

# SIEMENS

## SIMIT (V8.1)

### Operating Manual

Start	1
Couplings	2
Simulation model	3
Automatic model creation	4
Diagnostics & visualization	5
Automatic control interface	6
Libraries	7
Appendix	8

## Legal information

### Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

<b>⚠ DANGER</b>
indicates that death or severe personal injury <b>will</b> result if proper precautions are not taken.

<b>⚠ WARNING</b>
indicates that death or severe personal injury <b>may</b> result if proper precautions are not taken.

<b>⚠ CAUTION</b>
indicates that minor personal injury can result if proper precautions are not taken.

<b>NOTICE</b>
indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

### Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

### Proper use of Siemens products

Note the following:

<b>⚠ WARNING</b>
Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

### Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

### Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

# Table of contents

<b>1</b>	<b>Start.....</b>	<b>13</b>
1.1	What's new?.....	13
1.2	Product variants.....	15
1.3	Installation and update.....	16
1.3.1	Scope of delivery.....	16
1.3.2	Installation and update.....	17
1.3.3	Adjustable timeout times.....	18
1.3.4	Uninstalling the SIMIT software.....	18
1.4	Basics of SIMIT.....	19
1.4.1	Overview.....	19
1.4.2	SIMIT in DEMO mode.....	19
1.4.3	Portal view.....	21
1.4.4	The SIMIT graphical user interface.....	23
1.4.5	Creating simulations.....	26
1.4.6	Running simulations.....	28
1.4.7	Virtual time management.....	29
1.4.7.1	Synchronous and asynchronous operating modes.....	29
1.4.7.2	Acceleration and delay of the simulation time.....	30
1.4.8	Display of the simulation load.....	31
1.4.9	Changes in a simulation project while a simulation is running.....	33
1.4.10	Visualizing simulations.....	34
1.4.10.1	Visualization with graphics.....	34
1.4.10.2	Visualizing signals.....	38
1.4.10.3	Visualizing coupling signals.....	40
<b>2</b>	<b>Couplings.....</b>	<b>45</b>
2.1	The coupling concept.....	45
2.1.1	Couplings in the SIMIT architecture.....	45
2.1.2	Exchanging signals via couplings.....	46
2.1.3	Couplings to SIMATIC PLCs.....	48
2.2	Configuring and using couplings.....	49
2.2.1	Creating couplings.....	49
2.2.2	Editing couplings.....	50
2.2.3	Deactivating couplings.....	51
2.2.4	The signals of a coupling.....	52
2.2.4.1	Overview.....	52
2.2.4.2	Meaning of the coupling name.....	52
2.2.4.3	Sorting and filtering of signals in the coupling editor.....	53
2.2.4.4	Addressing signals.....	54
2.2.4.5	Fixing signals in the coupling editor.....	54
2.2.4.6	Interconnecting signals in the coupling editor.....	56
2.2.5	Using I/O signals.....	57
2.2.5.1	I/O signals for connection on charts.....	57
2.2.5.2	I/O signals for animations.....	59

2.2.5.3	I/O signals in controls on charts.....	60
2.2.5.4	I/O signals in trends.....	61
2.2.5.5	Multiple use of I/O connectors.....	61
2.2.6	Importing and exporting signals.....	61
2.2.6.1	Overview.....	61
2.2.6.2	The symbol table.....	62
2.2.6.3	The tag table.....	63
2.2.6.4	The signal table.....	64
2.2.6.5	The INI format of the OPC couplings.....	65
2.2.6.6	Import of signal properties.....	66
2.2.6.7	Export of signal properties.....	69
2.3	Couplings for SIMATIC.....	70
2.3.1	Common properties of SIMATIC couplings.....	70
2.3.1.1	Data types of signals.....	70
2.3.1.2	Scaling analog signals.....	71
2.3.1.3	Limitation of analog signals.....	74
2.3.1.4	Symbolic addressing.....	74
2.3.1.5	Accessing data blocks and bit memories.....	76
2.3.2	PROFIBUS DP coupling.....	78
2.3.2.1	How the PROFIBUS DP coupling works.....	78
2.3.2.2	Configuring the PROFIBUS DP coupling.....	88
2.3.2.3	Editing the PROFIBUS DP coupling.....	94
2.3.2.4	Redundant and fail-safe systems.....	101
2.3.2.5	Configuring the PROFIBUS DP interface module.....	104
2.3.3	PROFINET IO coupling.....	106
2.3.3.1	Functioning of the PROFINET IO coupling.....	106
2.3.3.2	Configuring the PROFINET IO Coupling.....	111
2.3.3.3	Editing the PROFINET IO coupling.....	119
2.3.3.4	Configuring the PROFINET IO interface module.....	125
2.3.3.5	Data record communication.....	127
2.3.4	PRODAVE coupling.....	127
2.3.4.1	Functioning of the PRODAVE coupling.....	127
2.3.4.2	Configuring the PRODAVE coupling.....	128
2.3.5	PLCSIM coupling.....	131
2.3.5.1	How the PLCSIM coupling works.....	131
2.3.5.2	Configuring the PLCSIM coupling.....	132
2.3.6	"Virtual controller" coupling.....	135
2.3.6.1	How the Virtual Controller coupling works.....	135
2.3.6.2	Configuring the Virtual Controller coupling.....	136
2.4	OPC coupling.....	137
2.4.1	Functioning of the OPC coupling.....	137
2.4.2	The OPC Server coupling.....	139
2.4.2.1	Adding an OPC Server coupling.....	139
2.4.2.2	Configuring I/O signals in the OPC server coupling.....	140
2.4.2.3	Properties of the OPC server coupling.....	141
2.4.2.4	Special features of the OPC server coupling.....	141
2.4.3	The OPC Client coupling.....	143
2.4.3.1	Adding an OPC Client coupling.....	143
2.4.3.2	Configuring I/O signals in the OPC client coupling.....	144
2.4.3.3	Properties of the OPC client coupling.....	145
2.4.3.4	Signal properties within the coupling.....	147

2.4.3.5	Special features of the OPC client coupling.....	149
2.4.4	DCOM configuration.....	150
2.4.4.1	Overview.....	150
2.4.4.2	Firewall.....	151
2.4.4.3	Domain and user.....	151
2.4.4.4	Granting access rights.....	151
2.5	Shared Memory coupling.....	152
2.5.1	How the SHM coupling works.....	152
2.5.1.1	Accessing the memory area.....	152
2.5.1.2	Structure of the memory area.....	153
2.5.1.3	Creating the memory area.....	156
2.5.2	Configuring the SHM coupling.....	157
2.5.2.1	Creating an SHM coupling.....	157
2.5.2.2	Configuring the signals in the SHM coupling.....	159
2.5.2.3	Signal properties.....	161
2.5.2.4	Properties of the SHM coupling.....	161
2.5.2.5	Importing and exporting signals.....	162
<b>3</b>	<b>Simulation model.....</b>	<b>165</b>
3.1	The Project Manager.....	165
3.1.1	View and functions of the Project Manager.....	165
3.1.2	Versioning.....	166
3.1.3	Write protection.....	168
3.2	The chart editor.....	170
3.3	The task cards.....	170
3.3.1	The "Controls" task card.....	170
3.3.2	The "Components" task card.....	171
3.3.3	The "Graphic" task card.....	174
3.3.4	The "Projects" task card.....	175
3.3.5	The "Signals" task card.....	177
3.3.6	The "Macros" task card.....	178
3.3.7	The "Templates" task card.....	179
3.4	The Macro Component Editor (MCE).....	180
3.4.1	The macro editor.....	180
3.4.2	Separating connectors of a macro component.....	183
3.4.3	Topological connectors of macro components.....	184
3.4.4	Default settings for macro component inputs.....	185
3.4.5	Defining parameters of macro components.....	185
3.4.6	Properties of macro components.....	186
3.4.7	Find and Replace in macro components.....	187
3.4.8	Using macro components.....	187
3.5	Migrating projects from previous versions of SIMIT.....	188
3.5.1	Overview.....	188
3.5.2	Migrating SIMIT projects.....	189
3.5.2.1	Aspects of project migration.....	189
3.5.2.2	Step-by-step migration of projects.....	190
3.5.2.3	Log file.....	192
3.5.3	Migrating charts and operating screens.....	192
3.5.3.1	Overview.....	192
3.5.3.2	Migrating charts with sheets.....	192

3.5.3.3	Migrating component names.....	192
3.5.3.4	Migrating signal names.....	193
3.5.3.5	Migrating data types.....	193
3.5.3.6	Specifics when migrating components.....	194
3.5.3.7	Migrating graphics and animation.....	195
3.5.3.8	Migrating operator controls.....	197
3.5.3.9	Rule-based implicit connections.....	202
3.5.4	Migrating couplings.....	202
3.5.4.1	Migrating the names of coupling signals.....	202
3.5.4.2	Migrating data types of coupling signals.....	204
3.5.4.3	Migrating data record communication.....	204
3.5.5	Migrating macro components.....	204
3.5.5.1	Overview.....	204
3.5.5.2	Step-by-step migration of macro components.....	206
3.5.5.3	Migrating charts with macro components.....	209
3.5.5.4	Problems arising during migration.....	211
3.5.6	Migrating templates.....	212
3.5.6.1	Overview.....	212
3.5.6.2	Step-by-step migration of templates.....	212
3.5.6.3	Specifics when migrating templates.....	213
3.5.7	Migrating scripts.....	214
<b>4</b>	<b>Automatic model creation.....</b>	<b>217</b>
4.1	Templates.....	218
4.1.1	Creating templates.....	218
4.1.1.1	Find and replace in templates.....	221
4.1.2	Instantiating templates by entering replacements.....	221
4.1.3	Creating tables from a template.....	223
4.1.4	Defining a folder hierarchy for templates.....	224
4.1.5	Addressing a module via the I/O address.....	224
4.1.6	Indirect addressing.....	224
4.1.7	Opening basic templates in the editor.....	225
4.2	The table import.....	226
4.3	The IEA import.....	229
4.4	The CMT import.....	232
4.5	Generic import.....	235
4.5.1	Importing the XML file.....	235
4.5.2	The syntax.....	236
4.5.3	Examples.....	241
4.5.3.1	Example of a single template instantiation.....	241
4.5.3.2	Example of a grouped template instantiation.....	241
4.5.3.3	Example of chart creation.....	242
4.6	Automatic parameter assignment.....	243
<b>5</b>	<b>Diagnostics &amp; visualization.....</b>	<b>249</b>
5.1	Trend and Messaging Editor (TME).....	249
5.1.1	The functions of the Trend and Messaging Editor.....	249
5.1.2	The Messaging System.....	250
5.1.2.1	Overview.....	250
5.1.2.2	The message editor.....	250

5.1.2.3	Message – the message component.....	252
5.1.2.4	Component-specific messages.....	252
5.1.2.5	Messages in the status bar.....	253
5.1.2.6	Limitations of the message system.....	253
5.1.3	The Archive.....	253
5.1.3.1	Overview.....	253
5.1.3.2	Configuring the archive.....	254
5.1.4	Trends.....	255
5.1.4.1	Overview.....	255
5.1.4.2	Adding and configuring trends.....	255
5.1.4.3	Displaying signal trends .....	258
5.1.4.4	Features of the trend editor.....	263
5.2	Find & replace.....	265
5.2.1	Find.....	266
5.2.1.1	Finding with the Find & replace editor.....	266
5.2.1.2	Search using signal properties.....	268
5.2.1.3	Searching for fixed signals.....	270
5.2.2	Replace.....	270
5.2.2.1	Replacing with the Find & Replace editor.....	270
5.2.2.2	Refresh.....	273
5.3	Consistency check.....	273
<b>6</b>	<b>Automatic control interface.....</b>	<b>277</b>
6.1	Overview.....	277
6.2	Handling of scripts.....	278
6.2.1	Creating a script.....	278
6.2.2	Executing a script.....	280
6.2.3	External scripts.....	280
6.3	The script syntax.....	281
6.3.1	Controlling the script.....	281
6.3.2	Composing scripts.....	283
6.3.3	Commenting scripts.....	283
6.3.4	Signals in scripts.....	284
6.3.5	Controlling the simulation.....	285
6.3.5.1	Initializing a simulation.....	285
6.3.5.2	Starting the simulation.....	286
6.3.5.3	Starting and waiting until an absolute time.....	286
6.3.5.4	Starting and waiting until a relative time.....	286
6.3.5.5	Starting and waiting for a certain number of time slices.....	286
6.3.5.6	Starting and waiting for an event.....	287
6.3.5.7	Stopping the simulation.....	287
6.3.5.8	Executing a single step.....	287
6.3.5.9	Saving a snapshot.....	288
6.3.5.10	Loading a snapshot.....	288
6.3.5.11	Resetting the simulation time.....	288
6.3.6	Logging.....	289
6.3.6.1	Opening and closing log files.....	289
6.3.6.2	Unformatted output.....	289
6.3.6.3	Formatted output.....	289
6.3.6.4	Outputting time and date.....	290

6.3.6.5	Outputting version information.....	291
6.3.6.6	The _printlog system function.....	291
6.3.7	Signal curves.....	291
6.3.7.1	Overview.....	291
6.3.7.2	Opening and closing a plot file.....	292
6.3.7.3	Specifying signals.....	292
6.3.7.4	Specifying a cycle.....	293
6.3.8	Setting signals.....	293
6.3.8.1	Setting individual values.....	293
6.3.8.2	Separating connected signals.....	294
6.3.8.3	Defining a signal curve.....	295
6.3.9	Conditional execution.....	295
6.3.10	Accessing the simulation time.....	296
<b>7</b>	<b>Libraries.....</b>	<b>297</b>
7.1	The basic library.....	297
7.1.1	General.....	297
7.1.1.1	Introduction.....	297
7.1.1.2	Component symbols.....	297
7.1.1.3	Symbols for the controls.....	299
7.1.1.4	Component connectors.....	300
7.1.1.5	Connectors for controls.....	300
7.1.1.6	Connecting connectors.....	301
7.1.1.7	Setting inputs.....	303
7.1.1.8	Properties of components.....	304
7.1.1.9	Component error messages.....	311
7.1.1.10	Properties of controls.....	311
7.1.2	Connectors.....	313
7.1.2.1	Global connector.....	314
7.1.2.2	I/O connectors.....	315
7.1.2.3	The topological connector.....	315
7.1.2.4	The Unit connector.....	315
7.1.3	Standard components.....	316
7.1.3.1	Overview.....	316
7.1.3.2	Analog functions.....	316
7.1.3.3	Integer functions.....	335
7.1.3.4	Mathematical functions.....	341
7.1.3.5	Binary functions.....	345
7.1.3.6	Converting values.....	354
7.1.3.7	General components in the Misc directory.....	363
7.1.4	Drive components.....	369
7.1.4.1	Overview.....	369
7.1.4.2	Valve drives.....	369
7.1.4.3	Pump and fan drives.....	373
7.1.4.4	PROFIdrive devices.....	375
7.1.4.5	SIMOCODE pro motor control devices.....	389
7.1.5	Sensor components.....	407
7.1.5.1	SIWAREXU components.....	407
7.1.5.2	Counter module FM350-1.....	413
7.1.6	Communication components.....	420
7.1.6.1	Components for SIMATIC.....	420
7.1.6.2	Components for SINUMERIK.....	422



7.1.7	Controls.....	427
7.1.7.1	Controls for displaying signal values.....	427
7.1.7.2	Controls for entering signal values.....	433
7.1.7.3	Miscellaneous controls.....	443
7.1.7.4	The 3D Viewer control.....	446
7.2	The FLOWNET library.....	458
7.2.1	Introduction.....	458
7.2.2	Flownets.....	459
7.2.2.1	Flownet basics.....	460
7.2.2.2	Variables used in flownets.....	463
7.2.2.3	Modeling of flownet branches.....	463
7.2.2.4	Modeling of flownet nodes.....	464
7.2.2.5	Heat exchange with the environment.....	467
7.2.2.6	Parameter assignment of flownets.....	467
7.2.3	Components in FLOWNET library.....	471
7.2.3.1	The topological connector in the FLOWNET library.....	471
7.2.3.2	General components.....	473
7.2.3.3	Measuring components.....	487
7.2.3.4	Component types for "water/steam" medium.....	493
7.2.3.5	Component types for liquid medium.....	513
7.2.3.6	Component types for gas medium.....	528
7.2.4	Creating your own component types for flownets.....	543
7.2.4.1	Topological properties.....	544
7.2.4.2	Connection to the solution procedure.....	547
7.2.4.3	Constants and functions.....	553
7.2.4.4	Initialization of flownet simulations.....	558
7.3	The CONTEC library.....	559
7.3.1	Introduction.....	559
7.3.2	Material handling simulation.....	560
7.3.2.1	Principles of conveyor technology simulation.....	561
7.3.2.2	Modeling of the objects.....	562
7.3.2.3	Modeling the conveyor system network.....	564
7.3.2.4	Special features of conveyor technology simulation.....	567
7.3.2.5	Scalability.....	569
7.3.2.6	Generating the simulation of drives and sensors.....	572
7.3.3	Components of the CONTEC library.....	577
7.3.3.1	The topological connector in the CONTEC library.....	577
7.3.3.2	Component types for conveyor systems with vehicles.....	578
7.3.3.3	Component types for non-vehicular conveyor systems.....	593
7.3.3.4	Component types for simulating objects.....	626
7.3.3.5	Component types for simulating identification systems.....	632
7.3.4	Creating custom component types for material handling simulation.....	662
7.3.4.1	Topological properties.....	663
7.3.4.2	Connection to the solution procedure.....	666
7.3.4.3	System functions.....	674
7.3.4.4	System variables.....	683
<b>8</b>	<b>Appendix.....</b>	<b>685</b>
8.1	Automatic modeling > Table Import.....	685
8.2	Automatic modeling > IEA import.....	685

8.3	Automatic modeling > CMT import .....	685
8.4	Automatic modeling > Create device level.....	685
8.5	Automatic modeling > Generic import.....	685
8.6	Automatic model creation > Automatic parameter assignment.....	686
8.7	Portal view > Start.....	686
8.8	Portal view > Couplings.....	687
8.9	Portal view > Simulation model.....	687
8.10	Portal view > Automatic model creation.....	687
8.11	Portal view > Diagnostics & visualization.....	688
8.12	"Select project" dialog box.....	688
8.13	"IM configuration" dialog box.....	690
8.14	"Add license key" dialog box.....	690
8.15	"No license key found" dialog box.....	691
8.16	"PROFIBUS DP import" dialog box.....	691
8.17	"PROFINET import" dialog box.....	692
8.18	"IEA import" dialog box.....	693
8.19	"Table import" dialog box.....	694
8.20	"Create device level" dialog box.....	695
8.21	"Automatic parameter assignment" dialog box.....	695
8.22	"Characteristic" editor.....	696
8.23	"Info" dialog box.....	697
8.24	"Print" dialog box.....	697
8.25	"CMT import" dialog box.....	699
8.26	"Instantiate template" dialog box.....	700
8.27	"Selection of new coupling" dialog box.....	701
8.28	Project > New project .....	702
8.29	Project > Open.....	703
8.30	Project > Close.....	703
8.31	Project > Save all.....	704
8.32	Project > Save as.....	704
8.33	Project > Archive.....	704
8.34	Project > Retrieve.....	705
8.35	Project > Migrate.....	705
8.36	Project > Exit.....	706
8.37	Edit > Cut.....	706

---

8.38	Edit > Copy.....	706
8.39	Edit > Paste.....	706
8.40	Simulation > Initialize.....	706
8.41	Simulation > Start.....	706
8.42	Simulation > Pause.....	706
8.43	Simulation > Single Step.....	707
8.44	Simulation > Exit.....	707
8.45	Simulation > Snapshot.....	707
8.46	Window > Tile Horizontally.....	707
8.47	Window > Tile Vertically.....	707
8.48	Window > Unsplit.....	707
8.49	Window > Close all.....	707
8.50	Options > Zoom.....	707
8.51	Options > IM configuration.....	707
8.52	Options > Assign coupling signals.....	708
8.53	Help > About.....	708
8.54	Help > Add License Key.....	708



# Start

## 1.1 What's new?

### SIMIT V8.1

Version V8.1 includes the following enhancements or changes compared with the previous version V8.0:

- **New "Couplings" folder**

There is a new "Couplings" folder in the project tree. All couplings of the current project are assembled in this folder and therefore no longer have to be listed in the project tree. This provides a better overview in the project tree when the project contains many couplings.
- **New "Virtual controller" coupling**

There is a new coupling type, "Virtual controller". This coupling creates a connection with the "SIMIT Virtual Controller (VC)" software. The SIMIT Virtual Controller is a separate product and therefore not part of SIMIT.  
You can find additional information on this in the section: "Virtual controller" coupling (Page 135).
- **Subdivision of the signal characteristics in the coupling**

The properties of the coupling signals are now divided into the following 4 categories:

  - General
  - Scaling
  - Limit
  - Interconnection

You can find additional information on this in the section: The signal table (Page 64).
- **Limiting coupling signals**

Analog coupling signals can now be limited. The following properties are provided for this:

  - The limitation can be switched on and off.
  - A minimum value and a maximum value can be determined in each case.

