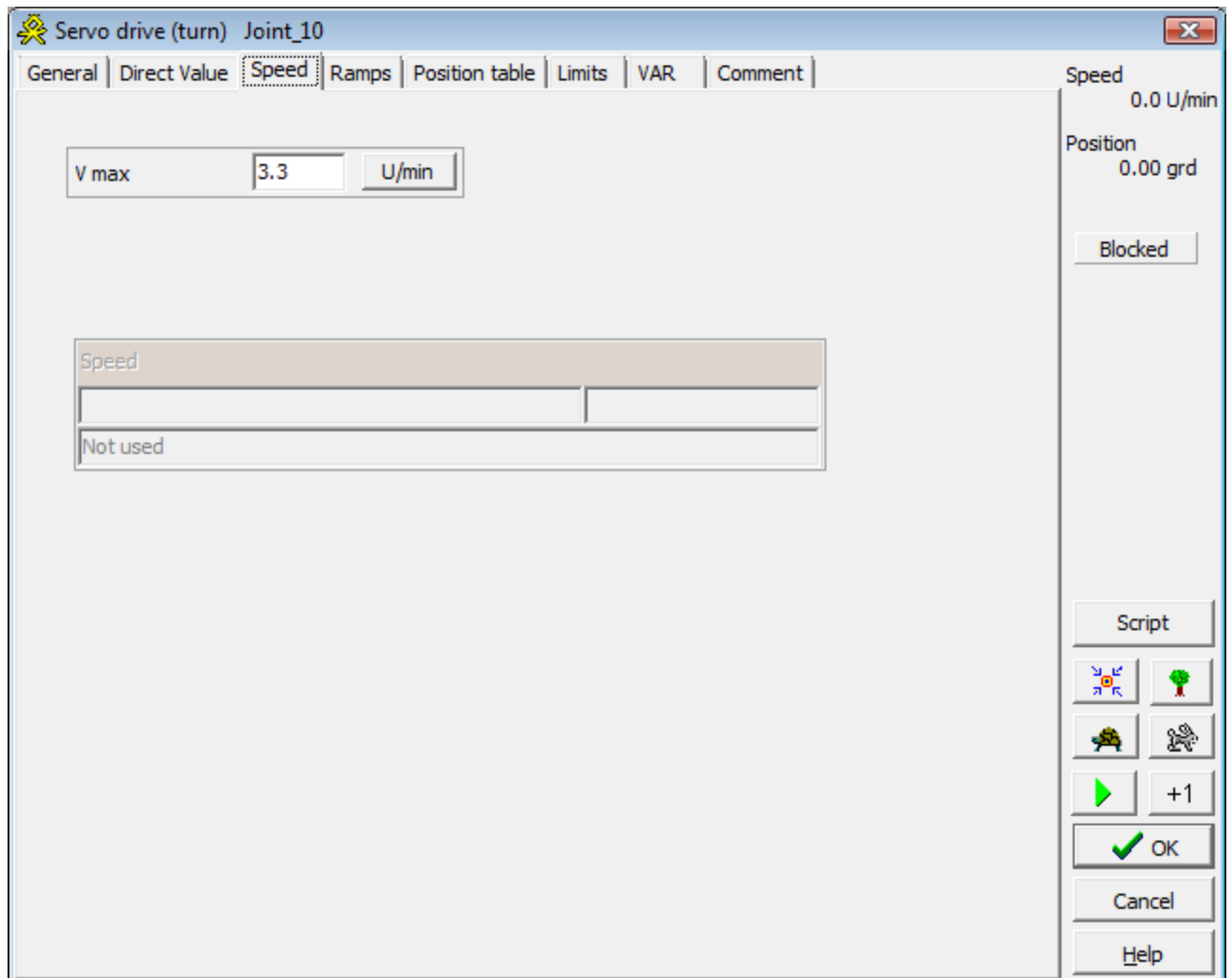
- [Heating / cooling drive by linear mover](#)



- [Servo drive](#) for linear mover



- [Servo drive](#) for joint



- [Absolute real drive](#) for joint (not supported anymore, please use the servo drive)

See also:

[Static elements of simulation](#)

2.6.1 Simple frequency converter

**

This drive corresponds to a frequency controlled motor or a proportional valve. It has at least to be connected to one word (E, A-, M-, or data) of the PLC. Next to the address you have to specify as well at which value of the word the maximum speed shall be reached. If the word in the real PLC has to head for an analog output that shall specify a frequency converter +/- 10V you have to enter 27648 here.

Either you can specify the start and stop slopes or modify them by the PLC. The start slope will always be used if the drive is released (even if the speed is reduced). The stop slope will only be

used if the drive is not released.

You can specify a release bit. If this bit is to be used the drive will only be released if it is active (exception: if the backward bit is to be used it will be enough if only that one is specified). The drive will run backward as well if a negative setpoint is specified.

Furthermore you can specify a backward bit. If this bit is specified the setpoint will be negated, that means by the help of this bit you can let the drive with only positive setpoint go backward and forward.

There are lots of frequency converters at the market and each is headed for in a little different way. We tried to design this drive so that it will be able to cope with as many demands as possible. Eventually you have to write some lines of codes in OB3 nevertheless to reach exactly the behaviour of the converter used by you.

This drive will also be used if none of the other drives seem to be appropriate, e.g. if you have got a motor with 3 speeds. Then you have to program the heading for the speed yourself, best in OB3.

The simple frequency converter gives also the possibility that a drive follows another one exactly. (Master slave operation). Therefore select the button 'extended'. You can select the strange element on the edit mask that appears which shall be followed by the drive. You have to specify a freewheeling bit as well. If this bit is specified the drive will not follow its master but it will behave like a completely normal frequency converter. If your drive is never to be freewheeling you will have to specify a bit here that is 'always 0'. Therefore we recommend the bit M 0.0 (and M 0.1 for 'always 1'). Even a drive that is configured as slave has to be released so that it runs. That means either you activate the check box 'use forward input' or you will have to see to it that the specified bit is '1' if the drive is to run.

The word motor is similar to the simple frequency converter but not that flexible. It still exists just because of reasons of compatibility and should not be used anymore if possible.

See also:

[Other drives](#)

2.6.2 Valve drive by linear mover

**

With this kind of a [drive](#) an element is not controlled by the PLC directly, but indirectly by a [linear mover](#) that has to exist already.

On the edit mask you have to specify a maximum flow. This flow is varied from 0 to 100% dependent on the position of the linear mover. Length and orientation does not matter, the one end position of the linear mover is always 0% the other one is always 100%. You can use the linear mover with integrated sensors as well.

Up to now this drive is only available for the [valve](#), the [joint](#), the [heating](#) and more inadvertently for the [pump](#).

2.6.3 Joint drive by linear mover

With this kind of a [drive](#) an element is not controlled by the PLC directly but indirectly by a [linear mover](#) that has to exist already. The absolute turning angle that is specified by the drive is varied dependent on the position of the hot spot of the linear mover between the minimum value and the maximum value. Length and orientation of the linear mover do not matter, the one end position always corresponds to the minimum value, the other one to the maximum value. You can use an linear mover with integrated sensors as well.

With the rattle functions ‘only forward’ resp. ‘only backward’ you can adjust that the joint will only be moved if the linear mover moves into the corresponding direction.

Up to now this drive is only available for the [valve](#), the [joint](#), the [heating](#) and more inadvertently for the [pump](#).

2.6.4 Heating drive by linear mover

**

With this kind of a [drive](#) an element is not controlled by the PLC directly but indirectly by a [linear mover](#) that has to exist already. The heating power specified by the heating is varied dependent on the position of the hot spot of the linear mover between the minimum value and the maximum value. Length and orientation of the linear mover do not matter, the one end position always corresponds to the minimum value, the other one to the maximum value. You can use a linear mover with integrated sensors as well.

Up to now this drive is only available for the [valve](#), the [joint](#), the [heating](#) and more inadvertently for the [pump](#).

2.6.5 Servo drive for linear mover

**

With this kind of a [drive](#) for the [linear mover](#) you can specify a number of positions on the edit mask. At runtime these positions can be selected by the PLC by their numbers.

The number of the target has to be assigned to a word of the PLC. If you have selected the option [bit-coded](#) the position of the least significant bit will be used as number in word.

The new target will be activated when the bit ‘start/enable’ has a **rising slope** or immediately if the appropriate check box ‘always 1’ is clicked. Even if you specify a bit you will be able to configurate that each setpoint changing is taken over immediately (without rising slope as well): activate the check box ‘immediately’ for doing this.

The servo drive will only run if the bit 'start/enable' is waiting or if the check box 'always 1' is clicked.

By the bit 'position reached' the servo drive will report if it remains under the distance to the target position that you have specified.

The position is headed for with the specified maximum speed by start- and stop-slopes. The steepness of the slope you do not specify by a time but by the distance after that the drive shall have the maximum speed resp. at which distance to the target it shall be started braking (while stopping). If you create the slope too small the drive will not be able to stop with control. This is not that different in reality and that is why we did not take great trouble to get rid of this effect. You can modify the speed as well as the slope by the PLC program by the help of a [poke](#).

With 'Shift+Ins' and 'Ctrl+Ins' you can insert new lines in the table of the positions. With 'Shift+Del', 'Ctrl+Del' and 'Ctrl+X' you can delete lines. Take care to modify the target numbers in the PLC if you change the numbering by inserting or deleting positions.

With the button 'teach' you can take over the current position of the linear mover. Adjust this position with the slide controller 'hot spot' on the edit mask of the linear mover before.

You can specify the set position of the hot spot [directly by the PLC](#) as well.

2.6.6 Servo drive for joint

With this kind of the [drive](#) for the [joint](#) you can specify as many target angles as you like which are selected by the PLC by their number while simulation time.

The number has to be assigned to a word of the PLC. If [bit-coded](#) is clicked the position of the least significant bit in word will be used as number. The new target will either be taken over if the bit 'start/enable' has got a **rising slope** or immediately if the check box 'always 1' is clicked.

The servo drive will only turn if the bit 'start/enable' is waiting or if the check box 'always 1' is clicked.

By the bit 'position reached' the servo drive will report if it remains under the distance to the target position that you have specified.

The target angle is headed for with the specified maximum turning speed by start- and stop-slopes. The steepness of the slope you do not specify by a time but by the angle after that the drive shall have the maximum speed (while starting) resp. at which angle distance to the target it shall be started braking (while stopping). If you create the slope too small the drive will not be able to stop with control. This is not that different in reality and that is why we did not take great trouble to get rid of

this effect.

You can specify the speed as well as the slope of the PLC program by a [poke](#). If you have selected 'degree' as unit you will have to poke both sizes in 0,1 degree/s resp. 0,1 degree (e.g. for the speed 20 degree/s you have to transfer 200 into the PLC-word of the poke). If you have selected 'rad' as unit you will have to poke in 0,001 rad/s resp. 0,001 rad.

With 'Shift+Ins' and 'Ctrl+Ins' you can insert new lines in the table of the positions. With 'Shift+Del', 'Ctrl+Del' and 'Ctrl+X' you can delete lines. Take care to modify the target numbers in the PLC if you change the numbering by inserting or deleting positions.

With the button 'teach' you can take over the current position of the joint. Adjust this position with the slide controller 'angle without loosen the axis mother' on the edit mask of the joint.

You can specify the set position of the hot spot [directly by the PLC](#) as well.

[Back to the joint](#)

2.6.7 Direct adjustments at servo drive

**

With the servo drive you can adjust the position (or turning angle) directly by the PLC-word as well. Doing this you activate the check box 'direct adjustment'. Now you have to transfer an INT-value into the program. A little different rules are valid for the linear mover and the joint:

With the linear mover you adjust the position in the global adjusted units (mm is default, adjustable in View|Options|Machine). The reference point is the beginning of the linear mover.

With the joint you can adjust the unit 'degree' or 'rad' immediately on the edit mask of the servo drive. If you have selected 'degree' as unit you will have to transfer the set value of the angle in 0,1 degree. If you have connected the servo drive e.g. to a word QW 50 and you want an angle of 72 degree you will have to program:

```
L 720
T QW 50
```

If you have adjusted 'rad' as unit you will have to specify the angle in 0,001 radian.

The other adjustments on the edit mask of the servo drive keep their function.

With the joint this operating mode corresponds to the old absolute real drive, but with more comfort.

If the range of values of an INT is not enough for its purpose you will be able to activate the check box 'DINT'. Now the setpoint has to be specified as DINT whose range of values should be enough for all realistic cases of use.

[Back to servo drive of the linear mover](#)

[Back to servo drive of the joint](#)

2.6.8 Word motor

**

This drive should only be used by teachers whose pupils use a version of TrySim smaller than V2.9.12. Professional users should use the new drive [simple Frq converter](#) that has got an enlarged functionality.

This drive corresponds to a frequency controlled drive or a proportional valve. It has to be connected to a word (E-, A-, M- or data-) of the PLC. Next to the address you have to specify as well at which value of the word the maximum speed shall be reached. If the word in the real PLC has to head for an analog output that shall specify a frequency converter +/- 10V you will have to enter 27648 here.

The word motor will also be used if none of the other drives seem to be appropriate, e.g. if you have a motor with 3 speeds. Then you have to program the heading for the speed yourself, best in OB3.

See also:

[Other drives](#)

2.6.9 Word-drive for pump and valve

**

You can switch on / off this drive and you can also adjust the pump power (resp. max. flow at the valve).

If this drive is used for the pump a negative value for 'Power' will lead to a redirection of the flow. For the valve the sign has got no importance.

2.6.9.1 Absolute real drive

Before you use this drive you should check whether the [servo drive](#) with [direct input](#) is more suitable.

This drive does not give a speed but an absolute position to its element. The number has to be given to the PLC as REAL in a DWord. At the moment this drive is only available for the [joint](#). You can adjust on the edit mask whether you want to specify the value in degree or in [radian](#).

If the absolute angle of rotation is not created in your PLC program but you define it (only for the simulation) by other data, you shall place the code for conversion in OB3.

See also:

[Other drives](#)

2.6.10 1 bit-drive for pump and valve

*

This is a quite simple drive. If the bit is set the pumping power will be approached considering the start ramp, if it is switched off again, it will be set to 0 again considering the stop ramp.

If this drive is used for the pump a negative value for 'Power' will lead to a redirection of the flow.

2.6.11 Drives of the linear mover

*

- Bit motor
- This drive can go backward and forward with one speed. Either you can specify one or two outputs. In addition you can select forward/backward with pulse response. In this case the drive behaves like a 5/2-way valve with pulse response.
- Bit motor with two speeds
Here you can specify another bit 'fast' and corresponding to that another speed.
- 4/2-way valve
Here you only have to specify a PLC output. If it is specified the drive will go forward, if not the drive will go backward.
- Simple [frequency converter](#)
This motor corresponds to a frequency controlled motor or a proportional valve. It has to be connected to a word (E, A-, M- or data-) of the PLC.
Next to the address you have to specify as well at which value of the word the maximum speed shall be reached. If the word in the real PLC has to head for an analog output that shall specify a frequency converter +/- 10V you will have to enter 27648 here.
- [Servo drive for linear mover](#) During creating the simulation you specify some positions here which are selected automatically by the PLC by its number and which are approached automatically.
You can also specify the set position directly.
- [Crank drive](#) here a joint / an axis is used to drive the linear mover like a connecting rod.

2.6.12 Bit motor for joint

*

With this kind of [drive](#) for the [joint](#) you have to adjust a speed in U/min.

There are several possibilities for the selection of the current speed which corresponds in the main with the ones of the linear mover. Take care that the number and the meaning of the bits which are needed for control are changing while selecting the drive (top left in the mask). You can find exacter

informations of the different alternatives [here](#).

You can adjust two kinds of slopes: start slope and stop slope. The adjusted time always corresponds to the time that the drive needs to get from zero to the 'speed normal' resp. from there to zero. The start slope is always used if the set value speed does not equal zero, otherwise the stop slope is used. That means, if you also adjust a drive from fast to slow the start slope is used, although the motor is going to be slower. The stop slope will only be used if the motor slows down. If there is a motor with integrated brakes in your real machine you shall adjust the stop slope on a very small value. If your real motor has only got one speed you will be able to explicitly modify the braking behaviour in this case by selecting a 2-speed-motor and adjust a 0 for small speeds and adjust the stop slope corresponding to the braking behaviour of your motor. If you use a motor with 2 speeds in reality then you will not have this possibility for lack of parameters, and you have to improve instead. An exception to this are the slopes at the 4/2-way valve. Here the start slope will always be used if the bit is '1' and the stop slope if it is '0'. By this you can reproduce differently choked connections of a double effective cylinder.

For all drives you can adjust limitings of the turning angle as well. Doing this you have to decide first whether the limiting angles shall be displayed in degree or in rad. The easiest way to adjust the limiting is to put the joint by the slide controller 'angle without loosen the axis mother' into the wanted position and to click the buttons 'teach min' or 'teach max' then.

Both speeds and the slopes can be changed by the [word pokes](#) by the PLC while runtime as well. The speeds are poked in 0,01 U/min the slopes in 0,001 s. Doing this it might be a little annoying that the pokes always turn around the joint - just [fade out](#) the pokes.

Ah, if you have adjusted a limiting for a two-speeds-drive you will have to see to it yourself, like in real life, that the drive is switched to slow before. Otherwise your parts of the machine crash into the stop with full power. In TrySim it does not matter but in real life it does!

See also:

[Servo drive for joint](#)

2.6.13 Bit drives of the joint

**

1.) Start-Stop

This is a motor that has only got one speed and can only run into one direction. Naturally it cannot be used with one limiting meaningfully.

2.) 4/2-way valve

This motor never stands still. Either it will run forward (if the output is '1') or backward (if the output is '0'). In most cases it will be a good idea to use it with limitation of the turning angle.

3.) For- Backward

This motor will run forward if the first bit is set and backward if the second bit is set, but always with the same speed. If no bit is set it will slow down with the stop slope. It can be used with as well as without limitation.

4.) Slow / Fast

This motor just runs forward but with two speeds. If the 'fast' bit is set it will run fast, independent of the stage of the 'slow' bit. (Remark: This is possibly different to your real connection: there it might be that the motor only runs fast if both outputs are set.)

A specification of limits is not useful because you will never get back the joint.

5.) For / Back / Fast

For the control of this motor you need three bits. The first and the second one adjust the direction of speed. With the third one you adjust the speed. If the 'fast' bit is '0' the joint will turn slowly, is it '1' the joint will turn correspondingly fast. This drive can be used with as well as without limitation.

6.) For / Back / Slow / Fast

This drive is nearly the same as the previous one (No.5), but here the small speed has to be set explicitly as well.

7.) Slow For / Fast For / Slow Back / Fast Back

This drive corresponds also to the drives 5.) and 6.) fundamentally. It needs 4 control bits whose meaning is self-explanatory. Take care that only one of the bits is active. There are mainly frequently converters which are set like this, although this could also be realized with conventional circuit engineering, but that is quite unusual in connection with a PLC.

[Back to bit motor for joint](#)

2.6.14 Crank drive

With this kind of the [drive](#) you have to create a [joint](#) first and connect it to the PLC. This joint can be used to operate a [linear mover](#) or another joint.

If you operate a linear mover with a crank the hot spot will be moved once from one end to the other one and back at each complete turning of the joint.

If you operate another joint with a crank both joints will be coupled synchronously, as if they are connected to each other by gear wheels and a chain.

You can also change the transmission ratio. If you enter a number that is bigger than '1' in the corresponding field the element will move faster, if you enter a number that is smaller than '1' it will move slower.

2.7 Analog value processing

We are often asked if there is analog value processing in TrySim.

Elements that can be connected with analog inputs are currently:

[Slide controller](#)

[Spy](#)

[Bar Code Scanner](#) (if you want to call a bar code analog)

[Resolver for Conveyor/Chain/Track](#)

[Distance Sensor](#)

[Thermometer](#)

[Level gauge](#)

[Flow Meter](#)

[Analyzer](#)

[Linear Mover with Integrated Sensors](#)

[Digital Display](#)

Elements that can be connected with analog outputs are currently:

[Conveyor belt](#)

[Free point](#)

[Generator](#) (the bar code of the dynamic is determinable)

[Linear mover](#)

[Chain](#)

[Joint](#)

[Arc Belt](#)

[Turnable conveyor](#)

[Turntable](#)

[Turner](#)

[Heating](#)

[Digital display](#)

[Oscilloscope](#)

[Pump](#)

[Valve](#)

[Slide controller](#)

If you need more elements with analog connection, contact us.

2.8 Dynamic elements of simulation

2.8.1 Normal dynamics

These elements named ‘dynamics’ correspond to the material which is processed by the machine. They are generated during simulation runtime by the [generator](#) and can also disappear again. You can let move and detect the dynamics by the machine in various ways.

In former versions of TrySim we called the dynamics **boxes** but this name turned out to be not very appropriate because on the one side there were always mix-ups with the static element [box](#), and on the other side it gave the impression that TrySim is just useful to shift boxes.

From version 2.1 on there are two kinds of dynamics:

- 1.) Normal dynamics
- 2.) Turnable dynamics

The turnable dynamics are elements with completely different properties compared to the normal ones, because it was not possible to make the normal ones turnable. In the following versions the behaviour of the turnable ones will be adjusted more and more to the normal ones. With increasing computer speed and capabilities we plan to maintain only one option, turnable dynamics to be precise. Until then we have both kinds available, because the turnable ones need much more computing time than the normal ones.

But there is the possibility to change both kinds into each other during simulation runtime by the [dyn-converter](#), so you can adjust the behaviour of the dynamics to the relevant situation.

In the edit mode you can move the dynamics with the mouse. So you can lead to specific test-situations well-aimed. You can turn the turnable dynamics around all axis with slide controllers, just select the check box ‘turn’ out of the [context menu](#).

Dynamics can be tagged with a bar code which can be detected with the help of the [bar code scanner](#).

The ‘drop in depth’ (adjustable on the edit mask of the dynamics) will serve to make the behaviour of the dynamics more realistic if level differences exist between different conveyors. It should not be specified bigger than 10 mm.

On the edit mask of the dynamics you can also specify whether the dynamics shall be stored. These storable dynamics (semi dynamics) are not deleted by destroyers. They will be useful if your machine has circulating pallets, carriages or traverses. Do not complain about TrySim if your control does not work anymore after switching over to storable dynamics and restart of TrySim afterwards. In a real machine you would be faced with the same problem after having changed a CPU that is faulty.

The normal dynamics are always orientated rectangular but there are also the [turnable dynamics](#) for which these restrictions are not valid.

The [light barrier](#) detects dynamics. You can specify their position with the [distance sensor](#). You can grab dynamics with the [hook](#) and manipulate them with the [shifter](#) and also with the [wedge](#) in the XY-plane. You can combine several dynamics to a single dynamic with the help of the [melter](#). The [divider](#) splits dynamics while the [press](#) makes dynamics smaller. With the help of the [sticker](#) static elements can be fixed to a dynamic as well.

If you have selected the option 'fillable' on the dynamic mask of the generator the dynamics behave like a [container](#), that means that you can fill up the dynamics with the [pipe](#), with the [valve](#) and with the [pump](#). With the [level gauge](#) you can detect the amount of the filling. The [level switch](#) checks the level to a limit. With the [analyzer](#) you can specify the components of the fluids in the dynamics and finally you can change the components with the [reactor](#).

A frequently asked question with fillable dynamics is: 'Why can I not see the level like in the sample bottling plant' Answer: 'In the x-y top view you cannot see the levels.'

See also:

[Destroyer](#)

[Conveyor](#)

[Static elements of simulation](#)

2.8.2 Turnable dynamics

In contrast to the normal [dynamics](#), which are always orientated rectangular, this type of dynamics can be turned in every direction.

To create turnable dynamics, select the button 'dynamics' on the edit mask of the generator and then activate the check box 'turnable'. You can also convert an existing normal dynamic using the [dyn converter](#).

The behaviour of these elements is (still) mainly defined by their edges. Many operations that can be done with normal dynamics will work with turnable dynamics only if at least one edge overlaps with the executing element. You can increase the number of active points by selecting the options 'Center points' (6 new active points) and 'edge points' (12 new active points).

Because a lot of applications need turnings in the x-y plane where a tipping over of the dynamics is rather disturbing you can exclude all the other dynamics by selecting the option 'just in XY-plane'

In the edit mode the dynamics can be moved by the mouse and so you have the possibility to lead to special test situations specifically or you can turn the dynamics around all axis with three shift controllers by selecting 'turn' out of the context menu.

The turnable dynamics cannot be stored yet.

With the real turns there are still several restrictions. So you cannot stack turnable dynamics on each other and you cannot shift them mutually. You can decrease the effects of the restrictions by using the [dyn converter](#) with which both kinds of the dynamics can be converted in each other. Because the simulation of turnings need a huge computing time effort the maximum simulation speed can be raised by the help of this element if lots of dynamics exist.

The static elements [arc belt](#), [turn table](#), [turnable belt](#) are specially created for the turnable elements. With the help of the [straightener \(dyn converter\)](#) inadvertently sloping dynamics can be straightend.

See also:

[How to select turnable elements](#)

[Static elements of simulation](#)

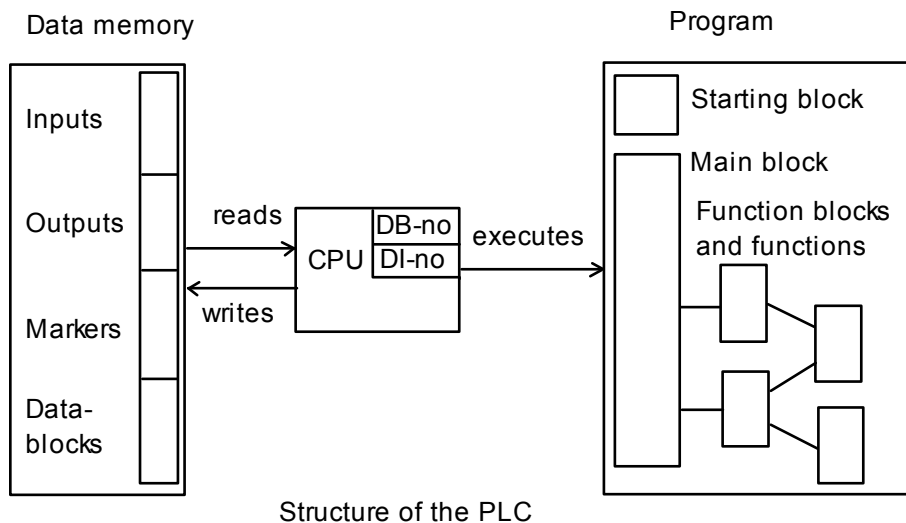
Part



3 PLC

3.1 Introduction

Although the PLC is a central part of TrySim you will never see it. If you want to know how a PLC looks like you will have to look at the catalogues of the producers. The structure of a PLC simulated by TrySim is the following:



In the following the single parts are described. General knowledge of the function of a computer is required. We refer you to the extensive available specialized literature.

See also:

[Program](#)

[CPU](#)

[Data Memory](#)

[PLC-Editor](#)

[Naming the Inputs, Outputs and Markers](#)

[Naming the Data in Data Blocks](#)

[List of Operations](#)

3.2 Data memory

3.2.1 General

In a real PLC you distinguish mainly between four different areas of the data memory. At the beginning of each machining cycle the inputs are set to the value which the connected sensor gives. The [CPU](#) reads the inputs and specifies by the program the state of the outputs.

The connected actuators are operated by the outputs at the end of the scan. The markers and data blocks are used to save provisional results and parameter.

Naturally we have taken over this structure, in contrast to a real PLC the division into inputs, outputs, marker and data blocks in TrySim has only got a symbolical character: You can operate actuators with inputs, e.g., and influence the state of markers by sensors. If your program shall run on a real PLC in future you will not make use of these possibilities, of course, but while testing they are of great use. Suppose, your machine stacks boxes and the controller shall save how many boxes are at a specific position. Anywhere in your program is a mistake and the number is not correct occasionally. With TrySim you just take a digital display, connect it to the data word with the number of boxes and put it directly below the stack. Now you can see at once whether the number is not correct, so that you can encircle the mistake quickly.

See also:

[PLC](#)

[Naming the Inputs, Outputs and Markers](#)

[Naming the Data in Data Blocks](#)

[Instance Data Blocks](#)

3.2.2 Naming inputs, outputs and memory bits

These data areas are organized byte by byte and consist respectively of 65536 bytes. You can access the data in these areas by bit, byte, word and double word.

Examples:

```
I 5.2      // Bit no. 2 of input byte 5 (Bit-numbers from 0 - 7)
QB 6       // Output byte 6
MW 10      // Marker word 10, consists of MB10 (high,h) and MB 11 (low,l)
ID 2       // Input double word 2 : IB 2(hh),IB3(hl),IB4(lh),IB5(ll)
```

In **View|Options|PLC-Editor** you can also adjust the IEC-Mnemonics, then inputs are not named with E but with I and outputs not with A but with Q.

See also:

[PLC](#)

[Data Memory](#)

[Naming the Data in Data Blocks](#)

[Instance Data Blocks](#)

3.2.3 Naming data in datablocks

There are up to 4095 data blocks which basically can contain 65536 bytes each. But how many of it you can use depends on the memory capacity of your computer and above all on the size of the data memory of your destination system. You have to [create](#) the data blocks and specify which size they have got at the same time. The data in the data blocks are likewise organized byte by byte and they are named exactly like the other data, but with the supplement in which data block they are.

Examples:

DB17.DBX 6.2	Bit no. 2 in data byte 6 of data block 17
DB22.DBB 8	Byte 8 in data block 22
DB39.DBW 11	Data word 11 in DB 39, consisting of DBB11 and DBB12
DB88.DBD 20	Data double word 20 in DB 88

If you want to access several data of the same data block one after the other - in order to simplify the spelling - you will be able to [open](#) a data block. Until you can open another data block you can leave out the name of the data block. But you have this possibility only inside the program, if you want to access data from out of the machine simulation you will always have to specify the data block. Internally the opening of a data block means that a number is put down in a register of the [CPU](#). This register is symbolized by the small box DB-Nr in the system of the PLC-structure.

See also:

[PLC](#)

[Data Memory](#)

[Naming the Inputs, Outputs and Markers](#)

[Instance Data Blocks](#)

3.2.4 Instance data blocks

The concept of the instance data blocks is explained in detail in the section [function blocks](#). In the data memory these are completely normal data blocks whose data you can access in the usual way. Usually this is not done. In the [CPU](#) there is another register (in the picture DI-No) in which the number of the currently opened instance data block is saved. If you want to access data of this block you will have to use the following notation:

DIX 7.2	bit no. 2 of byte 7 of the DB with the number in DI- register	
DIB 9	byte 9	''
DIW 10	word 10	''
DID 16	double word 16	''

This is just one possibility to access data, usually data out of instance data blocks are addressed by

the name that is specified in the declaration part of the accompanying FB.

See also:

[PLC](#)

[Data Memory](#)

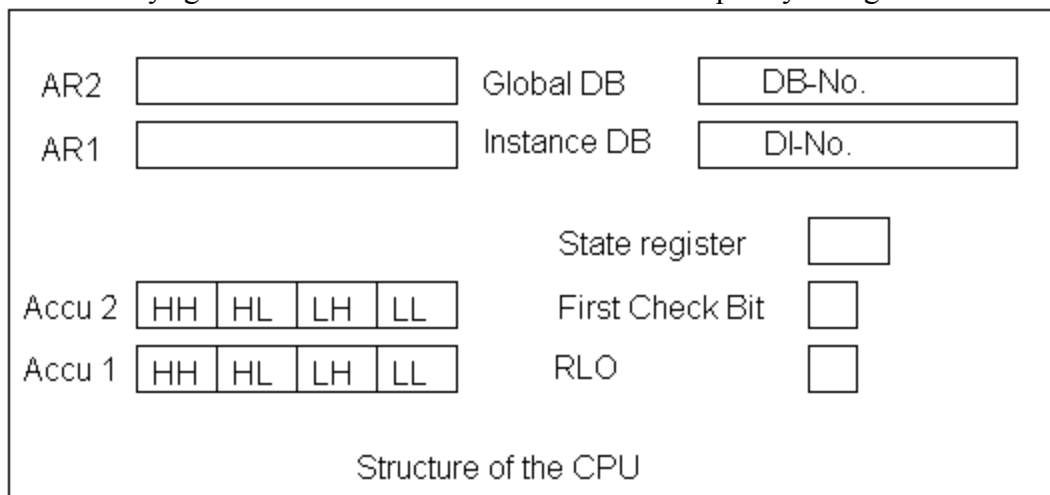
[Naming the In- and Outputs as well as the Marker](#)

[Naming the Data in Data Blocks](#)

3.3 CPU

3.3.1 Introduction

The CPU (Central Processing Unit) is every computer's heart. It reads the information of the data memory, processes these according to the instructions of the program and saves the results in the data memory again. For fulfillment of this task there are temporary storages available in the CPU:



See also:

[Accumulator 1](#)

[Accumulator 2](#)

[Accu 3 and 4](#)

[Result of Logic Operation \(RLO\)](#)

[First Request](#)

[Global Data Block Register](#)

[Instance Data Block Register](#)

[Address Register 1 and 2](#)

[CPU-Realtime Clock](#)

3.3.2 Accumulator 1

This 32-bit wide memory is the turntable of almost all informations which are processed byte by byte, word by word or double word by double word. Most of the instructions refer to the accu 1, either as data source or as data destination or both at the same time. If data are read out of the memory they reach the accu 1 first and if data are to be written into the memory they will be taken out of this. If calculations and manipulations are to be done the content of accu 1 will be part of it and the result will be saved here as well.

Accu 1 consists of a double word, two words, four bytes or 32 bits, according to the approach. The both words are distinguished by the addition H (high) and L (low). The four bytes are distinguished by adding HH (from high-word the high-byte), HL (from high-word the low byte) and corresponding LH and LL.

Examples:

Accu 1 H	Bits 31 - 16
Accu 1 LH	Bits 15 - 8

See also:

[CPU](#)

[Accumulator 2](#)

[Accu 3 and 4](#)

[Result of Logic Operation \(RLO\)](#)

3.3.3 Accumulator 2

This memory that is built exactly like accu 1 serves as intermediate storage for operations which need two operands. It is mostly loaded by loading a new value into accu 1. The previous content of accu 1 is shifted to accu 2 by that.

See also:

[CPU](#)

[Accumulator 1](#)

[Accu 3 and 4](#)

3.3.4 Accu 3 und 4

In the PLCs of the S7-400 series there are accu 3 and accu 4 which are also 4 bytes big that are used for saving intermediate results at complex calculations.

Operations with these accu :

[ENT](#)

[LEAVE](#)

[PUSH](#)

[POP](#)

See also:

[CPU](#)

[Accumulator 1](#)

[Accumulator 2](#)

S7-400 is a registered trademark of Siemens AG

3.3.5 Result of logic operation (RLO)

The RLO plays the same part for the bit-operations as [accu 1](#) for the other operations. But for the programmer who programs in [FBD](#) or [LAD](#) it will never appear. It is only used by the compiler that translates these languages into the machine code.

See also:

[CPU](#)

[Accumulator 1](#)

[First Request](#)

3.3.6 First Request

For this bit the same is valid as for the [RLO](#), if you program in [FBD](#) or [LAD](#) there will be no need to see to it.

See also:

[CPU](#)

[Result of Logic Operation \(RLO\)](#)

3.3.7 Global data block register

The DB referred by this register is used by the CPU every time a data word (-bit, -byte, double word) is accessed that is noted without the adding of a data block. This register is loaded by the operation [OPN](#) (open) DB.

Calling a new block the register stays unchanged, but its content is saved in the DB-stack and is recovered at a return of the accessed block. That means that if in the accessed block another DB is opened this will have no consequences for the current block.

See also:

[CPU](#)

[Naming the Data in Data Blocks](#)

[Instance Data Block Register](#)

3.3.8 Instance data block register

The DB referred by this register is used by the CPU every time an instance data word (-bit, -byte, double word) is accessed. Normally this register is loaded automatically at calling a function block, but it can also be loaded by the operation [OPN DI](#) with a new value. Calling a block the current content of this register is saved on the DB-stack and recovered at the return by the accessed block.

See also:

[CPU](#)

[Global Data Block Register](#)

3.3.9 Adressregister 1 and 2

In the CPU there are two address registers that are named AR1 and AR2 that can point to addresses in the I-, Q- or M-area or contain an unspecified address whose area is specified at runtime. The address registers are used for the [register-indirect addressing](#). The advantage of this kind of addressing compared to the memory-indirect addressing at complex memory operations is caused by the fact that there are special instructions ([+AR1,+AR2](#)) with which the address registers can be incremented or decremented without displacing the just processed numbers from accu 1 and 2.

You can also use these registers as normal computing register for special tasks, this is sometimes quite useful.

[!! Attention with Use of Address Register !!](#)

See also:

[LAR1 or 2](#) Load address register

[TAR1 or 2](#) Transfer address register into accu 1

[CPU](#)

[Accumulator 1](#)

[Global Data Block Register](#)
[Instance Data Block Register](#)

3.3.10 Status register

In the status register internal bit states of the CPU are saved, the RLO and the first check bit belong to it as well. The status register in TrySim is not a precise copy of a S7, because such a copy would have had distinct effect on the simulation speed and would only have been helpful in a few cases of use. The instruction L STW (Load Status Word) can be programmed in TrySim but cannot be executed.

See also:

[CPU](#)
[Accumulator 1](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.4 Program

3.4.1 Introduction

By the program it is specified what the [CPU](#) is supposed to do. To be able to formulate complex tasks clearly as well it is divided into organization blocks, function blocks and functions. All kinds of blocks are provided with the same operation set and can be displayed in all three kinds of representation (FBD, LAD and STL).

[Organization Blocks](#)
[Functions](#)
[Function Blocks](#)

3.4.2 Organization blocks

Organization blocks (OB) are always called by the operating system. Within a real PLC plenty of OBs are existing which are called at the most different events. In TrySim only five OBs are called by the operating system: the OB 1, 2 and 3 as well as the OB 35 and the OB 100. You are able to program more OBs although you have to call them explicitly, however.

1 OB 1

OB 1 contains the main program. Every time the simulation of the machine is finished, this block is called. When the OB 1 is executed the simulation of the machine is started again. OB 1 normally

calls mainly functions and function blocks. However you can also program quite normal in OB 1. Small programs that do not need any structure are completely programmed in OB 1 which is the only block of the program then.

2 OB 100

OB 100 is the start-up block. Here initializations are done and in a real PLC special modules are prepared. In TrySim the OB 100 will be executed if you start the program for the first time or after you have reset the PLC by the menu item [PLC|Reset PLC](#).

3 OB 2 and 3

Here you can program operations that are only needed for simulation.

If you program an activity check for contactors, for example, this will signal errors permanently without taking precautions. To avoid this just program in the [OB 2 or OB 3](#) that the check input will always be set if the output is set as well.

The OB 2 is processed before the OB 1, the OB 3 after it. Of course you do not have to export these blocks to STEP®7 with your program.

4 OB 35

The [OB 35](#) is accessed in fixed intervals of the virtual time. You can adjust these intervals in **PLC|CPU-Properties|Interrupts**.

In organization blocks you can declare [Temporary variables](#), as in all other code blocks.

See also:

[Functions](#)

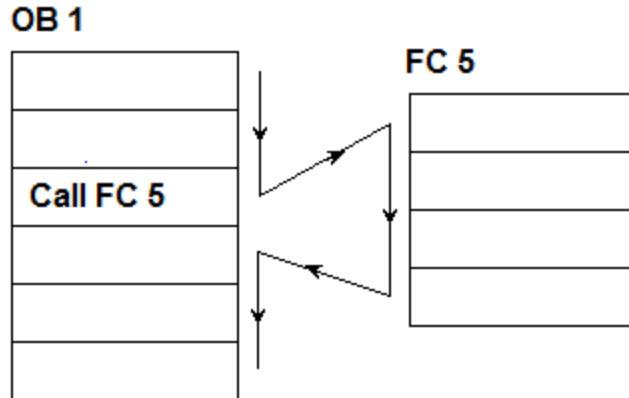
[Function Blocks](#)

STEP®7 is a registered trademark of the Siemens AG.

3.4.3 Functions

1 Task

In functions you formulate subtasks of the program. If you [access](#) a function in OB 1 or another block the instructions of the function will be executed. The process within the accessed block will be continued after the access at the end of the function.



Execution of instructions when calling a function

Functions are prepared to be called nestedly, i.e. they are able to call a function within another one. Functions with continually returning tasks are prepared to be called repeatedly.

2 *Parameter*

Functions can also be provided with parameter (place holders). These enable the function to work with different raw data at every call and to pass back values from the function to the calling block. Within a function the parameter are named formal-parameter.

While programming the function you use the parameter like in- and outputs, markers or data words. Calling the function you predefine which data are really used by the PLC then. These data are named actual-parameter. Which parameter shall have your function you adjust in the header of the function. For each parameter you have to declare these properties which are explained in the following:

- | | |
|---------------------|--|
| 1. Declaration type | In, Out, InOut, Temp |
| 2. Name | By this the parameter within the function is named |
| 3. Data type | By this the size is adjusted |
| 4. Default value | Has no importance within the functions! |
| 5. Comment | In order to make others understand your program |

3 *Declaration type*

There are three kinds of parameter and one kind of variables within the functions:

- In-parameter
- Out-parameter
- In-out-parameter
- Temporary variables

Calling a function In-parameter are set to the value of the actual-parameter. In-parameter should only be read within the function. Indeed you are able to write them onto In-parameter and to work with the modified value within the function then, but this change does not have an effect on the

actual-parameter in the calling block.

The operands connected to the Out-parameter are set to the value calculated in the function when returning to the calling block. Within a function you are able to read the Out-parameter but if you do this before you have assigned a value to the parameter the result will be an accidental one because Out-parameter are not set to the value of the connected operand when calling a function.

In-Out-parameter finally combine the behaviour of the In- and Out-parameter. Calling a function they are set to the value of the connected operand. They can be read and modified within the function. Returning to the calling block the modified value is copied to the connected operand.

The parameter model used in TrySim differs just slightly from that one of a S7. Please pay attention to the section [divergently implemented properties](#).

To save intermediate results which are only needed within the function you can use the [temporary variables](#).