The limitation is made in physical values.  
This function is not available for the OPC connection and the SHM coupling.
- **New "Custom" scaling type**

There is a new scaling type, "Custom". With this type of scaling, you can specify the input of a high and low raw value.
- **Changing scaling values while a simulation is running**

The high and low scale value can now be changed even while the simulation is running and are applied immediately.
- **Implicit interconnection of coupling signals**

Signals can now be implicitly interconnected in the couplings. This makes it possible to route a signal without the use of charts.

- **Improved export and import of signal properties**

The export and import of signal properties was standardized and adapted to the new features of this version of SIMIT.  
The import of the hardware configuration for the Profibus DP and Profinet IO coupling is now separate from the import of the signal properties.  
For more information on this, see the section: Importing and exporting signals (Page 61).
- **Changes to couplings while a simulation is running**

The only such changes that can be made in couplings during active simulation are those that can also be applied.
- **Implicit interconnection of component outputs**

Implicit interconnections can now be entered in component outputs.
- **New script commands for fixing signals**

There are 2 new script commands for separating and restoring signal connections:

  - FIX
  - UNFIX

You can find additional information on this in the section: Separating connected signals (Page 294).
- **Improvements in automatic model generation**

All functions that include automatic model generation were grouped under a new menu command, "Automatic modeling". This menu command is available in the portal view, in the project view and from the short menu of chart folders.
- **New system variables to query the system time**

There are new system variables to query the system time in component types.  
You can find additional information on this in the manual "SIMIT - Component Type Editor > Syntax of the behavior description > Internal variables and constants".
- **New system functions for processing character strings**

There are new system functions for processing character strings in component types.  
You can find additional information on this in the manual "SIMIT - Component Type Editor > Syntax of the behavior description > The instruction-oriented approach > Internal variables and constants".
- **New search option for the identification of output connectors**

There is a new search option, "Only output connectors". This enables you to determine the source connector when searching for connectors.  
For more information on this, see: section: Search using signal properties (Page 268).
- **New control for opening charts and trends.**

There is a new "Action" control to open charts and trends.  
You can find additional information on this in the section: Action (Page 445).
- **New function in the RCI**

There is a new function in the remote control interface: "RenameProject"  
You can find additional information on this in the manual "SIMIT - Remote Control Interface > Service calls > Simulation control > Rename project".

## 1.2 Product variants

SIMIT is offered in three variants:

- SIMIT STANDARD
- SIMIT PROFESSIONAL
- SIMIT ULTIMATE

### SIMIT STANDARD

SIMIT STANDARD contains the following modules:

- Macro Component Editor  
for creating and editing macro components.
- Dynamic Graphics Editor  
for creating and editing graphics on charts and for animating graphic objects.
- Trend & Message Editor  
for visualizing signal trends and for displaying messages.
- Automatic Control Interface  
for script-controlled execution of simulations, including an editor for creating and editing scripts.
- PROFIBUS DP coupling  
for connecting SIMIT to PROFIBUS DP.
- PROFINET IO coupling  
for connecting SIMIT to PROFINET IO.
- PRODAVE coupling  
for connecting SIMIT to SIMATIC S7 controllers using the PRODAVE interface.
- VIRTUAL CONTROLLER coupling  
for connecting SIMIT to SIMIT VC. SIMIT VC software must also be installed for this coupling to work.

The basic library with its component types and controls is included.

### SIMIT PROFESSIONAL

SIMIT PROFESSIONAL contains all features as provided in SIMIT STANDARD and additionally the following modules:

- PLCSIM coupling  
For connecting SIMIT to S7 PLCSIM.
- OPC coupling  
For connecting SIMIT to an OPC DA server or to OPC DA clients.
- Model change during running simulation  
For changing the simulation model while simulation is running.

### 1.3 Installation and update

- Automatic model creation  
For transferring information from PCS 7 files (IEA and CMT) and Excel files.
- Virtual Time Management  
For running simulation slower/faster than real-time and to set "synchronous" operating mode
- Remote Control Interface (RCI)  
For simulation control and monitoring of the simulation state with an external application.

## SIMIT ULTIMATE

SIMIT ULTIMATE contains all features as provided in SIMIT PROFESSIONAL and additionally the following modules:

- Component Type Editor (CTE)  
For creating your own library components including an editor for creating and editing component types.
- Shared memory coupling  
For connecting SIMIT to other applications using a shared memory area.
- Generic XML import  
For creating simulation models from an XML file.

## 1.3 Installation and update

### 1.3.1 Scope of delivery

The SIMIT product package comprises

- A SIMIT software CD
- A USB dongle with a keyring pendant and lanyard.

You may have bought one or more simulation units for connecting SIMIT to controllers via Profibus DP or Profinet IO. The supplied software must be installed for this.

The SIMIT software CD also includes manuals in electronic form.

---

#### Note

The full range of SIMIT features can only be used when the dongle is plugged in. The dongle must be plugged into a USB port when SIMIT is started and it must remain plugged in until SIMIT is closed.

---



## 1.3.2 Installation and update

The SIMIT installation files are contained on the SIMIT software CD. This CD is a fully functional Installation CD. The SIMIT software CD can also be used to update the installation.

### Requirements

As a prerequisite, you need to be familiar with general use of a personal computer and Windows.

Before you can use SIMIT, the SIMIT software needs to be installed on your computer. Note that you need to be logged onto your PC as administrator to install the software.

Your computer must meet the following minimum requirements:

- Standard PC
- CD drive
- Available USB slot (do not use USB hubs)

Windows operating systems

- Windows 7 Professional SP1, 32-bit, 64-bit
- Windows 7 Ultimate SP1, 32-bit, 64-bit

Additional requirements:

PLCSIM version 5.4 SP5 is needed to use the PLCSIM coupling.

PRODAVE version 6.2 is needed to use the PRODAVE coupling.

The SU software as of version 8.0.13 is required for using the PROFIBUS and PROFINET coupling.

### Installation

To install SIMIT, launch the *Setup.exe* program, which can be found on your SIMIT software CD. Additional required software can be installed along with it.

To install the software, follow the instructions in the installation program. Internet access is not required to install SIMIT.

If you have newly purchased SIMIT, your license for SIMIT Standard, Professional or Ultimate is contained on the dongle. You do not need to enter a license key in this case.

License keys are required to upgrade to a new version of SIMIT, for another SIMIT product version or for additional libraries. Please contact [SIMIT-Activation.industry@siemens.com](mailto:SIMIT-Activation.industry@siemens.com) for license keys. Please provide the dongle number in your email as well as the security code that was delivered to you. You can enter additional license keys using the menu command "Help > Add license key".

---

### Note

The setup for SIMIT was changed for version 8.0. If you have a version of SIMIT older than 8.0 on your computer, it will remain unchanged and version 8.1 is installed in addition.

SIMIT projects that were created with an older version can be edited as normal with the version 8.1. However, once such a project is opened with V8.1 it can no longer be opened or edited with an older version.

---

**Updating**

Run the *setup.exe* file on the SIMIT software CD to update an existing SIMIT installation. Then select *Update* in the installation dialog.

The update does not affect any of your existing projects.

**Note**

To avoid problems arising from country-specific table separators, the import of data files has been generally converted to \*.txt format as of SIMIT V7.1. Data in this format are separated by tabs.

Import files may need to be converted to \*.txt format.

**1.3.3 Adjustable timeout times**

Various operations are monitored by SIMIT and terminated if they exceed a pre-defined time in order to maintain SIMIT operability.

These timeout times are set so that as a general rule they are not exceeded. If you still receive a message that a timeout has been exceeded, check first to see whether it is due to a SIMIT error. If not, you can increase the timeout times in the Windows registry. You can find the corresponding keys under "HKEY\_LOCAL\_MACHINE\SOFTWARE\Siemens\SIMIT\8.0" or "HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Siemens\SIMIT\8.0" on 64-bit operating systems.

Table 1-1 Keys for timeout times

Name	Default setting [ms]	Minimum value [ms]
timeout1	4000	4000
timeout2	10000	10000
timeout3	60000	60000

**1.3.4 Uninstalling the SIMIT software**

SIMIT can be removed via the Control Panel (*Control Panel > Programs and Features > Uninstall a program*).

During removal all files and registry entries that were created during installation are removed. SIMIT projects and user-created objects are not uninstalled.

## 1.4 Basics of SIMIT

### 1.4.1 Overview

SIMIT can be used as an input and output simulator for test signals for your controller, but also as a complete plant simulator. SIMIT offers application options for testing and commissioning automation software in every situation.

Even if you initially only use SIMIT as a user interface for signal testing, you can add simulation models later at any time. This allows you to simulate your plant behavior and perform dynamic tests.

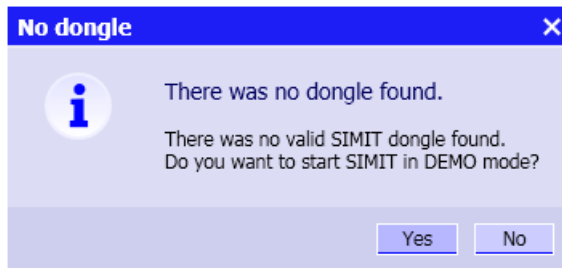
The following aspects are of particular interest when using SIMIT:

- **Coupling**  
You can define the interface via which you wish to couple SIMIT to your automation system and specify the signals that you want to access in SIMIT.  
You already have the option of specifying and displaying signals within SIMIT when creating a coupling.
- **Visualization**  
You can create charts with operating and display controls by either manually creating individual charts or generating them automatically from existing data – such as the data in the coupling.  
Graphic tools such as lines, rectangles, ellipses, etc. can be used statically or animated with signals, thus allowing you to efficiently build comprehensive two-dimensional animations.  
These visualizations provide an ideal overview of all signals in your system. Values belonging to the same part of your system can also be shown in a grouped fashion, independent of the physical addresses they use.
- **Simulation**  
You do not need any expert knowledge in simulation in order to build a simulation. Just use the graphical user interface to combine existing components from the SIMIT libraries and enter the matching parameters.  
Components with a diverse set of simulation functions are available in the SIMIT basic library. These components cover a wide range of applications: from simple arithmetic and logical relationships to drive simulations and complex system simulations. Detailed descriptions of the library components and how to use them in charts can be found in the section: Libraries (Page 297).

### 1.4.2 SIMIT in DEMO mode

#### Starting SIMIT in demo mode

If you start SIMIT without plugging a SIMIT dongle into your computer, you are asked whether you want to start SIMIT in demo mode.



If you have a valid SIMIT license and therefore a SIMIT dongle, this demo mode is not relevant for you. It is intended purely to give you an idea of the operation and performance of SIMIT before you buy a license.

Only the following SIMIT function modules are available in demo mode, regardless of any license keys you may have:

- The Macro Component Editor (MCE)
- The graphics editor
- The message system and trends
- Automatic modeling.

In demo mode, SIMIT offers only limited functionality with regard to the following elements:

- **Saving and archiving**  
You can save projects, templates and macro components in demo mode, but you can only use the projects, templates and macro components created in demo mode on the computer on which they were created.

---

#### Note

The projects, templates and macro components created in demo mode are not compatible with the full version of SIMIT.

You cannot archive projects in demo mode.

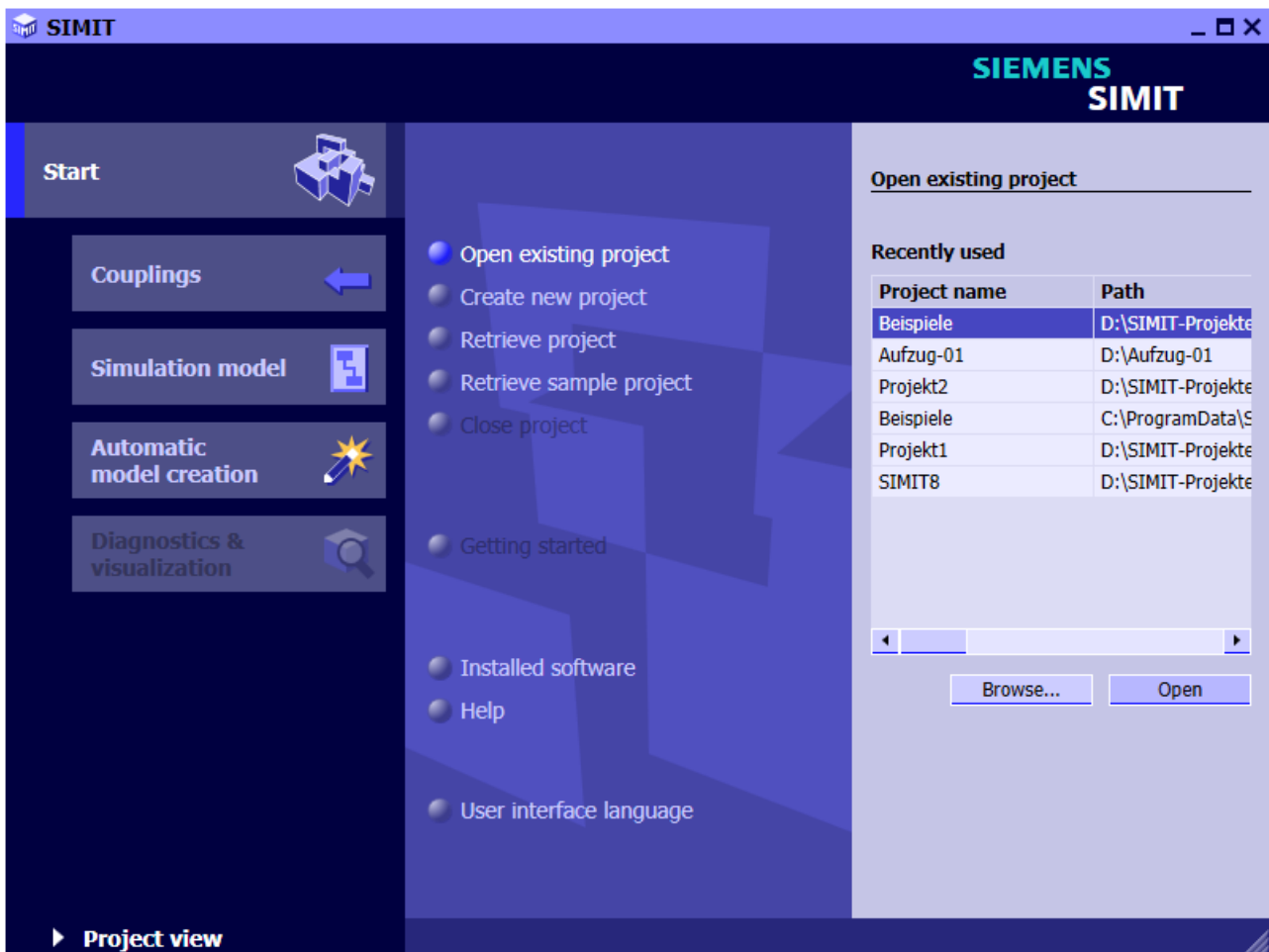
---

- **Opening and retrieving**  
In demo mode, you can only open projects that were saved on this particular computer in demo mode. Projects of a full version cannot be opened.  
You can retrieve projects that have been archived in a full version. If the retrieved project is changed in demo mode, it can no longer be used in the full version.
- **Address area**  
The address area of the coupling is limited to the following:  
IB0 – IB7 and IB64 – IB85  
QB0 – QB7 and QB64 – QB79
- **Run time**  
You can use SIMIT in demo mode for as long as you want, but the run time of a simulation is limited to 45 minutes. At the end of the 45 minutes, the simulation will end automatically. You can restart it once it has ended.
- **Number of couplings**  
You can only create one coupling in a SIMIT project in demo mode. Only one PLCSIM coupling or Virtual Controller coupling can be selected as the coupling type.

- **Project folder**  
In demo mode, you can only store projects in a designated location in the work area of SIMIT.
- **Libraries for macro components and templates**  
In demo mode, you can only store macro components and templates within the work area of SIMIT . You cannot open other library folders.

### 1.4.3 Portal view

Once SIMIT starts, the portal view opens.



The clear layout of the portal view helps you to quickly become familiar with SIMIT. The most important basic functions can be selected directly:

- Manage projects
- Create couplings
- Create simulation models
- Automatic data import from different file formats

Additional functions are:

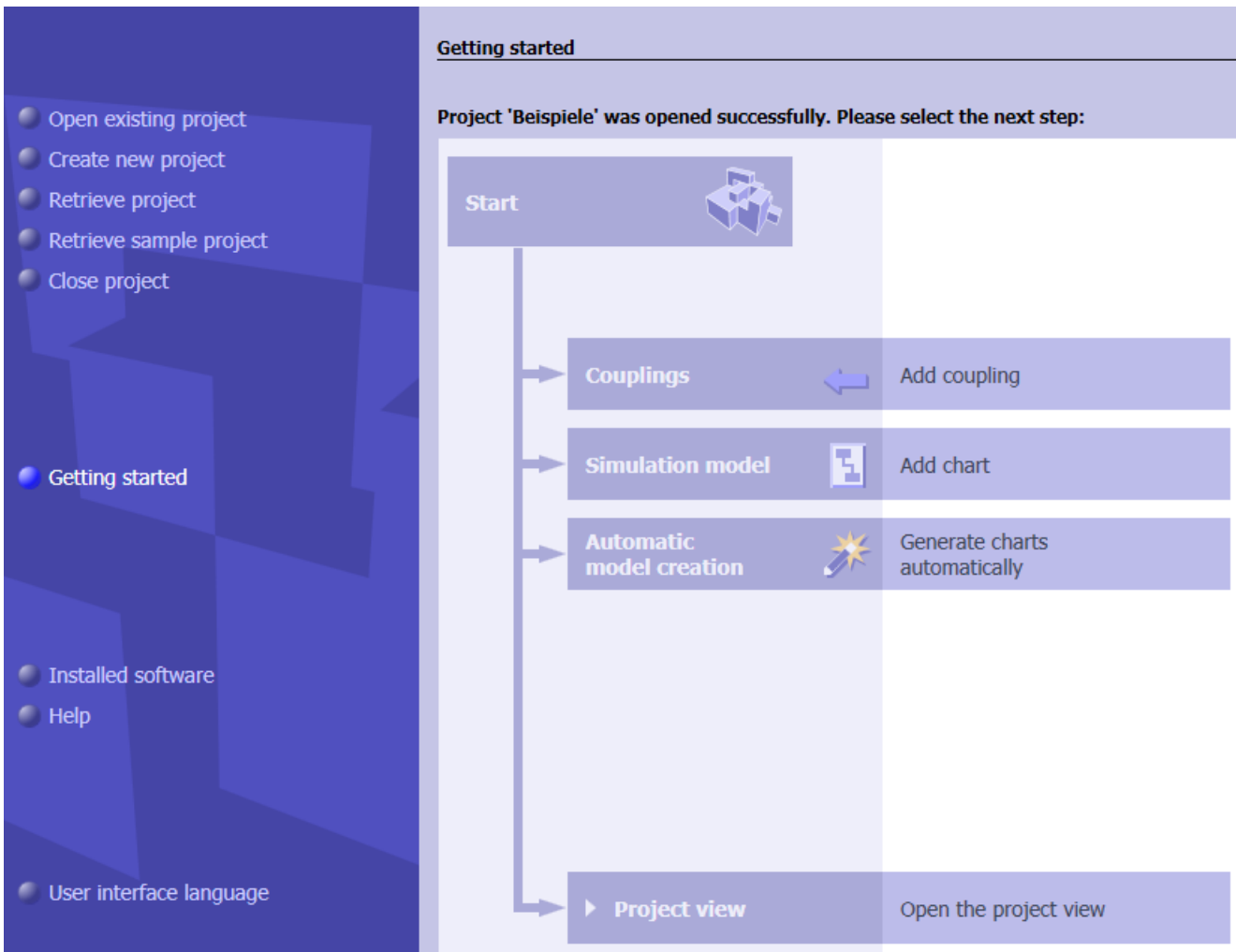
- Perform consistency checks
- Open the "Find & Replace" function
- Create a new trend
- Process archives

Click "Project view" to change to the project view. Full functionality of SIMIT is only available in the project view.

**Getting started**

After opening, creating or retrieving a project, the "Getting started" section opens. This is where you can add the basic objects that are required for the function of a SIMIT project.

- Add coupling
- Add chart
- Generate charts automatically



You can perform these steps here or go to the project view to continue editing and perform simulation.

## 1.4.4 The SIMIT graphical user interface

The SIMIT user interface consists of the following components:

The **menu bar** and the **toolbar** offer you easy access to the functions available in SIMIT. There are additional functions available in the shortcut menus.

The **Project window** shows the currently open project in a tree view.

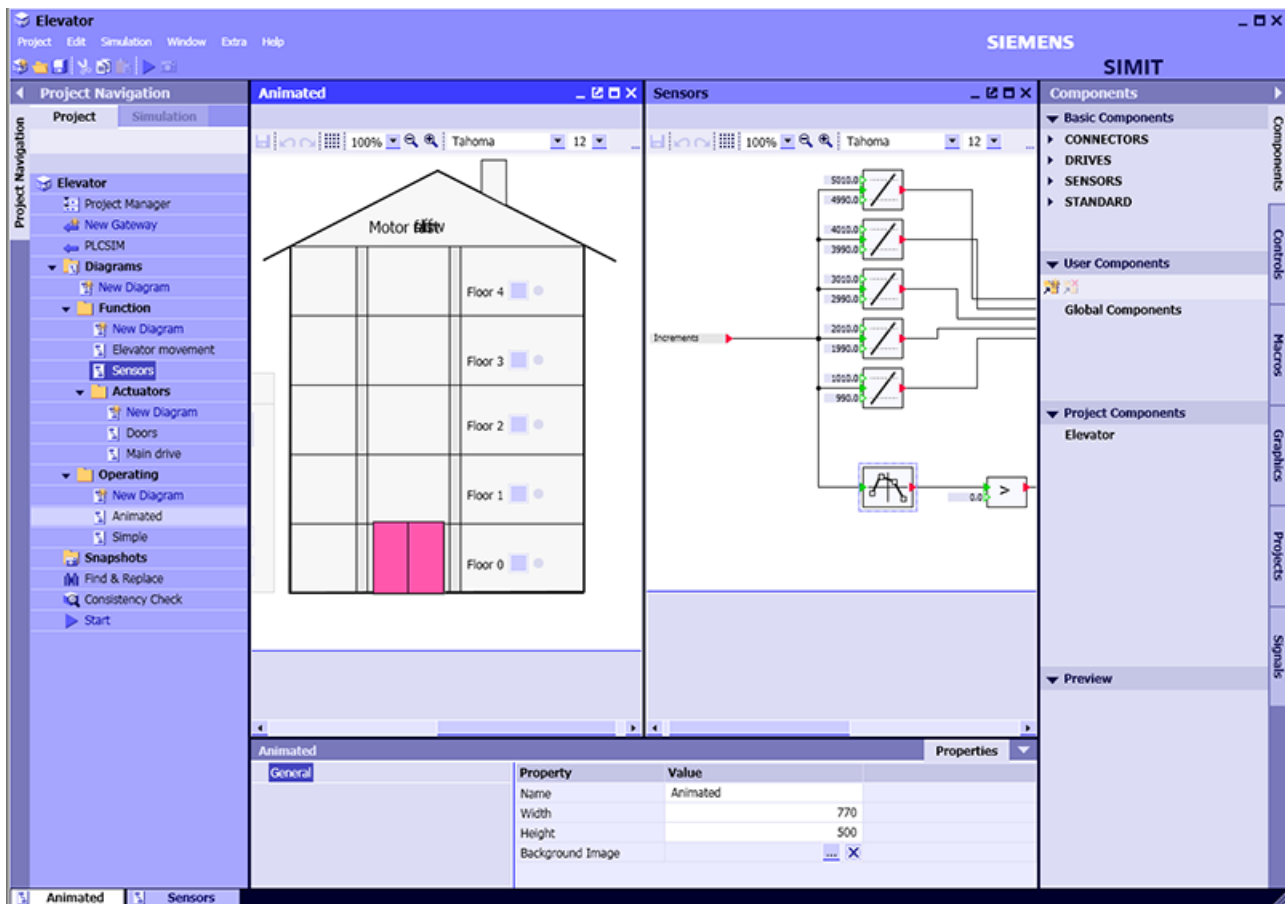
The editors are opened for editing in the **work area**. Every editor contains a toolbar for quick access to the editor-specific functions.

Editor-specific tools such as library components, controls and graphic tools can be found in their own task cards in the **Tools window**.

The properties of a currently selected object are shown in the **Properties view**.

The **editor bar** at the bottom left of the interface allows you to toggle between opened editors.

The **status line** in the bottom right corner of the interface shows information on the current status of SIMIT.



All editors are opened in the work area. The tools window only contains the specific task cards for each editor. The work area can be divided in order to open two editors side-by-side or underneath one another in the work area.

**The window buttons of an editor:**

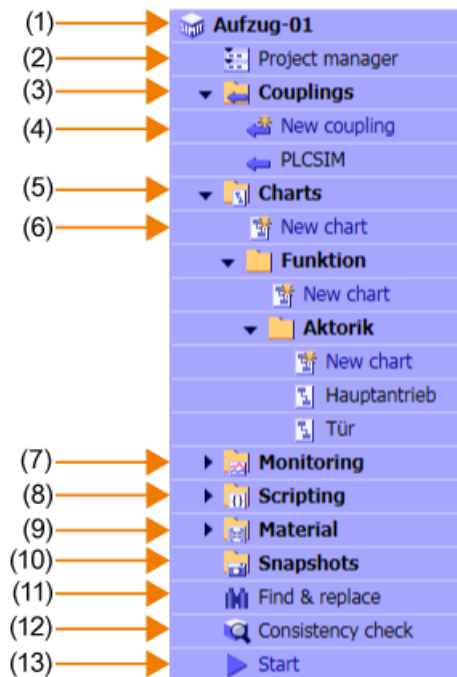


The following functions can be performed using the window buttons of an editor:

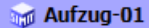
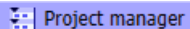
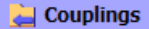

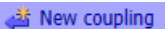
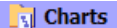


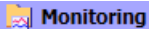
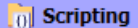
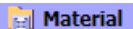
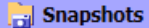
- Minimized , i.e. reduced to its entry in the editor bar
- Reduced to occupy only part of the work area
- Maximized to occupy the entire work area
- Close
- Detach an editor with tool and property window from the work area as a separate window
- Insert an editor window that was detached from the work area back again into the work area
- Anchor a detached editor window in the foreground




**The project window**

Simulation projects are administered in the project view. A SIMIT project is divided into the following tree nodes:



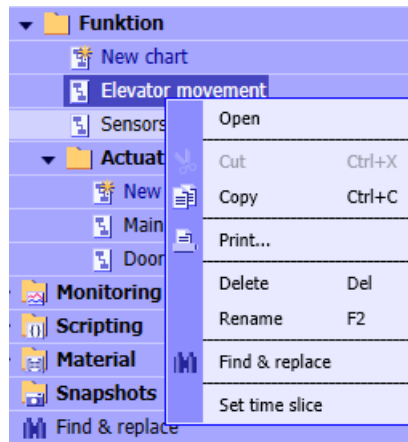


- |      |  |  |
|------|--|--|
| (1)  |  <b>Aufzug-01</b>       | Current project name   |
| (2)  |  <b>Project manager</b> | Tree node for the project manager. Elements from other projects can be copied into the currently opened project in the project manager.<br>You can find additional information on this in the section: View and functions of the Project Manager (Page 165).   |
| (3)  |  <b>Couplings</b>       | "Couplings" folder: Folder for storing the couplings of the current project.<br>You can find additional information on this in the section: Couplings (Page 45).   |
|      |                         | Symbol for the couplings. Couplings create the connection between SIMIT and a controller or another application.   |
| (4)  |  <b>New coupling</b>    | Tree node to create a new coupling.<br>You can find additional information on this in the section: Creating couplings (Page 49).   |
| (5)  |  <b>Charts</b>          | "Charts" folder: Folder for the charts.  |
|      |                         | Symbol for the charts. Charts contain a simulation model created using library components and controls.  |
| (6)  |  <b>New chart</b>      | Tree node to create a new chart.<br>You can find additional information on this in the section: Creating simulations (Page 26).  |
| (7)  |  <b>Monitoring</b>    | "Monitoring" folder: This folder contains the following functions and is also their storage location: <ul style="list-style-type: none"> <li>• New trend</li> <li>• Messages</li> <li>• Archive</li> </ul> You can find additional information on this in the section: The functions of the Trend and Messaging Editor (Page 249). |
| (8)  |  <b>Scripting</b>     | "Scripting" folder: This folder contains the "New script" function and is the storage location for existing scripts.<br>You can find additional information on this in the section: Creating a script (Page 278).  |
| (9)  |  <b>Material</b>      | "Material" folder: This folder contains the "New list" function and is the storage location for existing material lists.<br>This folder is only relevant if the "CONTEC" library has been installed.<br>You can find additional information on this in the section: Modeling of the objects (Page 562).                            |
| (10) |  <b>Snapshots</b>     | "Snapshots" folder: Folder for storing snapshots of the simulation. When simulation is running, the folder also contains the "New snapshot" function.<br>You can find additional information on this in the section: Running simulations (Page 28).  |


- (11)  Find & replace Tree node for Find & Replace. Searches can be made for signals, components, connectors, and graphic texts. You can find additional information on this in the section: Find & replace (Page 265).
- (12)  Consistency check Tree node for performing the consistency check. The consistency check verifies the project for formal errors. You can find additional information on this in the section: Consistency check (Page 273).
- (13)  Start Tree node for starting simulation.

A simulation project does not necessarily require all these elements. The respective functions are executed with a double-click.

Alternatively, you can access the features of any element in a SIMIT project in its shortcut menu (right-click the corresponding tree node). The figure below shows the shortcut menu of a chart by way of example. Comparable shortcut menus exist for many elements of a SIMIT project.



### 1.4.5 Creating simulations

A simulation in SIMIT is built with the help of pre-defined components and controls that are placed on **charts**. Charts can be added to the project using the *New chart* entry () in the project tree; they can be opened by double-clicking the chart in the project tree. The tools window will show components and controls that can be used on charts.

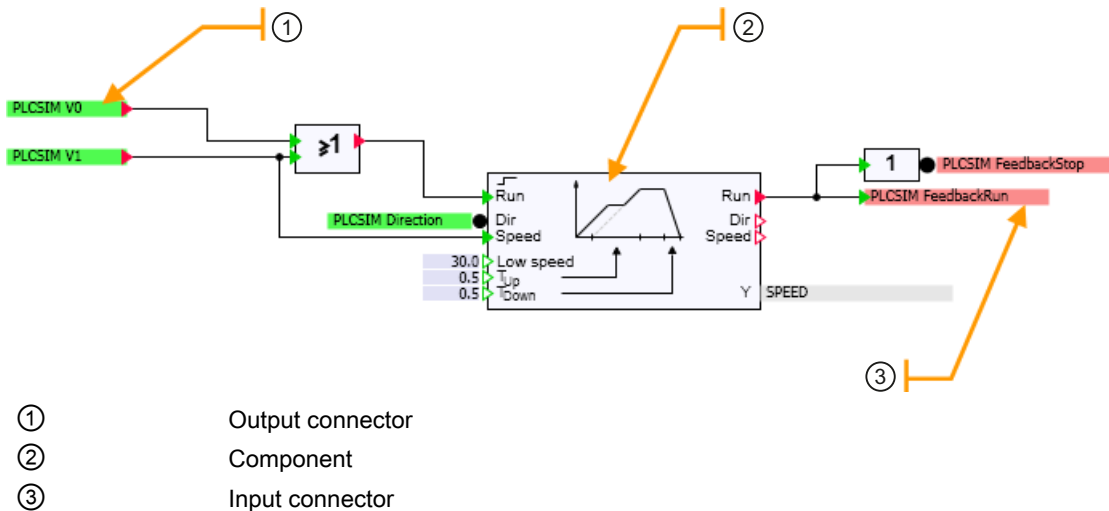
**Components** can be found in the "Components" task card. The *STANDARD* folder provides components for various logical and arithmetic functions. The "Components" task card also contains components for simulating drives (*DRIVES* folder) and sensors (*SENSORS* folder).

The "Components" task card provides elements for entering and displaying values. Input elements may be used to set values while a simulation is running. Similarly, display elements may be used to dynamically show values from a running simulation.

Simulations are assembled on charts using components and controls and connected to the PLC using connectors.

### Note

The minimum size for a chart is 20 x 20 pixels.



To do this, drag-and-drop components or controls from the library onto a chart, connect their connectors and enter the parameter values.

In order to connect connectors, which means the inputs and outputs of components and controls, move the mouse over the connector you want to connect. When the appearance of the mouse cursor changes, a connection can be established by clicking the left mouse button. Now move the mouse cursor over the connector that is to be connected and click the left mouse button again. A connection is now established and is shown as a connection line. You may also drag-and-drop one connector on top of another to establish a connection.

You can assign the parameters for a component or control in the fields at the inputs or in the property view of the component or control. To enter a value in an input field, open the input field with a double-click. Close the input by pressing *RETURN*. The property view can be opened by left-clicking a component or control.

The input and output signals of the PLC are administered in the **couplings**. On charts, these signals are shown as **connectors**: Output signals are shown as green output connectors (*Output*), input signals are shown as red input connectors (*Input*). Input and output connectors can be placed from the coupling onto charts with drag-and-drop. Split your work area using the *Window > Horizontal* menu command, and open both the coupling and the chart. Drag the signal of interest from the coupling onto the chart by grabbing it at the left border of the coupling and pressing the *Shift* key. Connect the connection point of the connector with the connector of a component.

### Note

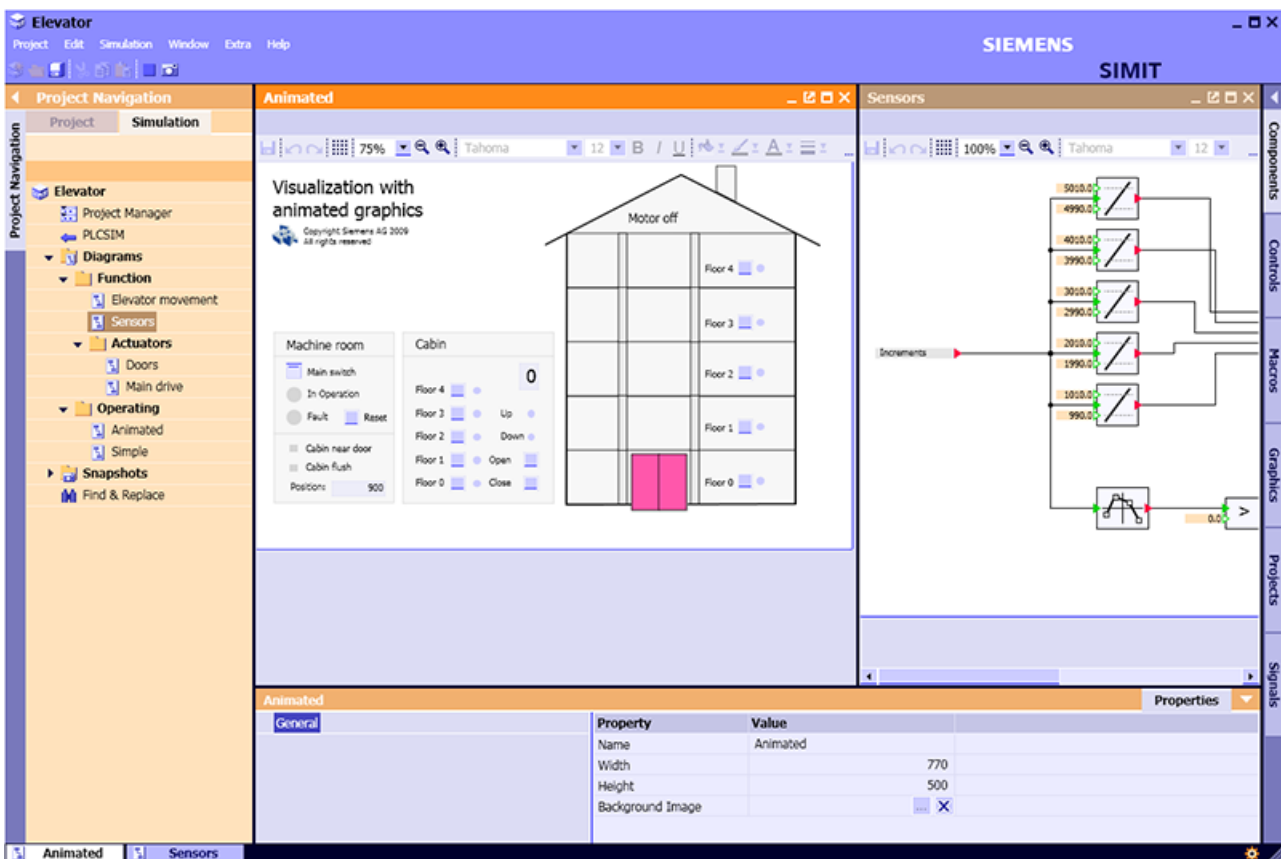
The coupling configuration must first be saved before a signal can be extracted from it.

You may also drag-and-drop input and output connectors from the *Signals* task card onto a chart. It is advisable to filter the signals according to their origin *Coupling* and according to their signal type *Input* or *Output*. Now drag the required signals onto the chart while keeping the *Shift* key pressed.

### 1.4.6 Running simulations

You can start a simulation using the toolbar (▶), the menu bar (*Simulation* > *Start*) or by double-clicking *Start* (▶ *Start*) in the project window. Once the simulation is running, the SIMIT color scheme changes from blue to orange.

While the simulation is running you may open and close any chart or coupling. While a simulation is running you can select components on charts using the right mouse button. Open the property view to observe the changes in the values at the inputs and outputs of selected components while the simulation is running.



You can use the *Simulation* > *Snapshot* command from the menu bar at any time to save a snapshot, which means the current state of your simulation. You can also create a snapshot using the toolbar (📷) or *New snapshot* (📷 *New Snapshot*) in the project window. Each snapshot is saved in the Snapshots folder (📁 *Snapshots*) and is assigned the current date and time as its name. You may rename or delete snapshots or move them into a subfolder.

When you load a snapshot using the shortcut menu, the simulation is set to the same state that is stored in the snapshot, and the simulation will continue running starting from this state. If the simulation is not running yet, it is automatically started when you load a snapshot.

Use the menu command *Simulation > Exit* or the toolbar (  ) to end the simulation.

## 1.4.7 Virtual time management

### 1.4.7.1 Synchronous and asynchronous operating modes

There are two different strategies according to which SIMIT processes the model and the couplings: asynchronous and synchronous. The operating mode depends on the project. It is set in the Project manager:

Projekt1	
Property	Value
Project Location	D:\SIMIT-Projekte\Projekt1\Projek...
Project Version	AA12320-926734-0.3 ...
Readonly	<input type="checkbox"/>
Default Scale	1 pix : 1 mm ▼
Cycle 1 [ms]	50
Cycle 2 [ms]	100
Cycle 3 [ms]	200
Cycle 4 [ms]	400
Cycle 5 [ms]	800
Cycle 6 [ms]	1600
Cycle 7 [ms]	3200
Cycle 8 [ms]	6400
Operating mode	Asynchronous ▼

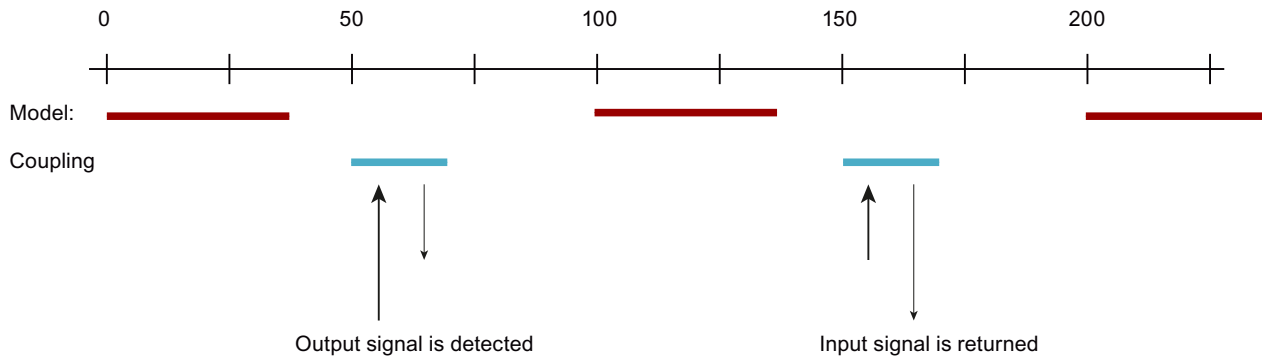
#### Asynchronous operating mode

The model calculation of the individual time slices and processing of the couplings in SIMIT is performed time-controlled.