4 *Name*

The parameter and the variables get a name each which you use while programming to access them. In the program code the names are marked with a '#' (double cross) which stands in front of the names. With clear names which are neither used in the symbol table nor correspond to a keyword you do not have to enter the sign '#', just in case that mix-up possibilities are given the input is necessary.

5 *Data type*

Except of the specification whether a parameter shall have in-, out- or in-out-character you have to declare the data type of the parameter as well.

These are, for example:

Type	Size	Example for connectable actual-parameter
BOOL	1 Bit	M 1.2
BYTE	1 Byte	IB 2
CHAR	1 Byte	MB 5
INT	2 Bytes	QW 10
WORD	2 Bytes	DBW 10
REAL	4 Bytes	DBD 20

See also:

[Data types](#)

6 *Default value*

This column is just relevant with [function blocks](#).

7 *Comment*

With this you can explain the meaning of the parameter and variables so that others can understand your program as well. As you are one of the 'Others' as well within one year you should use this possibility extensively.

8 *ENO-Output*

If you call a function in FBD or LAD you will be able to let carry out other operations (or further calls) in dependence on calculation results in the function by connecting the ENO-output of the function. The operations which are connected to this output will be executed if the BR-bit within the function is set to '1'. They will be ignored if the BR-bit is '0'.

See also:

[Organization Blocks](#)

[Function Blocks](#)

STEP®5, STEP®7, S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.4.4 **Function blocks**

1 *Task*

Function blocks (FB) are very similar to [functions](#). The difference between these two is that calling a function block nearly always a data block is specified in which the data, the block shall work with, are saved. This data block is named instance-data block. Its structure is adjusted while programming the FB and the access to its data is very comfortable within the FB.

In function blocks there are further kinds of variables: the [static variables](#). These variables keep their value from one call of the function block to the next one.

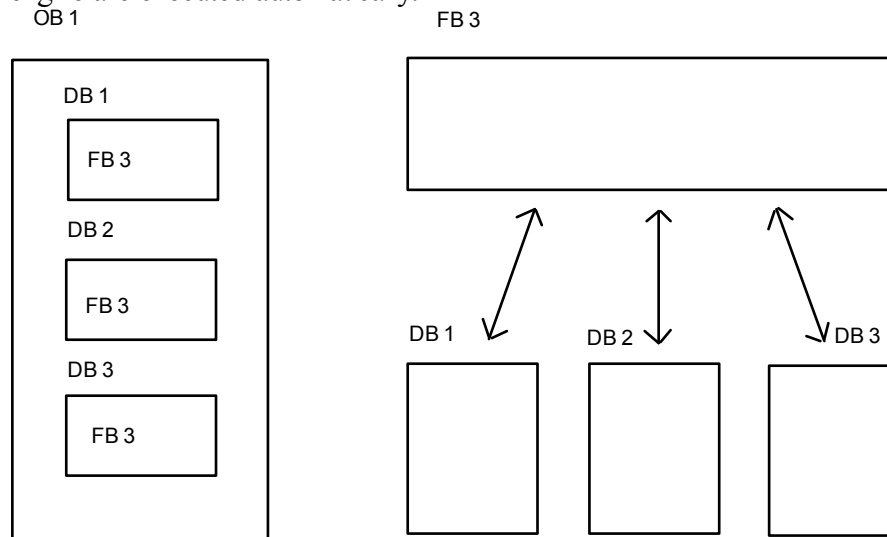
2 *Instance-data block-model*

After having declared the parameter and variables of a function block and after having programmed the block you have to call it anywhere otherwise it is not executed. If you program the call you will have to specify a data block number. The development system now creates a data block with this number and makes it so big that all parameter and (static) variables have got enough space in it. Also the names, [data types](#) and the comments are taken over out of the declaration part of the function block. The values of the data words (-bits, -bytes, -double bytes) are set to the default values that are specified in the header of the function block while creating.

Executing the program this data block will be opened automatically if the function block is [called](#). Now it is the instance-data block. The values of the actual-operands which are declared as In-Out-parameter are copied in the data block. All write- and read-accesses in the function block which you have programmed by using the parameter names refer to the data in the instance-data block. Returning to the calling block the In-Out- and Out-parameter are copied from the instance-data block into the connected operands. The use of the procedure will be clear if you program several

calls for the same function block, and this with different data blocks. Each of these DBs is created according to the mask of the function-block-header.

A standard example for this use of the instance-data-model is an operating time counter in a FB in whose header you have declared an In-BOOL-parameter with the meaning 'Engine Runs' and a Stat-INT-variable with the meaning 'Operating Minutes'. This FB you call for motor 1 with the DB 1, for motor 2 with the DB 2 etc.. At each process of the FB the data which are assigned to the engine are executed automatically.



Each Time the DB 3 is call,
another DB is invoked
as instance data block.

The FB 3 has access to the data in the
current data block only.

See also: [Local instance](#)

3 *Temporary variables*

The [temporary variables](#) in function blocks are saved on the stack just like the ones of functions. They are not part of the instance data block.

3.4.5 **Static variables**

Saving results which keep their value within a FB (not OB or FC) from one scan to the next one you can declare static variables.

The static variables are saved like the In-, Out- and In-Out-parameter in the instance data block of the FB.

[Temporary variables](#)

[Functions](#)

[Function Block](#)

[Organization Block](#)

[Data Type](#)

3.4.6 Temporary variables

Saving temporary results that are only needed within a block (OB, FC, FB) you can declare temporary variables. Calling a block these temporary storages have no defined value. That is why you have to note that you have to assign a value to them before the first read-access. After the process of the block the here saved values are not available anymore.

In the STEP®5 – world this task was done by a marker area that was called ‘Smear Marker’, which could be used by different program parts at the same time if there was a lack of markers. The nice thing about temporary variables is their name. As smear markers were used in different meanings it was only possible to give them the not very expressive comment ‘Smear Marker’. But the temporary variables can be named according to their content.

Internally the temporary variables are saved on the stack.

[Static Variables](#)

[Functions](#)

[Function Block](#)

[Organization Block](#)

[Data Types](#)

3.5 External (Soft) PLC

3.5.1 General

No english help available yet.



Bitte lesen Sie unbedingt die [Sicherheitshinweise bei externer \(Soft-\)SPS](#) !

Ein häufiger Einsatz von TrySim ist, nur die Anlage zu simulieren und das SPS-Programm auf einer realen (oder Soft-) SPS laufen zu lassen. In den allermeisten Fällen ist dies mit zusätzlichen Kosten verbunden, sei es für die notwendige Hardware, Lizenzgebühren für Treiber anderer Hersteller oder Anpassungen von TrySim an eine bestehende Konfiguration. Sprechen Sie mit uns, wenn Sie die TrySim - Anlage extern steuern wollen.

www.trysim.de

Mit TrySim können Sie keine externe SPS programmieren.

Über [MPI](#) können Sie TrySim problemlos an eine wirkliche S7 anbinden.

Die Anbindung über [TCP/IP](#) ist an jeden Partner möglich, den man entsprechend programmieren kann.

Die Anbindung an die S7-300/400 Simulation [S7-PLCSim](#) funktioniert recht gut.

Die Anbindung über [Profibus](#) funktioniert schnell und einfach, erfordert allerdings eine recht teure Schnittstellenkarte und ist in dieser Hilfe nicht dokumentiert.

Die Anbindung an [Allen-Bradley](#) RSLogix5000 ist über RSLinx möglich, aber in dieser Hilfe noch nicht dokumentiert.

Die Anbindung an SPS **beliebiger Hersteller** ist über die [offene Schnittstelle](#) möglich.

Die Anbindung zur [IBH softec SoftSPS](#) ist problemlos möglich.

Die Anbindung an [3S -CoDeSys](#) - IEC1131- SoftSPSen ist möglich, aber ebenfalls in dieser Hilfe noch nicht dokumentiert.

Falls Sie Interesse an einer dieser Anbindungen haben, wenden Sie sich bitte an [uns](#).

Die Anbindung an Soft-SPS / SPS-Simulatoren anderer Hersteller ist geplant.

Für Anwender, die ein wenig unter Windows programmieren können, besteht jedoch außerdem die Möglichkeit, die Anbindung an TrySim selbst vorzunehmen, da die [Schnittstelle](#) von TrySim offengelegt ist. Im Prinzip kann mit jedem Compiler ein Programm erstellt werden, das die TrySim-Anlage steuert.

Anmerkung.: o.g. Möglichkeiten stehen in der TrySim Lite Version nicht zur Verfügung.

S7-300, S7-400 und S7-PLCSIM sind eingetragene Warenzeichen der Siemens AG.
MPI ist eingetragenes Warenzeichen der Firma Schildknecht.

RSLogix und RSLinx sind Warenzeichen von Rockwell Software Inc.
Alle anderen Warenzeichen sind eingetragene Warenzeichen der jeweiligen Rechteinhaber.

3.5.2 Safety guidelines

No english help available yet.



Mißachtung dieser Hinweise kann zu Sachschaden, Körperverletzung oder Tod führen.

Wenn TrySim über irgend eine Schnittstelle an eine externe SPS oder Soft-SPS angeschlossen wird, an die wiederum eine real existierende Maschine angeschlossen ist, kann das Verhalten dieser Maschine von TrySim aus beeinflußt werden. Die Anlagensimulation in TrySim verhält sich dann quasi selbst wie eine SPS, da sie in Abhängigkeit von den eingehenden Informationen die ausgehenden Informationen modifiziert. Das Verhalten dieser "Quasi-SPS" ist aber, obschon deterministisch, nur schwer mit ausreichender Genauigkeit vorherzusagen. Das Verhalten einer SPS wird durch das explizit vorliegende Programm bestimmt, welches Schritt für Schritt analysiert werden kann. Das Verhalten der Anlagensimulation wird aber nur implizit durch die Anordnung und Funktion der Elemente bestimmt.

Aus diesem Grund empfehlen wir dringend, TrySim nicht zusammen mit Maschinen zu betreiben, von denen möglicherweise Gefahren für Menschen und Sachen ausgehen.

Wir weisen ausdrücklich darauf hin, daß wir keinerlei Haftung, aus welchem Rechtsgrund auch immer, für Schäden übernehmen, die aus einer Mißachtung dieser Empfehlung herrühren.

See also:

[External PLC](#)

3.5.3 Open interface of TrySim

No english help available yet.

TrySim stellt anderen Anwendungen den gesamten E-,A- und M- Bereich zur Verfügung. Die Eingänge und Ausgänge (aus der Sicht der SPS) bestehen jeweils aus 64KB. Insgesamt besteht der Schnittstellenbereich aus 4 x 64K:

- 1.) Besondere Dienste, 64K noch in Entwicklung begriffen
- 2.) Ausgänge, 64K
- 3.) Eingänge, 64K
- 4.) Merker, 64K

Dieser Bereich wird von TrySim als "Mapped Object" eingerichtet. Alle anderen Anwendungen können durch den Aufruf von Windows-API-Funktionen auf diesen Bereich zugreifen. Dazu muß folgender Code programmiert werden:

C Pascal (Delphi)

Schliessen Sie in TrySim alle SPS-Fenster und wählen Sie **SPS|Externe SPS**. Sie werden gefragt, ob jetzt das Schnittstellen-Testprogramm gestartet werden soll. Dies ist ein sehr einfaches Programm, mit dem Sie Ausgänge setzen und Eingänge abfragen können. Unter **Optionen | Externe SPS** können sie stattdessen Ihren Schnittstellentreiber angeben.

Die Schnittstelle ist bei Win95/98, bedingt durch das Zugriffsverfahren, nicht besonders schnell (ca. 50 - 100 ms Antwortzeiten). Bei WinNT jedoch liegen die Antwortzeiten unter einer Millisekunde.

3.5.4 Interface, C-Code

No english help available yet.

handle hfile; // handle auf das mapped file

```
hfile = OpenFileMapping(FILE_MAP_ALL_ACCESS,
                       FALSE,           // handle soll nicht geerbt werden
                       "TRYSIM_SCHNITTSTELLE");
```

// Bei manchen Compilern ist es notwendig, den Schnittstellennamen als PChar - Variable zu übergeben.

// hier sollte überprüft werden, ob hfile <> NULL ist. Mit GetLastError läßt sich der Fehlercode ermitteln

```
*byte EAdrSpace; // pointer auf das EB 65535 !!!
*byte AAdrSpace; // pointer auf das AB 65535 !!!
*byte MAdrSpace; // pointer auf das MB 65535 !!!
```

```
AAdrSpace = MapViewOfFile(hfile,           // Handle to mapping object.
                          FILE_MAP_ALL_ACCESS, // Read/write permission
```

```

0, // Offset HiDWord
1 * 64 * 1024, // Offset LoDWord
64 * 1024); // Length

```

// hier sollte überprüft werden, ob AAdrSpace <> NULL ist.

```

EAdrSpace = MapViewOfFile(hfile, // Handle to mapping object.
FILE_MAP_ALL_ACCESS, // Read/write permission
0, // Offset HiDWord
2 * 64 * 1024, // Offset LoDWord
64 * 1024); // Length

```

// hier sollte überprüft werden, ob EAdrSpace <> NULL ist.

```

MAdrSpace = MapViewOfFile(hfile, // Handle to mapping object.
FILE_MAP_ALL_ACCESS, // Read/write permission
0, // Offset HiDWord
3 * 64 * 1024, // Offset LoDWord
64 * 1024); // Length

```

// hier sollte überprüft werden, ob MAdrSpace <> NULL ist.

Die Reihenfolge der Bytes im Adressraum ist verkehrt herum !!!

// Beim Beenden des Programms sollten Sie das mapped object wieder freigeben:

```
CloseHandle(hfile);
```

Siehe auch:

[Schnittstelle](#)

3.5.5 Interface, Pascal-Code

No english help available yet.

```
uses windows;
```

```
const MaxAdr = $FFFF;
```

```
type PAdrSpace = ^TAdrSpace;
```

```

TAdrSpace = packed array [-MaxAdr..0] of Byte;

var EAdrSpace: PAdrSpace;
    AAdrSpace: PAdrSpace;
    MAdrSpace: PAdrSpace;

var hfile : THandle;

{-----}
function InitSchnittstelle: boolean;

const OutputOffset = 64 * 1024;
      InputOffset  = 64 * 1024 * 2;
      MerkerOffset = 64 * 1024 * 3;
      Length       = 64 * 1024;

var mappedname: PAnsiChar;

begin
  mappedname:= 'TRYSIM_SCHNITTSTELLE';
  hfile:= OpenFileMapping(FILE_MAP_ALL_ACCESS,
                        FALSE,
                        mappedname);

  if hfile = 0 then begin
    result:= false;
    EXIT;
  end;

  AAdrSpace:= MapViewOfFile(hfile, // Handle to mapping object.
                           FILE_MAP_ALL_ACCESS, // Read/write permission
                           0, // Offset HiDWord
                           OutputOffset, // Offset LoDWord
                           Length); // Size of View

  if AAdrSpace = nil then begin
    CloseSchnittstelle;
    result:= false;
    EXIT;
  end;

  EAdrSpace:= MapViewOfFile(hfile, // Handle to mapping object.
                           FILE_MAP_ALL_ACCESS, // Read/write permission

```

```

                                0,                // Offset HiDWord
                                InputOffset,       // Offset LoDWord
                                Length);           // Size of View

if EAdrSpace = nil then begin
  CloseSchnittstelle;
  result:= false;
  EXIT;
end;

MAdrSpace:= MapViewOfFile(hfile, // Handle to mapping object.
                           FILE_MAP_ALL_ACCESS, // Read/write permission
                           0,                // Offset HiDWord
                           MerkerOffset,     // Offset LoDWord
                           Length);         // Size of View

if MAdrSpace = nil then begin
  CloseSchnittstelle;
  result:= false;
  EXIT;
end;

  result:= true;
end; { InitSchnittstelle }
{-----}
procedure CloseSchnittstelle;
begin
  CloseHandle(hfile);
  hfile:= 0;
  AAdrSpace:= nil;
  EAdrSpace:= nil;
  MAdrSpace:= nil;
end; { CloseSchnittstelle }
{-----}

```

Siehe auch:
[Schnittstelle](#)

3.5.6 Connection to external S7 via MPI-Datasnake

No english help available yet.



Bitte lesen Sie unbedingt die [Sicherheitshinweise bei externer \(Soft-\)SPS](#) !

Diese Schnittstelle wird nicht mehr unterstützt. Bitte lesen Sie [Anbindung an Externe SPS über MPI \(ProDave\)](#).

MPI ist eingetragenes Warenzeichen der Firma Schildknecht.

3.5.7 SoftPLC not found

For the connection to the IBH SoftPLC TrySim needs the dll 'plc32.dll'. TrySim searches for these dll in the path, which you can enter in **PLC|ExternalPLC|Interface-Parameter**. (Pay attention to the selection of 'IBH SoftPLC' before you click 'interface-parameter'.)

If you have not got a clue where the dll can be you will be able to use the Windows function **Start|Search** for 'plc32.dll'.

If you use the demo-version of the S5-simulator, the dll will be named 'plc32dem.dll'.

See also:

[External PLC](#)

3.5.8 SoftPLC not active yet

Trying to get into contact with the IBH SoftPLC by the DLL 'plc32.dll', an error has occurred.

Therefore two reasons are possible:

- 1.) You have not started the IBH SoftPLC / simulator yet. Do it now and after that select **PLC|External PLC** again.
- 2.) If the IBH SoftPLC is already active you will have more than one copy of 'plc32.dll' on your computer. It is important that TrySim uses exactly the same copy of the dll as the SoftPLC! The path you can enter in **PLC|ExternalPLC|Interface-parameter**, points at one of the copies, which is **not** used by the IBH SoftPLC. (Pay attention to the selection 'IBH SoftPLC' before you select 'interface-parameter'.)

Using the Windows function **Start|Search** you can search for 'plc32.dll'. If the worst comes to the worst you will have to try all the data which are found there.

If you start the SoftPLC by the network you will have to put the complete computer name ('\\

\ComputerX\C\.....') in front of the path. Or you connect the network with a disk drive letter over the button 'network'.

If you use the demo version of the S5-simulator the dll will be named 'plc32dem.dll'.

See also:

[External PLC](#)

3.5.9 Connection to external S7 via MPI (Prodave)

No english help available yet.

Bitte lesen Sie unbedingt die [Sicherheitshinweise bei externer \(Soft-\)SPS !](#)

Um die TrySim - Anlage durch eine MPI angebundene S7 zu steuern, benötigen Sie einen MPI - Adapter:

- Entweder den USB-Adapter von Siemens
- oder eine MPI- ISA/PCI - Card von Siemens zum Einbau in den PC
- oder einen MPI-fähigen CP von Siemens
- oder einen MPI-Adapter von Deltalogic (bitte vorher Rücksprache halten)
- oder einen anderen MPI-Adapter, der sich von PRODAVE-MPI ansprechen lässt.

Falls Sie ein Programmiergerät einsetzen oder einen PC, der bereits über eine solche Schnittstelle verfügt, benötigen Sie keine weitere Hardware.

Weiterhin benötigen Sie eine Siemens-Lizenz für [PRODAVE-MINI](#) , die Sie bei uns erhalten können. Wenn Sie diese Lizenz nicht bei uns beziehen, achten Sie darauf, dass Sie nicht nur die Lizenz, sondern auch die Daten bekommen, auf die sich die Lizenz bezieht.

Mit PRODAVE-MINI ist nur die Übertragung von Datenbausteinen möglich. Daher müssen Sie alle Ein-/Ausgänge der Simulation in DBs in der SPS übertragen. Im zu testenden SPS-Programm muss dann der Eingangs-DB im ersten NW des OB1 auf das Prozessabbild der Eingänge übertragen werden und im letzten NW muss das Prozessabbild der Ausgänge auf den Ausgangs-DB übertragen werden. Es ist also eine nur für die Simulation notwendige Veränderung des zu testenden Programms notwendig. Details dazu finden Sie im Verlauf dieses Kapitels unter "Festlegung eines Blockes der I/O - Konfiguration für MPI"

Es gibt auch eine Lösung, die ohne jede Modifikation des SPS-Programms auskommt, aber diese ist teurer. Falls Sie daran Interesse haben, setzen Sie sich bitte mit uns in Verbindung.

Die kleineren S7-300-CPUs unterstützen nur eine MPI-Baudrate von 187 kB/sek. Damit ist selbst bei einem kleinen Prozessabbild nur eine Daten-Aktualisierungs-Rate mindestens 100 ms möglich. Die DP-CPUs erlauben auch eine Baudrate von 12 MB/sek. Damit wird die Aktualisierungsrate deutlich kleiner. Wirklich befriedigende Ergebnisse erhält man aber nur mit CPUs der S7-400-

Reihe.

1.) Falls Sie auf Ihrem Rechner nicht bereits Software von Siemens installiert haben, die MPI verwendet, müssen Sie Prodave-Mini einmalig installieren, damit das Programm "PG-PC-Interface" verfügbar ist. Wenn dieses Programm bereits auf Ihrem Rechner verfügbar ist (z.B. wenn Sie dort einmal STEP7 installiert hatten), brauchen Sie Prodave nicht zu installieren. Das Installationsprogramm befindet sich auf der CD (vorausgesetzt, Sie haben eine Kopierlizenz bei uns erworben) unter d:\ProdaveMini\Disk1\setup.exe.

2.) Konfigurieren Sie Ihren Adapter/MPI-Card mittels des Programms "PG-PC-Interface" unter Start|Programme|Prodave_S7_Mini. Dazu können Sie einen neuen Zugangspunkt mit Namen "TRYSIMMPI" einrichten oder sie können einen bereits vorhandenen Zugangspunkt verwenden. Wichtig ist nur, dass der in TrySim eingegebene Zugangspunkt mit dem im "PG-PC-Interface" aktivierten übereinstimmt.

3.) Wählen Sie **SPS|Externe SPS**. Mit dem Button "I/O - Konfiguration" rufen Sie das Fenster zur Eingabe der Ein- und Ausgänge auf, die zur/von der SPS übertragen werden sollen.

4.) Legen Sie die Blöcke fest, die zur SPS übertragen werden sollen. (Auf der Maske steht, weil es intuitiver ist, "Eingänge", aber Sie können hier genauso gut Ausgänge, Merker oder DB angeben). Alle hier angegebenen Blöcke werden nach der Bearbeitung der Anlagensimulation zur SPS übertragen. Beachten Sie, dass bei Prodave Mini alle Daten [in Datenbausteine der SPS übertragen werden](#).

Wichtig!

Bei E-/A-/M- Bereichen beginnt der Zielbereich im DB in der SPS (gleichgültig was die Start-Adresse ist) immer am DBB 0. Im D-Bereich hingegen werden die Daten in die gleichen Datenwörter übertragen, aus denen sie stammen.

5.) Legen Sie genauso die Blöcke fest, die von der SPS zu TrySim übertragen werden sollen.

6.) Da die Verbindung über MPI wegen des aufwendigen Protokollrahmens recht langsam ist, sollten Sie nur die Blöcke angeben, die tatsächlich benötigt werden. Da ein Großteil der Übertragungszeit auf den Protokollrahmen fällt, die für jeden Bereich einzeln anfällt, ist es günstiger, wenige, aber große Bereiche festzulegen, anstelle vieler Kleiner. Dies gilt auch dann, wenn dadurch nicht benötigte Daten übertragen werden.

7.) Wählen Sie "S7 über MPI".

8.) Geben Sie in der IO-Liste unter "Node-Nr" die MPI-Adresse des Partners an und "Slot-Nr" den Steckplatz der CPU an. Die MPI-Adresse ist meistens 2, die Slot-Nr bei S7-300 immer 2, bei S7-400 meistens 3.

9.) Wenn es noch nicht getan haben, schalten Sie jetzt die SPS ein und stellen Sie sicher, dass das Kabel eingesteckt ist.

10.) Schließen Sie das “Externe SPS”-Fenster mit OK. Danach werden Sie aufgefordert, zu bestätigen, dass Sie die externe SPS anbinden wollen. Lesen Sie die Sicherheitshinweise und bestätigen danach mit OK.

11.) Der Aufbau der Verbindung dauert etwa 5 sec, dann schließt sich das “Externe SPS” - Fenster.

12.) Jetzt ist die TrySim-Anlage an die SPS angeschlossen, als sei sie eine wirkliche Anlage. Sie sollten die Simulationsgeschwindigkeit in TrySim unter **Anlage|Echte Zeit** auf Echtzeit einstellen, da der Ablauf der Zeiten in der SPS nicht beeinflussbar ist. Die Simulationswiederholrate sollten Sie unter **Ansicht|Optionen|Simulation** auf mindestens 100 ms einstellen, bei einer kürzeren Einstellung kann nicht mehr jeder Simulationsschritt zur SPS übertragen werden.

S7-300, S7-400 und ProDave sind eingetragene Warenzeichen der Siemens AG.
MPI ist eingetragenes Warenzeichen der Firma Schildknecht.

3.5.9.1 PRODAVE MPI mini from Siemens

No english help available yet.

Dies ist ein Treiber, der die Kommunikation mit einer S7 über MPI gestattet.

Auch wenn Sie auf dem TrySim-Rechner den Simatic-Manager installiert haben und eigentlich schon eine MPI-Verbindung zur SPS haben, benötigt TrySim diesen Treiber, um an der Kommunikation teilhaben zu dürfen.

Mit PRODAVE können nur Daten, keine Programme zwischen SPS und PC ausgetauscht werden.

Sie können diese Software von uns erwerben oder von Siemens direkt.

S7-300, S7-400, Simatic und ProDave sind eingetragene Warenzeichen der Siemens AG.
MPI ist eingetragenes Warenzeichen der Firma Schildknecht.

3.5.10 Interface to SoftPLC of IBH softec

No english help available yet.



Bitte lesen Sie unbedingt die [Sicherheitshinweise bei externer \(Soft-\)SPS](#) !

Um die TrySim - Anlage durch die SoftSPS oder die S5/S7 Simulatoren von IBH steuern zu lassen, gehen Sie wie folgt vor:

- 1.) Wählen Sie **SPS|Externe SPS**. Mit dem Button "I/O - Konfiguration" rufen Sie das Fenster zur Eingabe der Ein- und Ausgänge auf, die zu/von der SPS übertragen werden sollen.
- 2.) Legen Sie die Blöcke fest, die zur SPS übertragen werden sollen. (Auf der Maske steht, weil es intuitiver ist "Eingänge", aber Sie können hier genauso gut Ausgänge, Merker oder DB's angeben). Alle hier angegebenen Blöcke werden nach der Bearbeitung der Anlagensimulation auf des Prozessabbild der SoftSPS kopiert.
- 3.) Legen Sie genauso die Blöcke fest, die von der SoftSPS nach TrySim übertragen werden sollen.
- 4.) Seien Sie bei der Festlegung der Blöcke ruhig großzügig, nur bei sehr langsamen Rechnern (< 200MHz) lohnt es sich, nur die tatsächlich benötigten Blöcke zu übertragen, sonst machen ein paar zig Bytes mehr oder weniger kaum einen Unterschied in der Bearbeitungsgeschwindigkeit.
- 5.) Wenn es noch nicht getan haben, Starten Sie jetzt die IBH SoftSPS.
- 6.) Wählen Sie "IBH SoftSPS".
- 7.) Geben Sie auf der Maske "Schnittstellen-Parameter" den Pfad zur SoftSPS ein. (Siehe auch: [SoftSPS läuft noch nicht](#) und [IBH SoftSPS nicht gefunden](#)).
- 8.) Bestätigen Sie alle Masken mit "OK" bis Sie wieder im Hauptmenu angelangt sind.
- 9.) Jetzt ist die TrySim-Anlage an die IBH SoftSPS angeschlossen, als sei sie eine wirkliche Anlage. Sie sollten die Simulationsgeschwindigkeit in TrySim unter **Anlage|Echte Zeit** auf Echtzeit einstellen, da bislang keinerlei Synchronisation zwischen der IBH SoftSPS und TrySim vorhanden ist.

Siehe auch:
[Externe SPS](#)

3.5.11 Connection to external Allen-Bradley PLC

No english help available yet.



Bitte lesen Sie unbedingt die [Sicherheitshinweise bei externer \(Soft-\)SPS](#) !

Diese Anbindung ist nur möglich, wenn Sie die entsprechende Option erworben haben.

Um die TrySim - Anlage durch eine über RSLinx angebundene Allen-Bradley-PLC zu steuern, benötigen Sie die entsprechende Hard- und Software von Rockwell.

Auf der PLC-Seite benötigen Sie eine CNB (ControlNetBridge) oder eine Ethernet-Baugruppe.

TrySim spricht nur auf die auf Ihrem Rechner laufende RSLinx-Software an und hat nur wenig mit der Weiterleitung der Daten zur angepeilten CPU zu tun.

Zum Aufbau einer Verbindung müssen Sie aber in der IO-Konfiguration festlegen:

1.) die Control-Net-Node-Nr. der von Ihnen eingesetzten CNB-Baugruppe: Diese wird über kleine Räder auf der CNB eingestellt, die man nur nach Ausbau der Baugruppe erreichen kann.

2.) die Steckplatznummer (Slot-No) der CPU: Der erste Platz neben dem Netzteil hat die Nummer 0! Das ist anders als bei Siemens-SPSen und sollte daher für Umsteiger immer im Kopf behalten werden.

3.) Die Tag-Names. Sie können von TrySim aus nur Controller-Tags ansprechen! Wenn Sie die Infos von TrySim in den Tags innerhalb Routinen benötigen, müssen Sie sie selbst (z.B. mittels Alias) dorthin transferieren.

RSLinx ist eingetragenes Warenzeichen der Rockwell Software Inc.

3.5.12 Setting of I/O configuration

No english help available yet.

Diese Maske muß nur beachtet werden, wenn Sie die TrySim-Anlage durch eine externe (Soft-) SPS steuern wollen.

Auch wenn Sie einen eigenen Schnittstellentreiber verwenden, müssen Sie keine Ein- und Ausgänge konfigurieren, da Ihnen dann der gesamte E- A- und M-Bereich von je 65.536 Bytes ohne jede Konfiguration zur Verfügung gestellt wird.

Auf dieser Maske geben Sie an, welche Daten von TrySim zu einer externen SPS übertragen werden sollen, und welche Daten von der externen SPS zu TrySim übertragen werden sollen. Für jede Übertragungsrichtung können Sie mehrere Blöcke angeben. Ein Block besteht jeweils aus der Bezeichnung des Operanden-Bereichs (Eingänge, Ausgänge, Merker, Datenbausteine), der Startadresse und einer Größe, die in Bytes angegeben wird. Bei Datenbausteinen müssen Sie natürlich noch die Nummer des Datenbausteins nennen.

Wenn Daten von mehreren Stellen aus verändert werden können, tauchen oft schwer zu findende Fehler auf. Wenn Sie also TrySim mit einer externen SPS betreiben und es treten anscheinend unerklärliche Fehler auf, schauen Sie sich noch einmal in Ruhe diese I/O-Konfiguration an, gegebenenfalls auch das Gegenstück auf der SPS-Seite, und prüfen Sie, ob hier die Ursache für den Fehler liegt.

Siehe auch:

[Externe SPS](#)

3.5.13 Setting of a block of I/O configuration

No english help available yet.

Ein Block besteht jeweils aus:

- der Bezeichnung des Operanden-Bereichs (Eingänge, Ausgänge, Merker, Datenbausteine),
- bei Datenbausteinen aus der DB - Nummer
- der Startadresse (immer in Bytes ! auch bei S5 - Datenbausteinen)
- und einer Größe, die in Bytes angegeben wird

Bei der Eingabe des Operandenbereiches wird nur der erste Buchstabe berücksichtigt (E,A,M oder D), auf Groß- und Kleinschreibung brauchen Sie nicht zu achten.

[Externe SPS](#)

[I/O - Konfiguration](#)

3.5.14 Setting of a block of I/O configuration for MPI

No english help available yet.

Ein Block besteht jeweils aus:

- der Bezeichnung des Operanden-Bereichs (Eingänge, Ausgänge, Merker, Datenbausteine),
diese bezieht sich **nur** auf den Adressbereich von TrySim
- bei Datenbausteinen aus der DB - Nummer
diese Nr. bezeichnet **nur** einen Datenbaustein innerhalb von TrySim
- der Startadresse (in Bytes)
wie bei den beiden Werten vorher, nur innerhalb von TrySim
- der Datenbaustein-Nummer in der S7. Diese DB müssen Sie vor der ersten Verbindungsaufnahme eingerichtet haben.

- der Startadresse im DB in der S7. Diese sollte 0 sein, oder der Quelladresse entsprechen.
es ist zwar auch jeder andere gerade Wert möglich, aber das führt zu unnötiger Verwirrung.
- und einer Größe, die in Bytes angegeben wird

Bei der Eingabe des Operandenbereiches wird nur der erste Buchstabe berücksichtigt (E, A, M oder D), auf Groß- und Kleinschreibung brauchen Sie nicht zu achten.

Wenn Sie die Anbindung über PRODAVE MPI MINI vornehmen, müssen Sie auch für die E-Bereiche jeweils einen Datenbaustein angeben, der in der SPS eingerichtet werden muss. In diesen DB wird der E-Bereich, beginnend ab der "StartAdresse in der SPS", kopiert. Im ersten Netzwerk des OB 1 können Sie dann die SFC 20 verwenden, um die Daten auf das Prozessabbild der Eingänge zu kopieren.

Beispiel: Ihre SPS hat den Eingangsbereich EB20 - EB29. Sie richten in der SPS einen Datenbaustein DB 2 mit einer Größe von 10 Bytes ein. In TrySim erstellen Sie unter Externe SPS| IO-Konfiguration einen "Eingänge der SPS" - Block.

Bereich : E
DB-Nr. : 2
StartAdr. : 20
Größe : 10.

Im ersten Netzwerk des OB1 programmieren Sie:

```
call SFC 20
SRCBLK : P#DB2.DBX0.0 BYTE 10
RET_VAL : #Ret_val // zuvor als Temp-Word deklarieren!
DSTBLK : P#E20.0 BYTE 10
```

Mit den Ausgängen verfahren Sie sinngemäß. Die SFC 20 muss jetzt im letzten NW des OB1 aufgerufen werden. Für den Ausgangsbereich AB48 - AB63, der über den DB3 transportiert werden soll würde das dann so aussehen:

```
call SFC 20
SRCBLK : P#A48.0 BYTE 16
RET_VAL : #Ret_val // zuvor als Temp-Word deklarieren!
DSTBLK : P#DB3.DBX0.0 BYTE 16
```

Und der zugehörige "Ausgänge der SPS" Block in TrySim würde sein:

Bereich : A
DB-Nr. : 3
StartAdr. : 48
Größe : 16.

Falls Sie auch Merker übertragen wollen, verfahren Sie genauso.

MPI ist eingetragenes Warenzeichen der Firma Schildknecht.

3.6 Reference PLC

3.6.1 Data types

[BOOL](#)
[BYTE](#)
[CHAR](#)
[WORD](#)
[INT](#)
[DWORD](#)
[DINT](#)
[POINTER](#)
[TIMER](#)
[COUNTER](#)
[ARRAY](#)
[STRUCT](#)
[S5TIME](#)
[TIME](#)
[DATE](#)
[DATE AND TIME](#)
[TIME_OF_DAY](#)
[ANY](#)
[BLOCK_FB](#)
[BLOCK_FC](#)
[BLOCK_DB](#)
[REAL](#)
[STRING](#)
[UDT](#)

Not belonging to the data types but often searched here:

[Constants syntax](#)

3.6.1.1 BOOL

This data type is only one bit big. Both states are named besides '0' and '1' also 'False' and 'True'.

Examples for the use of BOOL-types:

```
A      I 5.2
=      #Lamp           // Lamp is declared as Out-BOOL
```

```
S      M[MD8]           // MD 8 is loaded with P#4.2
```

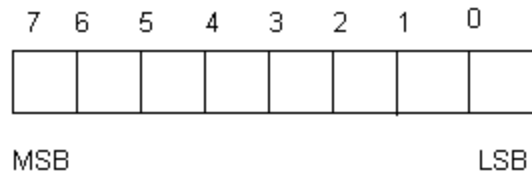
See also:

[Data Types](#)

3.6.1.2 BYTE

This data type has got a size of 8 bits. The range of values contains the numbers 0 – 255 resp. \$00 - \$FF

or $0 - (2^8 - 1)$. The bits are numbered from 0 till 7, bit 7 is the most significant bit (MSB most significant bit).



Numbering of the bits in the byte

Examples for the use of BYTE-types

```
T      MB 177
L      #Byte2           (Byte2 has been declared as In-BYTE)
L      B#16#0A         ('10' is loaded in accu 1)
```

See also:

[Data Types](#)

3.6.1.3 CHAR

This data type has got a size of one byte. It serves for saving ASCII-signs. In TrySim it is compatible for assigning to the type BYTE, that means always there where you can use a BYTE you can also use a CHAR and the other way round. If you want to export your program to STEP®7 you must not make use of this compatibility because it does not exist there.

Examples for the use of CHAR-types

```
L      'T'              (First letter of TrySim)
T      #Word[3]         (#Word is declared as ARRAY[1..5] of CHAR)
```

With the help of the operation L (Load) you can load up to 4 signs at the same time into the accu:

```
L      'afGh'
```

But if you want to connect a CHAR-parameter of a FB/FC, the constant must not be longer than one sign.

ASCII-Table :

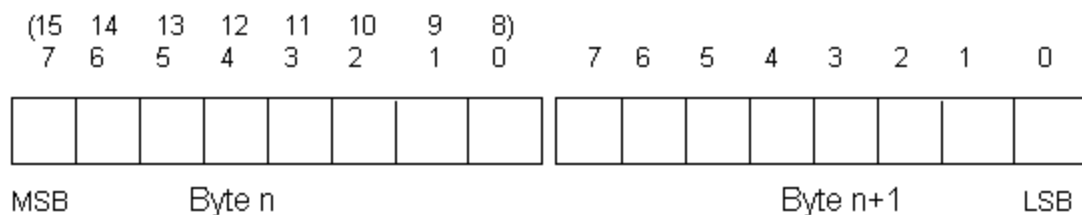
SP - 32	0 - 48	@ - 64	P - 80	` - 96	p - 112
! - 33	1 - 49	A - 65	Q - 81	a - 97	q - 113
" - 34	2 - 50	B - 66	R - 82	b - 98	r - 114
# - 35	3 - 51	C - 67	S - 83	c - 99	s - 115
\$ - 36	4 - 52	D - 68	T - 84	d - 100	t - 116
% - 37	5 - 53	E - 69	U - 85	e - 101	u - 117
& - 38	6 - 54	F - 70	V - 86	f - 102	v - 118
\ - 39	7 - 55	G - 71	W - 87	g - 103	w - 119
(- 40	8 - 56	H - 72	X - 88	h - 104	x - 120
) - 41	9 - 57	I - 73	Y - 89	i - 105	y - 121
* - 42	: - 58	J - 74	Z - 90	j - 106	z - 122
+ - 43	; - 59	K - 75	[- 91	k - 107	{ - 123
, - 44	< - 60	L - 76	\ - 92	l - 108	- 124
- - 45	= - 61	M - 77] - 93	m - 109	} - 125
. - 46	> - 62	N - 78	^ - 94	n - 110	~ - 126
/ - 47	? - 63	O - 79	_ - 95	o - 111	- 127

See also:
[ASCII-Table](#)
[Data Types](#)

STEP®7 is a registered trademark of the Siemens AG.

3.6.1.4 WORD

This type consists of two bytes. The more significant byte is always the one with the **lower** address number. This is quite unusual for programmer of the PC area because there it is exactly the other way round. The number area of the WORD contains the positive numbers from 0 - 65535 resp. \$0000 - \$FFFF.



Numbering of the bits in WORD

The numbering of the bits in the more significant byte from 8 - 16 is a little bit of academic nature,

because you can also use only the numbers 0-7 for the access to these bits all the time. The name of a word always relates to the more significant byte. AW 6 means byte 6 and byte 7.

Examples for the use of WORD-types:

```
T           MW 18
L           2#0000 0101 1111 0011
OPN DB[#DBau] // #Dbau is declared as Temp-WORD
```

See also:

[Data Types](#)

3.6.1.5 INT

The data type INTEGER corresponds to our normal numbers with sign, its range of values reaches from -32.768 till +32.767. It needs two bytes, just like the WORD.

Examples for the use of INT-types

```
T           'Sum' // 'Summand' is declared as INT in the symbol table
L           -5
```

See also:

[Data Types](#)

3.6.1.6 DWORD

This data type consists of 4 bytes (32 bits). Its range of values contains the numbers 0 till +4.294.967.295 resp. \$0000 0000 - \$FFFF FFFF.

This data type is used for the coding of bit patterns and as [POINTER](#) more often than for the representation of such big numbers in the PLC-programming.

Examples for the use of DWORD-types:

```
L           MD 46624 // In TrySim the marker area goes up to about 65 000)
A           M[#Idx] // #Idx is declared as Temp-DWORD
L           DW#16#0800 090A 040C 0E07
```

See also:

[Data Types](#)

3.6.1.7 DINT

The data type DOUBLE INTEGER will be used if the range of values of the INT is not big enough. It is 4 bytes big and its range of values contains the numbers -2.147.483 till +2.147.483.647.

Using INT and DINT - sizes at the same time you have to think about the fact that INT-numbers are always positive out of the view of a DINT. Numbers with signs will always be negative if their most significant bit is '1'. If you load a number out of an INT (Word) the both left bytes of accu '1' will be loaded with '0'. If you carry out a 'D' operation (+D, -D,...>D,<D,..) with this value after this the number will always be positive, even if it is normally negative.

See also:

[Data Types](#)

3.6.1.8 POINTER

1.) Pointers are 4 bytes big and they are used to point at bits in the I-, Q or M- or D-area. No process data are saved in it but an address inside the PLC. The internal format is difficult to read but if you use the P# - notation you realize immediately to which memory position the pointer points:

```
P#5.7    points to the Bit 5.7 of any data area
P#I 4.5  points to the Bit I 4.5
P#Q 3.1  points to the Bit Q 3.1
P#M 12.0 points to the Bit M 12.0
P#D 3.2  this notation is not permitted,
          you have to access data bits
          in another way.
```

A common example:

```
L P#6.8
T MD 10
....
....
A I[MD10]
```

2.) The data type POINTER in the declaration part of FBs/FCs is 6 bytes big, because in this case a possible DB number is saved as well. We admit honestly that we still do not know what to do with the pointers in STEP®7 in the declaration part. You cannot load them (because they are too big for the 4 bytes wide accu), you cannot use them in the L xx[AR1,P#I 5.3]-notation instead of the P#I. 5.3 and you cannot transfer them as parameter to other FBs/FCs. The only use that we have found is with some SFCs. We are very grateful for advices how to use these POINTERS.