If SIMIT does not complete the simulation model of a time slice within the scheduled time, one or more processing cycles are omitted and the other time slices are not calculated. The calculation of the simulation model only continues when all time slices can calculate once again.

If SIMIT cannot calculate a coupling in the scheduled time, one or more processing cycles are omitted once again but the calculation of the couplings in other time slices and the calculation of the simulation model are not affected. This means if couplings are blocked by a communication problem, this does not affect the overall processing.

In order to achieve the fastest possible response times, couplings are scheduled with a time offset compared to the model calculation:

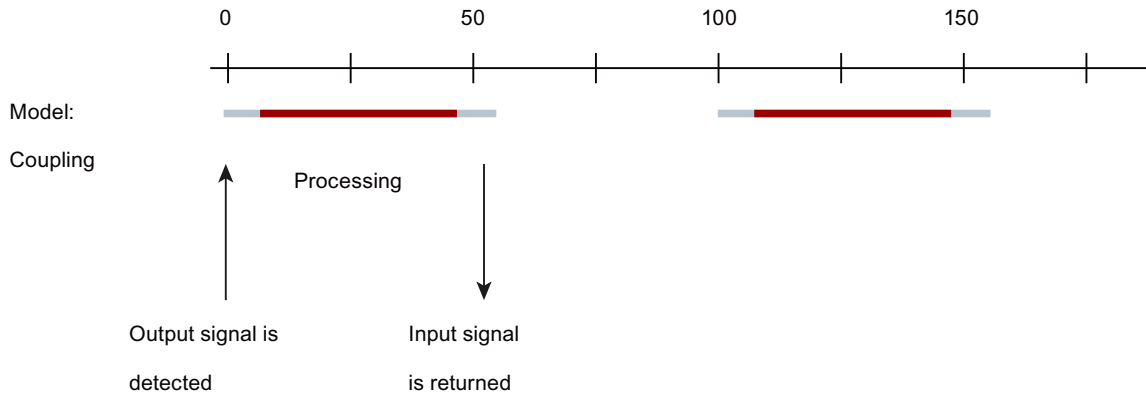


If the model calculation or the data communication of couplings takes longer than half of the cycle time, this time offset does not result in faster response time.

**Synchronous operating mode**

With synchronous mode, modules and couplings are calculated in a precisely specified sequence. The next action does not start until the previous action has finished.

The coupling is divided into the acquisition of output signals and the writing of input signals:

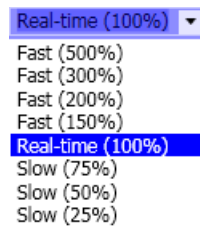


This results in better response times with short cycle times. However, each module and each coupling can block the entire simulation execution.

**1.4.7.2 Acceleration and delay of the simulation time**

In order to slow the course of the simulation or accelerate it, the relation of simulation time to time actually elapsed can be changed.

This value is set via a drop-down list in the toolbar:



Maximum acceleration is limited in such a way that the fastest cycle time of the project is at least 10 ms taking into account the acceleration. This means acceleration may not be possible depending on the cycle time. Only suitable values are offered for selection.

### 1.4.8 Display of the simulation load

A simulation model consists of

- The library components used
- The couplings

If you are using components of the FLOWNET or CONTEC libraries, a higher-level solution procedure is provided in addition.

All simulations are computed cyclically. The cycle time can be set with a time slice. Each component of a chart and each coupling are assigned a time slice. The solution procedures are calculated in the time slice in which the components are involved. You set the absolute cycle time for a time slice in the project manager.

The cycle time specifies the time frame in which the calculations are to be performed (for example, every 100 ms). If not all the calculations of the time slice can be performed within the cycle time, however, this time frame cannot be met.

SIMIT determines the simulation load to assess the degree to which the computer is busy. It indicates the percentage ratio of actual computation time for the configured cycle time. Based on the simulation load, you can assess whether the cycle time for all calculations is sufficient. SIMIT displays only the total worst value.

Example:

	Time slice 1	Time slice 2
Configured cycle time	100 ms	200 ms
Calculation time	60ms	80ms
Simulation load	60%	40%
Total simulation load (load value)	60%	

Meaning of the percentage load value:

- **Load value well below 50%:**  
The simulation can be calculated in the configured cycle times.  
The data is exchanged as planned between the simulation model and the couplings.
- **Load value reaches or exceeds 50%:**  
The simulation can be calculated in the configured cycle times.  
The data is no longer exchanged between the simulation model and couplings with a time offset. Additional cycle offsets are created.  
The simulation values are passed to the couplings or read from the couplings in a later processing step.  
If no couplings are contained in a simulation project, the 50% limit is irrelevant.
- **Load value reaches or exceeds 100%:**  
The simulation is in overload.  
The simulation model can no longer be calculated in the scheduled cycle time.  
Computational steps are omitted.

The simulation load is shown while simulation is running via an symbol in the low right pane. The current simulated load can be read from the green circle segment (full circle corresponds to 100%):



### Load fluctuations

The load value is not always constant. A fluctuating load can be caused by:

- The simulation model itself can create more load in each calculation step due to the configuration
- A time slice is interrupted by a time slice with a higher priority.
- Other applications are running in the background in addition to SIMIT
- Internal processes in both the operating system and in the .NET environment used by SIMIT may delay the simulation calculation

In order to keep the graphical representation of the interface calm, the load values are smoothed over several increments.

---

### Note

If the load value always jumps back and forth between two values, the problem may be that the processor of your computer keeps changing the clock frequency for energy saving reasons. You can prevent this by adjusting the system settings of your computer, for example, (Control Panel > Power Options > Select a power plan > High performance).

---

### Note

The Windows Task Manager is not suitable for evaluating the simulation load. On the one hand, the distribution of the simulation model on multiple processors or cores has to be taken into consideration; on the other hand, the task manager only indicates the computing power but not whether the configured cycle time cannot be met due to communication delays.

---



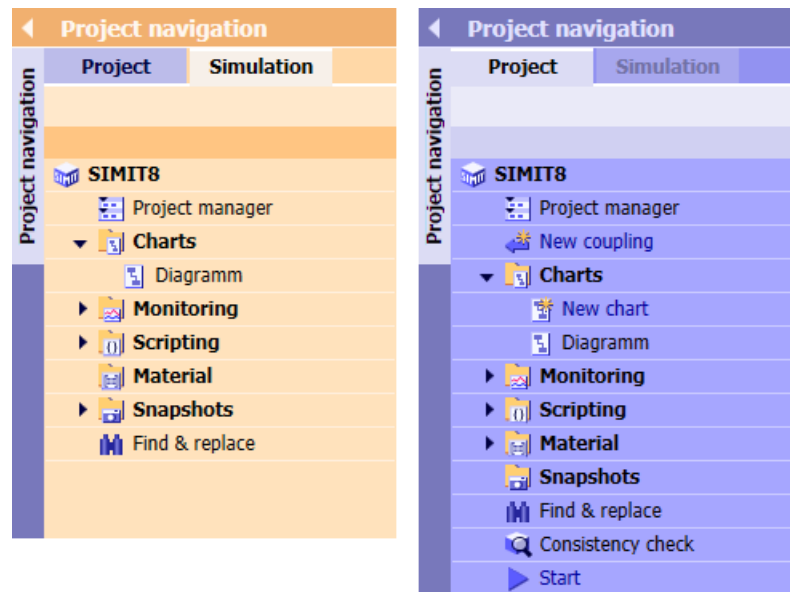
**Note**

If the simulation load is too high, you can get information about the cause using the "SimulationLoad" component.

You can find additional information on the component in the section: SimulationLoad - simulation load (Page 364)

### 1.4.9 Changes in a simulation project while a simulation is running

You can make changes to the simulation project while the simulation is running. To make changes while the simulation is running, switch between *Simulation* and *Project* in the project tree.



You can open charts in both project views, but for different purposes:

- If you are in the blue project view, charts are opened for editing. The changes that you make take effect the next time the simulation is started or when you select the "Activate changes" menu command.
- If you are in the orange simulation view, charts are opened for operation. Their content corresponds to the simulation that is running and cannot be changed.

The same chart can be opened simultaneously in both modes. When the simulation ends, the chart opened from the simulation tree closes automatically.

While the simulation is running, the following restrictions apply to using SIMIT:

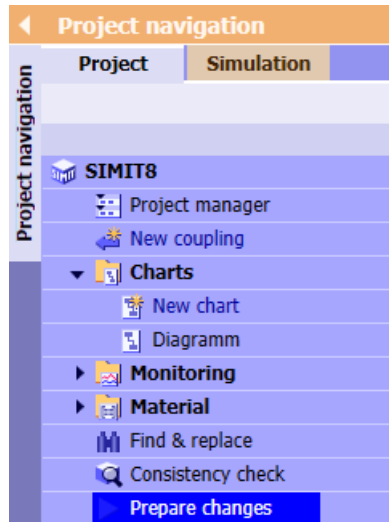
- No new couplings can be created and existing ones cannot be deleted.
- Only implicit interconnections can be edited in the couplings.
- All functions of "Automatic modeling" are disabled.

## Preparing and activating model changes

Model changes can also be activated during the simulation.

Follow these steps:

1. Switch to the project view while simulation is running.
2. Make the desired changes to the simulation model.
3. Click "Prepare changes". The preparation may take several minutes.



4. Once the "Prepare changes" function is completed, the menu command changes to "Activate changes". Click "Activate changes" to apply the changes to the ongoing simulation.

---

### Note

While the changes are activated, the simulation model does not process any values for some time. During this time, I/O devices may not respond. The length of time depends on the size of the simulation model.

If the current simulation state is transferred to the new configuration, the values can be identified and assigned based on the signal name. Values for which there is no equivalent are assigned initial values.

---

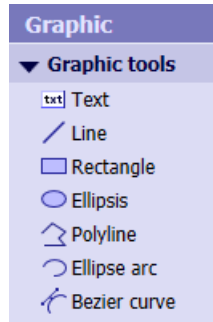
## 1.4.10 Visualizing simulations

### 1.4.10.1 Visualization with graphics

Graphics enable you to visualize the simulation model and add an explanatory texts. You can create static and animated graphics and link them together. Graphics are displayed and edited in the chart editor.

## Static graphic

To use a static graphic, drag the desired graphic element from the "Graphics" task card and drop it into the chart editor.



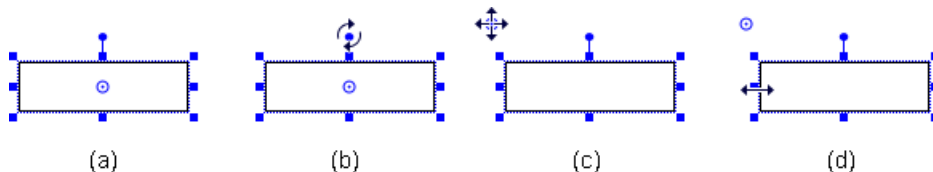
Edit the graphic element with the functions provided in the chart editor toolbar:



The following functions can be called up from the toolbar:

- Text: Set the font, font size and color for text.
- Lines: Select a color for the fill and lines, change the weight and style of lines.
- Graphic element: Flip graphics, mutually align, distribute or group multiple graphics. Group graphics, move individual or grouped graphics to the foreground or background.

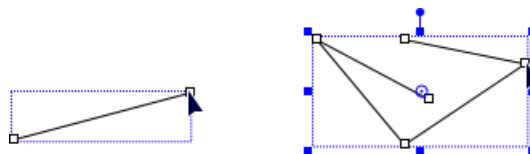
Multiple graphics can be selected with a lasso frame or by holding down the Shift or Ctrl key.



When a graphic object is selected, a selection frame is displayed as shown (in the figure "Editing rectangles (a)"). You can change the size of the graphic element using the superimposed blue squares. To do this, move the mouse cursor over a square until its appearance changes as shown in the figure "Editing rectangles (d)". With the mouse button pressed, drag the graphic element to the desired size.

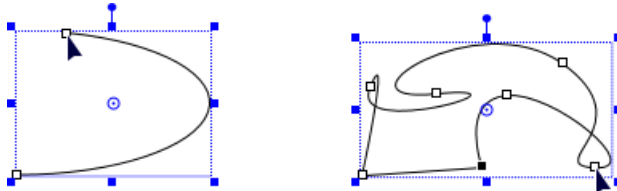
You can also rotate the graphic element by any desired angle with the upper blue point. To do this, move the mouse cursor over the blue point until its appearance changes as shown in the figure "Editing rectangles (b)". With the mouse button pressed, rotate the graphic element by the desired angle.

You can move the angle of rotation as desired (figure "Editing rectangles (c)").



Both endpoints of a line can be moved, whether for a simple straight line or for a line segment within a polyline. After a line is selected, a small rectangle appears at both endpoints. Move the mouse cursor over an endpoint until its appearance changes as shown in the figure above. You can now move the endpoint while pressing the left mouse button.

In the same way, you can move the endpoints of curved lines, which means you can move the endpoints of elliptical arcs or the points of a Bezier curve:



Graphics are arranged on a specified grid. You can remove the grid when editing graphics by pressing the Alt key.

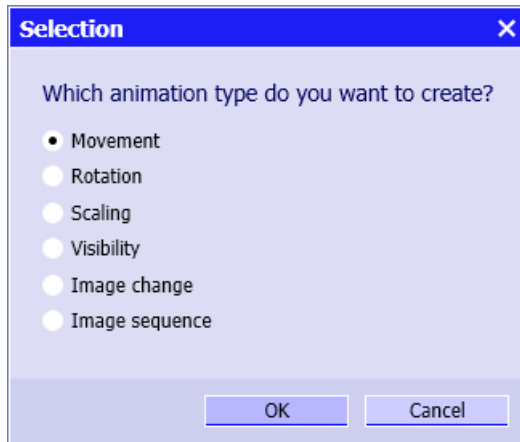
The graphics can be edited in the properties view.

### Animated graphic

Animations can be created for each graphic element. For example, if you draw a rectangle, you can see the "Animations" property in the properties view of the rectangle.

Rectangle		
General	Property	Value
Appearance	Name	Rectangle
Layout		
▼ Animations		
<b>New Animation</b>		

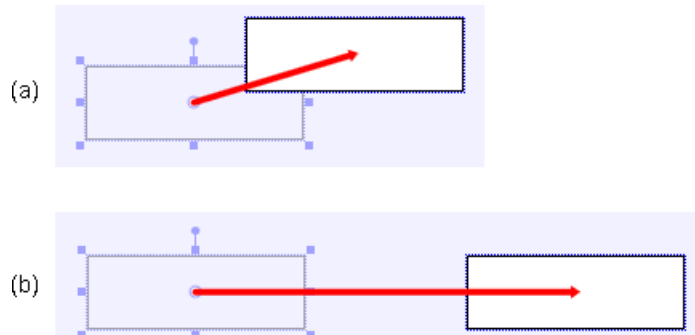
You create a new animation for the selected graphic element by double-clicking "New animation". The following dialog box opens:



- *Movement* of the graphic element on the chart surface,
- *Rotation* of the graphic element around the rotation axis

- *Scaling*, which means changing the size of the graphic element
- *Visibility* to show or hide the graphic element
- *Image change* and *image sequence* to show images contained in image files on the graphic element.

You may define several animations for a single graphic element and can both move and scale an object, for example. Image change and image sequence are mutually exclusive, which means you can create either an image change or an image sequence.



If you select "Movement", for example, a copy of the graphic element appears offset from the original object. A red arrow connects the copy to the original element as shown in the figure above under (a). This red arrow visualizes the movement of the object. Now use the mouse to move the copy to the desired destination position as shown in the figure above under (b) for horizontal movement.

In the properties view (see figure below), enter the signal that supplies the values for the movement. To do this, simply drag the signal from the "Signals" task card into the property field. Enter the initial value and end value. If you then start the simulation and change the signal value, the graphic element is moved horizontally. For signal values within the range specified by the initial and end values, movement takes place along the specified arrow. For signal values outside the defined range, the movement continues in a straight line beyond the arrow. The arrow thus specifies the direction of the linear movement, and the initial and final values only define the scale of the graphic element on the chart surface.

Rectangle		
General	Property	Value
Appearance	Signal	
Layout	Initial Value	0.0
▼ Animations	Final Value	100.0
<b>New Animation</b>	Distance	X: 100.0 Y: -30.0
<b>Movement</b>		

Several movement animations for a graphic object are superimposed and thus allow for movement along curved paths.

Given a group of several graphic objects you may animate both the group as a whole or each individual graphic object within the group.

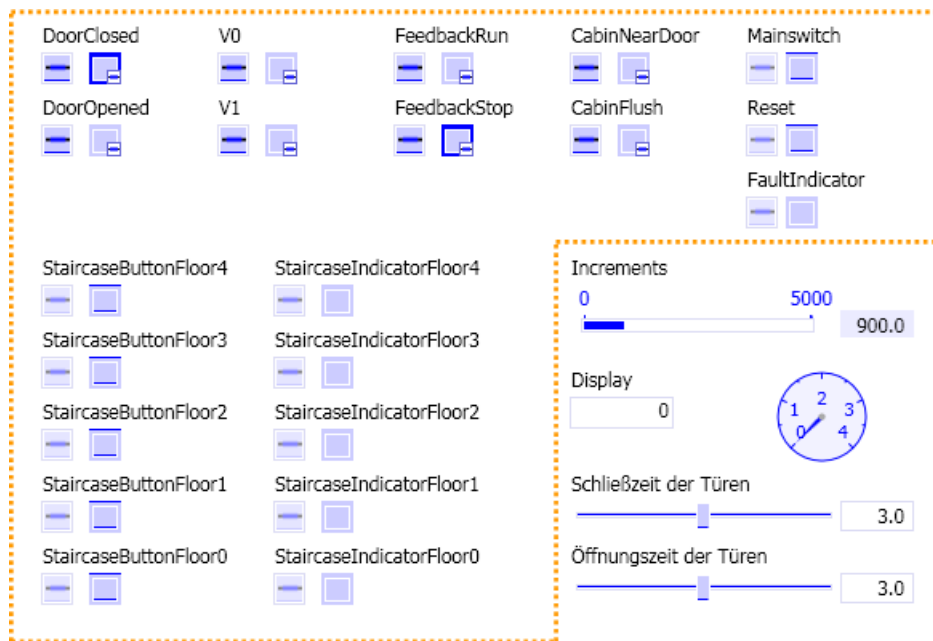
The signal currently assigned to which the animation is displayed under "Signal" in the properties view:

Cabin		
Property	Value	
General	Signal	PLCSIM Increments
Appearance	Initial Value	1000.0
Layout	Final Value	5000.0
▼ Animations	Distance	X: 0.0 Y: -279.5
<b>New Animation</b>		
<b>Movement</b>		
▶ Empty cabin		
▶ Symbol		
▶ Left door		
▶ Right door		

### 1.4.10.2 Visualizing signals

SIMIT allows you to arrange controls on charts and use these controls to display and set signals. You can thus assemble controls on various charts in arbitrary groupings. For each chart, you can specify not only the signals that are displayed but also their arrangement.

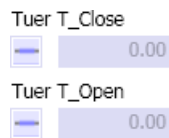
The following figure shows an example for the visualization of coupling signals. All controls shown within the dotted frame can be dragged and dropped from the "Signals" task card or from the coupling without any further modification.



The output signal Display on the other hand is displayed using two controls: one digital display and one analog display. You can, for example, obtain a digital display by dragging the signal from the coupling or the task card and deleting the signal splitter. You also may create a new digital display or analog display by dragging the control from the Controls task card onto the chart and then entering the signal name in the properties. You may enter the signal name manually or drag-and-drop the signal from the task card into the property field.

Analog Display#1	
General	Name
Connector	Signal
View	Profibus Display

The two signals for the closing and opening times of the doors show that not only coupling signals but also component input signals can be linked with controls. When dragging these two signals from the "Signals" task card, you will see – as shown in the figure below – that for each one a control and a signal splitter are inserted into the chart. Because these are analog signals, the *Digital input* control is used for this purpose by SIMIT.



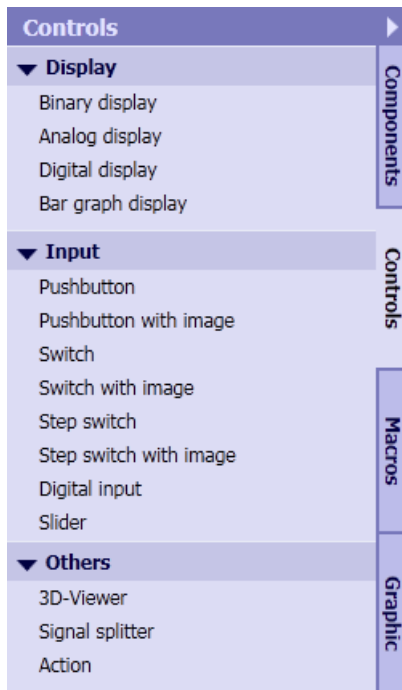
Controls for signals can thus be inserted for any input or output signal in the "Signals" task card. This means that not only coupling signals but also controls with signal splitters for the input and output signals can be inserted into a chart per drag-and-drop.