Internally in a HSB of a pointer is initially saved to which data area it points:

```
I      81h    // Input area
Q      82h    // Output area
M      83h    // Marker area
D      84h    // Global-DB
ID     85h    // Instance-DB
L      86h    // Local data
FX     87h    // Parameter area of a function (further local data)
```

In the next 24 bits the byte address follows but it is shifted left about 3 bits. In the 3 bits which are released by this the bit number is finally saved.

See also:

[Data Types](#)

STEP®7 is a registered trademark of the Siemens AG.

3.6.1.9 TIMER

This data type does not name a special format in which values are displayed in the PLC like many of the other data types, but it is the name of one of the 512 timers that exist in the PLC. In the symbol table it will be selected automatically if the address starts with 'T'. In the declaration part of functions and function blocks it is only permitted as In-parameter, inside the block you can use the timer as usual, but it is always another timer that is started or evaluated, depending on which timer you have specified at calling the block as actual-parameter. Do not mix up the data type TIMER with S5TIME and TIME.

Examples for the use of TIMER:

```
T 47  Mot1On      TIMER      On-delay for motor 1
(Line in the symbol table)
```

```
SF  #OffDel      // #OffDel is declared as In-TIMER
```

See also:

[Operations with Timers](#)

[Data Types](#)

[Timer corresponding to IEC 6-1131.3](#)

3.6.1.10 DATE

The data format DATE is 1 word big and displays the days since January 1st, 1990 (binary coded).

01.01.1990	D#1990-01-01	Value: 0
24.12.1998	D#1998-12-24	Value: 3279

See also:

[DATE AND TIME](#)

[TIME OF DAY](#)

[Data Types](#)

3.6.1.11 DATE_AND_TIME

This data type is 8 bytes big and that is why it cannot be loaded into the only 4 bytes big accu with 'L' anymore but only transferred as parameter. A typical DATE_AND_TIME-constant looks like this:

```
DT#99-12-24-18:30:39.7   Opening of presents 1999
```

The meaning of the bytes is (BCD-coded):

Byte 0	Year
Byte 1	Month
Byte 2	Day
Byte 3	Hour
Byte 4	Minute
Byte 5	Second
Byte 6	1/100 - Second
Byte 7	(High-Nibble): 1/1000 - Second

You can save the weekday in the low-nibble of byte 7, but this function is not supported by TrySim.

With the [IEC-Function](#) FC 3 you can assemble a DT-variable out of [DATE](#) and [TIME OF DAY](#).

The IEC-function FC6 extracts the date out of a DT variable.

The IEC-function FC8 extracts the time of day out of a DT variable.

The IEC-function FC9 compares two DTs to equal.

The IEC-function FC12 compares DT1 >= DT2.

The IEC-function FC14 compares DT1 > DT2.

The IEC-function FC18 compares DT1 <= DT2.

The IEC-function FC23 compares DT1 < DT2.

The IEC-function FC28 compares DT1 <> DT2.

Please note that DT parameter of functions are transferred in TrySim in a completely different way as in STEP ®7. That is why you should avoid accesses to DT parameter by the address register.

See also:

[Data Types](#)

3.6.1.12 TIME_OF_DAY

This data type is 4 bytes big and is displayed internally by the number of the run out milli seconds of the current day (binary coded). A typical TIME_OF_DAY-constant looks like this:

```
TOD#23:12:04.090    Time to go to bed
```

See also:

[DATE](#)

[DATE AND TIME](#)

[Data Types](#)

3.6.1.13 ANY

This data type is a [pointer](#) that points at any data area and determines at the same time how many data of which type can be found there.

A typical notation is:

```
P#M22.0 WORD 3
```

This expression names the marker words MW 22, MW 24 and MW 26. That is why the data type ANY has got many things in common with the data type [ARRAY](#). While calling a FB with an array parameter all elements of the array are really transferred, only the 10 bytes that are necessary for the specification of an ARRAY are transferred calling with an ANY parameter.

But you can also specify a marker word, a structure in a DB or in the local data as actual-parameter for an ANY.

ANYs are quite difficult to handle and if it is not necessary we recommend not to use them in TrySim. Unfortunately we have to admit that we did not understand the valid syntax and semantics of the ANYs in STEP®7 completely. In addition to this it seems to change slightly from STEP®7 version to STEP®7 version. If you have got any problems with ANYs please contact us immediately, here [a mistake by us](#) is so probably like in no other area of TrySim.

If you connect an operand to a parameter that is declared as ANY please note the following characteristics:

1.) You can write the operand as constant like it is described above:

e.g. P#DB100.DBX 3.0 S5TIME 3

Then the 10 bytes that are necessary for the specification of these constants are transferred to the called block without modification.

2.) You can also specify parameter and variables as operands which are declared in the block header. The translator determines automatically the correct address. (Unfortunately it writes it like this then, that is why you hardly recognize your operand but we are going to change this sometime.) During runtime the following conversions are still done:

- 2.a) If it is about a parameter or a [static variable](#) not the identification 85h (for instance data) will be transferred but the identification 84h (for global DB) and the current IDB-No.
 - 2.b) If it is about a local variable not the identification 86h will be transferred but the identification 87h (for further local data).
 - 2.c) You cannot transfer parameter of functions (FC) to an ANY-parameter
- 3.) But if you specify a parameter or a variable that is declared as ANY the address of this parameter is not transferred exceptionally but the 10 bytes that are at this address are transferred.

See also:

[Internal ANY Data Format](#)

[Pointer](#)

[Data Types](#)

STEP®7 is a registered trademark of the [Siemens AG](#).

3.6.1.14 BLOCK_FB

You can transfer a function block as parameter so that varying functions can be executed in the called block. For the call you have to use one of the operations [UC](#) or [CC](#). Please note the dangers which are described there calling function blocks which have In/Out-parameter or use [static variables](#).

The operation [CALL](#) is reserved for calls with specification of parameter. But parameter cannot be specified calling a BLOCK_FB because it is not known at the period of development which FB will be called.

See also:

[Data Types](#)

3.6.1.15 BLOCK_FC

You can also specify a function as parameter so that varying functions can be executed in the called block. For the call you have to use one of the operations [UC](#) or [CC](#). Note the dangers which are

described there at calling functions which have got In/Out-parameter.

The operation [CALL](#) is reserved for callings with the specification of parameter. But parameter cannot be specified at calling a BLOCK_FC because it is not known at the period of development which FC will be called.

See also:

[Data Types](#)

3.6.1.16 BLOCK_DB

You can transfer a block as parameter so that varying functions are executed in the called block.

See also:

[Data Types](#)

3.6.1.17 REAL

This data type has got a size of 4 bytes. The internal format is hardly to read for human beings. The number area goes from about -10 to the 38th (power) up to -10 to the -38th (power), nought and then from 10 to the -38th (power) up to 10 to the 38th (power). Entering REAL constants you have to enter a point in every case, that means '8.0' and not '8', otherwise your system interprets your entering as integer which leads to absurd results evaluating as REAL number.

[RND](#) Converts REAL into DINT

[DTR](#) Converts DINT into REAL

[Calculating Operations](#)

[Trigonometric Operations](#)

[Data Types](#)

3.6.1.18 STRING

STRINGS are implemented from version 1.5 onwards. You can declare Strings, use them as parameter of functions and import and export them. You access the single letters with the notation stringname[idx].

The following functions are available in the directory IECfuncs for the process of Strings until now:

FC 21 Determine the length of a String.

Others are in preparation, please ask if required.

Data types

3.6.1.19 UDT

This is an **user defined type**.

If big structures are needed quite often it will be annoying to have to write them out in the declaration parts of the blocks and in the DBs each time. That is why you can define structures as UDTs and use them like normal [data types](#) afterwards.

UDTs are defined like function block headers, apart of the fact that no declaration type has to be specified.

UDTs can also serve as 'Mask' to create new data blocks.

If you have entered an UDT in the symbol table you will be able to use the symbol as type name.

PLC-Editor

3.6.2 Constants

Constants can be entered in different representations:

Binary-constant e.g. 2#1100, each decimal place stands for one bit and can only be 0 or 1. This constant can be used for bytes, words and DWords. Naturally the maximum number of bits has to be considered.

Byte constant hex e.g. B#16#F5, each place stands for a half byte and can only be 0..9, A..F. The maximum value of these constants is FF hex or 255 decimal.

Word constant hex e.g. W#16#6F34, each place stands for a half byte and can only be 0..9, A..F. The maximum value of these constants is FFFF hex or 65535 decimal.

Word constant as two decimal bytes e.g. B#(152,43), the both numbers specify the both bytes of the word and are specified decimal. The maximum value of these constants is B#(255,255).

DWord constant hex e.g. DW#6FA42322, each place stands for a half byte and can only be 0..9, A..F. The maximum value of these constants is FFFF FFFF hex or about 4 billion decimal.

Integer constant e.g. -5. This constant corresponds to our whole numbers. The range of values goes from -32.768 up to +32.767.

Double integer constant e.g. L#334124. This constant corresponds to the integer constant with an enlarged values of -2.147.483.648 up to +2.147.483.647. If you use the double integer operations +D, -D, *D, /D, <D, >D and so on and load a negative constant it will be important that you put the L# in front of it.

REAL constants e.g. 3.5 or 5.342124e+002. If you calculate with REAL numbers or compare these and use constants doing this it will be important that you enter the '.' as well. 5 and 5.0 are completely different to a PLC!

S5-time constants e.g. S5T#1H2M. The syntax is described in [S5Time](#). You need this constant to start timer. Do not use a constant of the type TIME, e.g. T#200MS!

Counter constants e.g. C#59. With this you load the specified number [BCD](#)-coded. The range of values goes from C#0 up to C#999.

Time constants e.g. T#5D2H5M. With this you load the specified duration in ms as double integer (DINT).

[Data Types](#)

3.7 List of operations

See also: [List of operations \(alphabetical\)](#)

Bit-operations

A	A(AN	AN(
O	O(ON	ON(
X	X(XN	XN(
)(closing bracket)			
= (assignment) S		R	
SET	CLR	NOT	SAVE
FP	FN		

Operations with timers

SD	SF	SP
SEI	SS	ZW (Parameter)

Operations with counters

CU CD Set Counter Reset Counter
FR

Load and transfer operations

L T LC

Operations with integers

+I -I *I /I
MOD
==I <>I >I >=I
<I <=I NEGI
+ (Plus)

Operations with double integers

+D -D *D /D
MOD
==D <>D >D >=D
<D <=D NEGD
+ (Plus)

Operations with reals

+R -R *R /R
==R <>R >R >=R
<R <=R
ABS NEGR
SQRT SQR LN EXP

Jump operations

JU JCN JMZ JN
JP JPZ JZ JNB
JC JCB JBI JNBI
LOOP JL

Not implemented jumps:

JO JUO JOS

Shift and rotating operations

SRD SSD
RLD RLDA LD SLW
RRDA RRD SSI SRW

Logical word and double word operations

AD AW OW OD
XOD XOW
INVI INVI

BCD operations

[ITB](#) [DTB](#) [BTD](#) [BTI](#)

Other conversions

[DTR](#) [ITD](#) [TRUNC](#)
[RND](#) [RND+](#) [RND-](#)

Trigonometric operations

[SIN](#) [COS](#) [TAN](#)
[ASIN](#) [ACOS](#) [ATAN](#)

Misc. operations with accu 1

[TAK](#) [CAW](#) [CAD](#)
[INC](#) [DEC](#) [+ \(Plus\)](#)

Operations with accu 3 and 4

[PUSH](#) [POP](#) [LEAVE](#) [ENT](#)

Register indirect addressing

[LAR1 or 2](#) [+AR1 or 2](#) [CAR1 or 2](#) [CAR](#)

Operations for blocks

[OPN](#) [BEU](#) [BEC](#)
[CALL](#) [CC](#) [UC](#)
[CDB](#)

Nought / Zero operations

[NOP 0 or 1](#) [BLD](#)

Master control relay (not implemented)

[MCR\(\)MCR](#) [MCRA](#) [MCRD](#)

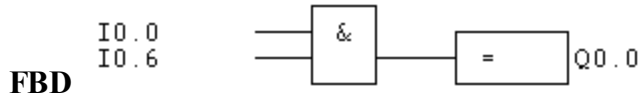
3.7.1 Bit-operations

3.7.1.1 A

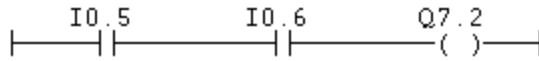
AND


STL

Combines the value of the operand and the [RLO](#) with the operation 'And' and stores the result in RLO. If the RLO is scanned (abgefragt??) before the value of the operand will be transferred into the RLO without logic operation (Verknüpfung??).



The output of the And-block will be '1' if all outputs are '1'. The And-block is created by the symbol '&' or by the list and selection of 'And'.



Here the AND-operation is created by series connection of several contacts. The whole chain will only conduct current if all contacts are closed. A new contact is created by the symbol  or by the list and selection of 'And'.

See also:

[AN](#)
[A\(](#)
[AN\(](#)
[O](#)
[ON](#)
[O\(](#)
[ON\(](#)
[X](#)
[XN](#)
[X\(](#)
[XN\(](#)
[\)](#)

[List of operations](#)

3.7.1.2 A(

AND with subsequent branch.

Starts a bracket expression whose result is combined with the [RLO](#) 'And' after the closing ')' bracket. The first operation after 'A(' transfers the value of the operand without logic operation into the RLO.

See also:

[A](#)
[AN](#)
[AN\(](#)
[O](#)
[ON](#)

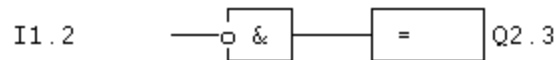
[O\(](#)
[ON\(](#)
[X](#)
[XN](#)
[X\(](#)
[XN\(](#)
[\)](#)

[List of operations](#)

3.7.1.3 AN

AND NOT

Combines the negated value of the operand and the [RLO](#) with the operation 'And' and stores the result in the RLO. If the RLO is scanned (??abgefragt) before the negated value of the operand will be transferred into the RLO without logic operation (??Verknüpfung).



See also:

[A](#)
[A\(](#)
[AN\(](#)
[O](#)
[ON](#)
[O\(](#)
[ON\(](#)
[X](#)
[XN](#)
[X\(](#)
[XN\(](#)
[\)](#)

[List of operations](#)

3.7.1.4 AN(

AND NOT with subsequent branch

Starts a bracket expression whose negated result is combined with the [RLO](#) 'And' after the closing

)' bracket. The first operation after 'AN(' transfers the value of the operand without logic operation into the RLO.

See also:

- [A](#)
- [AN](#)
- [A\(](#)
- [O](#)
- [ON](#)
- [O\(](#)
- [ON\(](#)
- [X](#)
- [XN](#)
- [X\(](#)
- [XN\(](#)
- [\)](#)

[List of operations](#)

3.7.1.5 O

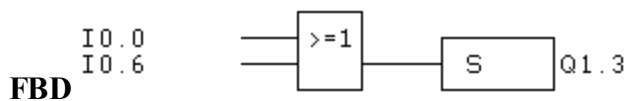
OR

STL


Starts an 'Or'-connection by the And-before-Or-rule. Internally 'O' is replaced by 'O(' without operand. Before the next RLO-evaluating operation the ')' is inserted automatically.

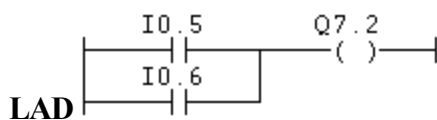
OR with Operand





Combines the value of the operand and the RLO with the operation 'Or' and stores the result in the RLO. If the RLO has been evaluated before, the value of the operand will be copied to the RLO without connection.



FBD

The output of the Or-block will be '1' if at least one of the inputs is '1'. The Or-block is created by the symbol  or via the list and selection of 'Or'.



Here the Or-connection is reached by the parallel connection of several contacts. The whole path will be conductive if at least one of the contacts is closed. Using the symbol  is the easiest way to create an Or-connection (doing this please note where the yellow cursor is). After inserting the desired contacts by  or  the branch is closed again by . Before doing this put the yellow cursor on the contact **behind** which the branch shall be closed again.

See also:

[A](#)

[AN](#)

[A\(](#)

[AN\(](#)

[ON](#)

[O\(](#)

[ON\(](#)

[X](#)

[XN](#)

[X\(](#)

[XN\(](#)

[\)](#)

[List of operations](#)

3.7.1.6 O(

OR with subsequent branch

Starts a bracket-expression whose result is combined with the RLO 'Or' after the closing ')' - bracket. The result is stored in the RLO. The first operation after 'O(' transfers the value of the operand without combination/logic operation into the RLO.

See also:

[A](#)

[AN](#)

[A\(](#)

[AN\(](#)

[ON](#)

[ON\(](#)

[X](#)

[XN](#)

[X\(](#)

[XN\(](#)

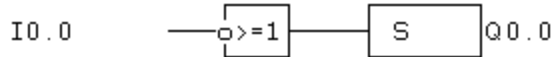
[\)](#)

[List of operations](#)

3.7.1.7 ON

OR NOT

Combines the negated value of the operand with the RLO and stored the result in the RLO. If the RLO has been evaluated before, the negated value of the operand is copied directly to the RLO.



See also:

[A](#)

[AN](#)

[A\(](#)

[AN\(](#)

[O](#)

[O\(](#)

[ON\(](#)

[X](#)

[XN](#)

[X\(](#)

[XN\(](#)

[\)](#)

[List of operations](#)

3.7.1.8 ON(

OR NOT with subsequent branch

Starts a bracket expression whose negated result is combined with the RLO 'Or' after the closing ')' -bracket. The result is stored in the RLO. The first operation after 'O(' transfers the value of the operand into the RLO.

See also:

[A](#)

[AN](#)

[A\(](#)

[AN\(](#)

[O](#)

[ON](#)

[O\(](#)
[X](#)
[XN](#)
[X\(](#)
[XN\(](#)
[\)](#)

[List of operations](#)

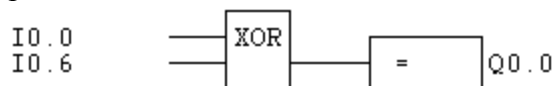
3.7.1.9 X

EXCLUSIVE OR

Combines the value of the operand with the [RLO](#) by the 'exclusive or' rule and stores the result in the RLO.

Value of the operand	Old RLO	New RLO
0	0	0
1	0	1
0	1	1
1	1	0

If the [RLO](#) has been evaluated before, the value of the operand will be loaded without logic operation.



See also:

[A](#)
[AN](#)
[A\(](#)
[AN\(](#)
[O](#)
[ON](#)
[O\(](#)
[ON\(](#)
[XN](#)
[X\(](#)
[XN\(](#)
[\)](#)

[List of operations](#)

3.7.1.10 X(

EXCLUSIVE OR with subsequent branch

Starts a bracket expression whose result will be combined with the [RLO](#) 'Exclusive Or' at the closing "(" bracket. The first operation after 'X(' transfers the value of the operand without logic operation into the RLO.

See also:

[A](#)

[AN](#)

[A\(](#)

[AN\(](#)

[O](#)

[ON](#)

[O\(](#)

[ON\(](#)

[X](#)

[XN](#)

[XN\(](#)

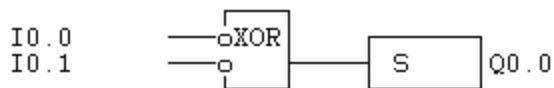
)

[List of the operations](#)

3.7.1.11 XN

EXCLUSIVE OR NOT

Combines the negated value of the operand with the [RLO](#) by the 'Exclusive Or' rule and stores the result in the RLO. If the RLO has been evaluated before, the value of the operand will be loaded without logic operation into the RLO.



This operation cannot be programmed in LAD.

See also:

[A](#)

[AN](#)

[A\(](#)

[AN\(](#)

[O](#)
[ON](#)
[O\(](#)
[ON\(](#)
[X](#)
[X\(](#)
[XN\(](#)
[\)](#)

[List of operations](#)

3.7.1.12 XN(

EXCLUSIVE OR NOT with subsequent branch

Starts a bracket expression whose negated result is combined with the [RLO](#) 'Exclusive Or' at the closing '(' bracket. The first operation after 'XN(' loads the value of the operand without logic operation into the RLO.

See also:

[A](#)
[AN](#)
[A\(](#)
[AN\(](#)
[O](#)
[ON](#)
[O\(](#)
[ON\(](#)
[X](#)
[XN](#)
[X\(](#)
[\)](#)

[List of operations](#)

3.7.1.13) (closing bracket)

With the ')' the last programmed but not yet closed bracket expression is closed and the result of the bracket expression above the operation, that is specified while opening the bracket expression, gets linked with the then [RLO](#) and is stored as new RLO.

The operations that can be specified while opening a bracket expression are:

[A\(](#)
[AN\(](#)
[O\(](#)
[ON\(](#)
[X\(](#)
[XN\(](#)

[List of operations](#)

3.7.1.14 = (assignment)

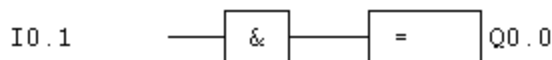
Operands:

I, Q, M
 DBX, DIX,
 BOOL parameter
 BOOL variables

STL

Syntax: = Q 5.2

The content of the [RLO](#) is copied to the specified bit operand. After an assignment a new linking chain is started, that means the first operation after a '=' always gives the value of an operand, indifferent, whether the operation is named 'A', 'O' or 'X'. The operations 'AN', 'ON' and 'XN' however load the RLO with the negated value of the operand.

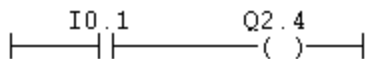


FBD

The result of the precede logic operation is copied to the specified operand. The assignment block is usually created automatically. But you can also call the [list](#) and '=' select assignment'.

You can change the '=' sign to 'S' or 'R' by the space bar or by 'Shift-ArrowRight'. To do so the yellow cursor has to stand on the block.

LAD



In LAD the assignment is usually named 'coil'. The coil will be switched on if current flows through it. Usually the coil is created automatically. But you can also call the [list](#) and '=' select assignment'.

You can change the ‘-()-’ sign to ‘-(S)-’ or ‘-(R)’ by the space bar or by ‘Shift-ArrowRight’. To do so the yellow cursor has to stand on the block.

See also:

[S](#)
[R](#)

[List of operations](#)

3.7.1.15 S

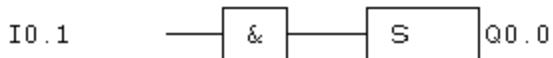
Operands: I, Q, M
DBX, DIX,
BOOL-parameter
BOOL-variables

STL

Syntax: A I 0.1 // e.g.
S Q 5.2

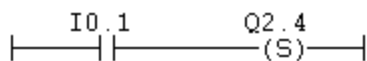
If the [RLO](#) is ‘1’ the specified bit operands will be set to ‘1’. If the RLO is ‘0’ nothing will happen at all. After this operation a new logic operations sequence ?? (Verknüpfungskette) is started, that means that the first operation after a ‘S’ always gives only the value of the operand, unimportant whether the operation is named ‘A’, ‘O’ or ‘X’. The operations ‘AN’, ‘ON’ and ‘XN’ in comparison load the RLO with the negated value of the operand.

FBD



If the result of the precede logic operation ?? (Verknüpfung) is ‘1’ the specified operand is set to ‘1’. If the result of the precede logic operation ?? (Verknüpfung) is ‘0’ nothing will happen at all, that means that the operand is not set back again. This has to happen at another position in the program, e.g. by the operation [R](#). You create this block by creating an [assignment block](#) first, putting the yellow cursor on it and pressing the space bar or ‘Shift-ArrowRight’.

LAD



Here the coil ?? (Spule) will only be switched on if current flows. If no current flows it will not be switched off again, this has to happen at another position in the program, e.g. by the operation [R](#).

You create the ‘-(S)-’ - sign by creating a normal coil ?? (Spule) first and pressing the space bar or ‘Shift-ArrowLeft’ after that. The yellow cursor has to be on the coil ?? (Spule) for doing so.

See also:

[= \(assignment\)](#)

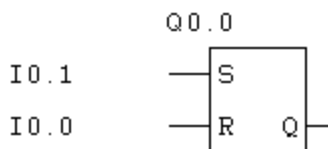
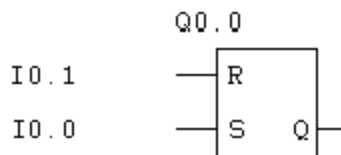
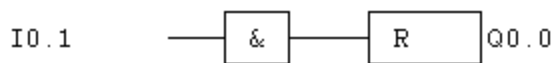
[R](#)

[FlipFlops](#)

[List of operations](#)

3.7.1.16 R

If the input is ‘1’ the output will be set back, otherwise nothing happens not all.



See also:

[S](#)

[List of operations](#)

3.7.1.17 SET

The RLO is just set to ‘1’, there is nothing else to say about it.

See also:

[CLR](#)

[NOT](#)

[SAVE](#)

[List of operations](#)

3.7.1.18 CLR

This instruction just sets the [RLO](#) to '0', there is nothing else to say about it.

See also:

[SET](#)

[NOT](#)

[SAVE](#)

[List of operations](#)

3.7.1.19 NOT

If the RLO is 1 it will become 0 and if it is 0 it will become 1.

This instruction is also used by the system to express the negation of a FBD-AND/OR-block in STL.

See also:

[CLR](#)

[SET](#)

[SAVE](#)

[List of operations](#)

3.7.1.20 SAVE

The current RLO is stored in the [BR-bit](#). The BR-bit is analysed when returning of a function or a function block, in FBD and LAD the BR-bit is the ENO-output.

See also:

[CLR](#)

[SET](#)

NOT

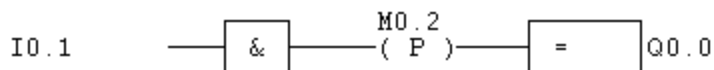
List of the operations

3.7.1.21 FP

If the input has got a rising (positive) edge the output will be set to '1' for one cycle. Please see to it that the edge trigger flag (in the picture M 0.2) is not used anywhere else (that means neither in the program nor in the machine)!

Syntax:

```
A      I 0.1
FN     M 0.2
=      Q 0.0
```



Remark:

In STEP®7 FBD/LAD there is also another display of the edge that is noted in STL like this:

```
A      I 0.1
BLD    100
FN     M 0.2
=      Q 0.0
```

This display is not supported by TrySim.

See also:

[FN](#)

List of operations

STEP®7 is a registered trademark of the Siemens AG.

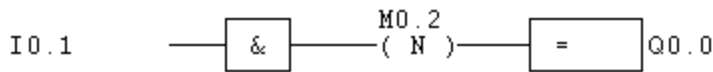
3.7.1.22 FN

If the input has got a falling (negative) edge the output will be set to '1' for one cycle. Please see to it that the edge trigger flag (in the picture M 0.2) is not used anywhere else (that means

neither in the program nor in the machine)!

Syntax:

```
A      I 0.1
FN     M 0.2
=      Q 0.0
```



Remark:

In STEP®7 FBD/LAD there is also another display of the edge that is noted in STL like this:

```
A      I 0.1
BLD    100
FN     M 0.2
=      Q 0.0
```

This display is not supported by TrySim.

See also:

[FP](#)

[List of operations](#)

STEP®7 is a registered trademark of the Siemens AG.

3.7.2 Operations with timers

In the PLC there are 512 ‘Timers’ that can be used for the delay of events or for the creation of pulses. The timers are numbered from T0 till T511, naturally you can name them properly by the [symbol table](#). It is not specified right from the beginning which function a timer has got but it is specified starting the timer. The following functions are possible:

[On-Delay](#)

[Off-Delay](#)

[Pulse](#)

[Extended Pulse](#)

[Saving On-Delay](#)

In the example ‘For_Beginners’ the use of timers is demonstrated.

[Insert Timers in FBD/LAD.](#)

How to enter the duration of a timer you can read [here](#).

The timers can also be specified memory-indirect or register-indirect.

Operati Operands Description

on

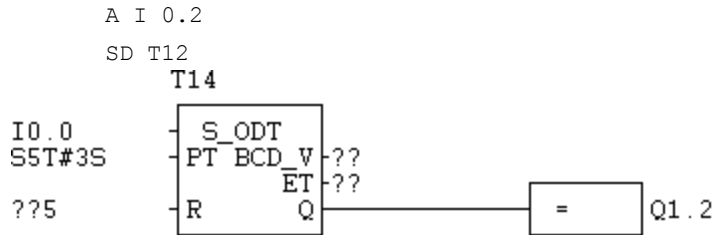
SE	T TIMER- Parameter	Will start a timer as on-delay if the RLO changes from '0' to '1'.
SA	T TIMER- Parameter	Will start a timer as off-delay if the RLO changes from '1' to '0'.
SI	T TIMER- Parameter	Will start a timer as pulse if the RLO changes from '0' to '1'.
SV	T TIMER- Parameter	Will start a timer as extended pulse if the RLO changes from '0' to '1'.
SS	T TIMER- Parameter	Will start a timer as saving on-delay if the RLO changes from '0' to '1'.
R	T TIMER- Parameter	If the RLO is '1' the timer will be reset. As long as the RLO is '1' the evaluation of the timer comes to '0' in every case and a query of the run off timer with L Txx or LC Txx is '0' in every case.
L	T TIMER- Parameter	Loads the current value of the timer into accu 1. The timers run from the value with which they are started down until '0'. With this instruction the current value is loaded binary coded in units of the time base. Evaluating of the so loaded number it has to be considered with which time base the timer was started.
LC	T TIMER- Parameter	Loads the current value of the timer BCD-coded with the time base into accu 1. A timer value that is loaded so can directly be used to start another timer.

The function 'FR', enable a timer is not implemented in TrySim.

See also:

[Data Types](#)

3.7.2.1 SD



[Timing diagram Einschaltverzögerung??](#)

See also:

[Operations with times](#)

[SP](#)

[SE](#)

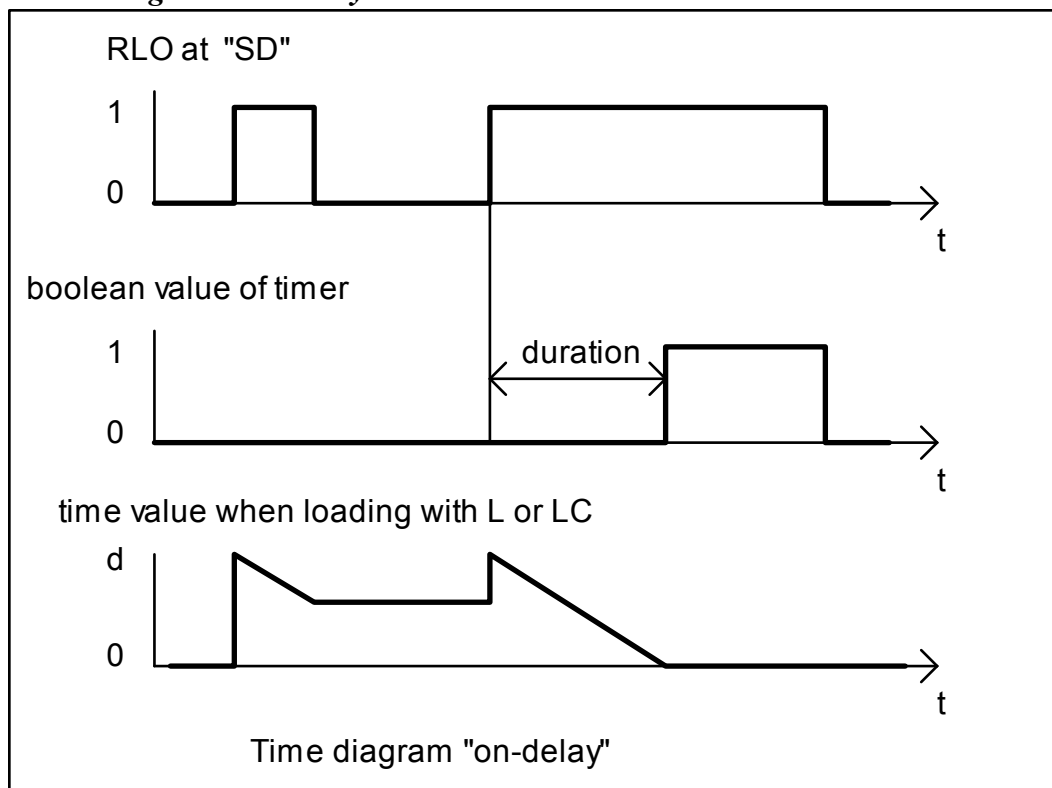
[SS](#)

[SF](#)

[List of operations](#)

3.7.2.1.1 Switch on delay

Time-Diagram 'On-Delay'

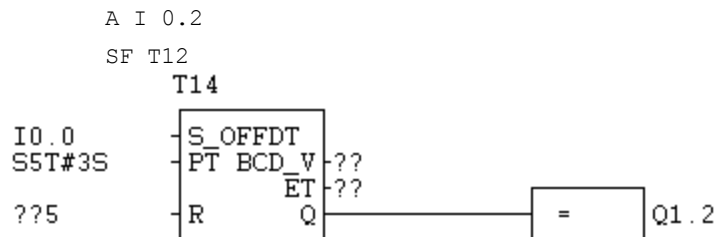


See also:

- [Time-Diagrams 'Off-Delay'](#)
- [Time-Diagrams 'Pulse'](#)
- [Time-Diagrams 'Extended Pulse'](#)
- [Time-Diagrams 'Retentive-On-Delay'](#)

[Operations with Timers](#)

3.7.2.2 SF



[Timing diagram Off-Delay Timer](#)

See also:

[Operations with times](#)

[SP](#)

[SE](#)

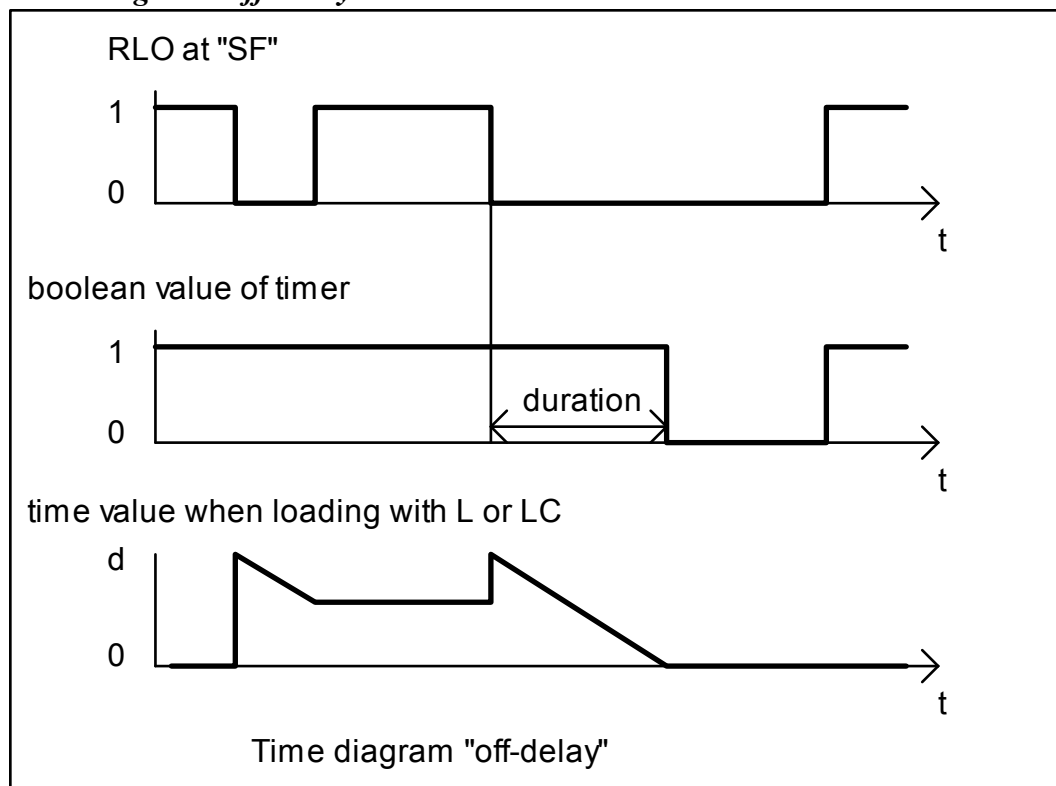
[SD](#)

[SS](#)

[List of operations](#)

3.7.2.2.1 Switch off delay

Time-Diagram 'Off-Delay'



See also:

[Time-Diagrams 'On-Delay'](#)

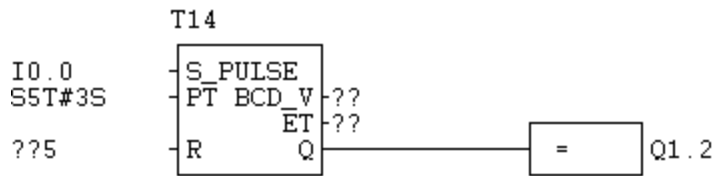
[Time-Diagrams 'Pulse'](#)

[Time-Diagrams 'Extended Pulse'](#)

[Time-Diagrams 'Retentive-On-Delay'](#)

[Operations with Timers](#)

3.7.2.3 SP



[Timing diagram impuls](#)

See also:

[Operations with times](#)

[SE](#)

[SD](#)

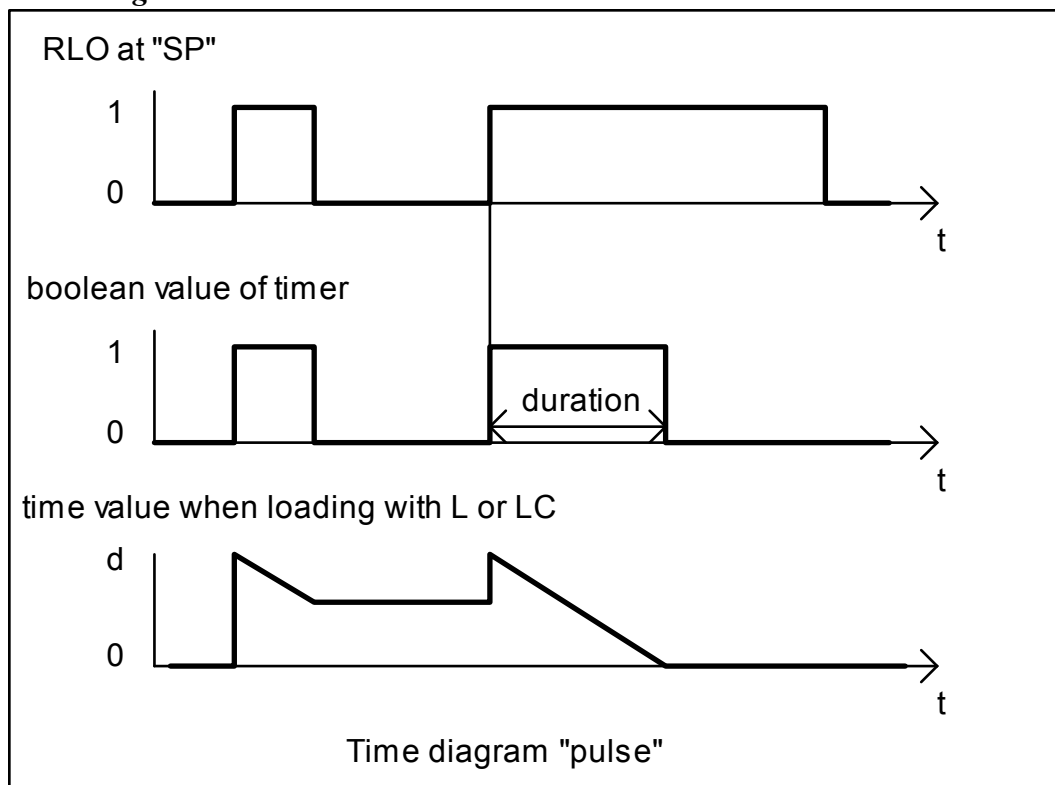
[SS](#)

[SF](#)

[List of operations](#)

3.7.2.3.1 Pulse

Time-Diagram 'Pulse'



See also:

[Time-Diagrams 'On-Delay'](#)

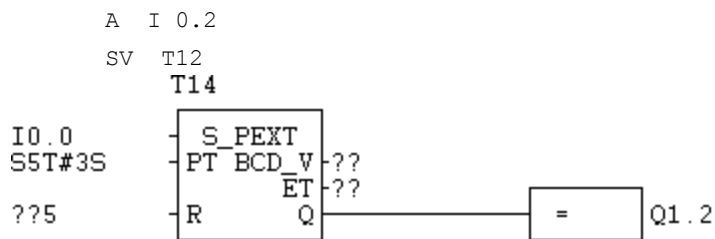
[Time-Diagrams 'Off-Delay'](#)

[Time-Diagrams 'Extended Pulse'](#)

[Time-Diagrams 'Retentive On-Delay'](#)

[Operations with Timers](#)

3.7.2.4 SE



[Timing diagram extended impulse](#)

See also:

[Operations with times](#)

[SP](#)

[SD](#)

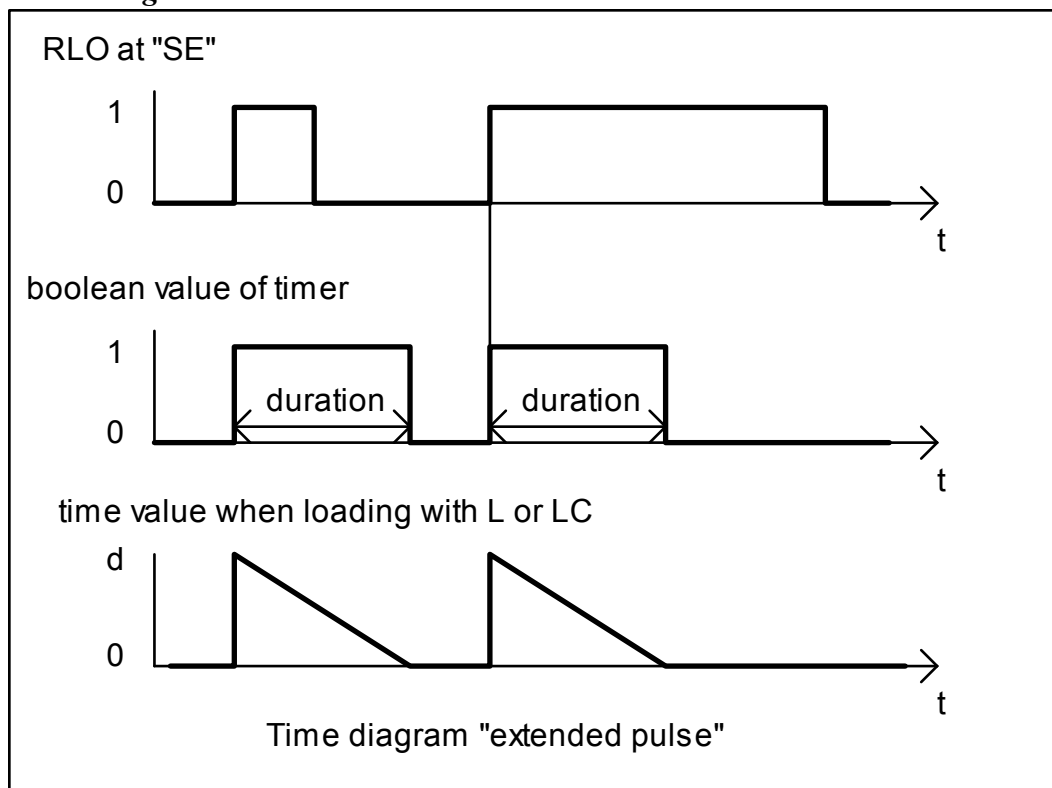
[SS](#)

[SF](#)

[List of operations](#)

3.7.2.4.1 Extended Pulse

Time--Diagram 'Extended Pulse'

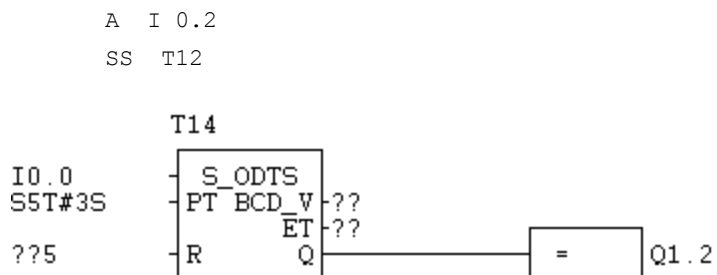


See also:

- [Time-Diagrams 'On-Delay'](#)
- [Time-diagrams 'Off-Delay'](#)
- [Time-Diagrams 'Pulse'](#)
- [Time-Diagrams 'Retentive-On-Delay'](#)

[Operations with Timers](#)

3.7.2.5 SS



[Timing diagram retentive on-delay timer](#)

See also:

[Operations with timers](#)

[SP](#)

[SE](#)

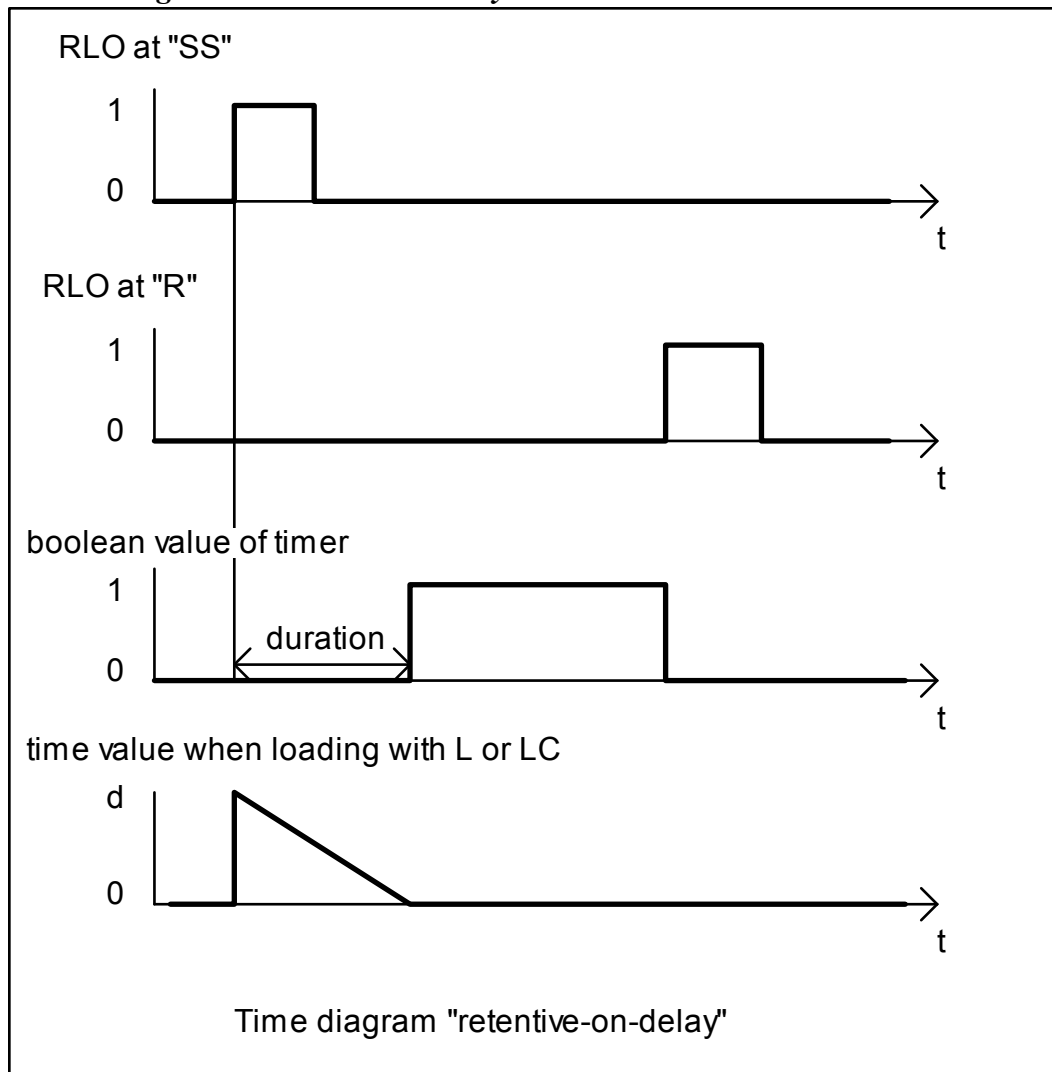
[SD](#)

[SF](#)

[List of operations](#)

3.7.2.5.1 Retentive on delay

Time-Diagram 'Retentive On-Delay'



See also:

[Time-Diagrams 'On-Delay'](#)

[Time-Diagrams 'Off-Delay'](#)

[Time-Diagrams 'Pulse'](#)

[Time-Diagrams 'Extended Pulse'](#)

[Operations with Timers](#)

3.7.2.6 PV

The counter will be set to this preset value if a rising slope occurs at the [S-input](#). It is very important that this value is specified as [BCD](#). If you want to specify a constant here you shall use the C#xyz - notation. If you want to use a number that is calculated and stored before as PV-value the function [ITB](#) will be helpful.

See also:

[Counter in general](#)

[S](#) Set the counter to the value ZW

[R](#) If this input is "1" the counter value will be set to '0'

[CU](#) Count up

[CD](#) Count down

[FR \(not implemented\)](#)

[List of operations](#)

3.7.2.7 Output DU of a timer

You can connect any word variable to this output of the timer. Into this variable the current value of the timer is transferred continuously. This is not needed for simple PLC programs. At this output the current value of the timer is displayed binary coded in the [time base](#) with which the timer was started. But the time base is not existing anymore in the displayed value, that is why you have to know with which time base the timer was started. In case of doubt it is better to use the value that is displayed at the [DEZ-output](#) to convert this one into milli seconds then.

[Timers in general](#)

3.7.2.8 Output DE of a timer

You can connect any word-variable to this output of the timer. Into this variable the current value of the timer is transferred continuously. For simple PLC programs this is not needed.

At this output the current value of the timer is displayed [BCD-coded](#) together with the time base with which the timer was started. That means that you can use it directly to start another timer.

If you want to convert the selected duration in ms you will be able to program the following STL-code, for example:

```

LC Tx          // 0 < x < 511; LC = Load coded
T #TimeW      // #TimeW is declared as word-variable
L W#16#0FFF   // Mask for BCD-coded rough-value
UW           // fade out most significant nibble
BTI          // BCD - INT converting
T #Roh       // #Rough is declared as Int-variable
L #ZeitW     // intermediate saved value
SRW 12      // shift 12 bits to the right side
SPL err     // Jump list
SPA R0      // Time base 10 milli seconds
SPA R1      // Time base 100 milli seconds
SPA R2      // Time base 1 second
SPA R3      // Time base 10 seconds
err : NOP 0
R0  : L 10          // 10 milli seconds the value
      SPA Mul
R1  : L 100        // 100 ms
      SPA Mul
R2  : L 1000       // 1.000 ms = 1 s
      SPA Mul
R3  : L 10000      // 10.000 ms = 10 s
Mul : L #Roh
      *D           // Attention! The result can be > 32767
                       // Now the time in milli seconds is binary-coded as 32 bit-
                       // number in accu 1.

```

[DU-Output](#)
[Timers in general](#)

3.7.2.9 Insert Timers in FBD/LAD

Position the yellow cursor at the desired position.

Select the desired timer-type out of the list.

Name the timer above the block with the [number of the timer](#).

Specify at the input at the top left when the timer is to be started. Naturally you can add a whole logic-tree there.

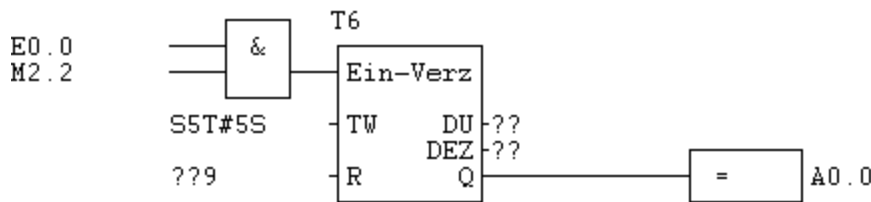
Specify the duration. In the easiest case this is a timer-[constant](#). You can also enter every word memory but you have to see to it that there is a valid duration.

You shall leave the reset input R unconnected if possible. Except of the saving on-delay it is not needed and it does more harm than good.

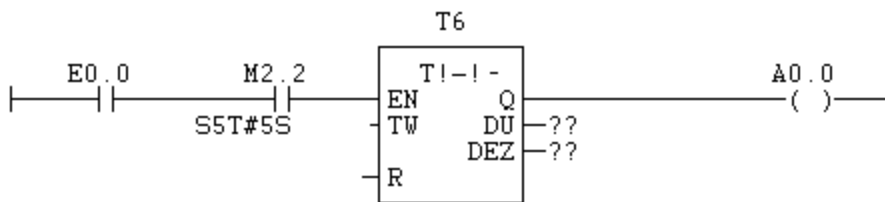
Also the DU- and DEZ-outputs are not needed in general. Here you are able to transfer the value of the run out timer in two formats into the word memory.

But the Q-output is important. Here it is indicated whether the timer is run out or not. If you use the timer not only in this network your program will be easier to read if you connect a boolean variable (marker, stat- or temp) here and use it in the further course.

Example of a timer in FBD:



Example of a timer in LAD:



And this is like a timer looks like in STL:

```

U E 0.0
U M 2.2
L S5T#5S
SE T 6
NOP 0
NOP 0
NOP 0
U T 6
= A 0.0
    
```

[Timers in general](#)

3.7.2.10 Specification of a timer

Here you have to select which timer you want to use. The names go from 'T 0' up to 'T 511', but please note that not every CPU of the S7-300/400 series has got all these timers.

You really have to note that you use each timer just once. The double use of timers is an often and quite hard to find error in the PLC programming. If two persons just use one egg timer to boil eggs at different times this often turns out well, but it will be a problem if they want to do this at nearly the same time. In the [cross reference list](#) you can check whether a timer was already used. If you enter each timer that you use consequently into the [symbol table](#) you will realize immediately if you enter a timer twice by mistake, because then a symbol is displayed instead of e.g. T56.

[Timers in general](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.7.2.11 Entering of duration while using timers

While starting the duration of timers has to be there in accu 1 resp. has to be specified in FBD or LAD at the TW-input in the 4-digit BCD format, the fourth decimal place (thousands) specifies the time base:

0	0,01s
1	0,1 s
2	1 s
3	10 s

The loading of the accu with such a coded number happens easiest by the instruction L S5T#. The duration is specified in the format '*hHmMsSmsMS*'.

Example:

S5T#1H20M	1 hour 20 minutes
S5T#3M10S400MS	3 minutes 10 seconds and 400 milli seconds
S5T#5S	5 seconds

The time base is selected with this entering format as small as possible automatically.

The maximum timer is $999 * 10 \text{ s} = 2 \text{ h } 46 \text{ min } 30 \text{ sec}$.

Because the duration that is specified in units of the time base can only go up to 999 the less significant decimal places will disappear if the timer is specified too exactly:

S5T#2H500MS changes into S5T#2H.

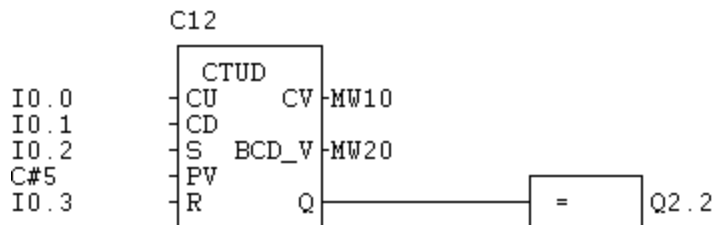
The T#...-format is not suitable for loading of the timer value in the correct format because so coded timer durations are loaded as double integer in ms into the accu!

[Timers in general](#)

3.7.3 Operations with counters

There are 'Counters' in the PLC 512 which can be used to count events. The counters are numbered from Z0 up to Z511, naturally you can name them sensibly by the [symbol table](#). Counters can be used as up-counter, down-counter and as up- and down-counter. The counting range goes from 0 up to 999.

Examples of a counter in FBD:



If there is a rising edge at the CU the count will be increased by 1. If there is a rising edge at the input ZR the count will be decreased by 1. If there is a rising edge at the input S the counter will be set to the value that is present at the [PV input](#). If '1' is the signal state at input R the counter will be set to '0' statically, that means not just with an edge. At the [output CV](#) the current count is written binary coded into any word variable. At the [output BCD_V](#) the count is written BCD-coded into any word variable. In this form it can be used again directly to serve another counter as PV-value. The output Q will be '1' if the counter is > 0. Negative counts or values > 999 do not exist. If there is an edge at the CD-input at the count 0 nothing will happen, this is the same as if at the state of 999 an edge will appear at the CU-input.

[ZCU](#) Count up

[ZD](#) Count down

[S](#) Set the counter to the value that is specified at PV

[PV](#) To this preset value the counter is set at a rising edge at S

[R](#) Set the counter to 0

[FR \(not implemented\)](#)

[List of operations](#)

3.7.3.1 Input PV of a counter

To this value the counter will be set if a rising edge is detected at the S-input. This input will not have to be connected if you do not need this function. But the pure down-counter is quite useless with an unconnected S- or ZW-input. The value that is fed into this input has to be [BCD](#) coded. The easiest way to do this is to write it as C# constant. Naturally, here can also be any word variable. The counter can only be set to values from 0 up to 999.

[Counters in general](#)

3.7.3.2 Output CV of a counter

At this output the current count is binary coded written into any word-variable. If you do not need this value you will be able to leave this output unconnected. The range of values of the here specified number goes from 0 up to 999. You cannot use this value as PV-value of another counter because this one has to be BCD-coded, for this use the displayed value at the [BCD V - ut](#).

[Counters in General](#)

3.7.3.3 Output BCD_V of the timer

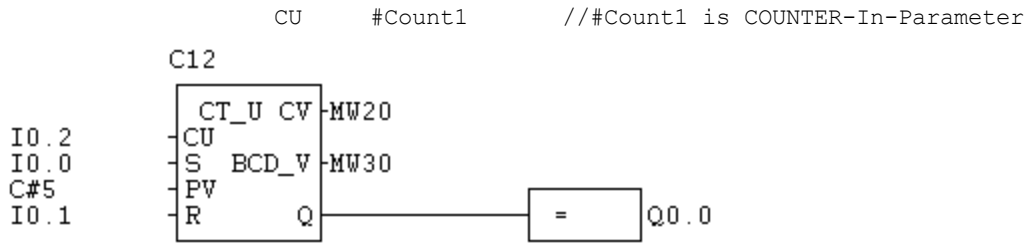
At this output the current count is written [BCD-coded](#) into any word-variable. If you do not need this value you can also leave the output unconnected. The range of values of the here displayed number goes from 0 up to 999. You can use this value as PV value of another counter. But you cannot process this value with the +I, -I, *I and /I operations. If you want to calculate with the count you shall use the binary coded value specified at the [DU-output](#). If you want to compare the value that is displayed at the DE-output with other numbers you will have to note that these are BCD-coded as well.

[Counters in General](#)

3.7.3.4 CU

At a rising slope of the [RLO](#) when executing this instruction the counter value is increased by one. If the counter value is already 999 it will not change.

```
Syntax:          CU      C 38
                  CU      C [#Widx]    // #Widx is Word with the number of the
counter
```



See also:

[Counter in general](#)

Set Set the counter to the preset value PV
[counte](#)

r

PV The counter is set to this value at a rising slope at S
Reset If this input is “1” the counter value will be set to ‘0’

[counte](#)

r

CU The same as CD, but up

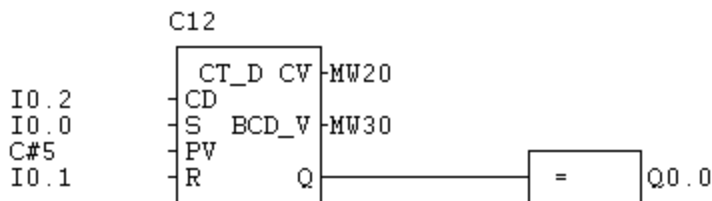
[FR \(not implemented\)](#)

[List of operations](#)

3.7.3.5 CD

At a rising slope of the [RLO](#) when executing this instruction the counter value is decremented by one. If the counter value is already zero it will not be changed.

Syntax: CD C 38
CD C [#Widx] // #Widx is Word with the number of the counter
CD #Count1 // #Count1 is COUNTER-In-Parameter



See also:

[Counter in general](#)

Set Set the counter to the preset value PV
[counte](#)

r

[PV](#) The counter is set to this value at a rising slope at S

[Reset](#) If this input is “1” the counter value will be set to ‘0’

[counte](#)

[r](#)

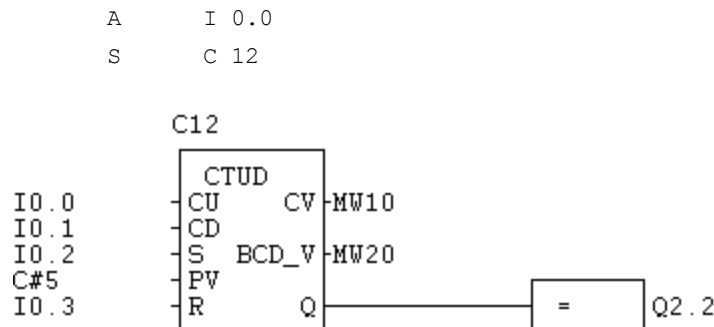
[CU](#) The same as CD, but up

[FR \(not implemented\)](#)

[List of operations](#)

3.7.3.6 Set counter

If the S-input of a counter goes from 0 to 1 the counter will be set to the value that is specified at the preset value (PV)-input. The number at the [PV-input](#) has to be BCD-coded, what it will become automatically if it is specified as C#.. constant.



See also:

[Counter in general](#)

[CW](#) To this value the counter is set with a rising slope at S

[Reset](#) If this input has the signal 1 the counter value is set to ‘0’

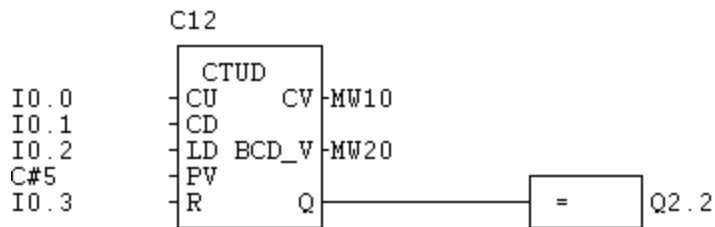
[counter](#)

[CU](#) Just like CR, but forward

[List of operations](#)

3.7.3.7 Reset counter

The specified counter is set back to ‘0’. Evaluations of the boolean state of the counter give ‘wrong’ as long as the counter is counted up again.



See also:

[Counter in general](#)

[CU](#) Count up(wards)

[CD](#) Count down(wards)

[S](#) Set the counter to the value PV

[PV](#) To this value the counter is set with a rising slope on LD ??

[List of operations](#)

3.7.3.8 FR

The functions release of times and counters are not implemented in TrySim.

See also:

[SD](#)

[SF](#)

[SP](#)

[SE](#)

[SS](#)

[CU](#)

[CD](#)

[List of operations](#)

3.7.4 Load and transfer operations

3.7.4.1 L

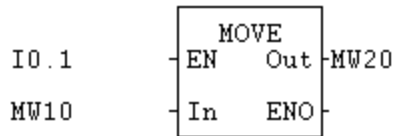
The specified operand is stored in accu 1. If the operand is not 4 bytes big the more significant bytes of accu 1 will be stored with '0'. Attention, this operation will be executed in any case, even if a former bit-connection results '0'. How many bytes are stored depends on the size of the operand.

```

L      DBB 2      // The content of DBB2 is copied to accu 1LL
L      MW 5       // The content of MW5 is copied to accu 1L
  
```

```
L      QD 10      // The content of QD10 is copied to accu 1
```

In FBD and LAD this instruction is only available with an destination specification:



The meaning is:

If input I0.1 is set you will have to load the content of MW10 into accu 1L and transfer it into the MW20 then.

See also:

[EN-input](#)

[LC](#)

[T](#)

[Constants which can be loaded](#)

[List of operations](#)

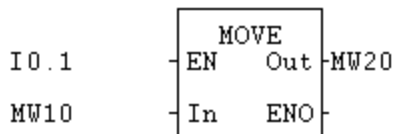
3.7.4.2 T

Transfer

The number in accu 1 is copied to the specified operand. Attention, this operation is executed in any case, unimportant which value the [RLO](#) has got. How many bytes are copied is dependent on the operand.

```
T      DBB 2      // The content of accu 1LL is copied to DBB 2
T      MW 5       // The content of accu 1L is copied to MW 5
T      AD 10      // The content of accu 1 is copied to AD 10
```

In FBD and LAD this instruction is only available with a source value.



The meaning is:

If input I0.1 is set MW10 is loaded into accu 1-L and this value is then transferred into MW20.

See also:

[L](#)

[EN-input](#)

[List of operations](#)

3.7.4.3 LC

Load current timer or counter as [BCD](#) into accu 1.

```
LC    T 5
LC    Z 7
```

See also:

[L](#)
[Operations with timers](#)

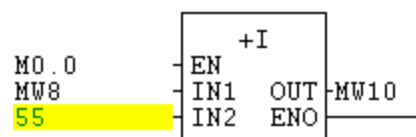
[List of operations](#)

3.7.5 Operations with integers

3.7.5.1 +I

Accu 1-L and accu 2-L are added as integers and the result is stored in accu 1. Accu 2 stays unchained. If the result is bigger than 32767 the OV-bit will be set.

```
L      2000
L      MW 12
+I
T      MW 14    // in MW 14 is now displayed 2000 + MW 12
```



See also:

[EN-input](#) and [ENO-output](#)

[+D](#)
[+R](#)
[-I](#)
[*I](#)

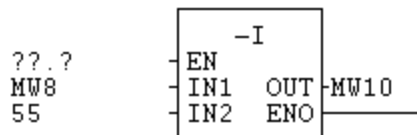
[/I](#)[List of operations](#)**3.7.5.2 -I**

Accu 1 is subtracted of accu 2 and the result is stored in accu 1. Accu 2 stays unchanged. If the result is smaller than -32768 the OV-bit will be set.

```

L      2000
L      MW 12
-I
T      MW 14    // in MW 14 is now displayed 2000 - MW 12

```



See also:

[EN-input](#) and [ENO-output](#)

[-D](#)

[-R](#)

[+I](#)

[*I](#)

[/I](#)

[List of operations](#)**3.7.5.3 *I**

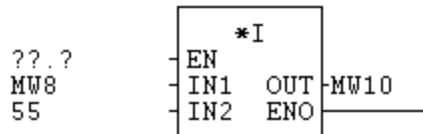
Accu 1-L and accu 2-L are multiplied and the result is stored in accu 1. If the result is bigger than 32767 or smaller than -32767 the OV-bit will be set.

```

L      2000
L      MW 12
*I

```

```
T    MW 14    // in MW 14 is now displayed 2000 * MW 12
```



See also:

[EN-input](#) and [ENO-output](#)

[*D](#)

[*R](#)

[+I](#)

[-I](#)

[/I](#)

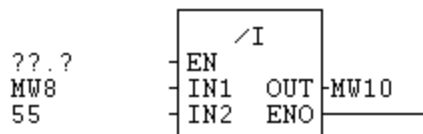
[List of operations](#)

3.7.5.4 //

Divide accu 2 by accu 1 as integer.

Accu 2 is divided by accu 1 and the result is stored in accu 1. Accu 2 stays unchanged. The remainder of the division is stored in accu 1-L. That is why the the accu 1 appears often as gigantic big number in the [monitoring mode](#) after dividing. As long as the following transfer operation relates only to words the result of the division is still correct. The remainder of the division kann be shifted with the operation SRD 16 (shift to the left side about 16 bits) into the word area of accu 1.

```
L      2000
L      MW 12
//I
T      MW 14 // in MW 14 is now displayed 2000 / MW 12
SRD 16
T      MW 16 // in MW 16 is now displayed the remainder of the division
```



See also:

[EN-input](#) and [ENO-output](#)

[/D](#)

[/R](#)
[+I](#)
[-I](#)
[*I](#)
[SRD](#)

[List of operations](#)

3.7.5.5 MOD

Accu 1 is loaded with the remainder of the double-integer-division of accu 2 by accu 1. Accu 2 stays unchanged.

With the integer-division this operation is executed automatically and the result is stored in both of the more significant bytes of accu 1.

See also:

[+D](#)
[-D](#)
[*D](#)
[/D](#)

[List of operations](#)

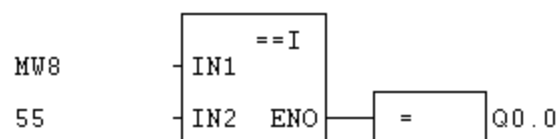
3.7.5.6 ==I

The 16 less significant bits of accu 1 and accu 2 are compared and if they are the same the [RLO](#) will be set to 1, otherwise to 0.

```

L      2000
L      MW 12
==I
=      Q 77.6 // if 2000 equals MW 12
                Q77.6 will be set to 1,
                otherwise to 0

```



See also:

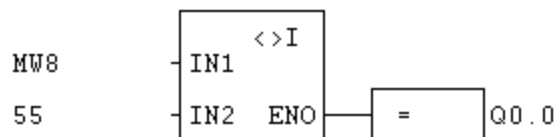
[L](#)
[<>I](#)

[List of operations](#)

3.7.5.7 <>I

Is integer in accu 2 <> accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```
L      2000
L      MW 12
<>I
=      Q 77.6 // if 2000 does not equal MW 12
           Q77.6 will be set to 1
           otherwise to 0
```



See also:

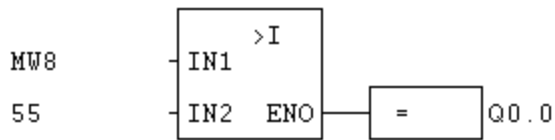
[L](#)
[<I](#)
[>I](#)
[<>D](#)
[<>R](#)

[List of operations](#)

3.7.5.8 >I

Is integer in accu 2 > accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```
L      2000
L      MW 12
>I
=      Q 77.6 // if 2000 is bigger than MW 12
           Q77.6 will be set to 1,
           otherwise to 0
```



See also:

[L](#)

[<I](#)

[>D](#)

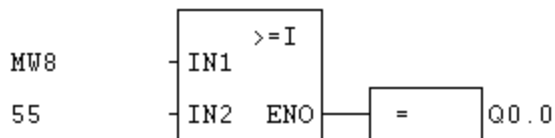
[>R](#)

[List of operations](#)

3.7.5.9 >=I

Is integer in accu 2 >= accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```
L      2000
L      MW 12
>=I
=      Q 77.6 // if 2000 is bigger or equals MW 12
           Q77.6 will be set to 1,
           otherwise to 0
```



See also:

[L](#)

[>I](#)

[>=D](#)

[>=R](#)

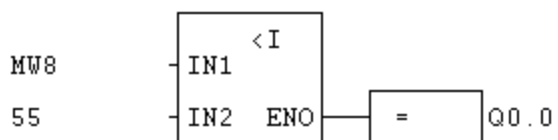
[List of operations](#)

3.7.5.10 <I

Is integer in accu 2 < accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```

L      2000
L      MW 12
<I
=      Q 77.6 // if 2000 is smaller than MW 12
           Q77.6 will be set to 1,
           otherwise to 0
    
```



See also:

- [L](#)
- [>I](#)
- [<D](#)
- [<R](#)

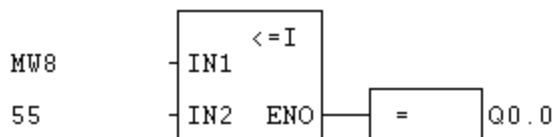
[List of operations](#)

3.7.5.11 <=I

Is integer in accu 2 <= accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```

L      2000
L      MW 12
<=I
=      Q 77.6 // if 2000 is smaller than or equals MW 12
           Q77.6 will be set to 1,
           otherwise to 0
    
```



See also:

[L](#)
[<I](#)
[<=D](#)
[<=R](#)

[List of operations](#)

3.7.5.12 NEGI

The integer-number in accu 1 is multiplied with -1.

See also:

[INVI](#)
[INVD](#)
[NEGD](#)

[List of operations](#)

3.7.5.13 + (Plus)

Syntax : + 5

The constant that is specified as operand is added to accu 1. The other accus and the displays will not be influenced.

See also:

[+I](#)
[+D](#)
[+R](#)

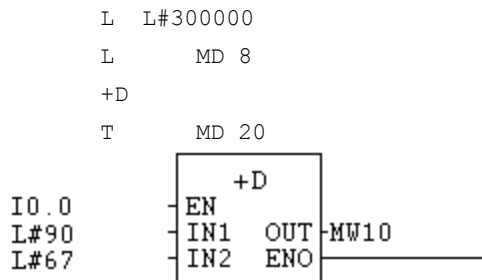
[List of operations](#)

3.7.6 Operations with double integers

3.7.6.1 +D

Accu 1 and accu 2 are added and the result is stored in accu 1. As long as the values in accu 1 and accu 2 are positive it is indifferent whether they are loaded out of a

word or a dword. But if one of the as word loaded operands can be negative '+D' will lead to absurd results. In this case the eventually negative operand has to be extended to the dword format with the operation [ITD](#) before adding.



See also:

[EN-input](#) and [ENO-output](#)

[+I](#)

[+R](#)

[-D](#)

[*D](#)

[/D](#)

[MOD](#)

[NEGI](#)

[NEGD](#)

[Load](#) and [Transfer](#)

[List of operations](#)

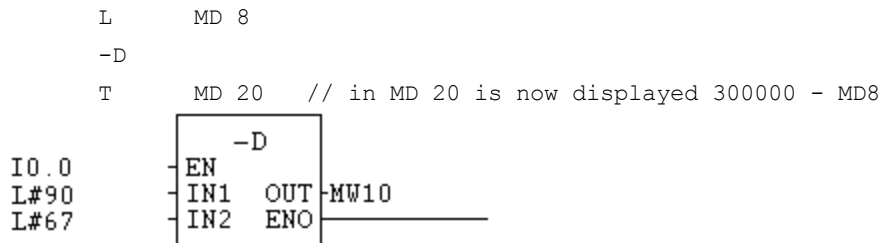
3.7.6.2 -D

Accu 1 is subtracted of accu 2 and the result is stored in accu 1. As long as the values in accu 1 and accu 2 are positive it is indifferent whether they are loaded out of a word or a dword. But if one of the as word loaded operands can be negative '-D' will lead to an absurd result. In this case the eventually negative operand has to be extended to the dword format with the operation [ITD](#) before subtracting.

```

L   L#300000

```



See also:

[EN-input](#) and [ENO-output](#)

[-I](#)

[-R](#)

[+D](#)

[*D](#)

[/D](#)

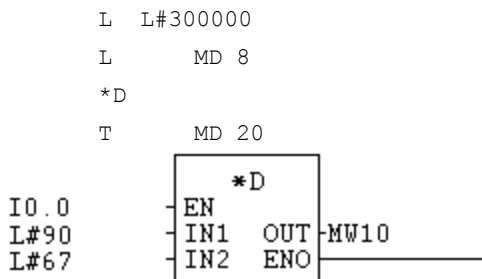
[MOD](#)

[List of operations](#)

3.7.6.3 *D

Multiply accu 1 with accu 2 as integer (32 bit).

Accu 1 is multiplied with accu 2 and the result is stored in accu 1. As long as the values in accu 1 and accu 2 are positive it is indifferent whether they are loaded out of a word or a dword. But if one of the as word loaded operands can be negative ‘*D’ will lead to an absurd result. In this case the eventually negative operand has to be extended to the Dword-format with the operation [ITD](#) before multiplying.



See also:

[EN-input](#) and [ENO-output](#)

[*I](#)

[*R](#)

[+D](#)

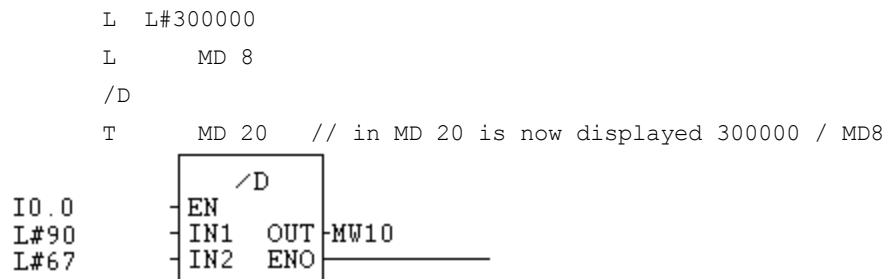
[-D](#)
[/D](#)
[MOD](#)

[List of operations](#)

3.7.6.4 /D

Accu 2 is divided by accu 1 and the result is stored in accu 1. The remainder gets lost. As long as the values in accu 1 and accu 2 are positive it is indifferent whether they are loaded out of a word or a dword. But if one of the as word loaded operands can be negative '/D' will lead to an absurd result. In this case the eventually negative operand has to be extended to the Dword-format with the operation [ITD](#) before dividing.

The remainder of the division can be kept with the operation MOD.



See also:

[EN-input](#) and [ENO-output](#)

[/I](#)
[/R](#)
[+D](#)
[-D](#)
[*D](#)
[MOD](#)

[List of operations](#)

3.7.6.5 MOD

Accu 1 is loaded with the remainder of the double-integer-division of accu 2 by accu 1. Accu 2 stays unchanged.

With the integer-division this operation is executed automatically and the result is stored in both of the more significant bytes of accu 1.

See also:

[+D](#)

[-D](#)

[*D](#)

[/D](#)

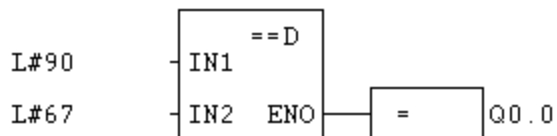
[List of operations](#)

3.7.6.6 ==D

Accu 1 and accu 2 are compared as DInts. If they are the same the [RLO](#) will be set to '1', otherwise to '0'.

As long as the values in accu 1 and accu 2 are positive it is indifferent whether they are loaded out of a word or a dword. But if one of the as word loaded operands can be negative '==D' will lead to an absurd result. In this case the eventually negative operand has to be extended to the Dword-format with the operation [ITD](#) before comparing.

```
L L#300000
L MD 8
==D
= Q 0.0 // if 300000 equals MD 8,
          Q 0.0 will be set to 1,
          otherwise to 0
```



See also:

[<>D](#)

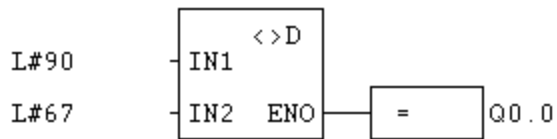
[List of operations](#)

3.7.6.7 <>D

Is double integer in accu 2 <> accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```

L L#300000
L MD 8
<>D
= Q 0.0 // if 300000 does not equal MD 8
           Q 0.0 is set to 1,
           otherwise to 0
    
```



See also:

- [<D](#)
- [>D](#)
- [<I](#)
- [<R](#)

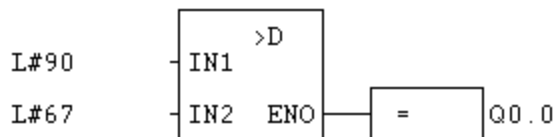
[List of operations](#)

3.7.6.8 >D

Is double integer in accu 2 > accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```

L L#300000
L MD 8
>D
= Q 0.0 // if 300000 is bigger than MD 8
           Q 0.0 will be set to 1,
           otherwise to 0
    
```



See also:

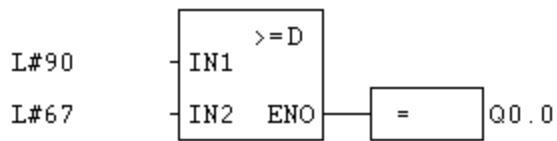
- [<D](#)
- [>I](#)
- [>R](#)

[List of operations](#)

3.7.6.9 >=D

Is double integer in accu 2 >= accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```
L L#300000
L MD 8
>=D
= Q 0.0 // if 300000 is bigger than or equals MD 8
           Q 0.0 will be set to 1,
           otherwise to 0
```



See also:

[>D](#)

[>=I](#)

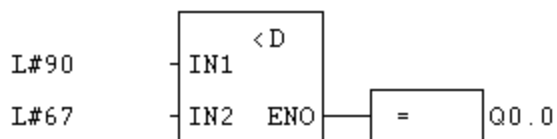
[>=R](#)

[List of operations](#)

3.7.6.10 <D

Is double integer in accu 2 < accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```
L L#300000
L MD 8
<D
= Q 0.0 // if 300000 is smaller than MD 8
           Q 0.0 is set to 1,
           otherwise to 0
```



See also:

[>D](#)

[<=D](#)

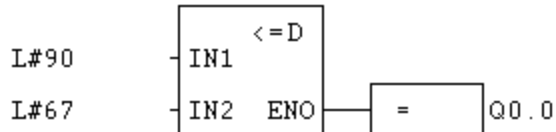
[<I](#)

[<R](#)

[List of operations](#)**3.7.6.11 <=D**

Is double integer in accu 2 <= accu 1? If it is so you will have to set [RLO](#) to 1, otherwise to 0.

```
L L#300000
L MD 8
<D
= Q 0.0 // if 300000 is smaller or equals MD 8
           Q 0.0 is set to 1,
           otherwise to 0
```



See also:

[<D](#)

[<=I](#)

[<=R](#)

[List of operations](#)**3.7.6.12 NEGD**

The double-integer number in accu 1 is multiplied with -1.

See also:

[INVI](#)

[INVD](#)

[NEGI](#)

[List of operations](#)**3.7.6.13 + (Plus)**

Syntax : + 5

The constant that is specified as operand is added to accu 1. The other accus and the displays will not be influenced.

See also:

[+I](#)

[+D](#)

[+R](#)

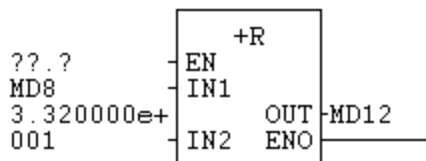
[List of operations](#)

3.7.7 Operations with reals

3.7.7.1 +R

The numbers in accu 1 and accu 2 are multiplied as REAL-numbers and the result is stored in accu 1. It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before adding. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```
L    MD 40
L    2.0          // The point is important!
+R
T    MD 44        // in MD 44 is now displayed MD40 + 2.0
```



See also:

[EN-input](#) and [ENO-output](#)

[+D](#)

[+I](#)

[-R](#)

[*R](#)

[/R](#)

[NEGR](#)

[ABS](#)

[Trigonometrical operations](#)

[List of operations](#)

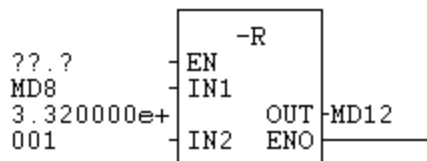
3.7.7.2 -R

Accu 1 is subtracted of accu 2 and the result is stored in accu 1. It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before subtracting. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```

L    MD 40
L    2.0          // The point is important!
-R
T    MD 44        // in MD 44 is now displayed MD40 - 2.0

```



See also:

[EN-input](#) and [ENO-output](#)

[-D](#)

[-I](#)

[+R](#)

[*R](#)

[/R](#)

[NEGR](#)

[ABS](#)

[Trigonometrical operations](#)

[List of operations](#)

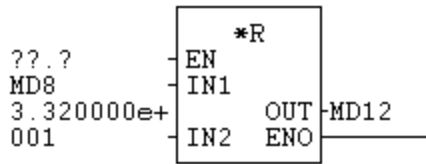
3.7.7.3 *R

Accu 1 and accu 2 are multiplied and the result is stored in accu 1. It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before multiplying. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```

L    MD 40
L    2.0          // The point is important!
-R
T    MD 44        // in MD 44 is now displayed MD40 - 2.0

```



See also:

[EN-input](#) and [ENO-output](#)

[*D](#)

[*I](#)

[+R](#)

[-R](#)

[/R](#)

[NEGR](#)

[ABS](#)

[Trigonometrical operations](#)

[List of operations](#)

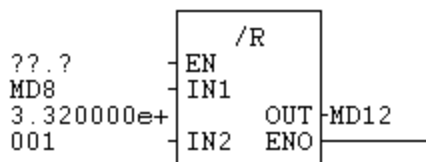
3.7.7.4 /R

Accu 2 is divided by accu 1 and the result is stored in accu 1. Accu 2 stays unchanged. It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before dividing. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```

L    MD 40
L    2.0           // The point is important!
/R
T    MD 44         // in MD 44 is now displayed MD40 / 2

```



See also:

[EN-input](#) and [ENO-output](#)

[/D](#)

[/I](#)

[+R](#)
[-R](#)
[*R](#)
[NEGR](#)
[ABS](#)
[Trigonometrical operations](#)

[List of operations](#)

3.7.7.5 ==R

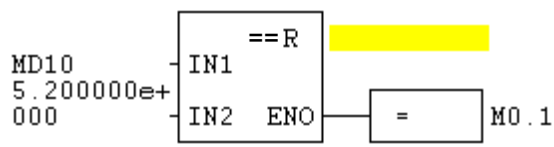
Accu 1 and accu 2 are compared and the [RLO](#) will be set to 1 if they are the same, otherwise to 0. It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before comparing. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

The comparison of REAL-numbers to 'equal' leads quite often to maloperations of the program because of rounding errors. Instead of '==R' you should better subtract the numbers that are to be compared and compare the absolute value of this with '<R' to an upper limit.