The example above, however, uses sliders rather than digital inputs to specify the two closing times. This control is available along with other controls in the "Controls" task card. The controls for displaying signal values can be found there.

- Binary display,
- Analog display,
- Digital display and
- Bar graph display

and the controls for entering signal values

- Pushbutton,
- Switch,
- Step switch,
- Digital input and
- Slider.



The control for pushbuttons, switches and step switches is also available *with an image*. You can define the appearance of these controls on the chart by assigning appropriate images, and you can use photographs, for example, for a very realistic visualization of the controls.

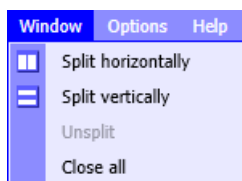
Controls can be linked with any signal from the "Signals" task card. In addition to using them with the input and output signals of couplings and components, you can also use controls to show state values of components and to set parameters that are online changeable, which means parameters that can be changed while the simulation is running.

### 1.4.10.3 Visualizing coupling signals

#### Dragging coupling signals from a coupling

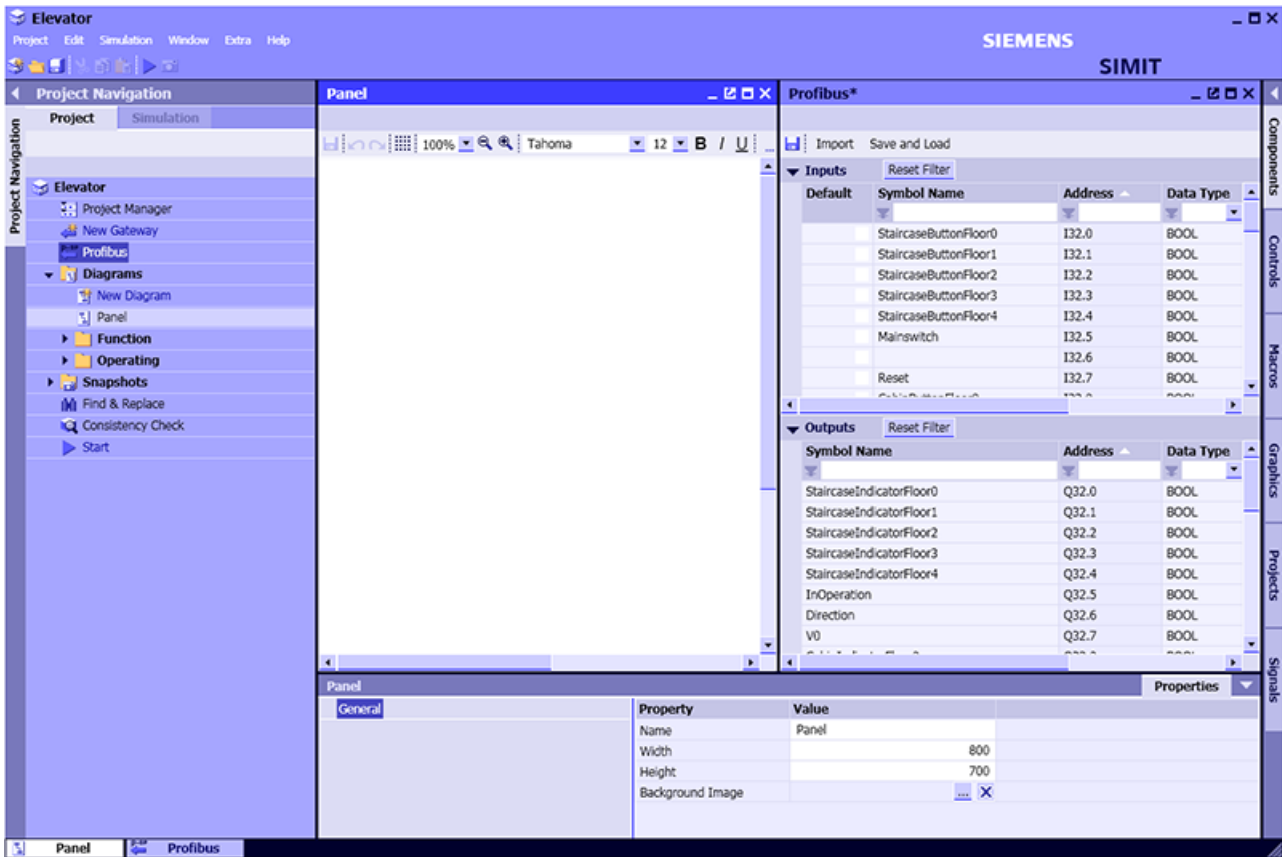
When a simulation is running, a control with a signal splitter is provided for each signal in the coupling. This control along with its associated signal splitter can also be placed on charts for any coupling signal.

To drag a signal from a coupling onto a chart you must open both the coupling and the chart in the work area. First split your work area using the menu command *Window > Split horizontally*.

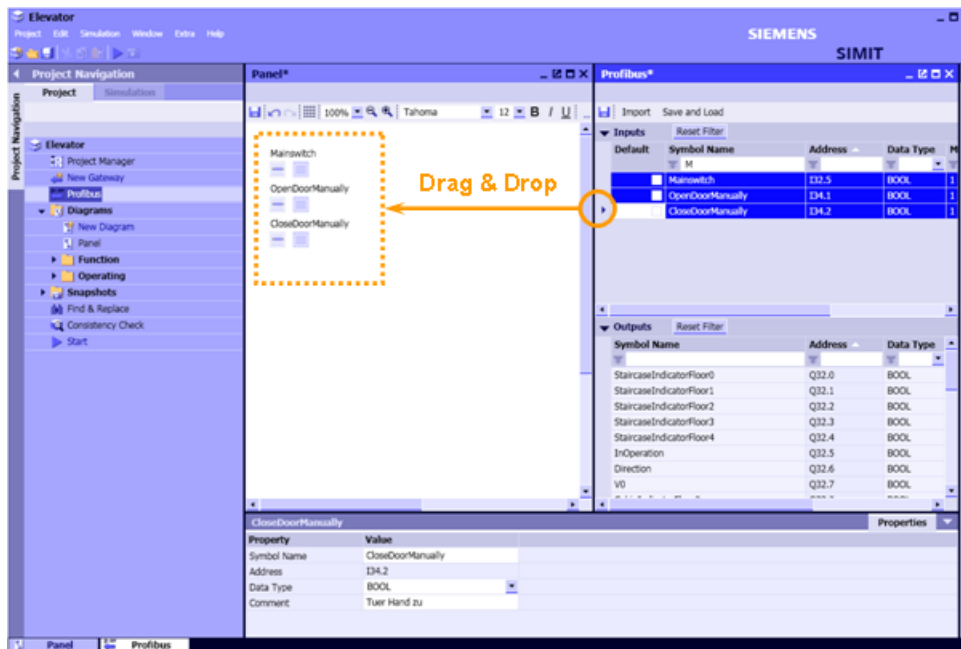


Then create a new chart and open it. If now you open the coupling, you should see both the chart and the coupling arranged in the work area as shown in the figure below:





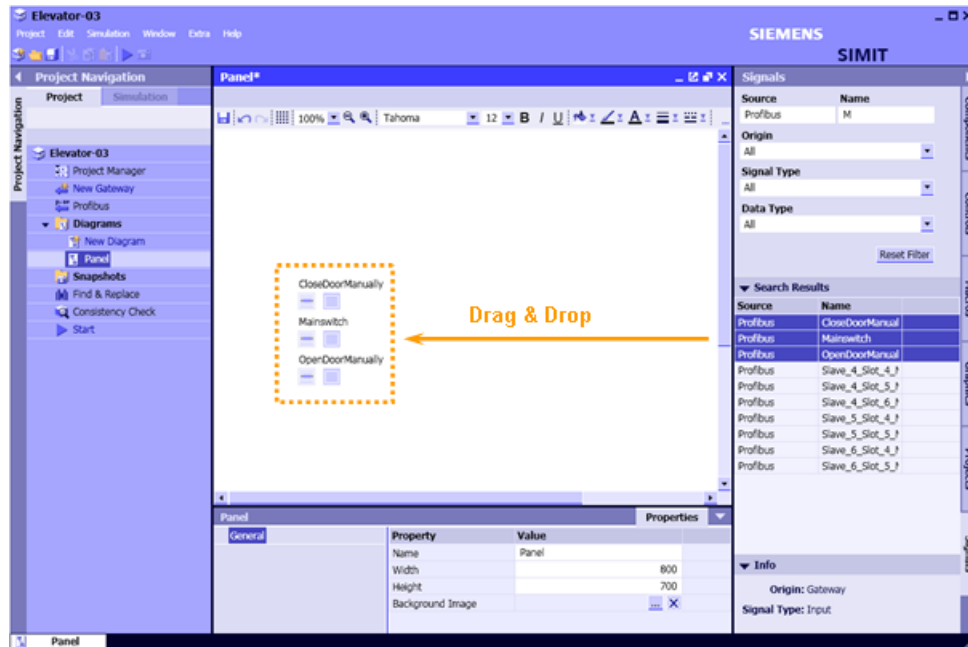
To select the required I/O signals you may use appropriate filter settings in the coupling window, for example, and select the required signal by clicking it with the mouse. You may select several signals while holding down the Ctrl key (single selection) or the Shift key (range selection).



Now drag the selected signals from the coupling onto the chart as shown in the figure above. To do this, "Grab" the selected signals in first column of the coupling indicated with the arrow. A switch is created for binary signals as a copy of the controls shown in the coupling, and a digital input is created for all other signals.


## Dragging coupling signals from the Signals task card

You may also drag coupling signals from the "Signals" task card onto a chart. To do so, directly open the chart in the work area and open the "Signals" task card in the associated tools window. All signals that are saved in the project are provided in this task card. To select signals, set the coupling name as a filter for the source and limit the signal list using an additional filter by signal name, if necessary. Then select signals from the list and drag-and-drop them onto the chart.

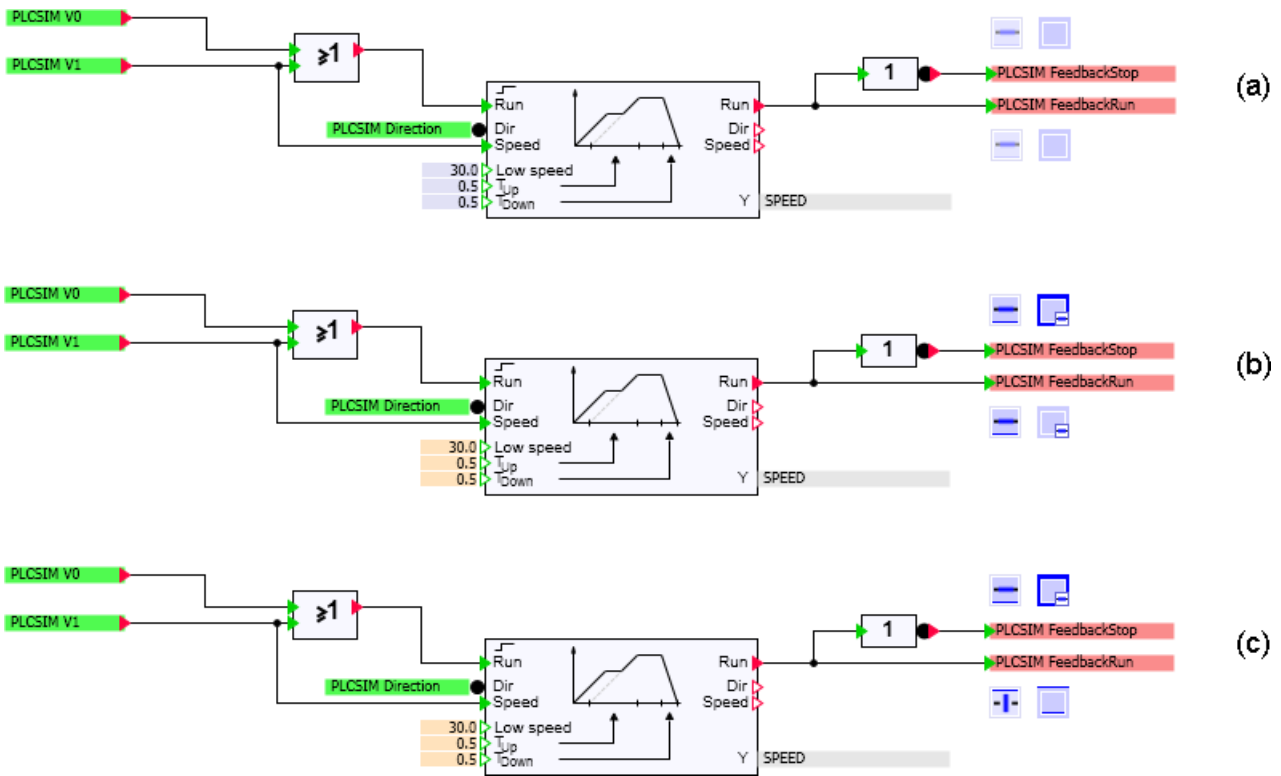


## Forcing coupling signals

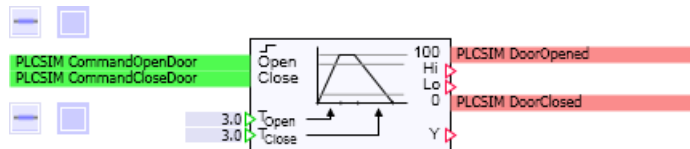
A sensible use of the controls of I/O signals results when they are inserted into charts to supplement the I/O connectors. By way of example, under (a) in the figure below, controls (switches) for the two feedbacks *FeedbackRun* and *FeedbackStop* are inserted in the chart for the simulation of the main drive.

When the simulation is started, both switches are shown with an overlay , as can be seen under (b) in the figure below. This overlay indicates that the switch is not operable until the associated signal splitter is activated. When the signal splitter is activated, the overlay on the associated switch disappears, as shown under (c) in the figure below, and the signal value can be set using the switch. The signal splitter enables **forcing** of the signal: The connection is broken between the simulation and the signal that is to be set, and the signal for the PLC can be set manually.

1.4 Basics of SIMIT



Forcing with a signal splitter and a control is not just for input signals of a coupling, but can similarly be used for output signals of a coupling. In this case, the connection between the output signal and the PLC is broken, and the signal can be set manually for the simulation.



The controls with their signal splitters have already been created for each signal in the coupling. Forcing is thus already prepared in the coupling for each I/O signal.

Controls and signal splitters for a specific signal can also be created multiple times on the same chart or on different charts. Forcing of this signal can then be performed with any one of these controls.

## Couplings

### 2.1 The coupling concept

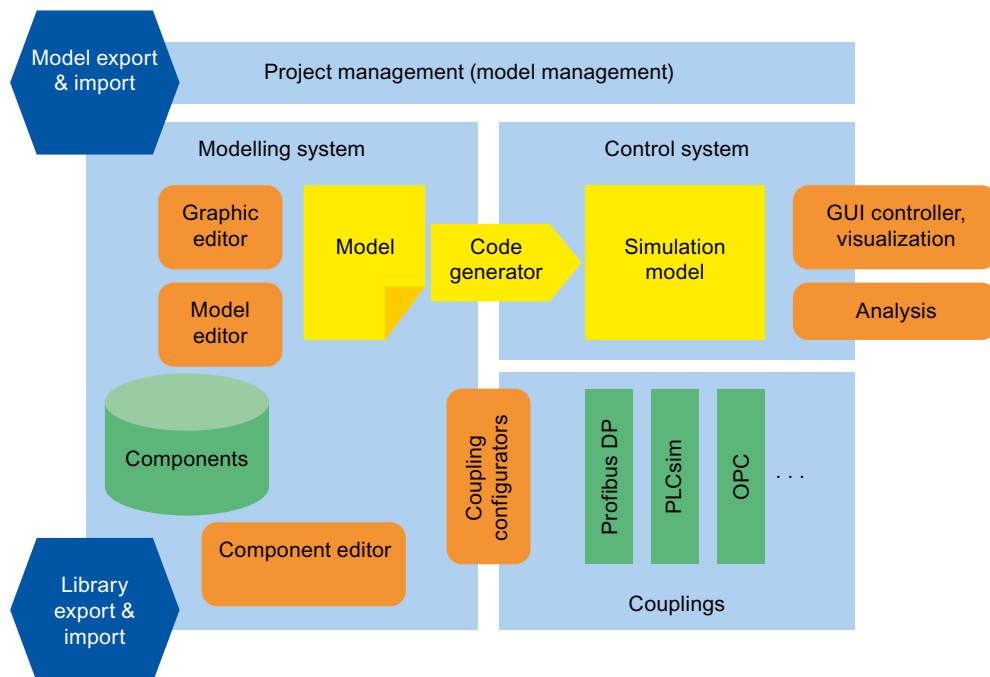
#### 2.1.1 Couplings in the SIMIT architecture

##### Introduction

Couplings form the interface of SIMIT to the automation. The input/output signals (I/O) signals are exchanged via a coupling. A coupling coordinates the exchange of signals between the simulation and the automation system and therefore contains binary, integer and analog input and output signals. Each coupling has a specific configurator for creating and editing the coupling. In general, automation systems as well as other applications can be connected to SIMIT via couplings.

##### Couplings in the SIMIT architecture

The core of every simulation system consists of a modeling system and a control system. The simulation model is created with the modeling system. The control system enables the controlled use of this model, which means running a simulation with this model.



A coupling forms the connection between the simulation, which means a simulation model implemented on the control system, and the automation system. This means it has two interfaces:

- The interface to the simulation model  
This interface is the same for all couplings. It ensures that the simulation model is independent of the type of coupling used. Couplings can therefore be interchanged within the simulation model without having to be modified.
- The interface to the automation system  
This interface must be adapted for each coupling to the system that is to be connected. It is therefore different for each coupling.

#### **Coupling configurator**

As the link between the simulation model and the automation system, the coupling has the task of establishing the exchange of signals. The coupling must therefore contain the following information:

- The information required to establish the connection
- Signals to be exchanged
- Access to the signals

This information is entered with the aid of a configurator for each coupling.

### **2.1.2 Exchanging signals via couplings**

#### **Signal exchange via couplings in asynchronous mode**

The signal exchange is not integrated into the cycle for calculation of the simulation model. Both the calculation of the model and the exchange of signals in the coupling are triggered cyclically by the control system. The cycle time for the model calculation is specified in the components or in the project properties. The cycle time for the exchange of signals is specified in the coupling properties (coupling configurator). Delays in the exchange of signals in a coupling therefore have no effect on the model calculation but only on the other couplings that are processed in the same cycle time.

To keep delays in the exchange of signals as small as possible, the time at which the coupling is triggered is generally shifted by half the set simulation cycle relative to the time at which the simulation model is triggered.

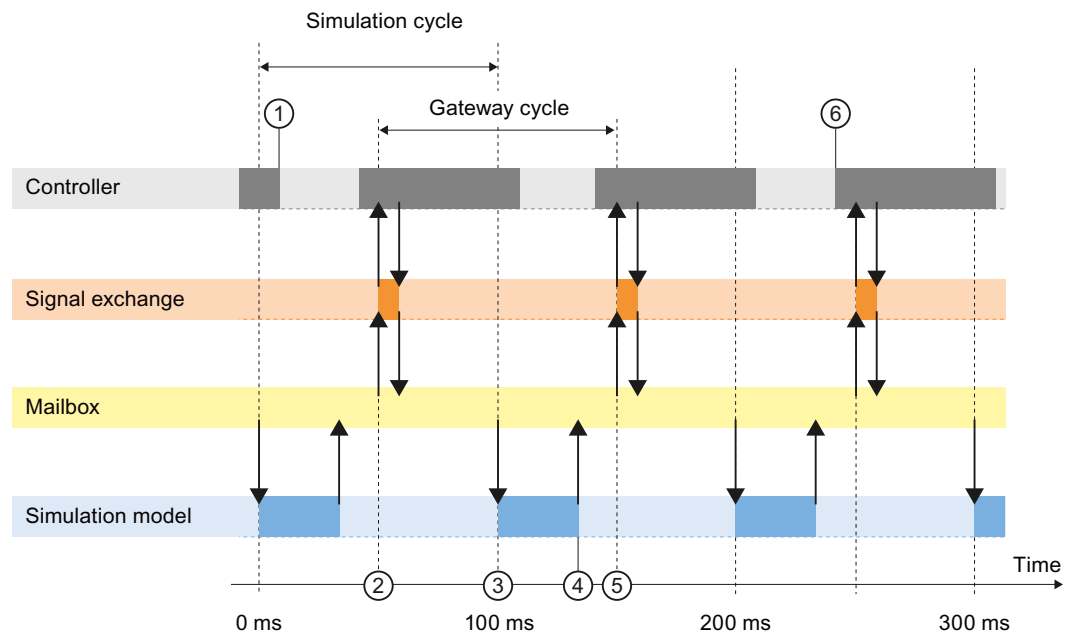
---

#### **Note**

The same value should be set for both the cycle time of the coupling (coupling cycle) and the cycle time for the calculation of the simulation model (simulation cycle).

---

Provided that the time necessary to perform a model calculation is no more than half the model calculation time, this results in the sequence shown in the figure below.



In this example, both the automation system and the calculation of the model have a cycle time of 100 ms. As the automation system and the simulation are not synchronized, their timing is usually shifted with respect to each other.

By way of explanation, assume that at time 1 the automation system has completed its calculation cycle and that the signals are available at the controller outputs. In the next coupling cycle at time 2, the controller outputs are stored in the coupling memory of SIMIT and can be used in the model calculation.

The model calculation is then triggered again at time 3. At the end of the model calculation cycle (time 4), the calculated input values are available to the controller in the coupling memory. With the next clock pulse of the coupling (time 5), these input signals are transferred to the controller.

If the controller only evaluates its inputs at the start of a cycle, the transferred input signals are only used in the control program at time 6. The reaction of the simulation therefore only takes effect two control cycles later.

### Signal exchange via couplings in synchronous mode

In synchronous mode, model calculation and signal exchange is performed in a specified sequence. The coupling detects the output signal, and writes back the input signal only after a model calculation. In contrast to asynchronous mode, no control cycles can be dropped.

You can find information on the operating modes in the section: Synchronous and asynchronous operating modes (Page 29).

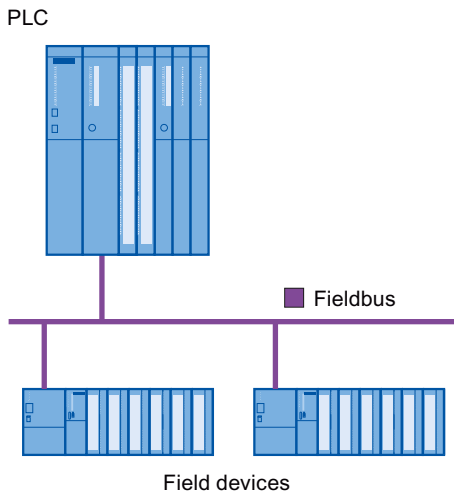
### 2.1.3 Couplings to SIMATIC PLCs

#### Introduction

SIMIT provides couplings to a SIMATIC controller, for example an S7-400, via PROFIBUS DP, PROFINET IO or PRODAVE, as well as a coupling to PLCSIM and to SIMIT VC. The data required for the configuration can be taken from the SIMATIC projects.

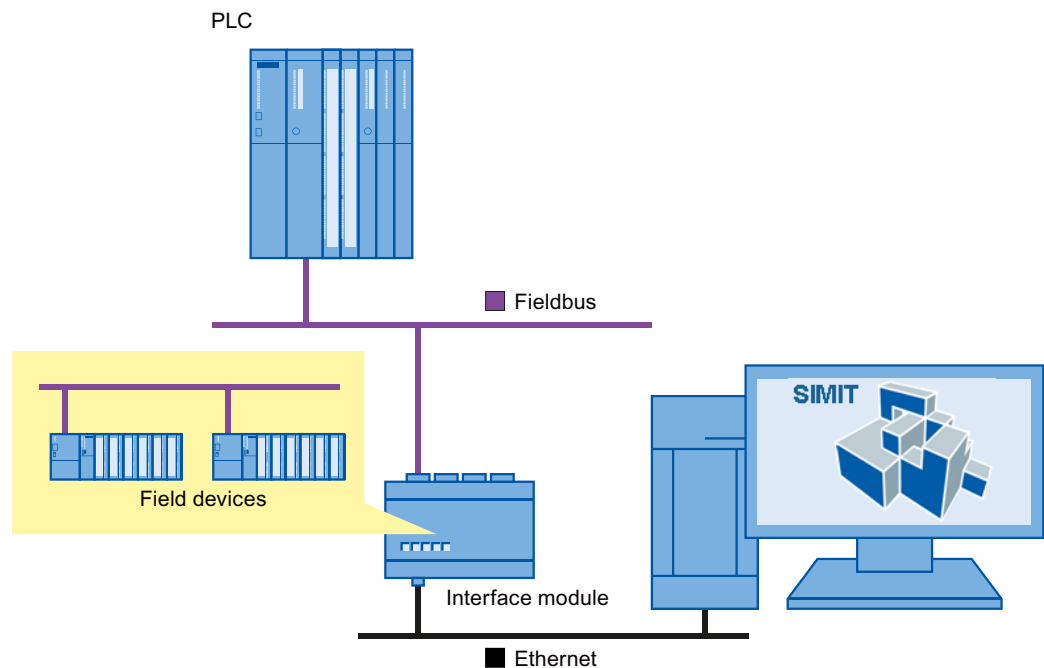
#### Coupling to a SIMATIC controller

As shown in the figure, a SIMATIC controller with a PROFIBUS DP or PROFINET IO field bus can be very easily connected directly to SIMIT at the field bus level, as shown in the figure.



In SIMIT, special interface modules are used for this purpose. These interface modules are capable of simulating the field devices. In the simplest configuration, such an interface module is connected with the PROFIBUS DP master or PROFINET IO controller of the PLC. The controller then communicates with this interface module, and therefore with SIMIT, in the same way as with the field devices in the real plant. The interface between SIMIT and the SIMATIC controller is the fieldbus cable.






An alternative to connecting SIMIT to a SIMATIC controller via field bus is to connect it via the PRODAVE interface. In this case, the physical connection between the PLC and the SIMIT PC is made either by means of the serial interface, USB, an MPI interface card or Ethernet. As a consequence of using the PRODAVE interface, this coupling lacks the high performance the field bus coupling provides. For detailed descriptions of all the couplings, refer to the following section: Couplings (Page 45).

### Coupling with PLCSIM

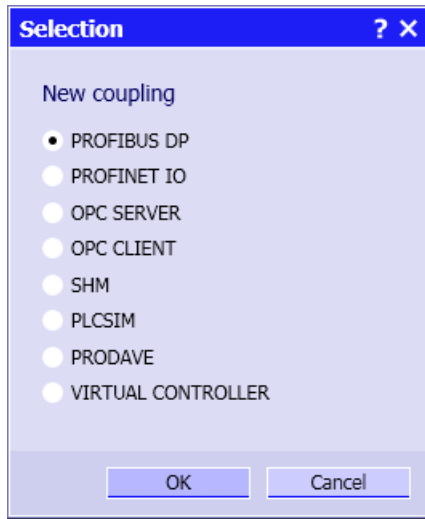
The SIMIT PLCSIM coupling facilitates the dynamic exchange of data between PLCSIM and SIMIT. This requires PLCSIM and SIMIT to be installed on the same PC. The PLCSIM coupling is explained in detail in the following section: PLCSIM coupling (Page 131).

## 2.2 Configuring and using couplings

### 2.2.1 Creating couplings

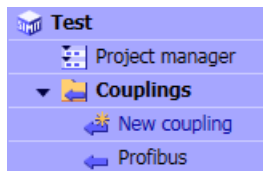
You can use one or more couplings in your SIMIT project. To create a new coupling, select the command "Couplings > Add new coupling" in the portal view, or create a new coupling by double-clicking on the tree node *New coupling* (  *New coupling* ) in the *Couplings* folder of the project view.

The following dialog box opens:



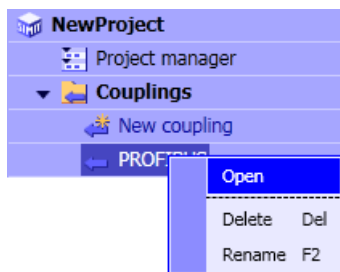
Select the desired coupling type.

You may choose the default name or enter any name of your choice as the name for the coupling. In the project tree, you will then see the newly created coupling with the coupling symbol (↵) and the name of the coupling:



### 2.2.2 Editing couplings

You can open the coupling for editing via its shortcut menu or by double-clicking it in the project tree.



The coupling editor for editing the coupling then appears in the work area. This editor looks somewhat different for each type of coupling, although in principle it always has the structure shown in the figure below.

**Profibus**

Save and load

▼ Inputs Reset filter

Default	Symbol name	Address	Data type	System	Slave	Slot	Comment	Scaling	Lower
0		EW522	WORD	1	3	4		No scaling	
0		EW524	WORD	1	3	4		No scaling	
0		EW526	WORD	1	3	4		No scaling	
0		EW528	WORD	1	3	4		No scaling	

▼ Outputs Reset filter

Symbol name	Address	Data type	System	Slave	Slot	Comment	Scaling	Lower	Upper
	AW522	WORD	1	1	4		No scaling		
	AW524	WORD	1	1	4		No scaling		
	AW526	WORD	1	1	4		No scaling		
	AW528	WORD	1	1	4		No scaling		

**EW522**

Property	Value
General	
Scaling	Symbol name
Limits	Address EW522
Connection	Data type WORD
	Comment

Each coupling provides a precisely defined set of signals that are exchanged between SIMIT and the connected controller. In the coupling editor, these coupling signals are separated in lists of input and output signals. You can find additional information in the section: The signals of a coupling (Page 52).

The properties of the coupling and the properties of the individual signals can be edited in the property view. The displayed properties vary according to the type of coupling.

### 2.2.3 Deactivating couplings

If, for example, you have configured a PROFIBUS DP coupling in your simulation project but wish to start the simulation without the coupling, you can set the hardware channel in the coupling properties to *Not assigned*.

**Profibus**

Property	Value
Profibus	
▼ Mastersystem 1	
▶ [4] ET 200M (IM153-2)	
▶ [5] ET 200M (IM153-2)	
▶ [6] ET 200M (IM153-2)	
Master Address	2
Hardware Channel	Not assigned
Baud Rate	1.5 Mbaud
H-System	No
F-System	No

The coupling is not activated when the simulation is next started, which means that no signals are exchanged and all values for the signals contained in this coupling are set to "0".

The "Disable couplings" function is only available at the following coupling types:

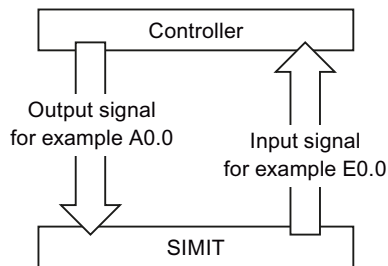
- PROFIBUS DP
- PROFINET IO
- PLCSIM
- PRODAVE
- OPC client

## 2.2.4 The signals of a coupling

### 2.2.4.1 Overview

Each coupling in SIMIT defines signals that can be linked to simulation components. As the terms "Input" and "Output" have specific meanings for a controller, the data direction in the coupling, which means whether a signal is defined as an input or output signal, is seen from the point of view of the connected controller in SIMIT. The following thus applies for SIMIT:

- An input signal is a signal that is calculated or output by the simulation and read by the controller.
- An output signal is a signal that is output by the controller and used as an input by the simulation.



All signals contained in a coupling are listed in the coupling editor. The symbol name or the absolute address is used to access a signal from the coupling, regardless of whether the signal is used in a chart, an animation, a control or in a trend.

### 2.2.4.2 Meaning of the coupling name

A signal name in SIMIT always consists of a source and a name. For any coupling signal the name of the coupling itself defines the source, and the symbolic name or the absolute address of the signal defines the name of the signal.

Source	Name
Profibus	I32.0

This guarantees that signal names remain unique throughout the entire SIMIT project, even if different couplings use identical symbol names or absolute addresses.

### Note

To avoid naming conflicts, couplings in SIMIT need to be assigned different names.

If you need to connect SIMIT to several PROFIBUS DP masters, for example, add a separate coupling in SIMIT for each PROFIBUS DP master system and assign meaningful coupling names. All coupling signals can then be uniquely assigned to the correct coupling.

### 2.2.4.3 Sorting and filtering of signals in the coupling editor

To find signals in the signal list more easily, you can sort and filter the signals by each of the properties given in the individual column. This is done by clicking on the heading of the column by which you want to sort. An arrow appears in the column heading. Every additional click on the column header toggles between alphabetically ascending and descending sort order. The arrow changes direction according to the selected sort order: ▲ for ascending order, ▼ for descending order.

Profibus			
Save and Load			
▼ Inputs Reset Filter			
Default	Symbol Name	Address ▲	Data Type
<input type="checkbox"/>		I0.0	BOOL
<input type="checkbox"/>		I0.1	BOOL
<input type="checkbox"/>		I0.2	BOOL

If you wish to further reduce the number of displayed signals, you may apply a filter to any of the properties, which means you can set a filter in any column. Then only those signals are displayed which match all of the filter criteria that are set. You may define a filter criterion by either using a selection box or entering text. The figure below shows the example of a text filter applied to the signal addresses. Only those signals are displayed that contain the provided text within their address.

Profibus			
Save and Load			
▼ Inputs Reset Filter			
Default	Symbol Name	Address ▲	Data Type
<input type="checkbox"/>		.1	
<input type="checkbox"/>		I0.1	BOOL
<input type="checkbox"/>		I1.1	BOOL
<input type="checkbox"/>		I32.1	BOOL

### 2.2.4.4 Addressing signals

In a SIMATIC PLC, different mechanisms are used to access the process image (I/O) and the I/O addresses (PI/PQ). For SIMIT itself, this access difference is not relevant. To use SIMATIC couplings in SIMIT, you can either use *QW* or *PQW* or *IW* or *PIW* in the address of the I/O signal. Signals are exchanged as follows, depending on the type of coupling:

- In the case of the PROFIBUS DP coupling, SIMIT exchanges signals with the controller, like in the real plant, as I/O signals of the field devices. For the PLC this is no different than the exchange of data with real I/Os. In the SIMATIC PLC, the I/Os or the process image can thus be accessed using the relevant mechanisms.
- In the case of the PRODAVE coupling, SIMIT only exchanges I/O signals with the process image of the controller. Access in the SIMATIC PLC to I/O addresses results in access errors.
- For the PLCSIM coupling, the mechanisms can be regarded as comparable to the PROFIBUS DP coupling: SIMIT communicates with PLCSIM on the basis of the I/O signals. The access mechanisms of the PLC are coordinated by PLCSIM.



### 2.2.4.5 Fixing signals in the coupling editor

With SIMIT, you can set signals and display the signal values in the coupling editor by creating a coupling in the project. You have to perform the following three steps:

- Create a coupling (You can find additional information on this in the section: Creating couplings (Page 49)),
- Configure the coupling.
- Start the simulation.

For each signal in the coupling, both of the automatically generated controls with fixation are used for this purpose. In SIMATIC, the fixation of a particular signal state is called "forcing". Fixing in SIMIT is a comparable function, but it only has an effect within the simulation model and does not cause "forcing" in SIMATIC. Therefore the term "Fix" was deliberately chosen for this function.

Table 2-1 Controls with fixation in the coupling

	"Toggle switch with fixation" for binary values
 900.0	"Digital input with fixation" for analog and integer values

For each coupling signal, the controls with fixation are available in the first column after starting the simulation: Toggle switch with fixation for the binary I/O signals, digital inputs with fixation for the analog and integer I/O signals.

▼ Inputs		Reset Filter	Symbol Name	Address	Data Type	Comment
			CabinButtonFloor2	I33.2	BOOL	Anwahl Innenkabine 2.OG
			CabinButtonFloor3	I33.3	BOOL	Anwahl Innenkabine 3.OG
			CabinButtonFloor4	I33.4	BOOL	Anwahl Innenkabine 4.OG
			FeedbackRun	I33.5	BOOL	Antrieb laeuft
			FeedbackStop	I33.6	BOOL	Antrieb steht
			DoorOpened	I33.7	BOOL	Sensor Tuer auf
			DoorClosed	I34.0	BOOL	Sensor Tuer zu
			OpenDoorManually	I34.1	BOOL	Tuer Hand auf
			CloseDoorManually	I34.2	BOOL	Tuer Hand zu

The toggle switch with fixation for binary values combines three functions:

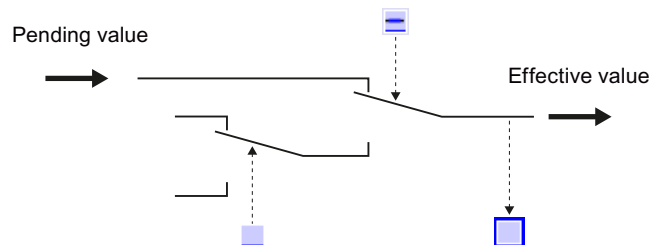
- Switching between pending value and fixed value, in other words fixation (),
- Switching the fixed value between zero and one ( and )
- Displaying the value that is effective at the output ( or ).

The switch is the "fixation switch".

How the input element is displayed in the running simulation tells you whether or not the corresponding coupling signal was used in the simulation model. Signals that are not used in the simulation model are displayed with an inactive fixation switch ().

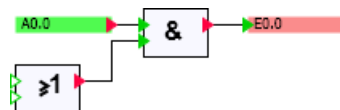
Signals that are used in the simulation model are displayed with an operable fixation switch. or : When the fixation switch is actuated, the toggle switch or becomes operable. The signal value can now be changed manually.

The function of the toggle switch with fixation is shown in the figure below:







Override is basically a changeover, which either transfers the actual value in the coupling or changes it to a value that is input manually. It is possible to preset both input and output signals, which means all I/O signals.



To further describe the function, let us assume that you have configured two signals in the coupling and have used them as follows in a chart:





In the coupling you can see which value the controller outputs for the signal Q0.0.

Value output by the controller	Q0.0 in the coupling	Value used in the simulation model
0		0
1		1



Override allows you to control your simulation model so that it uses a different value from the one output by the I/O. To do this, click on the fixation switch  for the signal Q0.0. This is then shown like this: . You can now use the toggle switch to directly enter the value to be used in your simulation model as follows:

Value output by the controller	Q0.0 in the coupling	Value used in the simulation model
0 or 1		1
0 or 1		0

The same applies to the input signals. For an input signal I0.0 you can first of all see what value the simulation model outputs to the controller:

Value calculated in the simulation model	I0.0 in the coupling	Value output to the controller
0		0
1		1

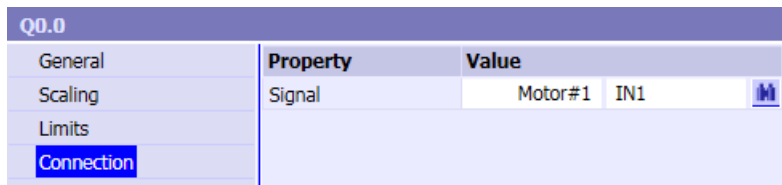
Override allows you to specify the value that is output to the controller, regardless of the value calculated in the simulation model. As before, use the fixation switch and specify the value using the toggle switch:

Value calculated in the simulation model	I0.0 in the coupling	Value output to the controller
0 or 1		1
0 or 1		0

This procedure applies in the same way to analog and integer signals. In contrast to the binary values, however, the target values are only entered using a digital input and a digital display is used instead of a binary display.

### 2.2.4.6 Interconnecting signals in the coupling editor

In the "Properties" area of a coupling, specify implicit interconnections for input and output signals:





You have the following options for interconnections:

- Signal exchange between 2 coupling signals
- Signal exchange between a coupling signal and an input or output of a component

---

### Note

Only signals of the same type can be connected. Only one signal can be specified for connection.

---

## 2.2.5 Using I/O signals

### 2.2.5.1 I/O signals for connection on charts

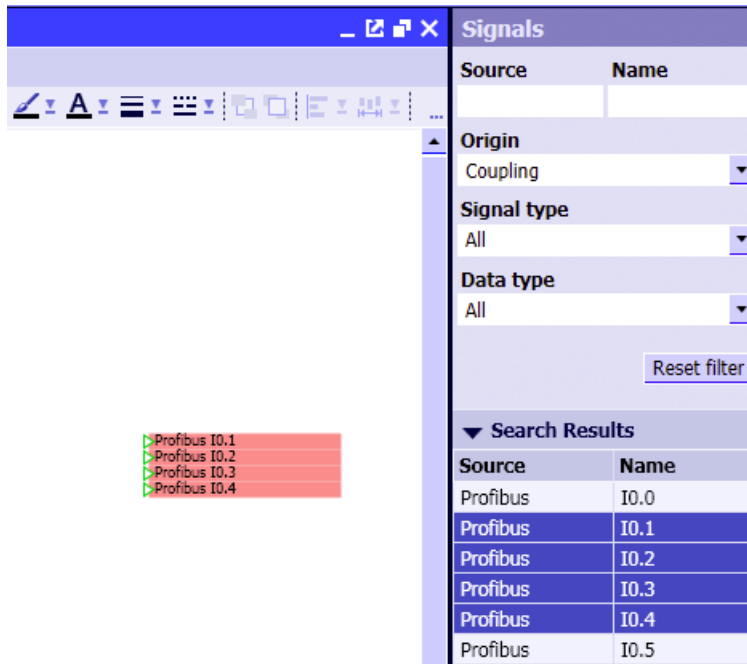
You can use I/O signals on charts and therefore include them in the simulation model. The simplest method of inserting I/O signals into charts is by means of the "Signals" task card. Open the chart for editing and select the "Signals" task card. If you wish to see I/O signals only, just set the filter *Origin* to *Coupling* within the "Signals" task card.