```

L      MD 40
L      2.0           // The point is important!
==R
Q      Q 22.4       // If MD40 equals 2.0 Q22.4 will be set to 1,
                   otherwise to 0

```



See also:

[-R](#)
[ABS](#)
[<=R](#)
[>=R](#)

[List of operations](#)

3.7.7.6 <>R

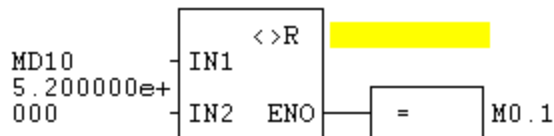
Accu 1 and accu 2 are compared and the [RLO](#) will be set to '1' if they do not equal, otherwise to '0'. It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before comparing. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

The comparison of REAL-numbers to 'not equal' leads quite often to maloperations of the program because of rounding errors. Instead of '<>R' you should better subtract the numbers that are to be compared and compare the absolute value of this with '>R' to an upper limit.

```

L     MD 40
L     2.0           // The point is important!
<>R
Q     Q 22.4       // If MD40 does not equal 2.0 Q22.4 will be set to 1,
                   otherwise to 0

```



See also:

[-R](#)

[ABS](#)

[<R](#)

[>R](#)

[<>D](#)

[<>I](#)

[List of operations](#)

3.7.7.7 >R

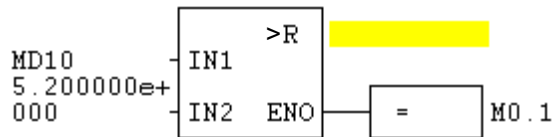
Accu 1 and accu 2 are compared and the [RLO](#) will be set to '1' if accu 2 > accu 1, otherwise to '0'.

It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before comparing. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```

L    MD 40
L    2.0           // The point is important!
>R
Q    Q 22.4       // If MD40 is bigger than 2.0 Q22.4 will be set to 1
                    otherwise to 0

```



See also:

[-R](#)
[ABS](#)
[<R](#)
[>D](#)
[>I](#)

[List of operations](#)

3.7.7.8 >=R

Accu 1 and accu 2 are compared and the [RLO](#) will be set to 1 if accu 2 >= accu 1, otherwise to '0'.

It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before comparing. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```

L    MD 40
L    2.0           // The point is important!
>=R
Q    Q 22.4       // If MD40 is bigger or equals 2.0
                    Q22.4 will be set to 1 gesetzt,
                    otherwise to 0

```



See also:

[-R](#)

[ABS](#)[<R](#)[>=D](#)[>=I](#)[List of operations](#)

3.7.7.9 <R

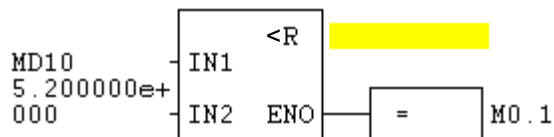
Accu 1 and accu 2 are compared and the [RLO](#) will be set to '1' if accu 2 < accu 1, otherwise to '0'.

It is important that both operands are already in the REAL-format. If one of the operands is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before comparing. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```

L    MD 40
L    2.0           // The point is important!
<R
Q    Q 22.4       // If MD40 is smaller than 2.0
                    Q22.4 will be set to 1
                    otherwise to 0

```



See also:

[-R](#)[ABS](#)[<D](#)[<I](#)[List of operations](#)

3.7.7.10 <=R

Accu 1 and accu 2 are compared and the [RLO](#) will be set to '1' if accu 2 <= accu 1, otherwise to '0'.

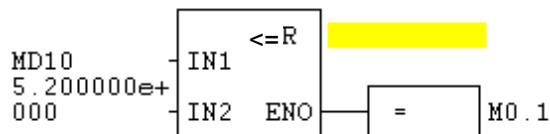
It is important that both operands are already in the REAL-format. If one of the operands is loaded

as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before comparing. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```

L    MD 40
L    2.0          // The point is important!
<=R
Q    Q 22.4      // If MD40 is smaller than or equals 2.0
                    Q22.4 will be set to 1,
                    otherwise to 0

```



See also:

[-R](#)

[ABS](#)

[<R](#)

[<=D](#)

[<=I](#)

[List of operations](#)

3.7.7.11 ABS

If accu 1 is positive the operation will achieve nothing, otherwise the value of accu 1 will be negated and stored in accu 1. After ABS accu 1 is positive (or '0') in any case.

You can only use ABS for numbers that are in the REAL-format. If you need the absolute value of Int- (DInt-) numbers you will program:

```

L    0
L    number
<I    (D)
JC    Ok
NEGI    (NEGD)

```

Ok :

See also:

[NEGR](#)

[+R](#)

[-R](#)

[*R](#)

[/R](#)

[List of operations](#)

3.7.7.12 NEGR

The REAL-number in accu 1 is multiplied with -1.

See also:

[+R](#)

[-R](#)

[*R](#)

[/R](#)

[ABS](#)

[List of operations](#)

3.7.7.13 SQRT

Of accu 1 the square root is calculated and stored in accu 1 again.

It is important that the operand in accu 1 is already in the REAL-format. If it is loaded as Int or DInt it will have to be converted in a REAL-number with [DTR](#) before the calculation.

See also:

[SQR](#)

[List of operations](#)

3.7.7.14 SQR

The square of the number in accu 1 is calculated and stored in accu 1.

It is important that the operand in accu 1 is already in the REAL-format. If it is loaded in Int or DInt it will have to be converted into a REAL-number before the calculation with [DTR](#). If an operand that is loaded as Int can be negative it will have to be extended with [ITD](#) to the Dword-format before as well.

See also:

[SQRT](#)

[List of operations](#)

3.7.7.15 LN

Logarithm to the base e of accu 1.

In accu 1 there has to be a REAL-number.

See also:

[EXP](#)

[List of operations](#)

3.7.7.16 EXP

e to the accu 1

Accu 1 has to be REAL-number.

See also:

[LN](#)

[List of operations](#)

3.7.8 Jump operations

3.7.8.1 Jump labels

Jump labels are used in STL-networks to mark lines to which it shall be jumped. Jump labels consist of max. 4 letters and numbers and must not contain special characters except of ‘_’. The capitalization is significant. Jump labels are closed with an ‘:’, after that an instruction has to follow,

at least [NOP 0](#).

Within a **block** a jump label can only appear once.

Unconditional jump:

[JU](#)

Conditional jumps:

[JNC](#) [JMZ](#) [JNJP](#)

[JPZ](#) [JZ](#) [JNB](#)

[JC](#) [JCB](#) [JBI](#) [JNBI](#)

Loops and jump distributor('Case'):

[LOOPJL](#)

Not implemented jumps:

[JO](#) [JUO](#) [JOS](#)

3.7.8.2 JU

Jump Unconditioned

```
JU    Lab1    // Lab1 is a jump label anywhere in the block
```

The program execution is continued at the specified position.

See also:

[JC](#) **Jump Conditioned** (if RLO = 1)

[JCN](#) **Jump Conditioned** (if RLO is **not** = 1)

[JCB](#)

[JNB](#)

[JBI](#)

[JNBI](#)

[JO](#)

[JOS](#)

[JUO](#)

[JZ](#)

[JP](#)

[JM](#)

[JN](#)

[JMZ](#)

[JPZ](#)

[JL](#)

[LOOP](#)

[List of operations](#)

3.7.8.3 JCN

Jump Conditioned (if RLO is not = 1)

```
JCN Lab1 // Lab1 is a jump label anywhere in the block
```

If the [RLO](#) = 0 it will be jumped to the specified position, otherwise the execution will be continued normally.

See also:

[JU](#) **Jump Unconditioned (without condition)**

[JC](#) **Jump Conditioned (if RLO = 1)**

[JCB](#)

[JNB](#)

[JBI](#)

[JNBI](#)

[JO](#)

[JOS](#)

[JUO](#)

[JZ](#)

[JP](#)

[JM](#)

[JN](#)

[JMZ](#)

[JPZ](#)

[JL](#)

[LOOP](#)

[List of operations](#)

3.7.8.4 JMZ

```
JCMZ Lab1 // Lab1 is a jump label anywhere in the block
```

If the result of a precede **Int- or DInt - computer operation** ≤ 0 it will be jumped to the

specified [jump label](#).

This jump does not function after REAL-operations! But see also [ABS](#).

See also:

[JZ](#) **J**ump, if (**z**ero)
 result = 0

[JN](#) **J**ump, if (**n**ot zero)
 result <> 0

[JP](#) **J**ump, if (**p**ositive)
 result > 0

[JPZ](#) **J**ump, if (**p**ositive or **z**ero)
 result >= 0

[JM](#) **J**ump, if (**m**inus)
 result < 0

[JU](#)
[JC](#)
[JCN](#)
[JCB](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.8.5 JM

```
JCM    Lab1    // Lab1 is a jump label anywhere in the block
```

If the result of a precede **Int- or DInt - computer operation** is < 0 it will be jumped to the specified [jump label](#).

This jump does not function after REAL-operations! But see also [ABS](#).

See also:

[JZ](#) **J**ump, if (**z**ero)
 result = 0

[JN](#) **J**ump, if (not zero)
 result \neq 0

[JP](#) **J**ump, if (positive)
 result > 0

[JMZ](#) **J**ump, if (minus or zero)
 result \leq 0

[JPZ](#) **J**ump, if (positive or zero)
 result \geq 0

[JU](#)
[JC](#)
[JCN](#)
[JCB](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.8.6 JN

```
JN     Lab1    // Lab1 is a jump label anywhere in the block
```

If the result of a precede **Int- or DInt - computer operation** \neq 0 it will be jumped to the specified [jump label](#).

This jump does not function after REAL-operations! But see also [ABS](#).

See also:

[JZ](#) **J**ump, if (zero)
 result = 0

[JM](#) **J**ump, if ((minus)
 result < 0

[JP](#) **J**ump, if (positive)
 result > 0

[JPZ](#) **J**ump, if (positive or zero)
 result \geq 0

[JMZ](#) **J**ump, if (minus or zero)

result <= 0

[JU](#)
[JC](#)
[JCN](#)
[JCB](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.8.7 JP

```
JP    Lab1    // Lab1 is a jump label anywhere in the block
```

If the result of a precede **Int- or DInt - computer operation** > 0 it will be jumped to the specified [jump label](#).

This jump does not function after REAL-operations! But see also [ABS](#).

See also:

[JZ](#) **J**ump, if (zero)
 result = 0
[JM](#) **J**ump, if ((minus)
 result < 0
[JN](#) **J**ump, if (not zero)
 result <> 0
[JPZ](#) **J**ump, if (positive or zero)
 result >= 0
[JMZ](#) **J**ump, if (minus or zero)
 result <= 0

[JU](#)
[JC](#)
[JCN](#)
[JCB](#)
[JNB](#)

[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.8.8 JPZ

```
JP    Lab1 // Lab1 is a jump label anywhere in the block
```

If the result of the precede **Int- or - DInt computer operation** ≥ 0 it will be jumped to the specified [jump label](#).

This jump does not function after REAL-operations! But see also [ABS](#).

See also:

[JZ](#) **J**ump, if (**z**ero)
 result = 0
[JM](#) **J**ump, if (**m**inus)
 result < 0
[JN](#) **J**ump, if (**n**ot zero)
 result <> 0
[JP](#) **J**ump, if (**p**ositive)
 result > 0
[JMZ](#) **J**ump, if (**m**inus or **z**ero)
 result <= 0

[JU](#)
[JC](#)
[JCN](#)
[JCB](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JL](#)

[LOOP](#)

[List of operations](#)

3.7.8.9 JZ

```
JZ    Lab1    // Lab1 is a jump label anywhere in the block
```

If the result of a precede **Int- or - DInt computer operation** = 0 it will be jumped to the specified [jump label](#).

This jump does not function after REAL-operations! But see also [ABS](#).

See also:

[JPZ](#) Jump, if (positive or zero)
result >= 0

[JM](#) Jump, if ((minus)
result < 0

[JN](#) Jump, if (not zero)
result <> 0

[JP](#) Jump, if (positive)
result > 0

[JMZ](#) Jump, if (minus or zero)
result <= 0

[JU](#)

[JC](#)

[JCN](#)

[JCB](#)

[JNB](#)

[JBI](#)

[JNBI](#)

[JO](#)

[JOS](#)

[JUO](#)

[JL](#)

[LOOP](#)

[List of operations](#)

3.7.8.10 JNB

```
JNB Lab1 // Lab1 is a jump label anywhere in the block
```

If the [RLO](#) = 0 it will be jumped to the specified position, otherwise the execution will be continued normally.

In addition the [BR-Bit](#) is set to the value of the RLO.

See also:

[JU](#)

[JC](#)

[JCN](#)

[JCB](#)

[JBI](#)

[JNBI](#)

[JO](#)

[JOS](#)

[JUO](#)

[JZ](#)

[JP](#)

[JM](#)

[JN](#)

[JMZ](#)

[JPZ](#)

[JL](#)

[LOOP](#)

[List of operations](#)

3.7.8.11 JC

Jump Conditioned

```
JC Lab1 // Lab1 is a jump label anywhere in the block
```

If RLO = '0' it will be jumped to the specified position, otherwise the execution will be continued normally.

See also:

[JU](#) Jump Unconditioned (without any condition)

[JCN](#) Jump Conditioned (if the RLO is not = 1)

[JCB](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JZ](#)
[JP](#)
[JM](#)
[JN](#)
[JMZ](#)
[JPZ](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.8.12 JCB

```
JCB Labl // Labl is a jump label anywhere in the block
```

If the [RLO](#) = '0' it will be jumped to the specified position, otherwise the execution will be continued normally.

In addition the [BR-Bit](#) is set to the value of the RLO.

See also:

[JU](#)
[JC](#)
[JCN](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JZ](#)
[JP](#)
[JM](#)
[JN](#)
[JMZ](#)

[JPZ](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.8.13 JBI

```
JBI    Lab1    // Lab1 is a jump label anywhere in the block
```

If the BR-Bit is '1' it will be jumped to the specified position.

See also:

[JNBI](#)
[JCB](#)
[JNB](#)

As well as:

[JU](#)
[JC](#)
[JCN](#)
[JO](#)
[JOS](#)
[JUO](#)
[JZ](#)
[JP](#)
[JM](#)
[JN](#)
[JMZ](#)
[JPZ](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.8.14 JNBI

```
JNBI   Lab1   // Lab1 is a jump label anywhere in the block
```

If the BR-Bit is '0' it will be jumped to the specified position.

See also:

[JBI](#)

[JCB](#)

[JNB](#)

As well as:

[JU](#)

[JC](#)

[JCN](#)

[JO](#)

[JOS](#)

[JUO](#)

[JZ](#)

[JP](#)

[JM](#)

[JN](#)

[JMZ](#)

[JPZ](#)

[JL](#)

[LOOP](#)

[List of operations](#)

3.7.8.15 Loop

The number in accu 1 is reduced by 1. If accu 1 is unequal zero it will be jumped to the specified [jump label](#).

This loop is done 10 times:

```
        L 10
lbl :   T MW 20
        ....
        ....
        L MW 20
        LOOP lbl           // lbl is jump label
```

See also:

[JU](#)

[JC](#)

[JCN](#)

[JCB](#)

[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JZ](#)
[JP](#)
[JM](#)
[JN](#)
[JMZ](#)
[JPZ](#)
[JL](#)

[List of operations](#)

3.7.8.16 JL

Jump List

```
Syntax:      JL      err      // err is jump label
              JU      L1      // L1 is jump label
              JU      L2      // L2 is jump label
              ..
              JU      Lx      // Lx is jump label
err:         .....
```

The number of program lines (which have to start with JU) that is to be found in accu 1 is skipped. If the list of JU-lines is not long enough it will be jumped to the label err instead.

See also:

[JU](#)
[JC](#)
[JCN](#)
[JCB](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JUO](#)
[JZ](#)
[JP](#)

[JM](#)

[JN](#)

[JMZ](#)

[JPZ](#)

[LOOP](#)

[List of operations](#)

3.7.9 Not implemented jumps

3.7.9.1 JO

Not implemented.

See also:

[JU](#)

[JC](#)

[JCN](#)

[JCB](#)

[JNB](#)

[JBI](#)

[JNBI](#)

[JOS](#)

[JUO](#)

[JZ](#)

[JP](#)

[JM](#)

[JN](#)

[JMZ](#)

[JPZ](#)

[JL](#)

[LOOP](#)

[List of operations](#)

3.7.9.2 JUO

Not implemented.

See also:

[JU](#)

[JC](#)
[JCN](#)
[JCB](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JOS](#)
[JZ](#)
[JP](#)
[JM](#)
[JN](#)
[JMZ](#)
[JPZ](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.9.3 JOS

Not implemented.

See also:

[JU](#)
[JC](#)
[JCN](#)
[JCB](#)
[JNB](#)
[JBI](#)
[JNBI](#)
[JO](#)
[JUO](#)
[JZ](#)
[JP](#)
[JM](#)
[JN](#)
[JMZ](#)
[JPZ](#)
[JL](#)
[LOOP](#)

[List of operations](#)

3.7.10 Shift and rotating operations

3.7.10.1 SRD

The 32 - bits of accu 1 are shifted right about the number that is specified as operand (0-32). Bits which are shifted out right get lost, from left it is refilled with zeros.

See also:

[SLW](#)

[SLD](#)

[SRW](#)

[SSI](#)

[SSD](#)

[List of operations](#)

3.7.10.2 SSD

The 32-bits of accu 1 are shifted right about the number that is specified as operand (0-32). From left it is refilled with the value of bit 31. With the help of this operation you can also divide negative numbers by right-shifting by powers of 2.

If no operand is specified accu 2-LL will be used instead.

See also:

[SLW](#)

[SLD](#)

[SRW](#)

[SRD](#)

[SSI](#)

[List of operations](#)

3.7.10.3 RLD

The 32 bits in accu 1 are rotated about the number (0-32) that is specified as operand to the left side, that means that doing a shifting the bits which are shifted out of the accu to the left side are

supplied on the other side again.

If no operand is specified the accu 2-LL will be used instead.

See also:

[RRD](#)

[RLDA](#)

[RRDA](#)

[List of operations](#)

3.7.10.4 RLDA

The 32 bits in accu 1 are rotated about one bit to the left side, that means that doing a shifting the bit which is shifted out of the accu to the left side is supplied on the other side again.

See also:

[RLD](#)

[RRD](#)

[RRDA](#)

[List of operations](#)

3.7.10.5 SLD

The 32 bits of accu 1 are shifted about the number that is specified as operand (0-32) to the left side. Bits that are shifted out to the left side get lost. From the right side zeros are filled in.

If no operand is specified the accu 2-LL will be used instead.

```
Syntax:      SLD x           // x is between 0 and 32
              SLD           // the content of accu 2-LL is used as parameter
```

See also:

[SLW](#)

[SRW](#)

[SRD](#)

[SSI](#)

[SSD](#)

[List of operations](#)

3.7.10.6 SLW

The 16 bits of accu 1-L are shifted about the number that is specified as operand (0-15) to the left side. The bits that are shifted to the left side beyond the accu 1-L-limit get lost, they are not transferred into the more significant bytes of the accu. Zeros are inserted from the right side.

```
Example:      L 1           // 1 in accu 1
              SLW 2        // Shift 2 bits to the left side
```

In accu 1 there is now W#16#0004 = 2#100

If no operand is specified the number in accu 2-LL is used instead.

```
Example:      L 4           // 5 in accu 1
              L 1           // 5 in accu 2 / 1 in accu 1
              SLW           // Shift 1 about 5 bits to the left side
```

In accu 1 there is now W#16#0010 = 2#1000

See also:

[SLD](#)

[SRW](#)

[SRD](#)

[SSI](#)

[SSD](#)

[List of operations](#)

3.7.10.7 RRDA

The 32 bits in accu 1 are rotated about one bit to the right side, that means that doing a shifting the bits which are shifted out of the accu to the right side are supplied on the other side again. Though with this operation a bit that is shifted out to the right side is not supplied immediately but is stored temporarily in a carry bit first.

See also

[RLD](#)

[RRD](#)

[RLDA](#)

[List of operations](#)

3.7.10.8 RRD

The 32 bits in accu 1 are rotated about the number (0-32) that is specified as operand to the right side, that means that doing a shifting the bits which are shifted out of the accu to the right side are supplied on the other side again.

See also:

[RLD](#)

[RLDA](#)

[RRDA](#)

[List of operations](#)

3.7.10.9 SSI

The 16-bits of accu 1-L are shifted right about the number that is specified as operand (0-15)- From left it is refilled with the value of bit 15. With the help of this operation you can also divide negative numbers by right-shifting by powers of 2.

If no operand is specified accu 2-LL will be used instead.

See also:

[SLW](#)

[SLD](#)

[SRW](#)

[SRD](#)

[SSD](#)

[List of operations](#)

3.7.10.10 SRW

The 16 - bits of accu 1-L are shifted right about the number that is specified as operand (0-15). Bits which are shifted out right get lost, from left it is refilled with zeros.

If no operand is specified the accu 2-LL will be used instead.

See also:

[SLW](#)

[SLD](#)[SRD](#)[SSI](#)[SSD](#)[List of operations](#)

3.7.11 Logical word and double word operations

3.7.11.1 AD

The 2 x 32 bits of accu 1 and 2 are AND-combined bit by bit and the result is stored in accu 1.

See also:

[AW](#)[OW](#)[XOW](#)[OD](#)[XOD](#)[List of operations](#)

3.7.11.2 AW

And Word

Accu 1-L and accu 2-L are combined with the operation And bit by bit and the result is stored in accu 1. The both more significant bytes of the accu are set to zero. Accu 2 stays unchanged.

See also:

[OW](#)[XOW](#)[AD](#)[OD](#)[XOD](#)[List of operations](#)

3.7.11.3 OW

The 16 less significant bits of accu 1 are set separately according to the following rule:
If one of the bits in accu 1 or accu 2 is '1' the bit will be set to '1', otherwise to '0'.

See also:

[AW](#)
[XOW](#)
[AD](#)
[OD](#)
[XOD](#)

[List of operations](#)

3.7.11.4 OD

The 32 bits of accu 1 are set separately according to the following rule:
If one of the bits in accu 1 or accu 2 is '1' the bit will be set to '1', otherwise to '0'.

See also:

[AW](#)
[OW](#)
[XOW](#)
[AD](#)
[XOD](#)

[List of operations](#)

3.7.11.5 XOD

EXCLUSIVE OR double word (32 bit)

Accu 1 and accu 2 are combined with the operation 'Exclusive Or' bit by bit and the result is stores in accu 1. Accu 2 stays unchanged.

See also:

[AW](#)
[OW](#)
[XOW](#)

[AD](#)
[OD](#)

[List of operations](#)

3.7.11.6 XOW

EXCLUSIVE OR word

Accu 1-L and accu 2-L are combined with the operation 'Exclusive Or' bit by bit and the result is stores in accu 1. The both more significant bytes of accu 1 are set to zero. Accu 2 stays unchanged.

See also:

[AW](#)
[OW](#)
[AD](#)
[OD](#)
[XOD](#)

[List of operations](#)

3.7.11.7 INVI

The 16 least significant bits of accu 1 are inverted, that means 1 becomes 0 and 0 becomes 1.

See also:

[INVD](#)
[NEGI](#)
[NEGD](#)

[List of operations](#)

3.7.11.8 INVD

The 32 bits in accu 1 are inverted, that means 1 becomes 0 and 0 becomes 1.

See also:

[INVI](#)

[NEGI](#)

[NEGD](#)

[List of operations](#)

3.7.12 BCD operations

3.7.12.1 BCD number

In the early times of computer technic, when the HMI (**human machine interface**) consisted of small lamps and handwheel-switches, one realized that people could hardly read the binary numbers of the machines and that it is difficult the other way round to draw a binary signal out of multidigit handwheel-switches. That is why the **binary coded decimal** number was invented, a chimera out of both representations. You just take each single digit of a number that is displayed in the decimal system and translate that one into a nibble (half byte) in binary representation. Nearly like this:

```

159   000 010 1001
      1   1   9
->    1   5

```

With a little practice man can learn the binary representation of the numbers from 0 up to 9, and also handwheel-switches can be constructed with limited effort so that they code this small counting range binary. Unfortunately, this inexpressible data format with which it is quite hard to calculate survived until now. The only comfort is that the waste of memory capacity which is combined with this does not matter anymore.

The easiest way to note a BCD-constant in the area of 0 - 999 is to use the counter format C#xyz, e.g. C#642.

A BCD-number will be interpreted as negative if all four bits of the most significant nibble equal 1. So you note -53(bcd) as W#16#F053, for example.

Seven-digit BCD-numbers can be noted by the use of the DW#16#-format.

Converting between BCD-numbers and binary format there are the following functions:

[BTI](#) 3 digit BCD-number with sign -> integer

[BTD](#) 7 digit BCD-number with sign -> double integer

[ITB](#) binary coded number from -999 up to +999 -> BCD

[DTB](#) binary coded number from -9.999.999 up to +9.999.999 -> BCD

3.7.12.2 ITB

The integer number in accu 1-L is converted into a 3-digit [BCD-number](#). With a negative number the four most significant bits of the result are set to '1'.

The number has to be between -999 and +999, there is no monitoring whether these limits are kept.

See also:

[BTI](#)

[BTD](#)

[DTR](#)

[ITD](#)

[DTB](#)

[List of operations](#)

3.7.12.3 DTB

Convert double integer number into 7-digit [BCD-number](#) with sign. With a negative number the 4 most significant bits of the result are all set to '1'.

The number has to be between -9 999 999 and +9 999 999. There will be no controlling whether these limits are kept.

See also:

[BTI](#)

[BTD](#)

[DTR](#)

[ITD](#)

[ITB](#)

[List of operations](#)

3.7.12.4 BTD

Convert BCD in accu 1 into double integer.

The accu 1 that is interpreted as 7-digit [BCD-number](#) is converted into a 4-bytes-binary number

and the result is stored in accu 1.

The 4 most significant bits are interpreted as sign. If they are all '1' the result will be negative.

There will be no controlling whether the number in accu 1 is a valid BCD-number.

Until version 2.1 this function has interpreted the accu 1 as 8-digit, always positive BCD-number by mistake. That is why programs that have made use of this property are faulty now.

See also:

[BTI](#)

[DTR](#)

[ITD](#)

[ITB](#)

[DTB](#)

[List of operations](#)

3.7.12.5 BTI

Convert BCD in accu 1 into integer.

The accu 1 that is interpreted as 3-digit [BCD-number](#) is converted into a 2-bytes-binary number and the result is stored in accu 1-L.

The 4 most significant bits are interpreted as sign. If they are all '1' the result will be negative.

There will be no controlling whether the number in accu 1 is a valid BCD-number.

Until version 2.1 this function has interpreted the accu 1 as 4-digit, always positive BCD-number by mistake. That is why programs that have made use of this property are faulty now.

The use of [BTD](#) will put things right.

See also:

[BTD](#)

[DTR](#)

[ITD](#)

[ITB](#)

[DTB](#)

[List of operations](#)

3.7.13 Other conversions

3.7.13.1 DTR

Convert double integer to REAL. If the number is stored as integer and could be negative it will have to be extended to the DInt-format with [ITD](#) before.

The reverse function is named [RND](#) (Round).

See also:

[BTI](#)

[BTD](#)

[ITB](#)

[DTB](#)

[List of operations](#)

3.7.13.2 ITD

The integer number in accu 1 is extended to a double integer number. For positive numbers absolutely nothing happens doing this, for negative numbers the both more significant bytes of the accu are filled with 1s.

See also:

[BTI](#)

[BTD](#)

[DTR](#)

[ITB](#)

[DTB](#)

[List of operations](#)

3.7.13.3 TRUNC

The REAL-number in accu 1 is changed into a DInt. Decimal positions are cut off.

See also:

[RND](#)

[RND-](#)
[RND+](#)

[List of operations](#)

3.7.13.4 RND

The REAL-number in accu 1 is changed into an integer by rounding.

See also:

[RND+](#)
[RND-](#)
[TRUNC](#)

[List of operations](#)

3.7.13.5 RND+

The REAL-number in accu 1 is changed into an integer by rounding, doing this it is always rounded up.

See also:

[RND](#)
[RND-](#)
[TRUNC](#)

[List of operations](#)

3.7.13.6 RND-

The REAL-number in accu 1 is changed into an integer, doing this it is always rounded off.

See also:

[RND](#)
[RND+](#)
[TRUNC](#)

[List of operations](#)

3.7.14 Trigonometric operations

3.7.14.1 Radian angle

The trigonometrical operations of TrySim expect angles in radians or dispense these in this format.

$$\begin{aligned} 90^\circ &= \text{Pi} / 2 &= 1.570796 \\ 180^\circ &= \text{Pi} &= 3.1415926 \end{aligned}$$

If you have an angle in degree and need it in radian angle, program:

```
L      NumberDeg // in Degree
L      180.0      // not 180 ! but 180.0
/R
L      3.1415926 // Pi
*R                                           // now you have the number in radian angle
```

If you have an angle in radian angle and need it in degree, program:

```
L      NumberDeg // in radian in REAL format
L      3.1415926 // Pi
/R
L      180.0      // not 180 ! but 180.0
*R                                           // now you have the number in degree
```

[SIN](#)
[COS](#)
[TAN](#)
[ASIN](#)
[ACOS](#)
[ATAN](#)

[REAL](#)

3.7.14.2 SIN

The SIN of accu 1 is calculated and the result is stored in accu 1. The angle in accu 1 has always to be there as REAL in [radian](#).

See also:

[ASIN](#)
[COS](#)
[ACOS](#)

[TAN](#)
[ATAN](#)

[List of operations](#)

3.7.14.3 COS

The COS of accu 1 is calculated and the result is stored in accu 1. The angle in accu 1 has to be there as REAL in [radian](#).

See also:

[SIN](#)
[ASIN](#)
[ACOS](#)
[TAN](#)
[ATAN](#)

[List of operations](#)

3.7.14.4 TAN

The TAN of accu 1 is calculated and the result is stored in accu 1. The angle in accu 1 has to be there as REAL in [radian](#).

See also:

[SIN](#)
[ASIN](#)
[COS](#)
[ACOS](#)
[ATAN](#)

[List of operations](#)

3.7.14.5 ASIN

The ASIN of accu 1 is calculated and stored in accu 1 in [radian](#).

It is important that the operand in accu 1 is already in the REAL-format. If it is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before calculating. If an operand that is

loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```
ASIN(-1)    = - Pi / 2 ( Pi = 3,141592... )
ASIN(0)     = 0
ASIN(1)     = Pi / 2 ( Pi = 3,141592... )
```

See also:

[SIN](#)

[COS](#)

[ACOS](#)

[TAN](#)

[ATAN](#)

[List of operations](#)

3.7.14.6 ACOS

The ACOS of accu 1 is calculated and stored in accu 1 in [radian](#).

It is important that the operand in accu 1 is already in the REAL-format. If it is loaded as Int or DInt it will have to be changed into a REAL-number with [DTR](#) before calculating. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```
ACOS(1)     = 0
ACOS(0)     = Pi / 2 ( Pi = 3,141592... )
ACOS(-1)    = Pi ( Pi = 3,141592... )
```

See also:

[SIN](#)

[ASIN](#)

[COS](#)

[TAN](#)

[ATAN](#)

[List of operations](#)

3.7.14.7 ATAN

The ATAN of accu 1 is calculated and stored in accu 1 in [radian](#).

It is important that the operand in accu 1 is already in the REAL-format. If it is loaded as Int or DInt

it will have to be changed into a REAL-number with [DTR](#) before calculating. If an operand that is loaded as Int can be negative it will also have to be extended to the Dword-format with [ITD](#) before.

```
ATAN(-infinite)      = - Pi / 2 ( Pi = 3,141592...)
ATAN(0)              = 0
ATAN(infinite)       = Pi / 2 ( Pi = 3,141592...)
```

See also:

[SIN](#)
[ASIN](#)
[COS](#)
[ACOS](#)
[TAN](#)

[List of operations](#)

3.7.15 Misc. operations with accu 1

3.7.15.1 TAK

The contents of accu 1 and accu 2 are exchanged with each other.

See also:

[CAW](#)
[CAD](#)
[PUSH](#)
[POP](#)

[List of operations](#)

3.7.15.2 CAW

Swap!

Exchange the bytes in accu 1-L:

LL, LH becomes LH, LL

See also:

[CAD](#)
[TAK](#)

[List of operations](#)

3.7.15.3 CAD

The bytes in accu 1 are exchanged:

HH, HL, LH, LL becomes LL, LH, HL, HH

See also:

[CAW](#)

[TAKPUSHPOP](#)

[List of operations](#)

3.7.15.4 INC

Add the argument to accu 1

Syntax: INC *number*

The *number* must be between 0 and 255, it is added to accu 1-LL and the result is stored again in accu 1-LL. The other bytes of the accu stay untouched.

INC is a nice abbreviation for:

L *number*

+I

especially because accu 2 stays untouched. But you have to take care that the result does not get bigger than 255 because there is no transfer into the higher bytes.

Just as practical is [+ \(Plus\)](#).

See also:

[CPU](#)

[DEC](#)

[+I](#)

[+D](#)

[List of operations](#)

3.7.15.5 DEC

Decrement accu 1.

Syntax : DEC *number*

The *number* has to be between 0 and 255, is subtracted from accu 1-LL and the result is stored in accu 1-LL again. The other bytes of the accu stay untouched.

DEC is a nice abbreviation for:

 L *number*
 -I

especially because accu 2 stays untouched. But you have to take care that the result does not get less than 0 because otherwise there will be an unpleasant surprise.

Just as practical is [+ \(Plus\)](#).

See also:

[CPU](#)

[INC](#)

[-I](#)

[-D](#)

[-R](#)

[List of operations](#)

3.7.15.6 + (Plus)

Syntax : + 5

The constant that is specified as operand is added to accu 1. The other accus and the displays will not be influenced.

See also:

[+I](#)

[+D](#)

[+R](#)

[List of operations](#)

3.7.16 Operations with accu 3 and 4

3.7.16.1 PUSH

The accu 1, 2 and 3 are shifted about one place to the top together. The value in accu 1 (V1) is in accu 1 and in accu 2 after that, the number in accu 4 gets lost.

```
      -->
V4      V3
V3      V2
V2      V1
V1      V1
```

This instruction is only available in PLCs of the S7-400 series.

See also:

[POP](#)

[ENT](#)

[LEAVE](#)

[Accu 3 and 4](#)

[List of operations](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.7.16.2 POP

The accu 2, 3 and 4 are shifted about one place to the bottom together. The value in accu 4 (v4) is in accu 4 and in accu 3 after that, the number in accu 1 gets lost.

```
      -->
V4      V4
V3      V4
V2      V3
V1      Z2
```

This instruction is only available in PLCs of the S7-400 series.

See also:

[PUSH](#)

[ENT](#)

[LEAVE](#)

[Accu 3 and 4](#)[List of operations](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.7.16.3 LEAVE

Accu 4 and accu 3 are shifted about one place to the bottom together. After this the value accu 4 (V4) is in accu 3 as well as in accu 2. The number in accu 1 gets lost.

```
      -->
V4          V4
V3          V4
V2          V3
V1          V1
```

This instruction is just available in PLCs of the S7-400 series.

See also:

[ENT](#)

[POP](#)

[PUSH](#)

[Accu 3 and 4](#)[List of operations](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.7.16.4 ENT

(Enter) Accu 2 and accu 3 are shifted together one place ahead. The content of accu 4 (V4) gets lost, accu 1 stays unchanged.