The screenshot shows the 'Signals' task card interface. It has a header 'Signals' and a table with 'Source' and 'Name' columns. Below the table are filters for 'Origin' (set to 'Coupling'), 'Signal type' (set to 'All'), and 'Data type' (set to 'All'). There is a 'Reset filter' button. Below the filters is a 'Search results' section with a table showing three rows of 'PLCSIM' source and 'CabinButtonFloor' name.

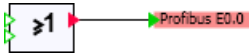
Source	Name
PLCSIM	CabinButtonFloor
PLCSIM	CabinButtonFloor
PLCSIM	CabinButtonFloor

Select a signal in the task card and drag-and-drop this signal onto the chart. Multiple selections are also possible. Select a contiguous block of signals using the *Shift* key or use the *Ctrl* key if individual signals are required; then drag the selected signals onto the chart.

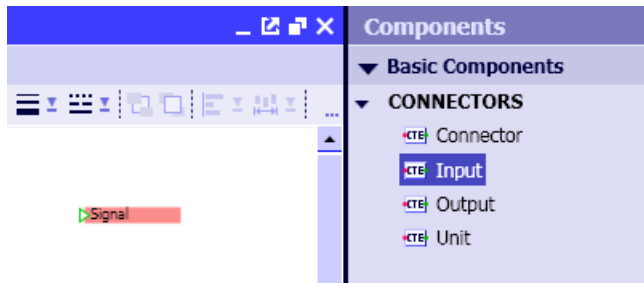
If you do it this way controls are created on the chart which can be used to set the selected signals. If you want to create I/O connectors for the signals instead, press the shift key when dragging and dropping the signals. This procedure then creates I/O connectors with the correct signal names .



As shown in the figure below you can now link the I/O connector to the connector of another component.



You can also use an I/O connector from the library "Components" task card but you have to enter the coupling and signal name manually in this case.



To enter the signal name, simply click on the I/O connector to select it and enter the name of the coupling and the signal name in the property view.

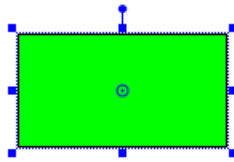
Profibus Signal		
General	Property	Value
	Signal	Profibus IO.0
	Display coupling name	<input checked="" type="checkbox"/>

**Note**

The name of the I/O connector must be written exactly as it is entered in the configurator. Therefore, pay attention to upper/lower case and do not use spaces in address names such as *I0.0*.

**2.2.5.2 I/O signals for animations**

You can use I/O signals to animate graphic objects. To do this, enter the signal name in the property view of the animation.

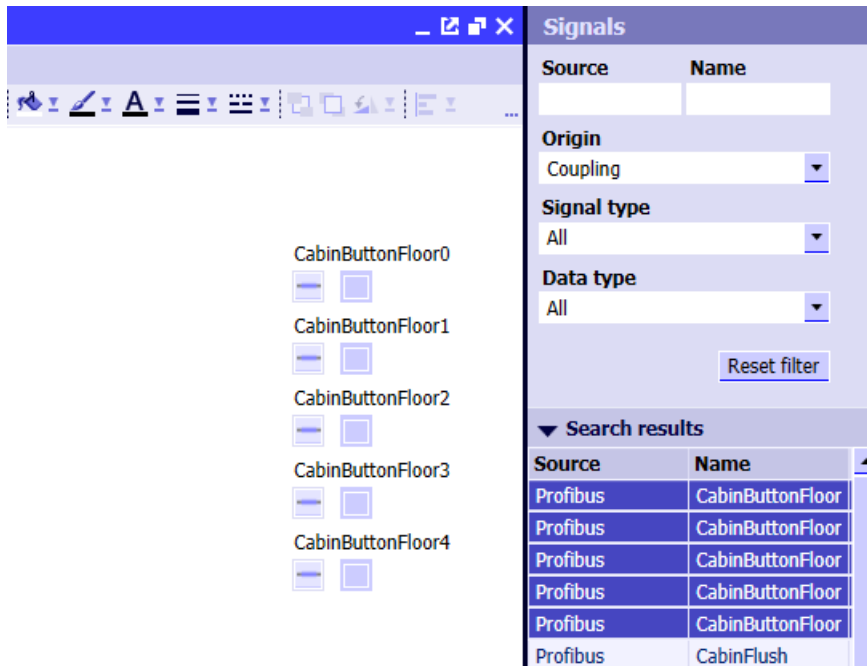


Rectangle		
General	Property	Value
Appearance	Signal	Profibus Q3.2
Layout	'Visible' if signal 'True'	<input checked="" type="checkbox"/>
▼ Animations	Offline visible	<input checked="" type="checkbox"/>
<b>New animation</b>		
Visibility		

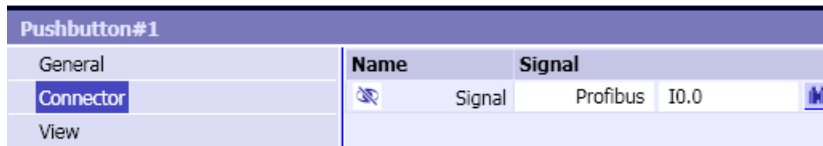
You can drag-and-drop the signals from the "Signals" task card.

### 2.2.5.3 I/O signals in controls on charts

The controls with fixation can be copied from the coupling into the chart. To do this, drag the I/O signals from the "Signals" task card into the chart. The name for the control is copied as a text element to the chart.



You can also use I/O signals directly in other controls, such as a pushbutton.



Either enter the name of the I/O signal in the property view of the control or drag-and-drop the I/O signal from the "Signals" task card into the property view of the control.

### 2.2.5.4 I/O signals in trends

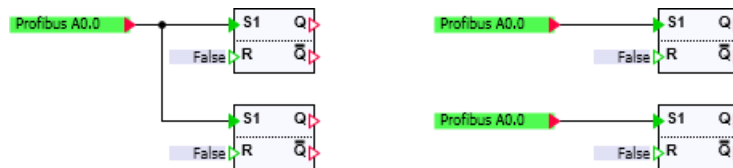
You can also use I/O signals in trends. Enter the signal with its source and name in the trend property view or drag-and-drop it from the "Signals" task card.

Properties					
Source	Name	Color	Range	Cursor	Alias
PLCSIM	CabinFlush		Binary		
PLCSIM	Increments		Auto (0 .. 0)		
*					

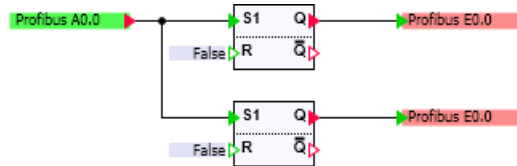
You can find the description of trends in the following section: Trends (Page 255).

### 2.2.5.5 Multiple use of I/O connectors

I/O input connectors with the same name can be used several times on the same chart. This means that the two charts below are equivalent:



On the other hand, a I/O output connector can only be used exactly once in the entire project. The connection below would lead to an error when the simulation project was started:



## 2.2.6 Importing and exporting signals

### 2.2.6.1 Overview

The coupling signals can be exported and imported in specific formats. The overview in the table shows which data formats can be used.

Table 2-2 Data formats for importing and exporting signal tables

Coupling type	Signal tables	
	Import format	Export format
PROFIBUS DP	asc, seq, txt or xlsx	txt, seq
PROFINET IO	asc, seq, txt or xlsx	txt, seq
Virtual Controller	txt	txt
PLCSIM	asc, seq, txt or xlsx	txt, seq

Coupling type	Signal tables	
	Import format	Export format
PRODAVE	asc, seq, txt or xlsx	txt, seq
SHM	asc, seq, txt or xlsx	txt, seq
OPC Server	ini, txt	txt, ini
OPC client	ini, txt	txt, ini

The standard import and export format in SIMIT is the "txt" format.

The signal table of the SIMATIC couplings (PROFIBUS DP coupling, PROFINET IO coupling, PRODAVE coupling and PLCSIM coupling) in the "asc" or "seq" format corresponds to a SIMATIC project symbol table. You can find additional information on this in the following section: The symbol table (Page 62).

Exported signal tables contain all coupling signals with their properties such as name (symbol), address, comment, etc. You can find additional information in the following section: The signal table (Page 64).

The import and export functions can also be used to copy signal configurations from one coupling to another.

### 2.2.6.2 The symbol table

The symbol table in a SIMATIC project can be exported in various formats using SIMATIC Manager. To use a symbol table in SIMIT, select the asc or seq format.

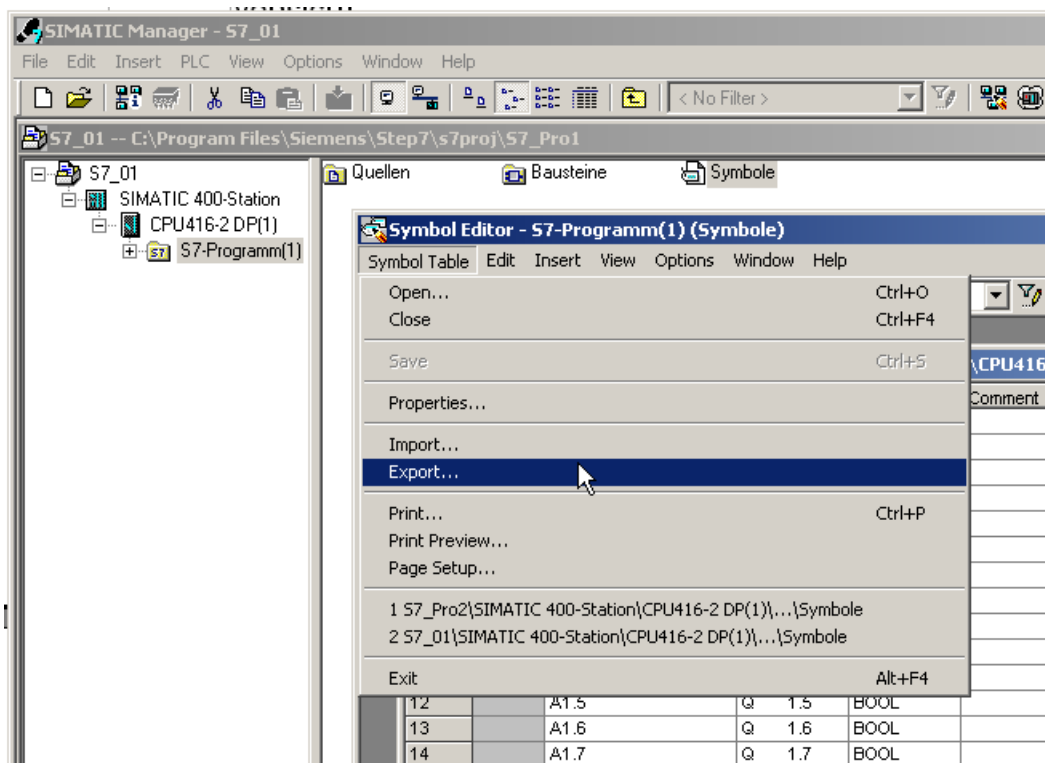
---

#### Note

Note that the comment is limited to 40 characters in a seq format, and it is very important to ensure that it does not contain any data types.

Data words have data type *WORD* when importing a seq file; double words have the data type *DWORD*. Conversions to *INT* or *REAL* need to be done manually, if required.

---



Importing symbol tables is supported by all SIMATIC couplings.

### Note

The asc format uses fixed column widths. If you edit this file with another text editor rather than the symbol editor, it is imperative that you do not alter the number of characters per line.

### 2.2.6.3 The tag table

You may import PLC tag lists (symbol tables) from the TIA Portal. The TIA Portal export these lists in xlsx format. Only input and output signals with following data types are transferred:

TIA Portal	STEP 7	SIMIT
Bool	BOOL	Binary
Bytes	BYTE	integer
Word	WORD	integer
Int	INT	integer
DWord	DWORD	integer
Dint	DINT	integer
Real	REAL	analog

### 2.2.6.4 The signal table

The signal table for the SIMATIC couplings has the structure described in the table "Format of the signal table". When a signal table is stored in a txt file, the tab character is used as the column delimiter.

The ID "#Signal properties" must be in the first row. The SIMIT version number with which this table was exported is added to this for the export from SIMIT. The column headings are in the second row.

Table 2-3 Format of the signal table

Column	Term	Heading	Meaning
1	Symbol	Symbol	Symbolic name of the signal
2	I/O	InOut	Encoded as <i>I, Q, IB, QB, IW, QW, ID, QD</i> , or in the corresponding international notation (I/Q)
3	Address	Address	Absolute address of the signal, for example <i>0.0</i> or <i>512</i>
4	Type	Type	Signal data type: <i>BOOL, BYTE, WORD</i> or <i>DWORD</i>
5	Comment	Comment	Text as comment
6	Default	Default	Initialization value that is the default of this signal
7	Signal source	ImplicitSource	Source of implicitly interconnected signal
8	Signal name	ImplicitSignal	Signal source of implicitly interconnected signal
9	Unit	Unit	Unit of the signal, e.g., physical unit for measured values
10	Scaling	Scaling	Type number of the scaling (for information, see the table in the following section: Scaling analog signals (Page 71))
11	Low limit	ScalingLowerPhys	Physical low or high limit of analog signals:
12	High limit	ScalingUpperPhys	
13	Low raw value	ScalingLower-RAW	Low or high raw value of scaled analog signals. Can only be freely selected with "Custom" scaling type.
14	High raw value	ScalingUpper-RAW	
15	Limiting on/off	LimitActive	Limitation of an analog signal is in enabled / disabled
16	Low limit	LimitLowerPhys	Low physical value of the limit
17	High limit	LimitUpperPhys	High physical value of the limit
18	Multiplier	Multiplier	OPC client, cycle multiplier
19	Can be read back	Readback	OPC client, signal can be read back

#### Note

The signal table has been revised with the SIMIT version 8.1 and expanded with column headings. If you want to import a signal table that was created with an older version of SIMIT, you must adjust this accordingly.

Due to the new "Custom" scaling type, the significance of the type numbers for scaling has moved.



**Note**

When you edit a signal table in Excel, all columns must have "Text format" so that Excel does not make any unintended format conversions.

You can directly open a signal table in EXCEL by clicking *Open with* in the Explorer shortcut menu and selecting EXCEL.

### 2.2.6.5 The INI format of the OPC couplings

OPC signals have their own import/export format for the following reasons:

- They do not recognize addressing
- The transmission direction can not be detected by the name
- There is no conversion between raw values and physical values

The signal table for OPC signals has the ini format consisting of 6 sections that are marked with the following keywords:

Keyword	Meaning
[AIN]	Analog input signals
[IIN]	Integer input signals
[BIN]	Binary input signals
[AOUT]	Analog output signals
[IOUT]	Integer output signals
[BOUT]	Binary output signals

These keywords must each appear in a line by themselves. This is followed by the signal names to be assigned to this section. The order of sectors is arbitrary. Not all sections must be used.

In each section, the signal table contains a single line per signal. The signals and their properties are exported in quotation marks and separated by a semicolon.

	OPC Server	OPC client
Inputs	<ul style="list-style-type: none"> <li>• Name</li> <li>• Comment</li> <li>• Default</li> <li>• Implicit interconnection (source/target)</li> <li>• Implicit interconnection (signal)</li> </ul>	<ul style="list-style-type: none"> <li>• Name</li> <li>• Comment</li> <li>• Default</li> <li>• Implicit interconnection (source/target)</li> <li>• Implicit interconnection (signal)</li> <li>• Multiplier</li> <li>• Can be read back ("True/False")</li> </ul>
Outputs	<ul style="list-style-type: none"> <li>• Name</li> <li>• Comment</li> <li>• Implicit interconnection (source/target)</li> <li>• Implicit interconnection (signal)</li> </ul>	<ul style="list-style-type: none"> <li>• Name</li> <li>• Comment</li> <li>• Implicit interconnection (source/target)</li> <li>• Implicit interconnection (signal)</li> <li>• Multiplier</li> </ul>

If a signal table only contains a signal name without any quotation marks, the signal is imported using the specified name without comment and with the standard default value.


**Note**

With an OPC client, the import format of SIMIT V8.1 is not backwards compatible with older versions of SIMIT. If you have specified the multiplier or the readback capability in existing import files, you must now add 2 additional empty columns for the implicit interconnection.

**2.2.6.6 Import of signal properties**

Signal properties can be imported from the following files:

- Symbol Table
- Tag table
- Signal table
- OPC signals

Click on the  symbol in the coupling editor. The following dialog box opens:



### Format

The format that can be read depends on the current coupling type. In general, the following file formats can be read:

- SIMATIC symbol table from STEP 7 or PCS 7  
You can find additional information on this in the section: The symbol table (Page 62).
- PLC tag table from the TIA Portal  
For more information, see section: The tag table (Page 63).
- Signal table (txt file)  
You can find more information on this in the section: The signal table (Page 64).

---

### Note

#### OPC coupling

If signal properties were saved in txt format, the file also contains SIMATIC addresses and information for scaling. However, this information cannot be processed by an OPC coupling. If a txt file is imported into a OPC connection, the only information included is that which an OPC coupling can also process.

---

- INI file

### File

The storage location of the import file. Click the "... " button to navigate to the desired file.

### Area "Which properties do you want to import?"

Here you select which signal properties should be transferred. The signal properties can only be selected if they are also present in the import file.

### Mode

The mode determines how the signals already present in the coupling and their properties are processed. The import modes that can be selected depends on the current coupling type. Basically, the following import modes are available:

- **Recreate signals**  
All existing signals are deleted and recreated with the imported information. The only properties imported are the once activated in the "Which properties do you want to import?" area. All other properties retain their default settings.
- **Add signals**  
All signals from the import file are added. The only properties imported are the once activated in the "Which properties do you want to import?" area. All other properties retain their default settings. Already existing signals are not changed.
- **Recreate properties**  
All properties activated in the "Which properties do you want to import?" area are set to their default settings and transferred from the import file.
- **Overwrite properties**  
All properties activated in the "Which properties do you want to import?" area are transferred from the import file and replace the existing signals.
- **Add properties**  
All properties activated in the "Which properties do you want to import?" area that still have their default setting are transferred from the import file and replace the existing signals.

**Automatically adjust data width**

---

**Note**

This check box is available only for the PROFIBUS DP and PROFINET IO coupling types.

---

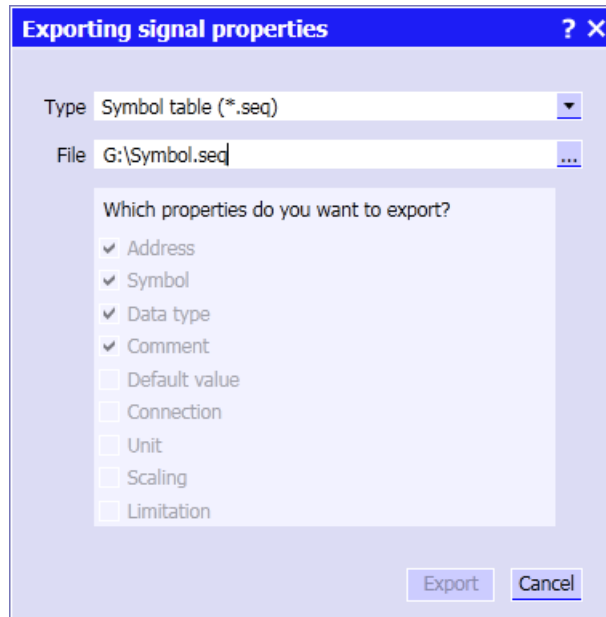
Select this check box if the data width of signals should be automatically adjusted to match the existing data width when importing. You can find additional information in the following sections:

- Merging and splitting signals (Page 96) (for the PROFIBUS DP coupling)
- Merging and splitting signals (Page 121) (for the PROFINET IO coupling)

If the check box is not selected and the width of the data to be imported does not match the existing signal, this signal is not imported.