```
      -->
V4          V3
V3          V2
V2          V2
V1          V1
```

This instruction is just available in PLCs of the S7-400 series.

See also:

[POP](#)

[PUSH](#)

[LEAVE](#)

[Accu 3 and 4](#)

[List of operations](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.7.17 Register indirect addressing

3.7.17.1 Warning using AR1 or AR2

Unfortunately we have to urge you to be careful using the [AR1](#) and [AR2](#). These registers are modified independently by the Siemens - S7 under specific circumstances or they must not be changed by the user because of the use for S7 internal purposes.

Although it was possible to program complex operations (especially in loops) very elegantly with these registers in principle, they will get quite useless because of the mentioned restrictions.

Using these registers you have to know very precisely whether and when the S7 operating system uses these for internal purposes so that no unexpected results occur. The search for the cause for maloperations which results from an unpermissible use will take much more time than needed if you do without a loop and write out explicitly 20 till 50 assignments or transfers instead.

The PLC simulated in TrySim does not change the contents of the AR register and does not need them for any operations which are not programmed by you explicitly. This can lead to the fact that a program functions with TrySim well but not in the real use. Such mistakes are quite difficult to find like we have been told and like we know from experience.

Please check the function of the parts of a program tested with TrySim which use the address register before starting up on a real S7 absolutely.

Naturally we will try to detect potential problem cases in TrySim and to give a corresponding warning announcement. But doing this he have to know precisely under which circumstances a S7 modifies the address registers or is dependent on a content that is not changed by the user. We are very grateful for tips on the matter. [Here you find how to contact us](#).

A safe alternative is the use of the memory-indirect addressing. Unfortunately there are not such operations as +AR1 and other annoying restrictions, like,e.g., that IN-parameters are not allowed to

be used as index but have to be transferred in a TEMP-variable first.

See also:

[Indirect addressing](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.7.17.2 LAR1 or 2

With this operation you can load the [address register1 or 2](#). It can be used with as well as without operand.

If it is used with operand this one will have to be a [pointer](#). If it is used without operand the content of accu 1 will be loaded into the corresponding address register. Of course in this case a valid pointer has to be loaded into accu 1 before.

Remark: TrySim allows more LAR1/2 operands than STEP®7. This can lead to the fact that a program that functions in TrySim cannot be exported to STEP®7. In such cases try to load the pointer into accu 1 first and use LAR1/2 without operand then.

Example

Do not write:

```
LAR1 P##Any1 // #Any1 is declared as In - ANY
```

But:

```
L P##Any1 // Load the pointer to #Any1 in Accu1
LAR1 // Load this pointer into AR1
```

[!! Warning with usage of the address register !!](#)

See also:

[L](#)

[ANY](#)

[CAR1 or 2](#)

[+AR1/2](#)

[Indirect addressing](#)

[List of operations](#)

STEP®7 is a registered trademark of the Siemens AG.

3.7.17.3 +AR1 or 2

This operation is used to increase the address register [address register 1 or 2](#) in [loops](#), for example. First the step width has to be loaded into accu 1 or this one can be specified as operand in the pointer format.

Accu 1 has to contain these values to obtain the desired step width

Step width	Pointer format	Hex Format	Example
1 Bit	P#0.1	0001h	Q 5.7 -> Q 5.8
1 Byte	P#1.0	0008h	I 7.3 -> I 8.3
1 Word	P#2.0	0010h	MW 6 -> MW 8
1 Dword	P#4.0	0020h	DBD10 -> DBD 14

[!! Warning using the address register !!](#)

Example of a loop with AR1

```

LAR1 P#Q6.0      // Load the address register with pointer to A6.0
L 4              // 4 iterations
lbl T #Idx       // #Idx is local INT variable
:
SET              // Set RLO to '1'
= [AR1,P#0.1]    // Aim is AR1 + P#0.1
+AR1 P#1.2       // Step width is 1 byte, 2 bits
L Idx
LOOP lbl

```

This has got the same effect as:

```

SET
= Q 6.1
= Q 7.3
= Q 8.5
= Q 8.7

```

See also:

[Indirect addressing](#)

[List of operations](#)

See also:

[Indirect addressing](#)

[List of operations](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.7.17.4 CAR1 or 2

The address register is transferred in the specified operand or, if none is specified into accu 1.

See also:

[CAR](#)

[LAR1 or 2](#)

[+AR1 or 2](#)

[Indirect address](#)

[List of operations](#)

3.7.17.5 CAR

The contents of both address registers are exchanged with each other.

[!! Attention with use of address register !!](#)

See also:

[CAR1 or 2](#)

[LAR1 or 2](#)

[Indirect address](#)

[List of operations](#)

3.7.17.6 Indirect addressing with AR1 and AR2

For some applications the procedure of the [memory indirect addressing](#) is not flexible enough. That is why there are two special registers (AR 1 and AR 2, address register 1 and 2) in the CPU and by their help the index is calculated only during the access. Before the first use of the address registers these have to be loaded with a pointer to an operand. If you want to access the operand I 5.6 you will program:

```
LAR1 P#5.6 // Load pointer to 'anything' 5.6 in AR1
```

Now the address 5.6 is in the address register 1. You ask the input with the following operation:

```
A I [AR1, P#0.0]
```

But if you want to ask the address I 5.7, program:

```
A I [AR1, P#0.1]
```

The [pointer](#) behind the comma is added to the value in the address register 1, and the result is interpreted as byte- and bit-number of the input. Doing this it is considered that bytes have only got 8 bits, so bit 5.9 becomes bit 6.0.

If you program

```
A I [AR1, P#0.2]
```

now, the input 6.0 will be evaluated.

You call the whole thing register-indirect-area-internal addressing. You call it area-internal because (in the example) only inputs are accessed.

Naturally you can access these not just by bytes, words and DWords.

Example:

```
L MW [AR2, P#2.0]
```

But there is also the [register-indirect-area-crossing addressing](#). That means that in the approach you do not even have to decide whether inputs, outputs or markers are accessed.

You do it like this:

You load the AR1 with the address Q 5.3:

```
LAR1 P#Q5.3
```

Then you assign the value of the RLO to the output Q 5.3 by programming:

```
= [AR1, P#0.0]
```

And corresponding you assign to the output Q 17.2

```
= [AR1, P#11.7] // (5.3 + 11.7 = 17.2)
```

Here you can also access to bytes, words and DWords:

```
L W [AR1, P#4.0]
```

Strangely the area crossing access to local data is not allowed in a S7-300, but it is in a S7-400. We have prohibited this access in TrySim.

[!! Attention with Use of the Address Registers !!](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.7.17.7 Indirect addressing

You do not have to specify the operands finally while writing your program, but you can specify them while runtime. This will be useful if always returning operations with different operands are to be executed. If you have, for example, saved 14 recipes in the data blocks DB 1 -14 and the number of the current recipe is saved in MW 20, you will program:

```
OPN DB[MW 20]
```

If there is a 5 in MW 20 the DB 5 will be opened by this instruction and the program works in the following with the recipe which is saved in it. You call this procedure 'Memory Indirect Addressing' because the index in the square brackets can be any memory place.

For indirect access to separate bits one word is not enough because of the big address area of inputs, outputs, markers and data bits of 65536 each, because one bit is already needed for the specification of the byte address. For the indirect addressing of these data areas a double word is needed. For reasons of unity this will also be valid if bytes, words or DWords are accessed. To code the bit number the three less significant bits of the DWord are used. Because of the fact that these bits are not available for the byte number this one is multiplied with 8 and added to the bit number. In the result all bits of a data area are numbered all the way through

Bit 0.2	gets the number	2	hex 0000 0002
Bit 1.0	gets the number	8	hex 0000 0008
Bit 8.3	gets the number	67	hex 0000 0043

In practise you will not have to care about these details if you use the [POINTER-Format](#), then you just program:

```
L      P#8.3
T      MD 24
A      Q[MD24]
```

to access the output Q 8.3.

See also:

[Register-Indirect-Addressing](#)

3.7.18 Operations for blocks

3.7.18.1 OPN

Open data block.

If you need data out of the same data block inside a program part quite often you will have to ‘open’ it. Addressing data words you do not have to specify in the following in which DB they are, the PLC takes them automatically out of the opened data block.

Warning: If you call an operand with the full qualified access while working with an opened DB the corresponding DB will be opened automatically! Example:

```
OPN      DB 10
L        DBW 20           // The DW 20 of the DB 10 is stored in accu 1
L        DB30.DBW 40     // The DW 40 of the DB 30 is stored in accu 1
L        DBW 20           // The DW 20 of the DB 30 is stored in accu 1
```

Internally the global DB register of the [CPU](#) is loaded with the specified DB by the operation OPN.

You can use the operation OPN with the syntax OPN DB as well as with OPN DI xx. In this case the global DB register is not loaded with the specified DB but the instance DB register. All accesses to data that contains an ‘I’ (DIX, DIB, DIW or DID) relates to the so opened DB subsequently. Attention: Even all operands that are marked with ‘#’ that are not out of the TEMP area are now in the DB that is opened as OPN DI.xx!

You can load the number of the just opened DB/IDB with L DBNO/DINO into accu 1.

DB7

OPN

DI7

OPN

See also:

[CALL](#)

[CC](#)

[UC](#)

[List of operations](#)

3.7.18.2 BEU

Block end unconditioned

If this instruction is reached while program processing the return to the calling block will happen. This instruction is usually only used in program parts, that are skipped sometimes, but also while fault locating it will be helpful if you want to preclude that the searched malfunction is caused by following data blocks.

See also:

[BEC](#)

[List of operations](#)

3.7.18.3 BEC

Block end conditioned

If the [RLO](#) is '1' while reaching this instruction the processing of the data block will be aborted and it will be returned to the calling block. Otherwise the RLO is set to '1' and the next instruction is processed. BEC is mostly used in data blocks which shall execute specific (less) operations always, the rest of the data block only if a condition is there.

See also:

[BEU](#)

[List of operations](#)

3.7.18.4 CALL

Call a block

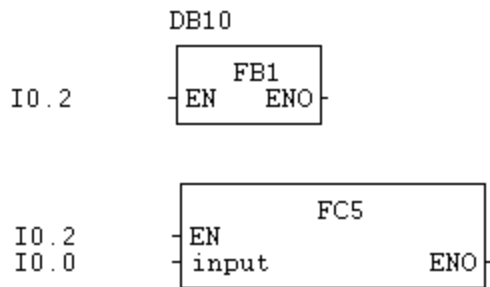
```
Syntax:CALL FB x, DB y  
        CALL FC x
```

With CALL you achieve the jump to the specified block that can be a [function\(FC\)](#) or a [function block\(FB\)](#). Calling a FB you have to specify the DB that shall be used as [instance-DB](#) by the FB.

If you program the instruction CALL the parameter of the called block appear automatically.

With FCs you have to specify an operand for each of the listed [formal parameter](#). Calling a FB you

do not have to do this: if you do not specify an operand the corresponding value of the instance DB will be used inside the FB.



The FB1 resp. the FC5 will only be called if the input I 0.2 is set to '1'. If the calling shall happen absolutely you will be able to leave the EN-input unused.

See also:

[EN-input](#)

[ENO-output](#)

[OPN](#)

[CC](#)

[UC](#)

[SFB](#)

[List of operations](#)

3.7.18.5 CC

Conditional call

Syntax: CC FBxx
 CC FCxx

The specified block will be called if the [RLO](#) is '1', otherwise the next instruction will be processed. No parameter list appears.

You should use this kind of calling just for functions and function blocks that have no parameter and no static [variables](#) because otherwise unexpected results will occur, in every case your program will be hard to understand.

If you do not specify an IDB when calling a FB the IDB of the calling block will be used instead! This does not lead to sensible results quite often.

See also:

[CALL](#) [UC](#)

[List of operations](#)

3.7.18.6 UC

Unconditioned Call

Syntax: UC FBxx
 UC FCxx

The specified block is called independent of the [RLO](#). No parameter list appears. You should use this kind of call only for functions and function blocks that do have neither parameter nor static [variables](#), otherwise unexpected results occur, your program will definitely be difficult to understand.

If you do not specify an IDB calling a FB the IDB will be used instead of the calling block! This usually leads to silly results.

See also:

[CALLCC](#)

[List of operations](#)

3.7.18.7 CDB

The contents of both data block registers (global- and instance data block register) are exchanged with each other.

Accesses with DBW refers to the current instance-DB afterwards, accesses with DIW refers to the present global-DB afterwards. This is also valid for all parameters that are marked with '#' and variables except of the TEMP-variables.

You shall only use this operation if the global-DB is also an instance-DB to the current function block, otherwise it will become quite confusing.

See also:

[OPN](#)

[CPU](#)

[List of operations](#)

3.7.19 Nought / Zero operations

3.7.19.1 NOP 0 or 1

Here the PLC does not do anything.

The instruction is used by the system as wildcard for not connected inputs of flipflops and the like, so that the retranslation from STL to FBD or LAD is easier.

If you program in STL you will usually use this instruction because of the fact that in one line with [jump label](#) an instruction **has to be**.

See also:

[BLD](#)

[List of operations](#)

3.7.19.2 BLD

This instruction causes absolutely nothing. You will never program it. It only appears in the STL-code of networks that are programmed in FBD or LAD. The programming system inserts it to guarantee the retranslation of the STL code into a FBD or LAD graphic.

See also:

[NOP](#)

[List of operations](#)

3.7.20 Master control relay (not implemented)

3.7.20.1 Master control relay

The instructions 'MCRA', 'MCRD', 'MCR(' and ')MCR' are understood by the editor but they cannot be executed. Transferring a block that uses these instructions into the [PLC](#) a warning is given.

[Not implemented properties](#)

3.7.20.2 MCR(

The functions for the master control relay are not implemented in TrySim.

See also:

[\)MCR](#)

[MCRA](#)

[MCRD](#)

[List of operations](#)

3.7.20.3)MCR

The functions for the master control relay are not implemented in TrySim.

See also:

[MCR\(](#)

[MCRA](#)

[MCRD](#)

[List of operations](#)

3.7.20.4 MCRA

The functions for the master control relay are not implemented in TrySim.

See also:

[MCR\(
\)MCR
MCRD](#)

[List of operations](#)

3.7.20.5 MRCD

The functions for the master control relay are not implemented in TrySim.

See also:

[MCR\(
\)MCR
MCRA](#)

[List of operations](#)

3.8 Differences between the TrySim-PLC to S7

Some properties of the real S7 are not available in TrySim, others are realized in another way, so that a differing behaviour can occur in special cases.

[Not implemented Properties](#)
[Varying implemented Properties](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.8.1 Not implemented properties

The following properties of S7 are not or not completely existing in TrySim:
(also this list is not complete).

[Master-Control-Relay](#)
[Process Interrupts](#)
[Time of Day Interrupts](#)
[Cyclic Interrupts](#)
[Limited ARRAYs](#)
[Limited Function of UDTs](#)

[Function 'FR' with Timers and Counters](#)
[JUO, JO and JOS](#)
[STRINGS](#)

S7-300 and S7-400 are registered trademarks of Siemens AG.

3.8.1.1 Master control relay

The instructions 'MCRA', 'MCRD', 'MCR(' and ')MCR' are understood by the editor but they cannot be executed. Transferring a block that uses these instructions into the [PLC](#) a warning is given.

[Not implemented properties](#)

3.8.1.2 Process interrupts

Process interrupts are not implemented. But you can simulate these interrupts limited because you can call OBs in TrySim as well. Program in a conditional OB call after evaluation of the value of the condition that shall trigger off the interrupt.

[Not implemented properties](#)

3.8.1.3 Time of day interrupts

These interrupts do not exist either. You can recreate time of day interrupts which are triggered off in bigger intervals than the scan time by a conditional OB call after some time.

From version 2.2 on a cyclic interrupt [OB 35](#) exists.

[Not implemented properties](#)

3.8.1.4 Clock interrupt

The tabsheet can be reached by [PLC|CPU Properties](#).

The OB 35 will be called in the here given intervals independent of the adjusted [simulation rate](#). Further clock interrupts are not yet available.

3.8.1.5 Limited ARRAYS

Arrays of structures, other complex types as well as arrays of arrays and multidimensional arrays are not possible yet.

In addition not all possibilities can be used for the default value instructions.

You can create arrays of following types:

BOOL, BYTE, CHAR, INT, WORD, DATE, S5TIME, DWORD, DINT, REAL,
TIME_OF_DAY, TIME

Arrays can be limited created by [UDTs](#) as well. This is a possibility to bridge the missing of arrays out of structures.

See also:

[Not implemented properties](#)

3.8.1.6 Limited function of UDTs

UDTs (user defined types) can serve as mask for creating DBs as well as an abbreviation for declaration of structured variables. But the processing is sometimes still faulty. If you want to use UDTs please ask us whether a better version is available.

[Not implemented properties](#)

3.8.1.7 Function 'FR' with timers and counters

This function is not implemented.

[Not implemented properties](#)

3.8.1.8 JUO, JO and JOS

These conditional jump instructions are not implemented.

[Not implemented properties](#)

3.8.1.9 STRING

STRINGS are implemented from version 1.5 onwards. You can declare Strings, use them as parameter of functions and import and export them. You access the single letters with the notation `stringname[idx]`.

The following functions are available in the directory IECfuncs for the process of Strings until now:

FC 21 Determine the length of a String.

Others are in preparation, please ask if required.

[Data types](#)

3.8.2 Variant implemented properties

[And before Or](#)

[Parameter-Actualisation with Functions](#)

[Out-Parameter of Functions](#)

[Data Type Check](#)

3.8.2.1 AND before OR

In a S7 this calculating rule is realized by the OR-Bit. Because the exact recreation of this process would have caused to a distinct reduction of the process speed the And before Or rule is already considered by the compiler in TrySim. If you program in FBD or LAD you will not notice any differences between the behaviour of the TrySim-PLC and a S7. But if you program jumps between different bracket levels in STL it will be possible that divergences occur. Avoid such jumps which lead to quite unreadable programs by the way.

[Differing implemented properties](#)

S7-300 and S7-400 are registered trademarks by the Siemens AG.

3.8.2.2 Parameter-updating at functions

The following differences just concern functions, not function blocks. Assigning to an Out- or In-Out-parameter whose actual-parameter is not another parameter of the calling block but an direct operand (in-/output, marker, data word) the actual-parameter is changed in the moment of the assignment in a S7. Accessing an In-parameter to which an operand is connected the actual state of

this operand will be detected even if this one has changed since the call of the block, in the same way.

In TrySim the updating of the actual-parameter which are connected to Out- and In-Out-parameter does not happen until the return to the calling block. The In- and In-Out-parameter are updated once at calling the block, after that the actual-parameter are not read again. But the differences caused by this will only occur if you access an operand directly as well as by a parameter.

Pay attention to this program example:

S7 TrySim

OB 1				OB 1		
		Status				Status
	SET	1			SET	1
	= M 1.2	1			= M 1.2	1
	CALL FC				CALL FC 2	
	2					
	Par1: M 1.2				Par1: M 1.2	
	U M 1.2	0			A M 1.2	0
FC 2				FC 2		
OUT	Par1	BOOL		OUT	Par1	BOOL
		Status				Status
	U #Par1	1			A #Par1	1
	U M 1.2	1			A M 1.2	1
	CLR	0			CLR	0
	= #Par1	0			= #Par1	0
	U #Par1	0			A #Par1	0
	U M 1.2	0			A M 1.2	1

Avoid such mixed accesses which can lead to program errors that are hardly to locate. In the upper example the state of the marker M 1.2 has changed from 1 to 0 on the left side during the program process and this is not perceptible by an explicit assignment.

[Differing implemented properties](#)
[Out parameter of functions](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.8.2.3 Out-parameter of functions

The following just concerns functions, not function blocks.

In a S7 Out-parameter have already got a defined value at calling contrary to the rules of the IEC 1131. Actually they behave like In-Out-parameter. This tempt some programmer into using Out-parameter like In-Out-parameter, for example by not assigning a value to an Out-parameter in many scans and only resetting it sometimes. We think that this practice is very dangerous. Because there is no guarantee that later revisions of a CPU show the same behaviour. The consequence is that a program does not run anymore on a new CPU after a failure of a CPU. You cannot pin the blame on Siemens in such a case: the new CPU does behave according to IEC 1131. Even the transfer of the program to another system that behaves conforming to IEC 1131 takes more time than expected by the misuse of the Out-parameter.

That is why in TrySim the Out-parameter do not have a defined value at calling a function regularly. This can lead to the fact that a program that runs on a S7 well will cause problems if it is processed by the TrySim-internally-PLC.

For the users of the standard- and professional-version we offer the option to recreate exactly the wrong behaviour of the real S7 in this point.

What is said about the Out-parameter on top is also valid for the In-parameter basically. But here there is a clear offence against the regulation of the IEC 1131 in the S7: An In-parameter must not be changeable in that way that the calling block has to calculate with other data after that. But the temptation to assign a new value to an In-parameter in a function is quite little, so that you must expect problems only in special cases. TrySim behaves in this point conforming to IEC 1131.

[Differing implemented properties](#)
[Parameter-updating with functions](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

3.8.2.4 Data type check

The data type check in TrySim is not that strict as in STEP®7 as long as the sizes of the operands correspond. So you are able to connect an In-parameter that is declared as CHAR to a BYTE, for example, without criticism of the compiler. In STEP®7 this will lead to a type conflict.

[Differing implemented features](#)

STEP®7 is a registered trademark of the Siemens AG.

Part



4 Program editor

4.1 Block functions

4.1.1 Create new blocks

- Click **PLC|New**
- Select [kind of block](#) and enter the number of the block.
- Select the desired kind of representation by clicking one of the buttons STL, FBD, LAD in the upper toolbar.
- You can adjust in **View|Options|PLC-Editor** which your favourite representation is.
- The first network is inserted automatically.

You can also create data blocks at runtime with the [SFC22](#).

See also:

[Save and Open Blocks](#)

[Export and Import Blocks](#)

[Edit OBs and FBs](#)

[Edit DBs](#)

[Transfer Blocks to PLC](#)

[Monitor Program Process](#)

[Cross Reference List](#)

4.1.2 Block types

- OB [Organization Blocks](#)
- FB [Function Blocks](#)
- FC [Functions](#)
- DB [Data Blocks](#)
- I-DB [Instance DB](#)
- UDT-DB [UDTs](#)
- SFB [System Function Blocks](#)
- SFC [System Functions](#)
- UDT [UDTs \(user defined type\)](#)

See also:

[Create new Blocks](#)

[Save and Open Blocks](#)

[Export and Import Blocks](#)

[Edit OBs and FBs](#)

[Edit DBs](#)

[Transfer Blocks to PLC](#)
[Monitor Program Process](#)
[Cross Reference List](#)

4.1.3 Open and save blocks

You can open program blocks in different ways:

- Select **PLC|Open** and select a block out of the list. With the buttons above the list you can filter the display by types of blocks.
- Select **PLC|NewOpen**. Now a list with all blocks that are opened before appears.
- You double click the line in the [cross-reference list](#), in which the desired block is. With this method you are already in the right network.
- You double click the corresponding block call in LAD, FBD or STL.
- In LAD, FBD or STL you click a block call with right and select 'Open' out of the context menu.
- Blocks will be opened at the right position automatically if an error occurs while transferring or at runtime.
- You enter the block in the [single step mode](#) with 'Trace'.

You can save program blocks in different ways:

- Select **PLC|Save**

Select **Project|Save All** or the symbol 

- You close the block and answer the question after the saving with 'Yes'.
- You activate the check box 'Auto Save' in **View|Options|PLC-Editor**. Now the blocks are saved at closing without annoying confirmation.
- The blocks are saved automatically before the transfer into the PLC.

See also:

[Kinds of Blocks](#)
[Create Blocks](#)
[Delete Blocks](#)
[PLC-Editor](#)

4.1.4 Delete a block

To delete a block you have to select **PLC|Open**, mark the block and press the key 'Del'. After that the block is deleted irrevocably and will be erased from the PLC.

You can also delete several marked blocks at the same time.

The item 'Delete' is also in the [context menu](#) of the Open-Block-Window.

And you can delete data blocks with the [SFC23](#) at runtime.

See also:

[Create New Blocks](#)

4.1.5 Export and import blocks

TrySim is a real offline-software. If you want to let run a program that is created by TrySim on a real PLC it will only be possible with the original development system of the manufacturer. Until now TrySim supports only the S7-300/400 series of Siemens.

Export to STEP®7

You can export the program and the [symbol table](#) to STEP®7, go on processing it there and transfer it afterwards into a real PLC.

[Export the Program](#)

[Export the Symbol Table](#)

[Problems with Export](#)

Import from STEP®7

You can import a program and the symbol table which you have written in STEP ®7 to TrySim and test it there. If you do changes you will be able to reexport it afterwards to STEP ®7.

[Import the Program](#)

[Import the Symbol Table](#)

[Problems with Import](#)

Import from STEP®5

The import of STEP ®5 is not possible anymore.

See also:

[PLC-Editor](#)

STEP®7, STEP®5 S7-300 and S7-400 are registered trademarks of the Siemens AG.

4.1.6 Edit OBs and FBs

This help point just refers you to other positions.

[STL](#)[FBD](#)[LAD](#)[Edit the Block Header](#)[Network Functions](#)[Open and Save Blocks](#)[Create New Blocks](#)[PLC-Editor](#)

4.1.7 STL

The STL-editor is a normal text-editor in general and can be operated with the key combinations which are conventional for windows.

A selection list does not exist for STL like it does for LAD and FBD.

Comments start with '//'.

You have to finish [jump labels](#) with a ':?'

By clicking right you get a context menu.

Double click a line with a block call opens that one.

Double click a line with full-qualified access opens the corresponding DB.

You can [take over](#) operands out of the machine directly by marking the corresponding element (in the graphic window, address table or element tree) and select the last line out of the context menu in the STL editor then. If you have marked text before that one will be replaced by the operand. If you have not marked text the operand will be inserted at the cursor position.

See also:

[FBD](#)[LAD](#)[PLC-Editor](#)

4.1.8 FBD

You find the instructions which are needed most frequently on the toolbar which will appear if the representation of the current block is 'FBD'. If this bar does not appear you might click with the right mouse button on the grey pane next to the existing symbols and select 'FBD' out of the appearing context menu.

There will be further instructions if you click the symbol .

For some instructions short keys are existing:

F2 And-Box

F3 Or-Box

F8 New Connection
F9 Negation on/off

If the network is still empty you do not have to note the position of the cursor. But if function blocks already exist you will have to position the cursor onto an operand or onto the connection between two function blocks before inserting new elements.

You open the editing field of the operands by the insert-key or by entering a letter, a number, '#' or the double inverted commas. After having inserted you close the field with the enter key, or, if you do not want to take over your insert you will press the **ESC**-key.

You can [take over](#) operands directly out of the machine. For this mark the element in the graphic window, in the address table or in the element tree. With operands which still start with a '?' (or which are completely empty) a click with the left mouse button is enough to insert the operand out of the machine. If you want to replace an already addressed operand you will have to select the last point out of the context menu.

You do not have to address the operands immediately as long as they start with a '?'. But as soon as you transfer the program into the PLC all obligatory operands have to be specified.

Parts of the network can be deleted by the **Del**-key.

By clicking right you get the context menu.

You open a block with a double click.

See also:

[STL](#)

[LAD](#)

[PLC-Editor](#)

4.1.9 LAD

You find the instructions which are needed most frequently on the toolbar which will appear if the representation of the current block is 'LAD'. If this bar does not appear you might click with the right mouse button on the grey pane next to the existing symbols and select 'LAD-View' out of the appearing context menu.

Further instructions can be selected if you click the symbol .

If the network is still empty you do not have to note the position of the cursor. But if rongs already exist you will have to position the cursor on a contact or on another function block before the inserting.

You open the editing field of the operands by the enter-key or by the entering a letter, a number, ‘#’ or the double inverted commas. After having inserted close the field with the enter-key, or, if you do not want to take over your insert you will press the **ESC**-key.

You can [take over](#) operands directly out of the machine. For this mark the element in the graphic window, in the address table or in the element tree. With operands which still start with a ‘?’ (or which are completely empty) a click with the left mouse button is enough to insert the operand out of the machine. If you want to replace an already addressed operand select the last point out of the context menu.

You do not have to address the operands immediately as long as they start with a ‘?’. But as soon as you transfer the program into the PLC all obligatory operands have to be specified.

Parts of the network can be deleted by the ‘Del’-key.

By clicking right you get a context menu.

You open a block with a double click.

See also:

[STL](#)

[FBD](#)

[PLC-Editor](#)

4.1.10 Transfer operand

Transfer an operand means to insert an operand that is already used in the machine in the PLC-program without typing it. If you create an indicator, for example, for displaying a specific state in the machine and you want to access that one in the PLC-program, you will only have to mark the indicator in the machine and click the corresponding assignment in the PLC-program afterwards. The address by that the indicator is accessed is transferred automatically in the PLC-program without typing it again. In FBD/LAD and STL a little different rules are valid how and when an operand is transferred.

[Transfer Operand in FBD](#)

[Transfer Operand in LAD](#)

[Transfer Operand in STL](#)

4.1.11 Fast operand-input by the numbers-block

To accelerate the input of operands on the edit mask, in the FBD-/LAD-editor as well as in the

symbol- and address-table some keys have got another meaning on the numbers block:

÷ : I
* : Q
- : M
, : .

4.2 Operations on networks

New Network

As long as you write one network after the other they are created automatically as soon as you leave the last network with 'Page Down' or with the double arrow on the scrollbar. If you want to add networks afterwards select **PLC |Network|New**. The new network is added in front of the current one. Note that you do not get an empty network with the function **Network|Add**, but the one that is deleted resp. copied last.

Delete Network

Select **PLC|Network|Delete**. The deleted network is saved in the clipboard and can be inserted again with **Network|Insert** on another position, even in another block.

Copy network

With **PLC|Network|Copy** you transfer the current network into the clipboard without deleting it. You can insert this network again on another position with **Network|Insert**

Insert Network

With **PLC|Network|Insert** you insert an already deleted or copied network in front of the current network. You can insert networks several times and also in other blocks.

See also:

[PLC-Editor](#)

4.3 Editing of a block header

You will only have to edit the block header if you want to provide the block with parameter or if you need [static](#) / [temporary](#) variables. After each changing of the block header you should check all calls of this block. The fastest way to do so is by the [cross-reference list](#), just double click each entry of the block.

Description of the columns

Address	Declaration	Name	Type	Default	Comment
0.0	in	Stop	BOOL	FALSE	Button Stop
2.0	out	Speed	INT	0	Speed in mm/s
	in_out				
4.0	stat	SMStart	BOOL	FALSE	Slope marker Start
0.0	temp	Idx	WORD		Index in table

In the first column the automatically specified address is displayed. A new declaration type always starts on an even address. With function blocks the numbering of the temporary variables starts with zero again, because these variables are not saved together with the other values in the [instance-DB](#) but on the stack.

In the second column you recognize the declaration type. Which declaration type a parameter / variable gets is specified by the position in the table (see a little bit further down).

In the third column you enter the name, do not use any special characters except for ‘_’ and no space characters. And the name must not start with a number. In the column type you enter the data type. You only have to enter the letters of the name of the type that are enough to specify the type. The following [data types](#) are available:

BOOL, BYTE, INT, WORD, DINT, DWORD, TIMER, COUNTER, S5TIME, TIME, DATE, BLOCK_DB, BLOCK_FB, BLOCK_FC, POINTER, ANY, ARRAY, STRUCT

In the column default it is best not to enter anything first. Then the system enters a zero value automatically in the correct format which you can adjust to your requirements afterwards.

In the comment line you should enter an as precise as possible description of the variables.

Insert new lines

Put the cursor on an already existing line of the desired declaration type. If you want to insert a new line in front of it you will have to press the key ‘Ins’, if you want to insert a new line behind it you will have to press the ‘Return’ (enter) key several times.

Delete lines

Put the cursor on the first column or click it. If you press the space bar the whole line will be marked. With the key ‘Del’ you delete the line. If you press the ‘Shift’ key at the same time the line will be saved in the clip board and you can insert it at another position (but not in another block) with ‘Shift’-‘Ins’.

Change the declaration type

As the declaration type is specified by the position in the block header you will only be able to change the type by marking the line (select field ‘Address’ and press space bar then), cutting out with ‘Shift’-‘Del’ and inserting with ‘Shift’-‘Ins’ in the correct section again.

See also:

[Edit OBs and FBs](#)

[PLC-Editor](#)

4.4 Edit DBs

Editing a DB you have to distinguish whether it is a freely created DB or a DB that is created as instance-DB to a [function block](#). As well the DB can be created by the model of an [UDT](#).

1 Editing of a free created DB

If you have opened a DB you will be able to choose between the declaration view and the data view in the menu **View**. Only in the data view you can see the currently saved values in the PLC and you are able to change them. In the declaration view on the other hand you can change the name, data type, default value and comment of separate lines. To delete separate lines put the cursor on the first column and press the space bar then. The line is marked. You can delete it with the 'Del'-key or shift it in the temporary storage with 'Shift' - 'Del' or insert it on another position with 'Shift' - 'Ins'. Unfortunately the marking of several lines is not possible yet.

2 Editing of instance DBs and DBs which are created by the model of an UDT.

In these DBs the structure is already specified irrevocably at the creating, so they are opened automatically in the data view and the only thing that you can edit are the data in the PLC:


If you monitor a data block at simulation runtime in the data view you will be able to activate the monitoring mode in the menu **View**, now the view is updated about every second with the values out of the PLC. If you need the current values just sometimes in the PLC you will be able to select **View|Update**, then the data are only read once out of the PLC.

See also:

[Editing of OBs and FBs](#)

[PLC-Editor](#)

4.5 Transfer a block into the PLC

To transfer an (open) block you select the menu item: **PLC|Transfer** or click the symbol: .

Other blocks that are needed by the just transferred block are transferred automatically. So you just have to transfer the OB 1 and then each block that you have changed.

Needed data blocks are also transferred to the PLC automatically. This is also valid for the data blocks that are called by the indirect addressing. Data blocks will also be transferred into the PLC if you open them. Opening a DB it can happen that the data of a block in the PLC do not correspond with the data that are saved on the hard disc. In this case you are asked to decide while opening whether the data shall be transferred from the file to the PLC or from the PLC to the file.

A block will also be transferred into the PLC automatically if you switch on the [monitoring mode](#).

TrySim is ([with restrictions](#)) a real offline-software. If you want to transfer blocks into a real PLC you will have to [export the program](#).

In real PLC systems you have to distinguish exactly between the blocks in the PLC (online) and the ones in the developing system (offline) because of safety reasons. But in TrySim this differentiation is not important but quite annoying and after a while you will see the transferring as 'Confirming'.

See also:

[Editing of OBs and FBs](#)

[PLC-Editor](#)

4.6 Watch the program execution

You enter the monitor mode by the menu item **PLC|Monitor** or

by the symbol .

On the status line at the bottom left appears a running progress bar. If this display does not move the block will not be executed presumably. Either the machine is in the state 'Stop' (red light in the machine window at the bottom left), or the block is not called, or 'Block End' is programmed in front of the monitored network, or the program is not executed because of a fatal error (but for this you get a message).

You switch the monitor mode off by renewed selection of the menu item, by renewed clicking the symbol or with the 'Esc'-key.

You can also monitor several blocks at the same time. Often it is useful to use the [single step mode](#) while monitoring. While monitoring no inserts are possible.

Monitoring in FBD

Active operands are displayed red, inactive grey. Underneath of byte- word- und DWord- operands the current value is displayed. Displaying timers the current value is at the upper right side, displaying counters it is in the lower part of the rectangle.

Monitoring in LAD

Active connections are displayed red, connections without current grey. The representation of the other elements is just as the one of FBD.

Monitoring in STL

Next to the program code on the right side following informations are displayed instead of the comment:

```
RLO | State of the operand | Accu 1 | Accu 2
```

The representation of the accu is adjusted as good as possible to the corresponding operand automatically. So the accu are displayed as integer after an integer addition, but after a wordwise and-connection as binary numbers as long as 12 digits are not exceeded, after that as hex-numbers.. The more precisely you declare the [data types](#) in the declaration headers and in the symbol table the better the system knows how to display the accu.

If you always save a REAL number in MD 24, for example, you shall declare this in the symbol table as well and change the data type from the default value 'DWord' to 'REAL'.

You recognize the hex representation in contrast to the integer representation because it is written with leading zeros.

If the often changing representation of the accu disturbs you you will be able to specify a permanent data format in [ViewOptions|PLC-Editor](#).

See also:

[STL](#)

[FBD](#)

[LAD](#)

[PLC-Editor](#)


4.7 Breakpoints and single step mode

By breakpoints you can interrupt the process of the PLC-program at any position. The process of the simulation is stopped then as well.

[Setting of a breakpoint in STL](#)

[Setting of a breakpoint in FBD/LAD](#)


After the program is stopped the CPU window is displayed and the LED at the bottom left becomes yellow. On the CPU window there are four possibilities to continue the process of the program:

1.) **Continue**: Here the process of the program is taken up normally again until the next breakpoint is reached. Instead of the button you can also use .

2.) **1 Network**: Here a network is processed and stopped after the last line has been processed. This is a little irritating in STL because one expect that the process is continued until the beginning of the next network. But a FBD- or LAD-network can only be displayed usefully in the [mode monitoring](#) after all instructions are processed. If the network contains a block call the process will be stopped prematurely so that you have the possibility to follow the process in the called block by 'Trace' (see below).

3.) **1 Step**: Here exactly one program line is processed. If the program line is a block call this block will be processed completely but not displayed. This button is not very useful in FBD/LAD.

4.) **Trace**: Even here the program is processed line by line, but at calling blocks the process is also done in the called block line by line.

If you click  the program will be processed until the end of the current scan and the simulation will be stopped. If a further breakpoint appears while processing that one will be ignored.

If the end of the scan is reached during the process of the program that is done step by step the machine simulation will be processed once, you recognize this by the marching on of the simulation time in the main window at the bottom left and by the short flashing of the green LED.

You **delete** a breakpoint in the same way as you have set it:
Click right and select the menu item 'Breakpoint'.

You can also specify [conditions](#) for the breakpoint: for this select 'Edit Breakpoint' after clicking right.

If you want to transfer a block into the PLC during the process that is done step by step (a transfer is also necessary for switching on/off of the monitoring mode) you will be told that this is not possible at the moment but that the block will be transferred automatically at the end of the scan.

See also:

[Speed-Trigger Monitoring](#)

Because of compatibility reasons to older versions there is also [another form of the breakpoint](#).

4.7.1 Set / delete a breakpoint in STL

You set a [breakpoint](#) by clicking the desired line in the STL-editor with the right mouse button and selecting 'Breakpoint' out of the context menu. The corresponding line is colored blue then.

If the process of the PLC-program reaches this point the block will be opened and the CPU-window appears. The process of the program is interrupted **before** the marked line is processed.

You edit an already set breakpoint by selecting 'Edit Breakpoint' out of the context menu.

You delete a breakpoint by selecting the line 'Breakpoint', which is checked, out of the context menu again.

If you have set the breakpoint in FBD/LAD it will be (invisible) below the last line of the network.

See also:

[Conditional Breakpoints](#)

4.7.2 Set / delete a breakpoint in FBD/LAD

You set a [breakpoint](#) by clicking the desired network with the right mouse button in the FBD- or LAD-editor and selecting 'Breakpoint' out of the context menu. A set breakpoint is displayed by a small blue box at the top left of the network.

If the process of the PLC-program reaches this point the block will be opened and the CPU-window will appear. The process of the program is interrupted **after** the network has been processed.

You edit an already set breakpoint by selecting 'Edit Breakpoint' out of the context menu.

You delete a breakpoint by selecting the line 'Breakpoint' which is checked out of the context menu again.

If you represent the network in STL later on the breakpoint will be (invisible) below the last line of the network.

See also:

[Conditional Breakpoints](#)

4.7.3 Conditional breakpoints

If the program is to be stopped just on specific conditions you will be able to set a conditional breakpoint or to supply an already set [breakpoint](#) with conditions afterwards. For doing this click the breakpoint with right in the STL-, FBD- or LAD-editor and select 'Edit Breakpoint'.

Please note that the condition in STL is evaluated before the process of the current line, but in FBD/LAD after the process of the whole network (see also [specific features in FBD/LAD](#)).

You can evaluate the RLO as follows:

- 1.) It shall not be considered, that means it will always be stopped if no condition is specified for the accu.
- 2.) It will only be stopped if the RLO is '1'.
- 3.) It will only be stopped if the RLO is '0'.
- 4.) It is only stopped at a rising edge of the RLO.
- 5.) It is only stopped at a falling edge of the RLO.
- 6.) It is stopped at every state change of the RLO.
- 7.) It will only be stopped if the RLO does not make any state change.

As a reference value for 4.) - 7.) the RLO is always valid like it has been at the last process of the line with the breakpoint. So 'Rising Slope' does not mean that the RLO goes from nought to one right now but that it was nought in the last scan and now one.

You can compare the accu 1 with a reference value. Doing this you have to specify how the content of the accu shall be interpreted: as INT, DINT or REAL.

4.7.4 Specific features in FBD/LAD

For an effective use of [conditional breakpoints](#) in FBD/LAD you have to note some points:

The RLO and the accu 1 are evaluated like they were found after process of the last line. That means just the last operation that changes the RLO resp. the accu decides whether the condition is fulfilled. If there are ambiguities it is recommendable to look at the network in STL for a short time.

If you have programmed a not evaluated SR/RS-flipflop, the value of the lower input decides about the RLO. But mostly one is interested in the value of the flipflop itself. In this case an assignment at the Q-output shall be programmed. Basically this is valid for all elements with a Q-output.

If you want to query a condition which is not in the RLO anymore at the end of a network, e.g. the upper input of a SR/RS-flipflop, so you have to look at the network in STL and set the breakpoint at the corresponding position. You can switch over to FBD/LAD but you have to note that just the signals are displayed correctly in the [monitoring mode](#), which were executed before the reaching of the breakpoint. Click '1 Net' in the CPU-window, then the process is continued until the end of the network and all signals are displayed correctly.

See also:

[Breakpoints](#)

4.8 Rewire

Rewiring is possible in the [address table](#) or, when all PLC- Windows are closed, by **PLC|Rewire**.


When an address is changed in the address table, all corresponding addresses in the PLC program are changed automatically. Before this is done you are asked to confirm the rewiring. By selecting the check box Do not ask further before confirming this question can be suppressed. At [View|Options|PLC-Editor|Rewire](#) you can re-activate the question.

When rewiring it must be sure that the new address is not already used in the program, otherwise two different addresses are united. If an address is already in use can be checked by browsing the [cross reference](#).

4.9 Cross reference list

In the cross reference list all operands and the symbol that belongs to it with the position of their occurrence in the program and the operation that is executed with them are listed. By double clicking a line or with 'Enter' the editor jumps to the corresponding position. If you realize that this is not the program position searched by you you will be able to close the block by a simple press of the button 'Esc'. The cross reference list is always sorted by operands. You have no influence on the sorting order.

There are 5 ways to reach the cross-reference list:

1. By the menu item **PLC|Cross-References**
2. By double clicking the status line (below) of the editor window while the block is open. If the cursor is on an operand the corresponding area is displayed in the cross reference list automatically.
3. Click right in the address table and select 'Cross References' out of the context menu.
4. If you have opened the edit mask of an element with PLC-connection you will see the symbol  in the [interface-panel](#). One click and you see immediately on which positions in the program the element is executed.
5. In STL, LAD or FBD you click with right the operand and select 'Cross Reference' out of the context menu.

The cross reference list can be printed by **Project|Print**.

By the [context menu](#) you reach the [symbol table](#) as well and you can check the elements that are connected to the operands in the [address table](#).

See also:

[PLC-Editor](#)

4.10 IEC-Functions

(Still in developing)

In the main directory of TrySim there is a subdirectory. 'IECFuncs'. That one contains preproduced functions which are needed to manipulate, compare and convert STRINGS, data formats and so on. In addition there are the headers of the functions which are in the Siemens-library 'TI-S7 Converting Blocks'.

You will have to import these functions into your project and if nec. rename them with **Project|Data**

Adding if the FC numbers collide with self programmed FCs.

Until now only a few of these functions are actually executed, most of them have only got the declaration. If you need some of these functions please ask us whether we have already finished them.

Already created IEC-functions:

[SFB3 \(TP\)](#) Time as Pulse

[SFB4 \(TON\)](#) On-delay

[SFB5 \(TOF\)](#) Off-delay

Already created TI-S7 functions

[FC84 \(ATT\)](#) Input for FIFO / LIFO

[FC85 \(FIFO\)](#) First In -> First Out

[FC87 \(LIFO\)](#) Last In -> First Out

See also:

[Special Functions and -Function Blocks](#)

[PLC-Editor](#)

4.10.1 SFB 3 (TP) Time as pulse

This IEC block corresponds essentially to the [S7-time-as-pulse](#). But it has to be loaded with T#..., not with S5T#.... In addition the time runs forward at the output ET, not backward.

See also:

[IEC-Functions](#)

4.10.2 SFB 4 (TON) On-Delay

This IEC block corresponds essentially to the [S7-on-delay](#). But it has to be loaded with T#..., not with S5T#.... In addition the time runs forward at the output ET, not backward.

See also:

[IEC-Functions](#)

4.10.3 SFB 5 (TOF) Off-Delay

This IEC block corresponds essentially to the [S7-off-delay](#). But it has to be loaded with T#..., not

with S5T#.... In addition the time runs forward at the output ET, not backward.

See also:

[IEC-Functions](#)

4.10.4 FC 87 (LIFO) Read out the youngest value of a table

With this you can read out the youngest value of a table that is added to it by the function [FC 84 \(ATT\)](#). Doing this the value is taken off the table.

The values in these tables are always 16 bit wide.

See also:

[FC 85 \(Fast In First Out\)](#)

[IEC-Functions](#)

4.10.5 FC 85 (FIFO) Read out the oldest value of a table

With this you can read out the oldest value of a table that is added to it by the function [FC 84 \(ATT\)](#). Doing this the value is taken off the table.

The values in these tables are always 16 bit wide.

See also:

[FC 87 \(Last In First Out\)](#)

[IEC-Functions](#)

4.10.6 FC 84 (ATT) Add a value to a FIFO/LIFO table

With this you can add a value to a table that can be read out (and deleted) with the functions [FC85 \(First In First Out\)](#) or [FC 87 \(Last In First Out\)](#) again after that.

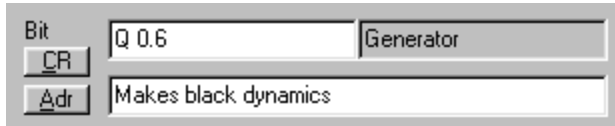
The values in these tables are always 16 bit wide.

See also:

[IEC-Functions](#)

4.11 Interface-panel


This is how TrySim names the groups which can be found in the [edit masks](#) of the elements with which the connection to the PLC is built up.




The operand can be entered into the little white field either in absolute spelling or as symbol. Names out of the declaration part of blocks that start with '#' are not allowed. I, Q and M as well as the point can also be entered by the [number block](#).

There are interface panels for bits, words and DWords, each panel only accepts operands that (at least in the size) correspond to its type.

In the big white field the comment out of the symbol table is displayed. You can also edit it here. If you have not entered the operand into the symbol table yet this will happen automatically by entering a comment. You cannot assign new symbols or change existing ones in the interface panel, for that you have to open the [symbol table](#).

With the button  you jump to the [cross reference list](#), in which all uses of the operand in the PLC-program are displayed. By a double click or by the [context menu](#) you go from the CR to the desired program position.

With the button  you jump to the [address table](#) in which all further uses of this address in the machine are listed. Please have a look at the points of the context menu as well.

See also:

[Data types](#)

[Naming inputs, outputs and memory bits](#)

[Naming data in datablocks](#)

4.12 Symbol table

PLC|Symbol table

Abbreviation:



In the symbol table you can assign a symbol, a data type and a comment to the absolute operands. Please do not confuse the symbol table with the [address table](#), there are all addresses used by the

machine summarized.

You can sort the symbol table by absolute operands as well as by symbols. To do this, click on the heading of the appropriate row. When clicking left, the table is sorted ascending. When clicking right you can choose, if the table is to be sorted ascending or descending.

The table is printed with **Project|Print** when it is active. The table is printed in the order it is currently sorted.

See also :

[Export the Symbol table](#)

[Import the Symbol table](#)

4.13 System function blocks and functions

These blocks fulfill manifold special tasks like STRING and date process, communication and so on. They are not exported by STEP®7.

Most of the system function blocks and -functions are not integrated in TrySim, that means, should the occasion arise, that you have to program them yourself so that the rest of your program works. Before you do this you should ask us whether the blocks needed by you are already programmed by us.

In the main register of TrySim there is a subdirectory 'Special Blocks'. You should copy all special blocks written by you or programmed by us into this subdirectory. Using them they are taken over into the current project automatically.

Programmed until now:

SFC 0	Set the CPU-clock
SFC 1	Read out the CPU-clock
SFC 20	Copy memory areas
SFC 21	Fill memory areas with a pattern
SFC 22	Create a new DB
SFC 23	Delete a DB
SFC 24	Find out the length of a DB
SFC 64	Read out the system time in ms

See also:

[IEC-Functions](#)

STEP®7 is a registered trademark of the Siemens AG.

4.13.1 SFC 0 Set the CPU-Clock

With this system function the [clock of the CPU](#) can be set. For this you have to declare a local [DATE AND TIME](#)-variable first and set this to the desired date and time with the [IEC-function](#) FC3 afterwards.

The [SFC 1](#) can be used to read out the clock.

4.13.2 SFC 1 Read out the CPU-Clock

With this system function the [clock of the CPU](#) can be read out. For this you have to declare a local [DATE AND TIME](#)-variable first and analyze this afterwards.

The [SFC 1](#) can be used to set the clock.

See also:

[System Functions](#)

4.13.3 SFC 20 Block move

With this system function you can copy the content of any memory area into another one. The source area as well as the destination area are displayed as [ANY](#). Just the number of bytes that exist in the source area as well as in the destination area is transferred. The return value RET_VAL is, not like in a S7, always '0'.

If you specify source- or destination area explicitly by a variable declared as [ANY](#), and the source- or destination area is in the local data, you will have to note that the identification \$87 is specified in ANY, not \$86 because this is interpreted by the SFC20 as their own local data, and these do not exist.

See also:

[System Functions](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

4.13.4 SFC 21 Fill

With this system function you can fill any memory area with a pattern. The destination area as well as the pattern are specified as [ANY](#). The pattern is copied into the destination area as often as this one is written completely. If the size of the destination area is not an integer multiple of the size of the

pattern the last copy of the pattern will be cut off correspondingly. The return value RET_VAL is, not like in a S7, always '0'.

If you specify pattern- or destination area explicitly by a variable declared as [ANY](#), and pattern- or destination area is in the local data, you will have to note that the identification \$87 is specified in ANY, not \$86 because this is interpreted by the SFC20 as their own local data, and these do not exist.

See also:

[System Functions](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

4.13.5 SFC 22 Create DB

With this system function you can create a DB while runtime. The area of the number of the new DB you specify with the parameter LOW_LIMIT and UP_LIMIT. The number of the really created DB is given back by the parameter DB_NUMBER. By the parameter COUNT that has to be even you specify the size of the DB that shall be created.

The Out-parameter RET_VAL can have the following values:

```
W#16#0000: no errors
W#16#80A1: LOW_LIMIT = 0 or bigger than UP_LIMIT
W#16#80A2: COUNT = 0 or uneven
W#16#80B1: No DB created because no number free.
```

The other realized error cases in a S7 are not realized.

See also:

[Delete a DB](#)

[System Functions](#)

4.13.6 SFC 23 Delete DB

With this system function you can delete a DB while runtime. You specify the number of the DB that shall be deleted with the parameter DB_NUMBER. Please note that the DB is not opened. It must not have been open in one of the calling blocks either as the call of the currently executed block happened!

The Out-parameter RET_VAL can have following values:

```
W#16#0000: no errors
W#16#80B1: the DB does not exist
```

The other realized error cases in a S7 are not realized.

See also:

[Create a DB](#)
[System Functions](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

4.13.7 SFC 24 Information about DB

With this system function you can find out the length of a DB while runtime. You specify the number of the DB with the parameter DB_NUMBER. The DB has to exist in the PLC, it will not be enough if it is only on the hard disk. The DB will be transferred into the PLC automatically if it is either opened in the editor once or if it is opened by the PLC program. The length (rounded up to the next even number) is given back by the parameter DB_LENGTH. The return value WRITE_PROT is, not like in a S7, always zero, because in TrySim DBs cannot be declared as write-protected.

The Out-parameter RET_VAL can have following values:

```
W#16#0000: no errors  
W#16#80B1: the DB does not exist
```

The other realized error cases in a S7 are not realized.

You can also get to know the length of the currently opened DB or of the current instance DB with [DBLG](#) or [DILG](#).

See also:

[System Functions](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

4.13.8 SFC 64 Read out the system time in ms

This system function only has got one out-TIME-parameter where the system time is displayed in ms.

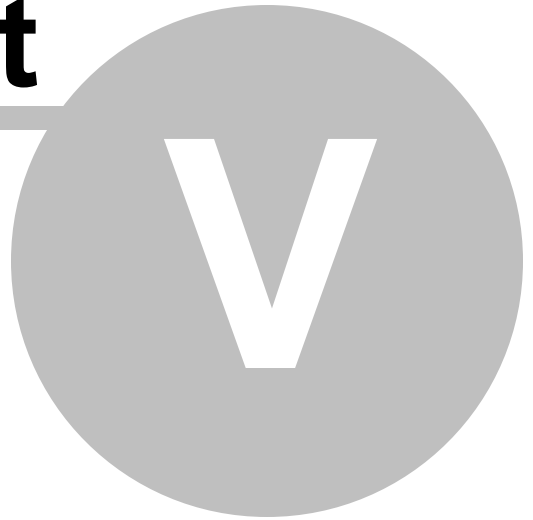
Please note that in a S7 this time starts again with zero after about 48 days so that unexpected results can occur when calculation a time difference. At the moment TrySim can only simulate 24 days so that you are not able to test this effect with TrySim.

See also:

[SFC1 Read Out the Clock](#)
[System Functions](#)

S7-300 and S7-400 are registered trademarks of the Siemens AG.

Part



5 Export and Import

5.1 Overview

TrySim is a real offline-software. If you want to let run a program that is created by TrySim on a real PLC it will only be possible with the original development system of the manufacturer. Until now TrySim supports only the S7-300/400 series of Siemens.

Export to STEP®7

You can export the program and the [symbol table](#) to STEP®7, go on processing it there and transfer it afterwards into a real PLC.

[Export the Program](#)

[Export the Symbol Table](#)

[Problems with Export](#)

Import from STEP®7

You can import a program and the symbol table which you have written in STEP ®7 to TrySim and test it there. If you do changes you will be able to reexport it afterwards to STEP ®7.

[Import the Program](#)

[Import the Symbol Table](#)

[Problems with Import](#)

Import from STEP®5

The import of STEP ®5 is not possible anymore.

See also:

[PLC-Editor](#)

STEP®7, STEP®5 S7-300 and S7-400 are registered trademarks of the Siemens AG.

5.2 Export to Step®7

For the export of the program you have to generate a source. A text file that contains the program is called 'Source'. You can edit a source with every text-editor.

- Select **Project|Export|Blocks**.
- Shift the blocks that are to be exported with the arrow buttons into the right list in the selection

mask. You can also shift all blocks at the same time. You do not have to think about the order of the blocks, it is specified automatically. You should not export [special function blocks and - functions](#). The files in the export filter (**Machine|Export|Edit Export Filter**) can be shifted to the right side but they do not create a source.

- Select 'Entire File', that is how you achieve that all blocks are written in one source. This file gets the name of your project with the extension 'STL'. The '*.STL' files are saved in the directory of your project. If there are already files with the same name these will be overwritten without warning.
- Start STEP ®7 and create a new project. Which steps are necessary for that is described in the documentation to STEP ®7. Open the box
Project|Station|CPU|S7-Program|Sources.
- Select the menu item **Insert|External Source**.
- Now you have to find the source file that you have created before.
The complete path is named (if you have accepted the defaults at installation)
C:\Programs\TrySim\Project\Your project.
- Having inserted you find the new source in the box that is named like your project. Open this source with a double click.
- Select the menu item **File|Translate** or **Insert|Translate**. If there are already blocks with the same name like the imported ones in the STEP ®7-project they will be overwritten without warning.
- If all goes well the message 'Compiler Result: 0 Errors, 0 Warnings' will be displayed in the lower window and your program is successfully exported to STEP ®7. If you get warnings these will mostly refer to the use of data words and they are normally harmless. In case you got error messages the translation had not been successful. For that please read the section [problems with export](#).

See also:

[Export the Symbol Table](#)

[Import the Program](#)

[Import the Symbol Table](#)

[Import and Export in General](#)

STEP®7 is a registered trademark of the Siemens AG.

5.3 Export of the symbol table

For the export of the [symbol table](#) there are no special steps necessary in TrySim because TrySim saves the table in a format that can be read by STEP ®7 immediately. But you should save the project so that also your last changes of the symbol table can be taken over.

- Start STEP ®7. If you have not created a project yet you will have to do this now.
- Open the symbol table in the box **Project|Station|CPU|S7-Program**.

The lines of the TrySim symbol table are added to the already existing ones in the STEP ®7 – table. If addresses or symbols can be found twice by that they will be in bold type and you have to delete

them separately by hand or rename them. If your list in TrySim contains all needed symbols you shall delete all lines in the STEP ®7 – table before importing. Select **Edit|Select|All** and press the key ‘Del’ afterwards.

- Select the menu item **Table|Import**.
- Select the file type ‘ASCII – Format (*.ASC)’.
- The symbol table of TrySim is called ‘SYMLIST.ASC’ and is in the project directory. The complete path is named (if you have accepted the instructions of the installation) C:\Programs\TrySim\Projects*Your Project*.
- Select the file and click ‘Open’.
- Now all lines of the TrySim-symbol table should appear in the table.

See also:

[Import the Symbol Table](#)

[Export the Program](#)

[Import the Program](#)

[Import and Export in General](#)

STEP®7 is a registered trademark of the Siemens AG.

5.4 Problems with the export

If you get error messages while translating in the STEP ®7 source editor these will be type conflicts in most of the cases. Errors will also occur if you have programmed recursively, that means that a block calls itself directly or indirectly. You cannot get rid of this error, recursiveness is not permitted in STL-sources. Finally there can also be an error in TrySim itself. Sometimes STEP ®7 criticizes also type conflicts that are not existing.

In every case you should deselect the check box ‘Create Blocks only with correct Translation’ in **Options|Customize|Source Files** while translating the source. Often it is possible to translate the source at the second or third try because more and more of the called blocks are existing. This multiple translating will only be necessary if you export a program to STEP ®7 for the first time. If you do changes in TrySim later and export the program again all blocks will already exist and there will be no error messages. Often it is useful to import the symbol table in STEP ®7 first because some instructions can only be displayed symbolically.

Type conflicts

These result from the fact that in TrySim the data type check is not that strict as in STEP ®7. In TrySim you can connect all actual-parameter which have got the same size as the formal-parameter to function blocks. But STEP ®7 accepts only the actual-parameter which have got exactly the same type as the formal-parameter. So the transfer of an variable that is declared as INT to a parameter declared as WORD is not permitted although both are 2 bytes big. So if the STEP ®7 – compiler reports type conflicts you will have to check the types in TrySim and adjust them correspondingly.

By the way, this has got the advantage that the accu-representation can be better selected while monitoring STL-blocks.

Recursive Calls

If you call a FB you will have to specify the instance DB. This one is created correspondingly to the FB header and so it has to be in the source after the FB declaration. Because the source compiler of STEP ®7 does not create the instance DB automatically (like normally in the editor) the instance DB will not exist yet if a FB calls itself. That is why recursive calls are not translateable in STL-sources. Please avoid that a block calls itself what is also not permitted according to IEC 61131-3.

Error in TrySim

Finally it is also possible, unfortunately, that we have made a program error and the right syntax is not created in the source. If you have a good command of the source syntax you will be able to get rid of the error in the source, if not you have no choice but to find the faulty position in the source editor (double click the error message), identify the corresponding block and generate the source in TrySim again, but without the faulty block. You have to enter this one by hand before translating in STEP ®7. If such an error occurs please contact us so that we can get rid of it in the next release of the program. Here is our [e-mail-address](#).

See also:

[Export the Program](#)

[Export the Symbol Table](#)

[Import and Export in General](#)

STEP®7 is a registered trademark of the Siemens AG.

5.5 Import from Step®7

To import a program you have to generate a source in STEP ®7 first. A text file that contains the program is named source. You can edit a source with every text editor.

- Start STEP ®7
- Open the project
- Start the LAD/STL/FBD-editor, for example by opening any block. Then you have to close the block again because sources cannot be created of opened blocks.
- Select the menu item **File|Generate Source**.
- Enter any name for the source that shall be created in the edit mask.

Select the blocks that shall be imported to TrySim; you do not have to think about the order. (See also: [import-filter](#)).

- Select 'Addresses: Absolute'
- Confirm with OK
- Start TrySim
- Open the project

- Select **Project|Import|Blocks|S7-STL-Source**
- Now you must find the STEP ®7 project. Presumably it is in C:\Step7_Vx\S7proj (a file is opened by a double click).
- Open the file with the name of your project.
- Then TrySim opens the file S7ASRCOM automatically. If this does not happen because of any reason you will have to do it yourself.
- In this file there are one or several files which are called 00000001, 00000002 and so on. In one of these files there is your STL-source.
- If you have opened the 0000000x - file the available STL-sources will be displayed in the right window. Open the desired one with a double click or mark it and confirm with OK.
(Remark.: In this file the sources are not saved under the name which you have entered but they are numbered corresponding to their creating date: '0000000x.stl', and x is a hex-number, that means the tenth entry is called '0000000a.stl', the eleventh '0000000b.stl' and so on. If TrySim cannot translate the coded names to the correct names you will have to select the file with the biggest number that is the one which you have just created.)
- If everything went right the message 'Import Successful' appears, otherwise a list with the blocks which could not be compiled will appear. See section [problems with import](#).

See also:

[Import of Symbol Table](#)

[Export of Program](#)

[Export of Symbol Table](#)

[Import and Export in General](#)

STEP®7 is a registered trademark of the Siemens AG.

5.6 Import-filter

Testing bigger programs it can happen that only parts of the machine of a simulation are accessible with TrySim. In these cases there are blocks in your program that is written in the STEP ®7 system that are needed in TrySim in a modified form. You can define an import filter that avoids an import of these blocks so that these blocks are not overwritten while importing the whole program.

Select **Project|Import|Blocks|Edit Import Filter**.

A typical example is a PLC with a counter module for position detecting. Here there are standard blocks by Siemens which handle the communication with the module. In the end these blocks have only got the job to save the position of the moved part at a specific position in the data memory. The (modified) block has to do exactly the same in TrySim as well so that the rest of the program works.

In this case you have to proceed like this:

- 1.) Import the standard block once, so that the interface is known in TrySim.

2.) Modify the code in the block so that it has got the desired functionality. Normally no code exists after the import because the blocks are declared as know-how-protected.

3.) Include the block into the import filter so that your changes are not overwritten by mistake at the next import of the program.

If you have got problems modifying such a block please contact us.

If you need the changed block in the following projects as well you will be able to add it with **Project|Add Files** to the new program.

See also:

[Import the Program](#)

STEP®7 is a registered trademark of the Siemens AG.

5.7 Import of the symbol table

For the import of the symbol table no precautions are necessary in TrySim because STEP ®7 can create the data format used by TrySim. If you have already created symbols in TrySim these will be overwritten by the import. If you want to avoid this you will have to export the TrySim table to STEP ®7 first, by this both lists are connected with each other.

- Start STEP ®7
- Open the symbol table
- Select **Table|Export**
- Select the file type 'ASCII Format (*.ASC)'
- Select 'Path name\Symlist.asc' as file name.
The complete path is named (if you have accepted the defaults at installation)
C:\Program files \TrySim\Projects*Your Project*.
- To the question whether you want to overwrite the file you answer with 'Yes'.
- The symbol table is available in TrySim now.

If you export the symbol table out of STEP ®7, while TrySim is already running, you will have to execute in TrySim **Project|Import|Symbol Table** with the symbol table in the TrySim project directory, because the symbol table is only read at the start of the program.

See also:

[Export the Symbol Table](#)

[Import the Program](#)

[Export the Program](#)

[Import and Export in General](#)

STEP®7 is a registered trademark of the Siemens AG.

5.8 Problems with the import

If you have seen to it that you have clicked ‘Operands Absolute’ while ‘Generating Source’ in STEP ®7 errors are usually attributed to the use of special function blocks and -functions. Sometimes incompatibilities with already existing blocks can be the cause of errors as well. Recursive block calls lead to errors which you cannot get rid of. Some constructions are not implemented in TrySim in this version and causes errors and warnings while importing. Finally an error can be in TrySim itself.

The importer of TrySim is only tested with sources that were generated by the Siemens SIMATIC ® -Manager. If you use another system and errors will occur, please [mail us](#) the error creating source code so that we can correct TrySim correspondingly.

Symbolical Representation of the Operands

If you have kept the default ‘Addresses: Symbolic’ while ‘Generate Source’ TrySim will not be able to import the source. Do this process again with ‘Addresses: Absolute’. But you should import the symbol table first in every case.

Special Function Blocks and –functions and Know-How-Protected

These blocks that are used in STEP ®7 to head for function modules and for special tasks will not be exported by STEP ®7. In the directory [special blocks](#) the declarations of all SFBs and SFCs are saved. They will be copied into your project automatically if you call them.

You should not export blocks that are declared as Know-How-Protected to TrySim. To avoid this you can use the [import filter](#).

Incompatibilities with available Blocks

If there are already blocks existing in your project which have got a differing declaration to the imported ones import errors may occur sometimes. Try to identify the error causing blocks by the help of the import log which is displayed automatically. In this cases delete the corresponding blocks and repeat this process.

Recursive Block Calls

You can program FBs in STEP ®7 which call themselves directly or by another block. A source that is created by such a program often cannot be retranslated. So avoid recursions please.

Not implemented Operations and Constructions

See: [Not implemented features](#).

Error in TrySim

If none of the other error sources is there it will be possible that a programming error by us is the cause of the problem. In this case you can only try to change the program parts that cause the import

error so that TrySim understands the source. Please contact us in such cases so that we can get rid of the errors. Here is our [e-mail-address](#).

See also:

[Import the Program](#)

[Import the Symbol Table](#)

[Problems with Export](#)

[Import and Export in General](#)

STEP®7 and SIMATIC® are registered trademarks of the Siemens AG.

5.9 Import from STEP®5

The import of STEP ®5 is not possible anymore.

STEP®5 is a registered trademark of the Siemens AG.

5.10 Import from STEP®5 Assignment lists

The import of STEP ®5 assignment lists is only limited possible.

- Select: **Project|Import|Symbol Table**
- Select for the file type: *.seq

Only the inputs, outputs, markers, timers and counters are imported. Space lines, comment lines, ‘;’ and pagings ‘.PA’ are ignored. The data type is selected correspondingly to the operand as BOOL, BYTE, WORD, DWORD, TIMER or COUNTER.

STEP®5 is a registered trademark of the Siemens AG.

Part



6 Example session

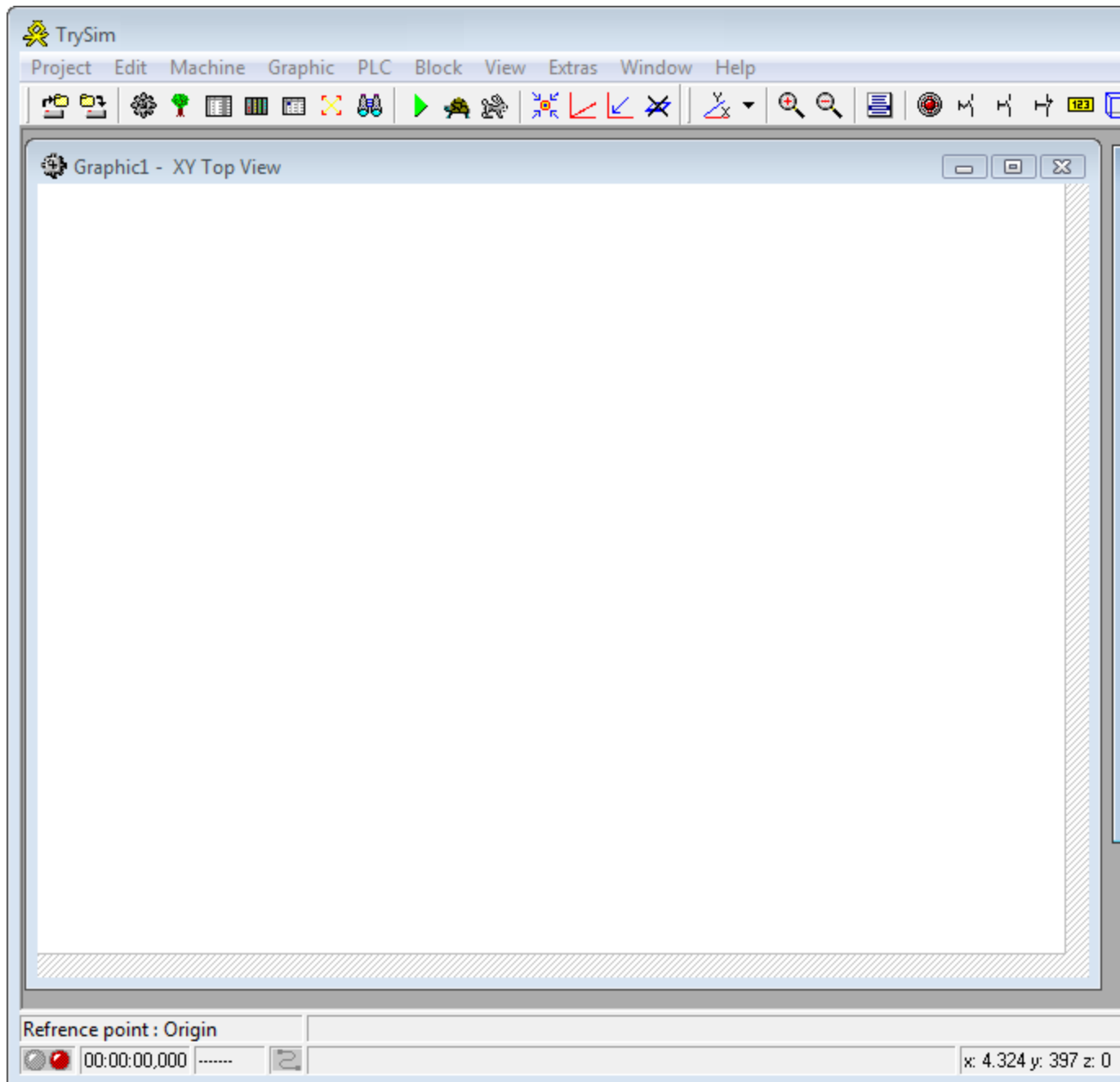
6.1 Example session

In this example session a very simple machine will be installed to familiarize you with the basics of TrySim. The task is making a wagon go forwards and backwards between two limit switches.

At the end of each section we will show you how your screen should look like.

Start TrySim. Select **Project|New**. An empty graphic window appears and a selection list with available elements.

[Continue](#)



[Continue](#)

6.2 Creating the machine

1.) Generate linear mover


Select the tabsheet 'Actuators' and generate a linear mover by catching the corresponding symbol with the mouse (keep pressing the left mouse button). Move the mouse about to the middle of the

graphic window and release the mouse button.

2.) Generate box

Just do the same thing in order to generate a box (tabsheet Misc). Position it a bit above the linear mover.

Now you have to determine that the box shall be fixed to the linear mover.

Therefore open the element tree . Click on the box and keep the left mouse button pressed. Then drag the box in the element tree onto the linear mover.

Open the edit mask of the linear mover by clicking on the line with the right mouse button. Please note that you click not before the cursor has been changed into a hand. Shift the edit mask aside so that you can see the linear mover and the box.

Shift the slide controller beneath the field *Hotspot* a little to the right. This will cause the hotspot of the linear mover and the box to move to the right. Close the edit mask by *OK*.

3.) Generate limit switch

Generate a limit switch (tabsheet Sensors). Position it a little to the left near the box. Make sure that it will be as near to the box that it will still reach it when the linear mover moves to the left.

Open the edit mask of the limit switch by clicking on the right mouse button. Click on the arrow near the input field *Master* (tabsheet VAR) and select the box. That is how the limit switch will know that it should be activated by the box. Enter "limit switch left" into the field *Name*.

Do just the same thing in order to generate another limit switch, which you position right beside the box. Name it "limit switch right".

In case the labels disturb you, drag them to another position by the left mouse button, or open the edit masks and inactivate the check box *Label*.

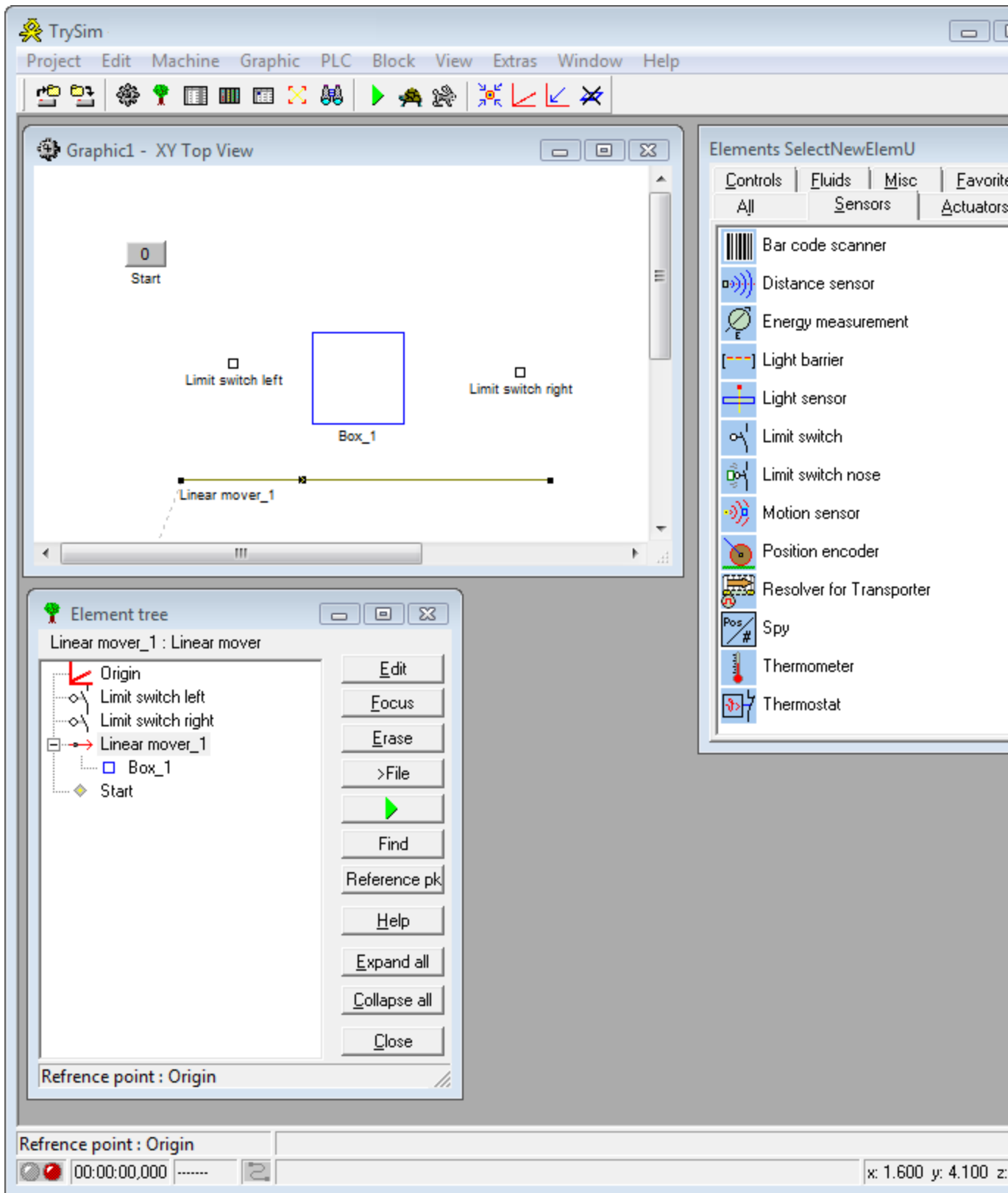
4.) Generate start-button

Generate a NO-pushbutton (tabsheet Controls) and place it somewhere in the window. Open the edit mask and name it "Start".

5.) Save

Now the machine is completed and you should save it: **Project|Save all**. You can either name the project yourself or accept the proposal "Project01".

[Writing the PLC-Program](#)

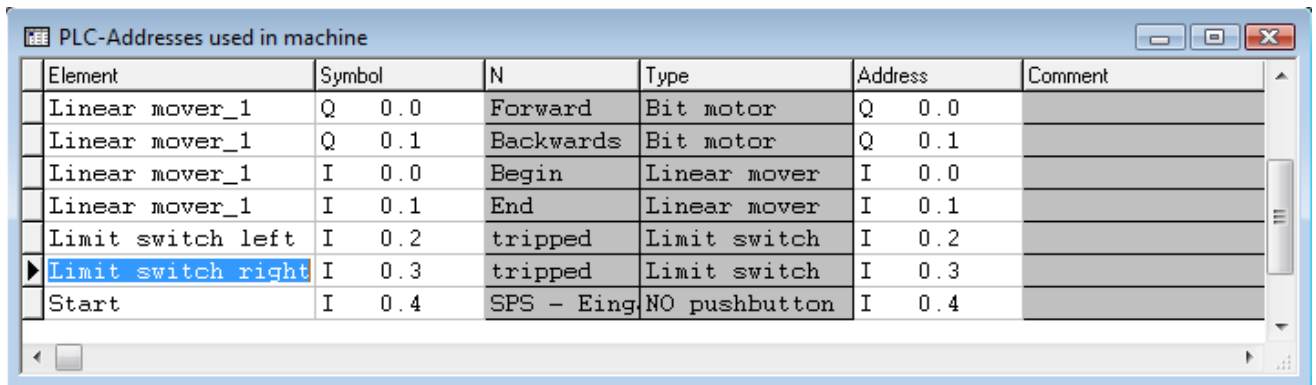


[Writing the PLC-Program](#)

6.3 Writing the PLC-program

TrySim automatically assigns PLC-addresses when creating new elements. If your program shall run at a real PLC later, you have naturally to adapt these addresses.

In order to look at all the given addresses clearly arranged and to alter them, if necessary, select **Machine|Address table**. If you have followed the order when generating the elements, this list will look as follows:



Element	Symbol	N	Type	Address	Comment
Linear mover_1	Q 0.0	Forward	Bit motor	Q 0.0	
Linear mover_1	Q 0.1	Backwards	Bit motor	Q 0.1	
Linear mover_1	I 0.0	Begin	Linear mover	I 0.0	
Linear mover_1	I 0.1	End	Linear mover	I 0.1	
Limit switch left	I 0.2	tripped	Limit switch	I 0.2	
Limit switch right	I 0.3	tripped	Limit switch	I 0.3	
Start	I 0.4	SPS - Eing	NO pushbutton	I 0.4	


1.) Open OrganizationBlock 1

Open the OB 1 (OrganizationBlock1) by **Block|Open**. You can also click onto the Symbol .

2.) Generate network for 'box forwards'

Generate a SR-flipflop by clicking the symbol (at the top right)  within the toolbar.

Enter the address Q 0.0 into the field directly above the flipflop. The corresponding edit field will automatically be active at the moment you enter a letter. You can also activate it explicitly by *ENTER*. Q 0.0 means *Motor forwards*. After the input you have to close the editing window by *ENTER*.

Put an Or-box in front of the S-input. Therefore click on the ??-name in front of the input, or move the cursor by the arrow keys. Then select the symbol  out of the toolbar.

Name the inputs of the Or-box I 0.2 (this is the limit switch on the left) and I 0.4 (this is the pushbutton *Start*).

Name the R-input with I 0.3 (this is the limit switch on the right). Instead of entering the text I 0.3, you can also click on the limit switch and after this on the operand ??.

Name also the assignment at the Q-output of the flipflop with Q 0.0, or delete it by the del-key.



3.) Generate network for 'box backwards'

Change into the following network by *page-down-key*, or click on the twin-arrow indicating below at the right side of the network-window.


Generate a further SR-flipflop Q 0.1 (*Motor backwards*).

Name the S-input with I 0.3 (limit switch on the right) and the R-input with I 0.2 (limit switch on the left) and delete the assignment at the Q-output.

4.) Download program to the PLC

Download the program to the PLC by clicking on the symbol  near the  on the toolbar.

5.) Save

The program is now finished, and you should save it by **Project|Save all** or .

[Start the simulation](#)

TrySim - Professional - C:\Users\Julia\Documents\TrySim\Projekte\Project01

Project Edit Machine Graphic PLC Block View Extras Window Help

PLC-Addresses used in machine

Element	Symbol	N	Type	Address	Comment
Linear mover_1	Q 0.0	Forward	Bit motor	Q 0.0	
Linear mover_1	Q 0.1	Backwards	Bit motor	Q 0.1	
Linear mover_1	I 0.0	Begin	Linear mover	I 0.0	
Linear mover_1	I 0.1	End	Linear mover	I 0.1	
Limit switch left	I 0.2	tripped	Limit switch	I 0.2	
Limit switch right	I 0.3	tripped	Limit switch	I 0.3	
Start	I 0.4	SPS - Eing	NO pushbutton	I 0.4	

OB 1 (OB 1) Network 1

Address	Declaration	Name	Type	Comment
0.0	temp	OB1_EV_CLASS	BYTE	
1.0	temp	OB1_SCAN_1	BYTE	
2.0	temp	OB1_PRIORITY	BYTE	

1/2 OB1 =

Reference point : Origin


00:00:00,000 Linear mover_1 : Linear mover x: 1.600 y: 4.100 z:

Note: This of course is not a nice program, for instance the outputs 'motor forwards' and 'motor


backwards' are not latched against each other. But it is easy to write and that was our priority here.

[Start the simulation](#)

6.4 Starting the simulation

Change again into the machine by clicking the machine-window, or select the gearwheel  on the toolbar.

1.) Start the simulation

Now start the simulation by clicking on the -symbol. Now the machine simulation and the PLC-program will be executed alternatively.


2.) Start the simulated machine

Click on the Start-button you have generated. Now, when all is done according to plan, the box should go to the right until it reaches the limit switch, then turn to the left limit switch, and go on forwards and backwards continually.

If the wagon does not move back and forth, see here: [Possible errors in the example](#)

3.) Adjustment of the start and stop ramps



When activated, the limit switches will turn red, which you can hardly detect, as the wagon will immediately move to the other direction again. You can take preventive measures by prolonging the ramp time of the motor:


Stop the simulation by clicking on . You can also stop by clicking anywhere on the white of the window with right mouse button.

Open the edit mask of the linear mover by clicking right (the cursor must have already the form of a hand when clicking) and click on the button *Drive*. Now the edit mask of the motor of the linear mover is opened. Enter *2 s* into the field *Duration* (tabsheet Ramps). Confirm 2 times by OK and start the machine again. Now you can see clearly how the limit switches will change their color. In case the wagon moves too slowly, click on the rabbit in order to accelerate the procedure.

[Monitor program execution](#)

6.5 Watch the program execution

Change again to the program by clicking on a visible part of the OB1-window, or click on the PLC-symbol  next to the gearwheel .



Click on the eye  in order to activate the monitor mode.

Now you can observe the state of the in- and outputs.

Change into the first network by page-up-key and shift the window, so that your button *Start* and the corresponding input at the Or-box is to be seen at the same time. If you press the button with the left mouse key, you will see how the input and the Or-box will turn red.

[Possible errors in the example](#)

6.6 Possible errors in the example

- Nothing happens:
 - Did you consider the notification of changes at the address table?
 - Did you start the simulation? (green LED in the lower left)
 - Did you transmit the program into the PLC)
 - Did you create the program as described?
 - If you have found and fixed an error in the PLC-program, then you have to retransmit the program!
- The HotSpot of the linear mover is moving but the box does not move:
 - You did not fix the box to linear mover.
 - Open the element tree  and move the box with the mouse onto the linear mover
- A box cannot reach a limit switch:
 - Stop the simulation by clicking the symbol  or click anywhere on the white of the window with the right mouse button.
 - Catch the limit switch with the mouse (keep pressing the left mouse button) and move it to the desired position.
- A limit switch moves together with the box and so it can never be reached:
 - You have fixed the limit switch to the box or the linear mover by accident.
 - Open the edit mask of the limit switch, and choose the origin as *Father*, so the limit switch is fixed to the origin, which cannot be moved per definition.
- The box covers a limit switch, but this one does not turn red:
 - The limit switch does not know that it shall be activated by the box;
 - open the edit mask of the limit switch and select the box out of the selection list which will appear if you click the arrow next to the field *Master*.
- The box covers a limit switch, this one turns red, but the box does not turn back:

Probably you made a mistake in the program. Check:

A 0.0 is set from E 0.2 or E 0.4

A 0.0 is reset from E 0.3

A 0.1 is set from E 0.3

A 0.1 is reset von E 0.2

If you have found and fixed an error in the PLC-program, then you have to retransmit the program!

End of the example session

Part



7 Menu items

7.1 Project

Here all functions are listed which run in many uses under “File”.

Some of the points have got changing meanings in dependence on the active window: “Print”, for example, always prints the object in the current window.

7.1.1 Project | New

By this you create a new machine which has no elements except of the origin and no PLC program blocks.

7.1.2 Project | Open

With this you open an already existing project. You do not have to close the current project, it will be closed automatically, and if you have done any changes you will be asked whether they shall be saved. Note that even letting run the simulation means a changing because the position of the moveable elements can be changed.

Projects of TrySim are always saved in a directory. That means that you do not have to find a special project data but the directory in which your project is in.

7.1.3 Project | Open again

Here all projects which were opened last are displayed.

7.1.4 Project | Save all

- The elements of the machine are saved with their current positions.
- All program blocks which are open are saved.
- All data blocks are saved with their current values.

Abbreviation:  You can only use the function key F2 if none of the program windows is

active, because here F2 has the meaning “And-Box” or “New NO-contact”.

7.1.5 Project | Save as

Here you have to enter a directory in which your project shall be saved. If the directory does not exist it will be created automatically. TrySim sets up an own file for each project with the name of the project. You can also save other informations in this directory, but you should not delete any data which were created by TrySim.

7.1.6 Project | Add files

This menu point is nearly the same as import. It is useful above all if you need program blocks out of projects which already exist in the current project as well. You can also copy the corresponding files on windows level.

7.1.7 Project | comment

The project comment is opened with **Project|Comment**. You can edit the comment with any editor that can read and write the rich text format “*.rtf”, e.g., MS-Word. Or you can save an existing project specification as **PrjKom.rtf** in the TrySim directory of the project. Take care to specify the “Rich Text Format”. With some editors it is not enough to change the extension into “*.rtf”.

See also:

[Block comment](#)

7.1.8 Project | Export

You can export:

- PLC-program blocks
- Symbol tables
- Machines

But the machine export is not a real transport, just the current TrySim machine is copied.

See also:

[Export of the program](#)

[Export of the symbol table](#)

[Export of the machine](#)

7.1.9 Project | Import

You can import:

- PLC-program blocks
- Symbol tables
- Machines

But the machine import is not a real import, just an existing TrySim machine is copied.

See also:

[Import of the program](#)

[Import of the symbol table](#)

[Import of the machine](#)

7.1.10 Project | Print

By this the object that is in the window which is active in that moment is printed.

This can be:

- The machine
- A program block
- A data block
- The symbol table
- The address table
- The 3-D view

It will always be printed with the current zoom, kind of display, arrangement in each case and so on.

It is not possible to print:

- Cross-reference list
- Element tree

If several program- and data blocks shall be printed, you are better to select [PLC | Print](#).

7.1.11 Project | Exit

With this you exit TrySim.

7.2 Edit menu

This menu contains different items, depending on what kind of window is just active. Particularly notable are those referring to the machine.

[Edit](#)

[Select all](#)

[Reverse marking](#)

[Fade Out](#)

[Fade out children](#)

[Fade In](#)

[Fade in with children](#)

7.2.1 Edit | Edit

Hereby the edit mask of the marked elements are activated. Same function as click short with right mouse button or click long and select “Properties”.

[Edit menu](#)

7.2.2 Edit | Select all

All elements are marked as selected, even if they are not visible.

If you call this function out of the [element tree](#) or the [address table](#), please make sure not to modify marked elements of the tree or the table before you perform the action. Both in the tree and in the table just one element can be marked.

[Edit menu](#)

7.2.3 Edit | Select all children

Hereby all elements that belong to the selected element are selected as well.

If you call this function out of the [element tree](#) or the [address table](#), please make sure not to modify

marked elements of the tree or the table before you perform the action. Both in the tree and in the table just one element can be marked.

[Edit menu](#)


7.2.4 Edit | Reverse marking

Hereby the current marking is reversed, this means that marked elements become unmarked and unmarked elements become marked. Please mind that this operation refers to all elements of the machine, also to these ones that are not visible in the current window.

If you call this function out of the [element tree](#) or the [address table](#), please make sure not to modify marked elements of the tree or the table before you perform the action. Both in the tree and in the table just one element can be marked.

[Edit menu](#)


7.2.5 Edit | Fade Out

Hereby all marked elements are suppressed in the top graphic window. The effect is the same as if you would open the element's edit masks and deselect the corresponding window in the graphic filter .

This function is especially useful while editing the current window in the element tree.

[Edit menu](#)

7.2.6 Edit | Fade out children

Hereby all children and children's children of the marked element are faded out in the top graphic window. The effect is the same as if you would open the children's edit masks and deselect the corresponding window in the graphic filter .

Please read also [hints to structuring huge Machines](#) under **Machine | Editing the machine | Groups | Structuring huge Machines**

[Edit menu](#)


7.2.7 Edit | Fade in

Hereby the marked element is shown in the top graphic window. You can archive the same effect utilizing the graphic filter on the editing mask.

This function is especially useful while editing in the element tree.

[Edit | Edit](#)

7.2.8 Edit | Fade in with children

Hereby all marked elements together with their children and children's children are faded in in the top graphic window. The effect is the same as if you would open the edit masks of all elements concerned and deselect the corresponding window in the graphic filter .

Please read also [hints to structuring huge Machines](#) under **Machine | Editing the machine | Groups | Structuring huge Machines**

[Edit menu](#)

7.2.9 Edit | Focus

Hereby the marked element is shown in the center of the top graphic window. To edit it press Enter.

Please read also [hints to structuring huge Machines](#) under **Machine | Editing the machine | Groups | Structuring huge Machines**

[Edit menu](#)

7.3 Machine

In this menu you find everything you need to create and modify the machine. As well you can start and stop the machine here and you can change the speed of the simulation.

[Menu PLC](#)

[Menu Window](#)

[Menu Project](#)

[OPC-Server](#)

7.3.1 Machine | Start

Abbreviation:  -Symbol in the toolbar

By this the edit mode is stopped and the simulation is started. Now the machine is simulated and then the PLC-program is executed alternately. By the first start of the simulation in the TrySim session the start block OB 100 is executed before the OB1.

Many first-time-users of TrySim have to get used to the existing of two start- and stop-buttons. With the buttons in the toolbar or by the menu you can start and stop the simulation. With the buttons which you create yourself the simulated machine can be stopped and started.

In the [single step mode](#) the execution of the PLC-program is continued till the next breakpoint by this menu point.

7.3.2 Machine | Stop

Abbreviation:  -Symbol out of the toolbar

By this the simulation is stopped and the machine is switched in the edit mode. You can edit and download the PLC-program while the machine is running.

Many first-time-users of TrySim have to get used to the existing of two start- and stop-buttons. With the buttons in the toolbar or by the menu you can start and stop the simulation. With the buttons which you create yourself the simulated machine can be stopped and started.

The simulation will also be stopped and switched to the edit mode if you add a new element or if you click right anywhere in the white area of the graphic window.

In the [single-step mode](#) the PLC-program is executed by this menu point till the end of the current scan and the simulation is stopped then. If during the execution of the program another breakpoint is found this one will be ignored.

7.3.3 Machine | Slower

Abbreviation:  Turtle in the toolbar.

You can let run the virtual time of the simulated machine slower or faster. By this you can analyse

critical moments calmly while you let processes which run already correctly pass quickly. The precision of the simulation is not influenced by the slow motion or the quick motion, the simulation repetition rate keeps always to the value which is adjusted in **View|Options|Simulation**.

You can also change the simulation speed with the [Speed-Trigger](#) automatically.

7.3.4 Machine | Faster

Abbreviation: Rabbit in the toolbar.

You can let run the virtual time in the simulated machine slower or faster. By this you can analyse critical moments calmly while you can let processes which run already correctly pass quickly. The precision of the simulation is not influenced by the slow motion or the quick motion, the simulation repetition rate keeps always to the value which is adjusted in **View|Options|Simulation**.

You can also change the simulation speed with the [Speed-Trigger](#) automatically.

7.3.5 Machine | Real time

By this an eventually activated slow motion or quick motion will be deactivated, the virtual time in the status bar runs like you are used to.

The Short-Key F8 will not function if a FUP- or KOP-block is activated, there it means “New Connection ” resp. “Open Fork”.


You can also change the simulation speed with the [Speed-Trigger](#) automatically.

7.3.6 Machine | Extended

Here you can adjust the speed of the simulation into the steps 100: 1, 10: 1, 1: 1, 1: 10 and 1: 100.

7.3.6.1 Slow motion


You can let run the virtuell time of the simulated machine slowlier or faster. So you can analyse critical moments calmly, while you let processes which already run correctly pass by quickly. The precision of the simulation is not influenced by the slow motion or the quick motion, the simulation repetition rate keeps always the value which is set in [View|Options|Simulation](#).

You can also get the slow motion if you click (several times) on the turtle .

And you can change the simulation speed with the [Speed-Trigger](#) automatically.

7.3.6.2 Quick motion

You can let run the virtual time of the simulated machine slower or faster. By this you can analyse critical moments calmly while you can let processes which already run correctly pass quickly. The precision of the simulation is not influenced by the slow motion or by quick motion, the simulation repetition rate keeps always to the value which is adjusted in [View | Options | Simulation](#).

You can also get the quick motion by clicking (several times) on the rabbit .

Please understand the term/name “1: 100” as “As fast as possible”. TrySim itself cannot let even simple machines run with 50 times the speed (by a simulation rate of 100 milli sec) not even on fast computers.

You can also change the simulation speed with [Speed-Trigger](#) automatically.

7.3.7 Machine | Library

You can save parts of a machine which you need often in a library. For this you need to mark the matching parts and select **Machine|Library|Save as**. Loading a part of a machine out of the library the PLC-addresses are given provisionally, they have to be adjusted to your demands. The names of the inserted elements are marked with a _Nr to reach clarity.

This menu point is just available if one of the graphic windows is activated.

See also:

[Groups](#)

7.3.9 Machine | Media

**

In a TrySim project there can be up to 16 different media. They are described by name, density and viscosity. You will create a new medium either by using the [media manager](#) or, if you select ‘new medium’ out of the selection list for media on the edit mask of a container.

The fluids in TrySim are represented as a mixture out of these 16 media. Density and viscosity are calculated as the arithmetical mean value. If this is a simplification too heavy for your application, you will be able to use the [reactor](#) to create a new medium with the desired properties.

See also:

[Fluids](#)

7.3.9.1 Media manager

**

You reach the media manager either by **Machine|Media** or by the button ‘media manager’ on the edit mask of [containers](#).

The media manager is used to define new [media](#) and to edit or to delete them.

If you want to delete media you will have to call the media manager by **Machine|Media**.

See also:

[Fluids](#)

7.3.10 Machine | Dynamics | delete normals

By [generators dynamics](#) are created, and by [destroyers](#) you should delete them again after the run through the machine. But at the beginning nearly no dynamic is able to run through the machine without any trouble. Otherwise there would have been no reason to develop TrySim. Sometimes there are lost dynamics everywhere, and with this menu point you can delete them all in one go.

When the simulation has stopped you can delete single or several dynamics: Mark them and press “Del”, or put them back to the right place with the mouse.

7.3.11 Machine | Reset time

By this the virtuell time which is running in the status line at the bottom is reset.

7.4 Graphic

7.4.1 Graphic | Properties

You can use up to 16 windows, each with another detail, another viewing direction and another zoom. On the edit masks of the elements you can specify by the [graphic filter](#) which elements are to be displayed in the separate windows. The windows are activated by the menu **graphic**.

The first item in this menu is **graphic|properties** with which you call up the properties editor of the currently active graphic. With this editor you can enter the name of the window and specify out of

which viewing direction the window shall display the machine. The viewing direction can be frozen as well to prevent a changing by mistake.

Furthermore you can specify which element types this window shall take. There are 3 possibilities: At 'if open' this element type will only be taken if the window is just opened while creating the element. At 'never' the element type is not taken into the list. Please note that changings of these adjustments are only valid for future elements. But for already created elements you can make the adjustments effective by the button 'now'.

With the check boxes 'fix vert. scroll bar' and 'fix horz. scroll bar' you can freeze the current detail so that it cannot be adjusted by the mouse by mistake. Even if only one of these check boxes is clicked you will not be able to neither change the zoom nor the viewing direction of the window.

You can also call up the graphic windows 1 till 9 by the short keys **Alt+1** till **Alt+9**.

See also:

[Simulation of machine](#)

7.5 PLC

In this menu you find everything you need to edit the PLC-program.

7.5.1 PLC | Reset PLC

By this all inputs, outputs, markers, times and counters with the exception of the ones which are excepted in [CPU-properties](#) are set back to "0". No blocks out of the memory are deleted, the data in the data blocks are left untouched.

See also:

[PLC master reset](#)

7.5.2 PLC | Master reset

By this you delete all blocks out of the PLC-memory and you reset all inputs, marker, times and counters. After this the PLC is in the same state like after the start of TrySim.

See also:

[Reset PLC](#)

7.5.3 PLC | Blocks in AS

Here all blocks in the PLC are displayed with its size. The displayed size (in bytes) does not correspond to the one which a block would have in a S7, but it is just a rough estimate.

7.5.4 PLC | CPU

If you click on CPU, the current PLC-cycle (OB2, OB1, OB3) is still stopped, after that the accus and the RLO as well as other informations are read out of the CPU and displayed in a little window. The machine and the PLC-program keep running, the data in the window are only a snapshot.

If you want to know the CPU-data on a special point in the PLC-program you have to use [breakpoints](#). Then the PLC and the machine stop at this position and the CPU-window appears with the current values. Every time you click on “forward”, a cycle of the PLC and a machine simulation is executed. If you want to continue the simulation normally you have to delete the breakpoint and download the block again (or click on the eye, so it is downloaded automatically before activating the monitor-mode).

7.5.5 PLC | CPU properties

You reach this adjustment by **PLC|CPU properties**.

Tabsheets:

[Clock memory](#)

[Clock interrupt](#)

[Retentive memory](#)

[Cycle time monitoring](#)

[Time system, CPU clock](#)

7.5.5.1 PLC | Clock memory

You can reach this tabsheet with [PLC|CPU properties](#).

If you click on the check box **activated**, you can enter a marker byte, whose bits will be switched on and off periodically by the operating system of the virtuell PLC.

The frequencies (in the virtuell time) of the single bits:

Bit 0 10 Hz 0.1 s

Bit 1	5 Hz	0.2 s
Bit 2	2,5Hz	0.4 s
Bit 3	2 Hz	0.5 s
Bit 4	1,2Hz	0.8 s
Bit 5	5 Hz	1.0 s
Bit 6	1 Hz	1.6 s
Bit 7	0,6Hz	2.0 s
	25	
	0,5	

If you want to head for a flashing light emitting indicator with the clock memory you should use the special form of the light emitting [indicator](#) because otherwise it would be quite difficult to see the flashing as the simulation speed is variable.

Using the 10Hz-Bits please notice that the simulation rate as standard is adjusted to 100 ms and that if the adjustment is maintained the bit always will be '1' or '0'. You can change the simulation rate with View|Options|Simulation

7.5.5.2 PLC | Clock interrupt

The tabsheet can be reached by [PLC|CPU Properties](#).

The OB 35 will be called in the here given intervals independent of the adjusted [simulation rate](#). Further clock interrupts are not yet available.

7.5.5.3 PLC | Retentive memory

This tabsheet can be reached by [PLC|CPU Properties](#).

If you select the menu item [Reset PLC](#), all markers will be set to “0”, and also all times and counters. Areas which you enter here are the exception.

7.5.5.4 PLC | Cycle time monitoring

This tabsheet can be reached by [PLC|CPU Properties](#).

The cycle time monitoring reacts if the program needs more than 1 sec in real time. In most cases you have programmed an endless loop then. By default it is not activated because it is not impossible that it causes unsettled crashes occasionally. A changing of this operation is effective as soon as you

restart TrySim.

If you have programmed an endless loop without an activated cycle time supervision and if you have also selected the option “start automatically” (View|Options|Machine), then you have got a problem: As soon as you start TrySim the PLC-programm is started and is in the endless loop.

Start TrySim with the option -n, that causes that no program will be started automatically anymore.

If you do not know how to start a program in Windows with an option you have to enter in the DOS-window: \program\trysim\trysim -n. (provided, that you have installed Try Sim into the suggested directory, otherwise you have to adjust the path correspondingly).

P.S. The new cray is so fast that it can bring an endless loop to an end in less than 24 h. :-)

7.5.5.5 PLC | Time system

This tabsheet can be found by [PLC|CPU Properties](#).

In the CPU there is a clock which is set to the right date and the right time by starting TrySim. This clock just works if the simulation works as well. You can set the clock under PLC|CPU Properties as well as with the [SFC 0](#). (before this, set a local DATE_AND_TIME correctly with the [IEC-function FC3](#)). You can read out the time with [SFC 1](#).

7.6 Block

7.6.1 Block | Open

By this you open one of the program blocks. In the dialog window you can filter which kinds of blocks shall be displayed.

This point is also used to delete program blocks. Mark the block which needs to be deleted in the dialog window and press “Del”, afterwards you leave the dialog window by “Abort”.

7.6.2 Block | Open again

Next to this menu point the blocks which were opened last are shown so you do not have to select blocks which you need very often from the open-blocks-dialog.

7.6.3 Block | Save as

The current block is saved under a different name. Useful to create a copy of the current block which shall be modified slightly then or to create a backup copy for changings of which you do not know whether you will regret them later.

At the moment a changing of the block type (OB,FB,FC) is not possible while saving as. If you want to do it you have to copy each network separately out of the old block (**PLC|Network|copy**) and insert them into the new block again (**PLC|Network|insert**).

7.6.4 Block | Print blocks

Select **PLC|Print**. A selection window with two areas appears. Mark the blocks which shall be printed on the left side and shift them with the arrow buttons to the right side. You can also mark several blocks by clicking them one after another. The blocks in the right field are printed in that order in which they are selected.

Symbolical/Absolute You can select whether the blocks shall be printed in that representation form in that you have saved them the last time, or you can force that it is printed symbolically or absolutely.

STL/FBD/LAD You can select whether the networks shall be printed in that language in that you have saved them the last time, or you can force one of the languages. In the first case the language can be different from network to network. If you want to force the printout in FBD or LAD, but a network cannot be represented in that way, it will be printed in STL.

Symbol-Comment This option has only got a function in STL. If it is selected for all lines which do not have a line comment the comment of the operand out of the [symbol table](#) will be printed instead.

If you only want to print the just processed block, select **Project|Print**. The block is printed like you can see it, that means with the current adjustment of 'With/Without Symbolical Representation' and with the kind of representation STL/FBD/LAD for each network specified by you.

See also:
[PLC-Editor](#)

7.6.5 Block | Block properties

The most important field of the block properties is the block comment.
The menu point is just existing if a PLC-window is active.

See also:

[Project-comment](#)

7.7 View

Most of the functions which are offered in this changing menu are self explaining.

7.7.1 View | Toolbar

With this you can switch on and off the visibility of the toolbar. But we recommend to let the toolbar switched on at every time, because it accelerates the work of TrySim considerably after the settling-in phase. Just users, who cannot select a higher resolution than 640 x 480 and who need each available square centimeter because of this, should switch off the toolbar and remember the short-keys instead.

7.7.2 View | Status Bar

Here you can select whether the status bar of the TrySim main window shall be displayed. In the status bar there are indications for the machine status (green: Simulation runs, red: Simulation has stopped, yellow: Single step mode) as well as for the virtual time and any messages.

7.7.3 View | Dimension

With this menu point you select out of which direction the machine shall be displayed in the current window.

XY: View

YZ: condescendingly (start from the condescending view)

XZ: View from the (start from the condescending view)

right side

View from the

“bottom”

If you wish a [3-D view](#) of the machine select **Graphic|3-D view**.

7.7.4 View | Symbolic representation

If you have assigned names to the operands of the PLC-program in the symbol table, these will be displayed instead of the absolute addresses. By deselecting the “Symbolic representation” the absolute addresses are displayed again.

If you just want to know the absolute address of one single operand it is better to click on it or to put the cursor on it: Then the absolute address is displayed in the statusbar.

7.7.5 View | Rezoom

With this the picture in the machine is enlarged or reduced that much that it fits perfectly into the current window.

After that the [View | Resize window](#) will be executed automatically. By doing this the high- / width-proportion of the window is adjusted to the one of the machine and there will be left more space for other views or PLC-blocks.

You can adjust the size of the machine in [View | Options | Machine](#).

7.7.6 View | Resize window

With this the current window is made as big as the machine that is displayed in the window. The grey border that you keep is necessary so that Windows does not add scroll bars automatically.

You can adjust the size of the machine in [View | Options | Machine](#).

See also:

[View | Rezoom](#)

7.7.7 View | Monitor (DB-window is active)

If you open a DB and if you let display the current data in the **data view**, the data will be read out of the PLC one time and will be displayed. But the data can be changed while the simulation is running. If you select this menu point the displayed data will be read anew out of the PLC nearly every second.

See also:

[View | Update \(DB-window is active\)](#)

7.8 Extras

7.8.1 Extras | Quick Com/Script

This “Stay on top”-Window ([Extras|Quick Com/Script](#)) is helpful to gain a quick overview of the comments, composed to elements, and the [scripts](#).

Editing is not possible in this window.

7.8.2 Extras | Quick Selected

This “Stay on top”-Window is helpful to gain a quick overview of selected elements.

So you can select elements:

- Click in mode 'stop' on an element in a 2-D graphic window
- Click on an element in the [element tree](#)
- For adding new elements keep pressed the shift- or control key
- You can mark multiple elements by surrounding them with the mouse
- In the menu 'Edit' there is the item 'Select all children'

So you deselect the selection:

- Click twice (no double click!) anywhere in the white of a graphic window
- Click on an already marked element whilst holding down the control key
- Click on an element in this quick display

So you open the edit mask:

- For several marked elements it is more favourable to press the enter button
- Short right-click in the graphic window
- A slightly longer right-click in the graphic window and then choose 'properties'
- For many elements and in the element tree a double click is also possible. But in the graphic windows a double click has got a different meaning for some elements (for example digital display and stripes)

If you often want to mark similar groups of elements, please read [Machine | Editing the Machine | Groups](#).

Editing is not possible in this window.

Under [Extras|Options](#) you can adjust if the window shall be opened automatically at the start.

7.9 Window

This menu is the same as in all Windows applications.

7.9.1 Window | Cascade

This is like in every Windows program.

7.9.2 Window | Next to each other

This is like in all Windows programs.

7.9.3 Window | One below the other

This is like in all Windows programs.

7.10 Help

You can only learn by trying how to use the help.

Part



8 Options

8.1 Overview

You reach the options via **View|Options**.

Simulation	Simulation rate, graphic repetition rate
Directories	Project directory
Toolbars	Active toolbars
Machine	Machine size, grid, auto start
PLC Editor	Representation, monitor, auto save

8.2 Options | Simulation

You reach the options via **View|Options**

Simulation repetition rate

In TrySim there is a virtual time with the resolution of 1 milli second.

Here you can adjust how many virtual milli seconds passes by between two simulation steps of the machine.

In reality this point nearly corresponds to the cycle time of the PLC.

The smaller you set this number the more exact is the simulation. But the maximum quick motion factor decreases by this in big machines.

The simulation rate will not be changed by the menu points **Machine|Faster / Slower**. By this just the speed of the virtual clock will be influenced.

CPU-Overspeed

With this you can adjust how often the OB 1 shall be executed between two machine-simulation-steps. In generell it is not useful to let the OB 1 run between two simulation steps several times because the inputs will always have the same value and the outputs will not be analysed, but in a single case it is possible that it makes sense.

Graphic refresh rate

Here can you adjust, how often the graphics are refreshed.

The drawing costs lots of time although the graphic of TrySim is quite simple. If you have programmed big projects and if you attach importance to a wide quick motion range, you should adjust a relatively slow graphic refresh rate. If small projects with fluent movements shall be represented the graphic repetition rate needs to be adjusted as small as possible.

Which adjustment is optimal depends on the speed of your computer and the size of your machine.

But a graphic refresh rate which is adjusted too small does not matter in generell, because TrySim draws less automatically if it is overloaded because of a simulation speed which is too fast.

8.3 Options | Directories

You reach the options via **View|Options**

Here you can give the default directory for projects.

8.4 Options | Machine

You reach the options via **View|Options**

Units of measure

The unit mm is now preset. Other units of earlier versions are not supported any longer.

Machine size

Here you can specify the white colored part of the machine. You can change these values at every time without any damage.

We recommend to adjust these values to the corresponding real machine size, because then it is easier to sort out the chaos of windows with the menu item **View|Resize window**.

Automatic start

When this option is activated, the simulation starts automatically when a project is opened.

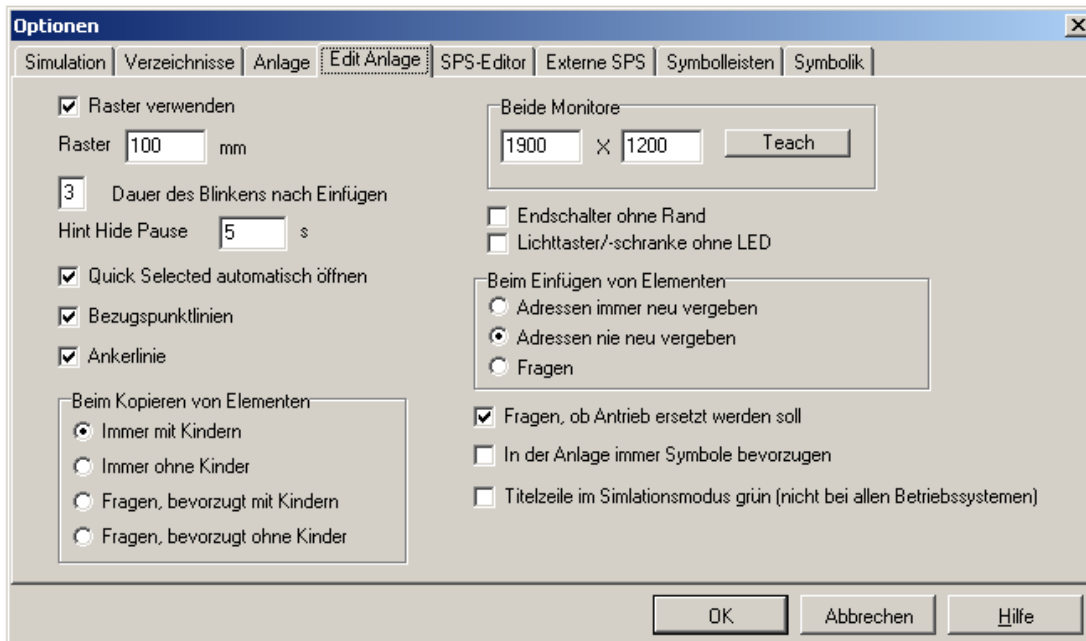
It is very useful for the passing on of the free runtime versions of TrySim to end customers, so that the function of future machines can be checked and discussed in their company.

This option is extremely disturbing if you have programmed an endless loop and the [cycle time monitoring](#) is not activated: TrySim will crash as soon as it is started. In this case start TrySim with the parameter "N" (for Not automatical start).

Use grid

The function grid, which does not need any explanation, is useful for the arrangement of control elements and for the construction of fantasy machines. If you want to create a real machine it is more effective to enter the values of the technical drawing than shifting elements with the mouse.

8.5 Options | Edit machine



- Use raster: When mouse is moved the elements are only positioned in indicates raster.
- Count of flashes after paste: The main element of insertion flashes so that the inserted element is seen better. This can be annoying for experienced user, hence it can be adjusted here.
- Hint Hide Pause : Hereby you can adjust for how long the tiny hints are displayed when pointing to an element.
- Autostart '[Quick Selected](#)': If you don't want to see this you can deselect it.
- While copying elements: In general 'always with children' is the best adjustment. According to working method and type of machine another adjustment can be more advantageous.
- Both screens: When two screens are used it is inconvenient to adjust the size of the TrySim main window, if this has been modified for example by a screensaver. The size set here can then quickly be restored in View|Both screens.
- Endschalter ohne Rand: Wenn die Endschalter sehr klein sind, besteht das Bild oft nur noch aus dem Rand und man kann den Zustand nicht mehr gut erkennen. Diese Option unterdrückt das Zeichnen des Randes. Bei sehr großen Anlagen trägt sie auch ein kleines bisschen zur Beschleunigung der Grafikerstellung bei.
- Light barrier without LED: Analogously here the same applies as for the limit switches.

- When inserting elements: TrySim assigns new addresses in order that no intersections appear (but does not function properly). Normally the addresses are fixed because of the electro plan, then it is more useful to keep the old addresses when copying, because you can recognize from the symbol and the comment how the new address shall be.
- Ask before exchanging drive: When you already have generated a machine with many conveyors or similar and you want to insert complex drives, for example frequency converter with a script subsequently the question is: 'Do you really want to substitute the existing drive?' Very annoying. But normally this question is a protection against overwriting (deleting) of a functioning drive in error.

8.6 Options | PLC editor

You reach the options via **Extras|Options**

Accu representation

In [monitor mode](#) the system tries to display the accus in an appropriate format. This leads to an often changing representation of the accu. Here you can default a fixed format.

View

Here you can adjust the default representation (FBD,LAD or STL) for new blocks. But you can change the representation for each network separately afterwards.

General

Auto save

When a block is closed it is saved without confirmation.

Compile on close

When a block is closed it is downloaded automatically to the PLC without confirmation.

So you know for sure that all (except of the still edited ones) blocks on the hard disk correspond with the ones in the PLC-memory.

This adjustment is annoying if you want to close an erroneous block. But you can still deselect it at that time. If the check box is selected saving happens automatically because just saved blocks can be downloaded to the PLC.

Monitor mode

Do not erase when STL monitoring

The RLO, the operand-state and the accus do not correspond with the current values in overjumped lines, but they keep their value of the last execution.

It is best to display these lines in a different color. Although it seems to be easy it was not possible for us to do it in STL yet. That is why we decided to delete the monitor values of the lines which are not valid anymore. But sometimes you definitely want to know which values were written into the

accus. That is why we have created this check box with that you can prevent the deleting of lines which are not up to date anymore (But see also: [Breakpoints](#)).

Progress bar when monitoring

The blue progress bar in the lower right corner when monitoring the program execution can be switched off.

The blue progress bar changes if the first line in the network is executed sometimes. This information prevents that you wait for a special event that will never happen because the machine simulation stopped, the PLC crashed or a block end was programmed before. But sometimes this progress bar can be annoying, so you can decide by yourself whether you want to see it or not.

Rewire

If you modify the address of an element in the [adress table](#) it is usually desirable that the corresponding addresses in the PLC program are adjusted automatically.

Here you can select if this should be done ever, never or after confirmation.

8.7 Options | Toolbars

You reach the options via **View|Options**

Here you can select the toolbars to be shown. We recommend to display every toolbar, because the symbols accelerate the work with TrySim.

You can also select and deselect the toolbar by clicking on the empty grey of the symbol area with the right button.

Index

-) -

)MCR 265

- + -

+AR1 256

+AR2 256

- 3 -

3-D control with the keyboard 26

3-D control with the mouse 26

3-D View 25

3-D view point 29

3-D Window
Properties 28

- A -

A 162

A(163

ABS 217

Absolut position sensor 51

absolute value 217

Absolute-Real-Drive 112

Accu 125

decrement 251

increment 250

Accu 3 and 4 125

Accu3+4

load 252, 253

retrieve 252

Accumulator 1 125

Accumulator 2 125

Actual parameter 129

AD 238

Add as DInteger (32 Bit) 202

Add as integer (16 Bit) 195

Add as real 210

Address table 30

adjust

window size 333

Zoom 333

Adressregister

Warnung 254

Akku3+4

load 253

Allen-Bradley 145

AN 164

AN(164

Analog control 68

Analog values 116

Analyzer 78

Anchor

element 45

AND 162

AND double word 238

AND NOT 164

AND NOT(164

AND Word 238

AND(163

Antenna

RFID 86

ANY 156

AR1 und AR2

Warnung 254

Arc belt 65

Arcus cosine of a real 248

Arcus sine of a real 247

Arcustangens of a real 248

Assignment 171

Attractor 97

Auto start 338

Automatic hook 60, 85

AW 238

- B -

Background 94

Balance 77

Bar 88

Bar code scanner 53

BCD

convert to 242

load 195

BCD number 241

BEC 261

BEU 261

Binary result 174

- Bit-Motor
 - as Drive for Joint 113
 - BLD 264
 - Block 157, 158
 - import from another project 318
 - Block end conditioned 261
 - Block Move 292
 - Block types 273
 - BLOCK_DB 158
 - BLOCK_FB 157
 - BLOCK_FC 157
 - Blocks
 - create 273
 - delete 274
 - download 281
 - monitor 282
 - print 331
 - Board 87
 - BOOL type 149
 - Box 87
 - Bracket 170
 - Breakpoint 283
 - conditional 285
 - LAD 285
 - STL 284
 - BTD 242
 - BTI 243
 - Button 65
 - ByPass 75
 - BYTE 150
- C -**
- CAD 250
 - CALL 261
 - unconditioned 263
 - CALL conditioned 262
 - CAR 257
 - CAW 249
 - CC 262
 - CD 191
 - CDB 263
 - Chain 61
 - Pawl 86
 - Resolver 52
 - CHAR 150
 - Characteristic curve 92
 - Check Valve 76
 - Child 40
 - Choice
 - Quick 334
 - Clock 330
 - Read out with SFC 1 292
 - Set with SFC 0 292
 - Clock interrupts 267, 329
 - Clock memory 328
 - Closing bracket 170
 - CLR 174
 - Color 43
 - Color depth 24
 - Comment
 - Element- 45
 - Project- 318
 - Compare DInteger (32 Bit) 206, 207, 208, 209
 - Compare integer (16 bit) 198, 199, 200, 201
 - Compare reals 213, 214, 215, 216
 - Conditional Breakpoint 285
 - specific features in FBD/LADD 286
 - conditional call 262
 - Connection
 - to PLC 44
 - Consistency sensor 78
 - Constants 159
 - Contacts
 - of master switch 72
 - Container 73
 - continuous
 - dynamic 56
 - Continuous Dynamics
 - Saw 79
 - Control
 - analog 68
 - Converter
 - for media 88
 - Conveyor 58
 - Cooling 92
 - Copy
 - memory areas 292
 - Cosine 247
 - COUNTER 189, 190, 191
 - reset 192
 - set 192
 - CPU 267, 328, 329
 - Clock 330
 - Tabsheet properties 328
 - Crank-

Crank-
 Drive 115
 Create DB 293
 Create new blocks 273
 Create new elements 20
 Cross 96
 Crossreference list 287
 CU 190
 Curve
 characteristic 92
 Cutter 81
 cycle time monitoring 329

- D -

Data block
 monitoring 333
 open 260
 Data chip
 RFID 86
 Data types 149
 DATE 154
 DATE_AND_TIME 155
 Datenaustausch 136
 C 137
 Delphi 138
 Pascal 138
 DB
 create while runtime 293
 delete while runtime 293
 get length while runtime 294
 DEC 251
 Declarations 129
 delay 178, 179
 delete 24
 blocks 274
 Delete DB 293
 Density 325
 Destroyer 56
 Diagonal transport 58
 Different implemented 266
 Digital display 66
 Digital input 66
 DINT 153
 convert to 242, 244
 negate 209
 Direction 24
 Directories 338

Display
 for strings 68
 Display background 94
 Distance sensor 52
 Divide as DInteger (32 Bit) 205
 Divide as integer (16 Bit) 197
 Divide as real 212
 Divider 81
 Download
 blocks 281
 Drive
 Absolute-Real 112
 Bit-Motor for Joint 113
 by linear mover 108, 109
 Crank- 115
 Pump 112, 113
 Servo direct set value 111
 Servo for joint 110
 Servo for linear mover 109
 Valve 112, 113
 Word- 107, 112
 DTB 242
 DTR 244
 DWORD 152
 Dynamic
 speed 54
 Dynamic converter 83
 Dynamic elemente 49
 Dynamic elements 49, 53, 54, 56
 Dynamic elements of simulation
 Packet 117
 Dynamics
 continuous 56
 turnable 118

- E -

Edit
 block header 279
 DBs 281
 OBs FBs und FCs 275
 Edit properties 22
 Editing tool Cross 96
 Editing tool Stripe 96
 Editor 36
 Einschaltverzögerung 288
 Element
 anchor 45

Element
 Comment 45
 library 325
 script 46
Element tree 31
Elements
 Common Properties 38
 create 20
 delete 24
 positioning 34
 select 21
 turnable 98
ENT 253
exchange 257
Exponential function 219
Export and import blocks 275, 297
Extended pipe properties 74
Extended pulse
 time diagram 183
External PLC
 Allen-Bradley 145
 Rockwell 145
 via MPI 142, 144
Externe SPS 134
 Sicherheitshinweise 136
 über MPI 140
Extruder 55

- F -

Fade in / out 321
Falling slope 175
Father 39
FBD 276
 Breakpoint 285
FC 84 289
FC 85 (FIFO) 289
FC 87 (LIFO) 289
File
 add 318
Fill 292
Filter
 Grafic- 27
 Import- 301
Find
 elements 36
Fix 23
fix on dynamics 85

Fixability 42
Fixing elements 23
Flange 76
Flashlight 66
Flow meter 77
Fluidor 79
Fluids 72
 Media 325
FN 175
Focus 322
Formal parameter 129
FP 175
FR 193
Free movable point 61
Frequency converter 107, 112
Front-View 24
Function blocks 132
Functions 129

- G -

Generator 54
 continuous 55
Grafic
 Filter 27
Graphic
 -window 326
Graphic editor 36
Grid 338, 339
Groups 32, 33
Guide
 Short- 20

- H -

hanging transporter 61
Heating 92
Hook 60
 automatic 85
 controlled by a Pawl 86
Hot spot 57
Huge Machines
 how to structure 37
Hysteresis 93

- I -

IBH
 Schnittstelle zu TrySim 144

IEC
 TOF 288
 TON 288
 TP 288

IEC functions 287

implemented
 differingt 266

Import
 blocks 275, 297
 -filter 301
 from S5 304

Impuls 288

In parameter 129

INC 250

Indirect access 259

Info over DB 294

In-Out parameter 129

Instance 132

Instructions 160

INT 152
 convert to 243
 negate 202

Interface 136
 Allen-Bradley 145
 C 137
 Delphi 138
 MPI 142, 144
 Pascal 138
 Rockwell 145

Introduction 15

INVD 240

INVI 240

ITB 242

ITD 244

- J -

JBI 229
 JC 227
 JCB 228
 JCN 221
 JM 222

JMZ 221
 JN 223
 JNB 227
 JNBI 229
 JO 232
 Joint 59
 Absolute-Real-Drive 112
 JOS 233
 Joystick 71
 JP 224
 JPZ 225
 JU 220
 jump 220, 221, 222, 224, 225, 226, 227, 228, 229
 Jump label 219
 jump list 231
 JUO 232
 JZ 226

- L -

Label
 jump - 219

LAD 277
 Breakpoint 285

LAR 255

LEAVE 253

LED 66

Level gauge 77

Level switch 79

Library 325

Light barrier 49

Light sensor 49

Limit switch 50

Limit switch nose 51

Linear mover 57
 with sensors 57

Link 88

List of operations
 by groups 160

LN 219

Load 193

Load BCD coded 195

LOOP 230

- M -

Machine 338

Mangle 63
 Mark
 turnable elements 99
 Marker
 retentive 329
 Marking
 Quick 334
 Master 50, 51
 Master Control Register 265, 266
 Master reset
 PLC 327
 Master switch 71
 contacts 72
 MCR(265
 MCRA 265
 MCRD 266
 Media manager 326
 Medium 325
 Melter 81
 memory areas 292
 memory-indirect 259
 MOD 198, 205
 modulo 198, 205
 Monitor 24, 25
 Monitor mode 282
 Monitoring
 data block 333
 Motor
 Word- 107, 112
 MPI
 Interface 142, 144
 Schnittstelle 140
 Multiple switch 71
 Multiply as DInteger (32 Bit) 204
 Multiply as integer (16 Bit) 196
 Multiply as real 211

- N -

Name 39
 negate
 DINT 209
 INT 202
 REAL 218
 NEGD 209
 NEGI 202
 NEGR 218
 New

Elements 20
 Node of a Chain 62
 NOP 264
 Nose 51
 NOT 174

- O -

O
 OR 165
 O(166
 OD 239
 ON 167
 ON(167
 online 275, 281, 297
 Open
 DB/DI 260
 Open and save blocks 274
 Operanden übernehmen 278
 Operations 160
 Optiones 338
 Options 338, 339, 341
 OR double word 239
 OR Not 167
 OR word 239
 OR(166
 Oscillograph 70
 Out parameter 129
 Overpressure valve 75
 OW 239

- P -

Parameter 129
 Pawl 86
 Peek 89
 Pipe 73
 pipe properties 74
 Pipeconnection 76
 Plane 84
 PLC 121, 124
 connection to 44
 master reset 327
 reset 327
 Plus 202, 209, 251
 POINTER 153
 Poke 90

POP 252
 Position 41
 Positioning elements 34
 Press 84
 Pressure Sensor 78
 Print
 Address table 30
 blocks 331
 project 319
 Programmable Logic Controller 121
 Project
 print 319
 Project comment 318
 Properties
 Elements 38
 Properties edit 22
 Proportional Valve 107, 112
 Pulse timer
 time diagram 181
 Pump 75
 Drive 112, 113
 PUSH 252
 Push button 65

- Q -

Quick
 Selected 334

- R -

R 173, 192
 Reactor 88
 REAL 158
 convert to 244
 negate 218
 Real time clock 330
 Real-Drive 112
 Reconnect 286
 reference point 35
 register-indirect 259
 Reset
 Bit 173
 COUNTER 192
 PLC 327
 Time 326
 Resolution 25

Resolver 51
 for Transporter/Chain 52
 Resolver wheel
 wheel 54
 retentive memory 329
 Retentive-on-delay
 time diagram 184
 Rewire 286
 RFID
 antenna & data chip 86
 Rider 63, 88
 Rider way 63, 88
 Rising slope 175
 RLD 234
 RLDA 235
 RLO
 set 173
 RND 245
 RND- 245
 RND+ 245
 Rockwell 145
 rotate 234, 235, 236, 237
 round 245
 RRD 237
 RRDA 236
 Running wheel 54

- S -

S 172, 192
 S5
 Import- 304
 SAVE 174
 Saw 79, 81
 Schnittstelle 136
 C 137
 Delphi 138
 IBH-softec 144
 MPI 140
 Pascal 138
 Screen 25
 Script
 element 46
 SD 178
 SE 182
 Search
 element 36
 Segment 62

- Select
 - turnable elements 99
- Select another view 24
- Selecting elements 21
- Sensor
 - Distance- 52
- Servo
 - drive for joint 110
 - Drive for linear mover 109
- set
 - bit 172
 - COUNTER 192
 - RLO 173
- SF 179
- SFB 291
- SFB3 288
- SFB4 288
- SFB5
 - Ausschaltverzögerung 288
- SFC 291
- SFC 0 292
- SFC 1 292
- SFC 20 292
- SFC 21 292
- SFC 22 293
- SFC 23 293
- SFC 24 294
- SFC 64 294
- Shelf 87
- shift 234, 235, 236, 237
 - elements 34
- Shifter 82
- Short guide 20
- SI 181
- Sicherheitshinweise
 - externe SPS 136
- Side-View 24
- Simulation
 - Start and stop 36
- Simulation tool Attractor 97
- Sine 246
- Single step 283
- Sink 73
- Size
 - elements 41
- SLD 235
- Slide controller 68
- Slope
 - falling 175
- Slope rising 175
- Slow motion 98
- SLW 236
- SoftSPS
 - IBH 144
- Source 73
- Speed-Trigger 98
- SPL 231
- SPS
 - Schnittstelle 134
- Spy 51
- SQR 218
- SQRT 218
- square 218
- square root 218
- SRD 234
- SRW 237
- SS 183
- SSD 234
- SSI 237
- Static
 - Variables 133
- Static elements of simulation 48
- Static variables 132
- Step switch 71
- Sticker 85
- STL 276
 - Breakpoint 284
- Stopper 82
- Straighter 83
- STRING 158, 269
- String display 68
- Stripe 96
- Structure of the system 16
- structuring
 - Huge Machines 37
- Subtract as DInteger (32 Bit) 203
- Subtract as integer (16 Bit) 196
- Subtract as real 211
- swimming switch 79
- Switch 50, 65
 - level - 79
 - Master- 71
 - Multiple 71
- Swivel table 65
- Symbol table 290
- System functions 291

System funktion blocks 291
 Systemzeit
 auslesen in ms 294

- T -

T 194
 TAK 249
 Take over operands
 FBD 276
 LAD 277
 Tangens 247
 TAR1 or 2 257
 Temp parameter 129
 Temperatur switch 93
 Temporary
 Variables 134
 Text display 70
 Thermal mass 91
 Thermometer 93
 Thermostat 93
 time diagram
 extended pulse 183
 pulse timer 181
 retentive-on-delay 184
 Time system 330
 TIME_OF_DAY 156
 Timer 154, 176, 178, 179, 182, 183
 enter the duration 188
 in FBD/LAD 186
 Tool bars 341
 Top-View 24
 Trace 283
 Track 55, 56
 Transfer 194
 Transition 63
 Transporter
 Arc- 65
 Resolver 52
 turnable 65
 with two rolls 63
 TRUNC 244
 turn 59
 Turn table 64
 Turnable
 Transporter 65
 turnable elements 98
 select 99

Turner 62
 types
 data- 149

- U -

UC 263
 UDT 159
 unconditioned call 263
 User Defined Type 159

- V -

Valve 75
 Check- 76
 Drive 112, 113
 overpressure 75
 Variables
 Static- 133
 Temporary- 134
 View
 select another 24
 view point
 3-D 29
 Viscosity 325
 Visibility 43

- W -

Wedge 82
 Window
 adjust size 333
 Graphic- 326
 WORD 151
 Word display 66
 Word-Motor 107, 112

- X -

X 168
 X(169
 XN 169
 XN(170
 XOD 239
 XOR 168
 XOR Double word 239
 XOR NOT 169

XOR NOT(170
XOR Word 240
XOR(169
XOW 240

- Z -

Zähler
 auf Wert setzen 185
Zeit
 auslesen in ms 294
Zeit als Impuls 288
Zoom 30
 adjust 333
ZW 185