### 2.2.6.7 Export of signal properties

To export signal properties, click on the "📄" symbol in the coupling editor. The following dialog box opens:



#### Type

File format of the export file. The file format depends on the current coupling. Generally, the following file formats can be exported:

- SIMATIC symbol table from STEP 7 or PCS 7  
You can find additional information on this in the section: The symbol table (Page 62).
- Signal table (txt file)  
You can find more information on this in the section: The signal table (Page 64).
- INI format

#### Note

##### OPC coupling

Since OPC signals do not have an address nor can they be transmitted as scaled raw values, the signal table can only be filled to a limited extent. Such a signal table can be imported into another OPC coupling, but import into couplings that expect SIMATIC signals is not possible.

#### File

The storage location of the export file. Click the "..." button to navigate to the desired storage location.

#### Area "Which properties do you want to import?"

Here you select the signal properties you want to export. The "Address" property is required to identify a signal and therefore cannot be deselected.

## 2.3 Couplings for SIMATIC

### 2.3.1 Common properties of SIMATIC couplings

#### 2.3.1.1 Data types of signals

##### Data types of signals

A signal in the SIMATIC coupling has one of the SIMATIC-specific data types *BOOL*, *BYTE*, *WORD*, *INT*, *DWORD*, *DINT* or *REAL*, which correspond to access to the address range of the PLC. When used as an I/O signal it has the data type binary, integer or analog. The table below shows the mapping of SIMATIC data types to the data types of the I/O signal.

Table 2-4 Mapping of SIMATIC data types to signal data types

SIMATIC	I/O signal
BOOL	Binary
BYTE	Integer
WORD	Integer
INT	Integer
DWORD	Integer
DINT	Integer
REAL	Analog

When a signal in the coupling of *WORD* data type is scaled, the data type changes from integer to analog. You can find additional information on this in the following section: Scaling analog signals (Page 71).

##### Importing PLC variable lists

You may import PLC variable lists (symbol tables) from the TIA Portal. These lists are exported from the TIA Portal in EXCEL format (\*.xlsx). Only I/O signals with the data types listed in the table below are copied from the variable list:

Table 2-5 Comparison of data types

TIA Portal	Step 7	SIMIT
Bool	BOOL	binary
Bytes	BYTE	integer
Word	WORD	integer
Int	INT	integer
DWord	DWORD	integer
DInt	DINT	integer
Real	REAL	analog

### 2.3.1.2 Scaling analog signals

In SIMATIC systems, like other automation systems, the analog signal values are transferred in a fixed decimal point format, which means in a fixed integer format. The value range depends on the resolution of the A/D converter and is typically between -27648 and +27648 for the SIMATIC S7.

By contrast, in the simulation, analog values are treated as floating point numbers. In particular, the absolute values of physical variables, such as pressures, temperatures, etc., are usually used in simulations. As input values for the connected automation systems, these values must not only be converted into fixed decimal point format, they must also be mapped onto the range of values for the corresponding measurement ranges. The analog values must therefore be scaled when they are sent from SIMIT to SIMATIC, and rescaled when sent in the opposite direction.

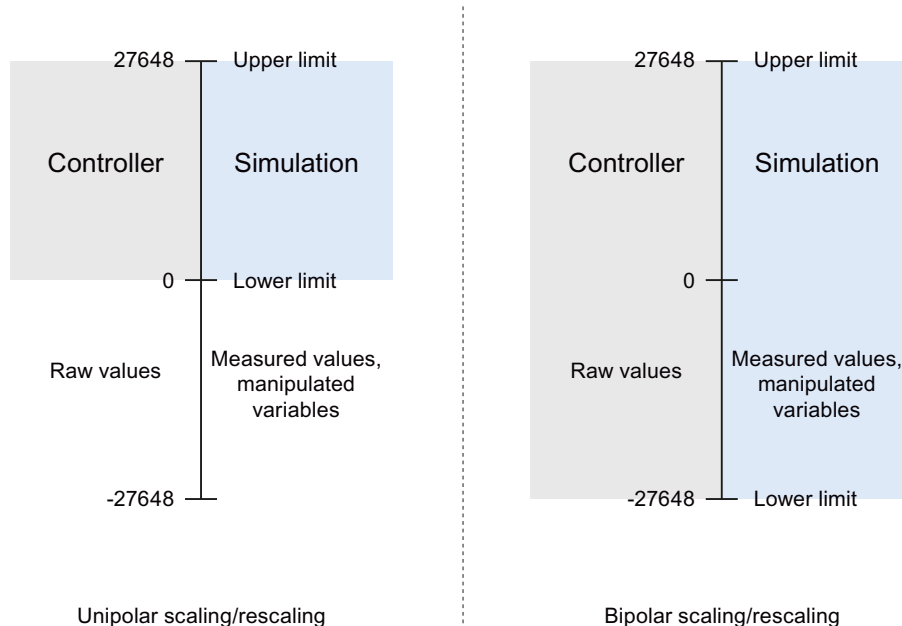
The scaling of input signals and the rescaling of input and output signals is a way of adapting the simulation to the characteristics of the controller. Scaling and rescaling are therefore performed at the interface between the simulation model and the controller in the couplings. The scaling/rescaling values are entered in the property view of the coupling signals. Alternatively, the scaling/rescaling values can be entered in the signal tables. Additional information on the signal table is available in the following section: The signal table (Page 64).

Pressure		
General	Property	Value
Scaling	Scaling	Unipolar ▼
Limits	Lower scale value	0
Connection	Upper scale value	15
	Unit	Bar
	Lower raw value	0
	Upper raw value	27648

Because all analog modules in the SIMATIC range use a resolution of two bytes, which are transferred as a word, scaling and rescaling in SIMIT is only available for analog values of *WORD* data type.

An example of the system used to scale unipolar and bipolar measurements is shown in the figure below:

- Unipolar measurements only provide positive raw values (0 to 27648)
- Bipolar measurements provide both positive and negative raw values (-27648 to +27648).



The table below lists the scaling/rescaling types that are supported by SIMIT.

Scaling types 4 to 18 for the scaling of temperature measurements can only be used for input signals. Unipolar and bipolar scaling (types 1 and 2) and "Custom" (type 3) can also be used to rescale output signals. Although for both types 1 and 2 the Start (*lower scale value*) and End (*upper scale value*) of the range are preset with 0 or -100 and 100, respectively, but they can be adapted to the measurement or adjustment range by entering the appropriate values.

For temperature measurement signals in a coupling, the limits of the measuring range are displayed only and cannot be changed.

High and low raw scores can only be edited when the "Custom" scaling type is selected.

Table 2-6 Scaling/rescaling supported by SIMIT

Scaling type		Measuring range		Raw values	
Number	Name	Start	End	Lower	Upper
0	No scaling				
1	Unipolar	0 (Default)	100	0	27648
2	Bipolar	-100 (Default)	100 (Default)	-27648	27648
3	User-defined	0 (Default)	100 (Default)	0 (Default)	27648 (Default)
4	PT x00 standard	-200 °C	850 °C	-2000	8500
5	PT x00 climate	-120 °C	130 °C	-12000	13000
6	Ni x00 standard	-60 °C	250 °C	-600	2500
7	Ni x00 climate	-60 °C	250 °C	-6000	25000



Scaling type		Measuring range		Raw values	
Number	Name	Start	End	Lower	Upper
8	Cu 10 standard	-200 °C	260 °C	-2000	2600
9	Cu 10 climate	-50 °C	150 °C	-5000	15000
10	Type B thermocouple	0 °C	1820 °C	0	18200
11	Type E thermocouple	-270 °C	1000 °C	-2700	10000
12	Type J thermocouple	-210 °C	1200 °C	-2100	12000
13	Type K thermocouple	-270 °C	1372 °C	-2700	13720
14	Type L thermocouple	-200 °C	900 °C	-2000	9000
15	Type N thermocouple	-270 °C	1300 °C	-2700	13000
16	Type R, S thermocouple	-50 °C	1769 °C	-500	17690
17	Type T thermocouple	-270 °C	400 °C	-2700	4000
18	Type U thermocouple	-200 °C	600 °C	-2000	6000

### Changing the scaling while the simulation is running

When the simulation is running, the measuring ranges of the scaling and the raw value ranges can be changed (for the "Custom" type). Changes take effect immediately.

#### Note

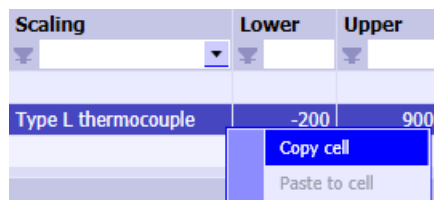
This function is not available with the Virtual Controller coupling.

#### Note

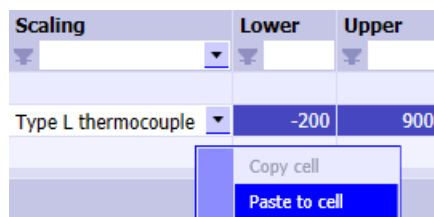
These changes apply only to the current simulation. They are not automatically applied to the configuration.

### Copying and inserting scaling values

It is very simple to copy scaling values from one signal to another or to a selection of signals. Select the cell you want to copy and then choose the *Copy cell* item in the shortcut menu.



You can then select one or more target signals and choose *Paste to cell* in the shortcut menu for the selection.



You can transfer the scaling type and the low and high value in this way. This information has to be processed column-by-column, however.

In addition, you can only paste to cells that are enabled for editing. If the lower and upper values therefore cannot be edited because no scaling has been set, for example, you need to set the scaling type first or transfer it from an existing signal. Only then can you transfer the associated values.

#### Transfer of floating point values (Float)

Some I/O devices - including those in the PROFIBUS PA range - transfer their measured values directly to the controller as physical variables in floating point format with a data width of 4 bytes (double word). In order to handle these floating point values correctly, the corresponding signals have to be assigned the data type *REAL* in the coupling.

### 2.3.1.3 Limitation of analog signals

All analog coupling signals, i.e. scaled measured values (WORD) or floating point numbers (REAL) may be limited. In this case, input signals sent to the controller or the connected partner are limited to the range between the specified low and high physical values, even if the simulation has actually calculated a value outside this interval. Conversely, output signals of the controller can be limited so that the simulation does not read values outside this interval.

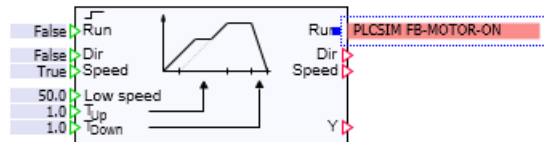
Pressure		
General	Property	Value
Scaling	Limiting	On
Limits	Low limit	0
Connection	High limit	5

The limitation can be switched on and off, even while simulation is running. The limit values themselves can also be changed while simulation is running.

### 2.3.1.4 Symbolic addressing

#### Displaying the absolute address of SIMATIC signals

The absolute address is now displayed both in the properties of the input and output connectors and in the *Signals* task card. The signals in SIMATIC couplings, which means in the PROFIBUS DP, PROFINET IO, PLCSIM and PRODAVE couplings, are addressed via symbolic names where they have been defined.



PLCSIM FB-MOTOR-ON		
General	Property	Value
	Signal	PLCSIM FB-MOTOR-ON
	Display Gateway Name	<input checked="" type="checkbox"/>
	Info	I0.0

**Signals**

Source	Name

**Origin**  
Coupling

**Signal type**  
All

**Data type**  
All

[Reset filter](#)

▼ **Search results**

Source	Name
PLCSIM	CabinButtonFloor

▼ **Info**

Origin: Coupling

Signal type: Input

Data type: binary

**About: E33.4**

### Symbolic addressing

In SIMATIC couplings, each I/O signal can be given a symbolic name.

Default	Symbol Name	Address	Data Type	Comment
<input type="checkbox"/>	FeedbackRun	I33.5	BOOL	Antrieb laeuft
<input type="checkbox"/>	FeedbackStop	I33.6	BOOL	Antrieb steht
<input type="checkbox"/>	DoorOpened	I33.7	BOOL	Sensor Tuer auf
<input type="checkbox"/>	DoorClosed	I34.0	BOOL	Sensor Tuer zu
<input type="checkbox"/>	OpenDoorManually	I34.1	BOOL	Tuer Hand auf
<input type="checkbox"/>	CloseDoorManually	I34.2	BOOL	Tuer Hand zu

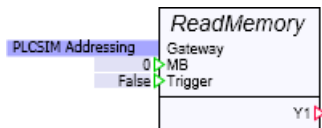
If a symbolic name was entered, the signal can only be accessed by using this symbol name. If there is no symbolic name, the signal is accessed by its address.

### 2.3.1.5 Accessing data blocks and bit memories

#### Overview

The PLCSIM coupling and the PRODAVE coupling enable access to the bit memory address area and the data block area. This access is not carried out by cyclic communication between the controller and signals that are listed in the coupling editor, but via components which read or write a specified address area of the controller on a trigger signal. The required component types can be found in the basic library in the directory *COMMUNICATION | SIMATIC*.

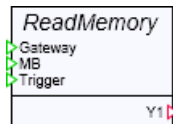
The components must be provided with a *Unit* connector at their *Gateway* input as shown in the figure below. You link the components with the relevant coupling by entering the name of the PLCSIM or PRODAVE coupling that you want to access using this component in the property window of the unit connector. Entering the address in the Unit connector is of no significance in this case.



To use this access method for a coupling, you must have already saved the coupling. To do this, open the coupling in the editor, define, for example, an input or output signal and then save the coupling.

#### ReadMemory – Reading a bit memory address area

##### Symbol



##### Function

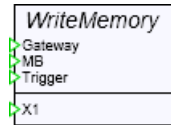
The *ReadMemory* component type reads one or more successive bytes from the bit memory address area of a controller. Enter the address of the first byte to be read at the input *MB*. The read operation is executed when a rising edge occurs at the *Trigger* input, which means a change from False to True.

The number N of outputs can be varied by "stretching" the component on a chart. You can specify a maximum of 32 outputs and read a maximum of 32 bytes with a component of this type. The bytes that are read are available at the outputs *Y1* to *YN*.

Exactly one read operation is triggered while the simulation is being initialized. This means initial values from the bit memory address area are available after initialization even though no trigger signals have been sent.

## WriteMemory – Writing to a bit memory address area

### Symbol



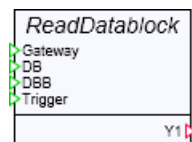
### Function

The *WriteMemory* component type allows you to write to one or more successive bytes in the bit memory address area of a controller. Enter the address of the first byte to be written to at the input *MB*. The write operation is executed when a rising edge occurs at the *Trigger* input, which means a change from False to True.

The number *N* of inputs can be varied by "stretching" the component on a chart. You can specify a maximum of 32 inputs, which means you can write to a maximum of 32 bytes in the memory bit area with a component of this type. The bytes to be written should be made available at inputs *X1* to *XN*.

Exactly one write operation is triggered while the simulation is being initialized. Initial values can therefore be written to the bit memory address area during the initialization process, even though no trigger signals have been sent.

## ReadDatablock – Reading a data block



The *ReadDatablock* component type enables one or more successive bytes from a data block of a controller to be read. Enter the data block number at the *DB* input and the address of the first byte to be read at the *DBB* input. The read operation is executed when a rising edge occurs at the *Trigger* input, which means a change from False to True.

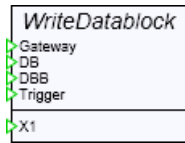
The number *N* of outputs can be varied by "stretching" the component on a chart. You can specify a maximum of 32 outputs, which means you can read a maximum of 32 bytes with a component of this type. The bytes that are read are available at the outputs *Y1* to *YN*.

Exactly one read operation is triggered while the simulation is being initialized. Initial values from the data block are available after initialization, even though no trigger signals have been sent.

### Note

This component only works with a PLCSIM or PRODAVE coupling.

## WriteDatablock – Writing to a data block



The *WriteDatablock* component allows you to write to one or more successive bytes in the data block of a controller. Enter the data block number at the *DB* input and the address of the first byte to be written at the *DBB* input. The write operation is executed when a rising edge occurs at the *Trigger* input, which means a change from False to True.

The number *N* of inputs can be varied by "stretching" the component on a chart. You can specify a maximum of 32 inputs, which means you can write to a maximum of 32 bytes in the data block with a component of this type. The bytes to be written should be made available at inputs *X1* to *XN*.

Exactly one write operation is triggered while the simulation is being initialized. Initial values can therefore be written to the data block during the initialization process, even though no trigger signals have been sent.

---

### Note

This component only works with a PLCSIM or PRODAVE coupling.

---

## 2.3.2 PROFIBUS DP coupling

### 2.3.2.1 How the PROFIBUS DP coupling works

The PROFIBUS DP coupling allows SIMIT to communicate with one or several PROFIBUS DP masters. To do this, SIMIT maps the characteristics of the PROFIBUS DP slaves on the bus and enables data communication between the PROFIBUS DP masters and SIMIT. Special DP interface modules are required for communicating with the PROFIBUS DP masters.

---

### Note

SIMIT simulates the characteristics of the PROFIBUS DP slaves. It basically does not matter which PROFIBUS DP master is connected. However, the configuration of the PROFIBUS DP coupling assumes that the hardware configuration to be simulated was created using the HW Config tool (STEP 7 software).

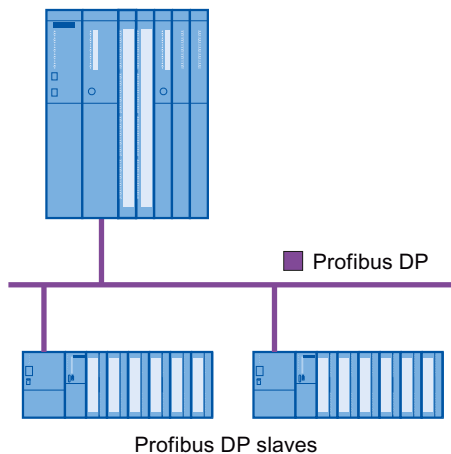
---

## How the PROFIBUS DP coupling works

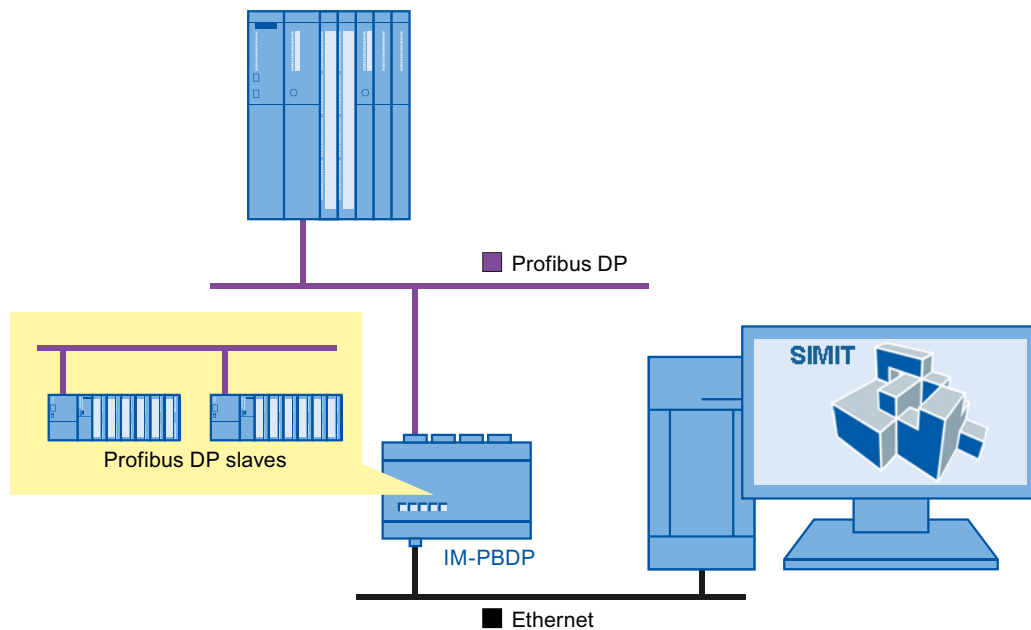
To illustrate how the PROFIBUS DP coupling of SIMIT works, the figure below assumes an automation configuration consisting of a non-redundant PLC and a PROFIBUS DP master system. The coupling between SIMIT and the PLC then occurs using a special PROFIBUS DP interface module, whereby the PROFIBUS DP master is no longer connected to the PROFIBUS slaves but rather to the interface module.

The interface module simulates the configured PROFIBUS DP slaves, which means it communicates with the PROFIBUS DP master of the PLC in the same way as the PROFIBUS DP slaves. Otherwise communication via the PROFIBUS DP would not be error-free.

PLC with one Profibus DP master



PLC with one Profibus DP master



## The PROFIBUS DP interface module

The PROFIBUS DP interface module is available in several versions:

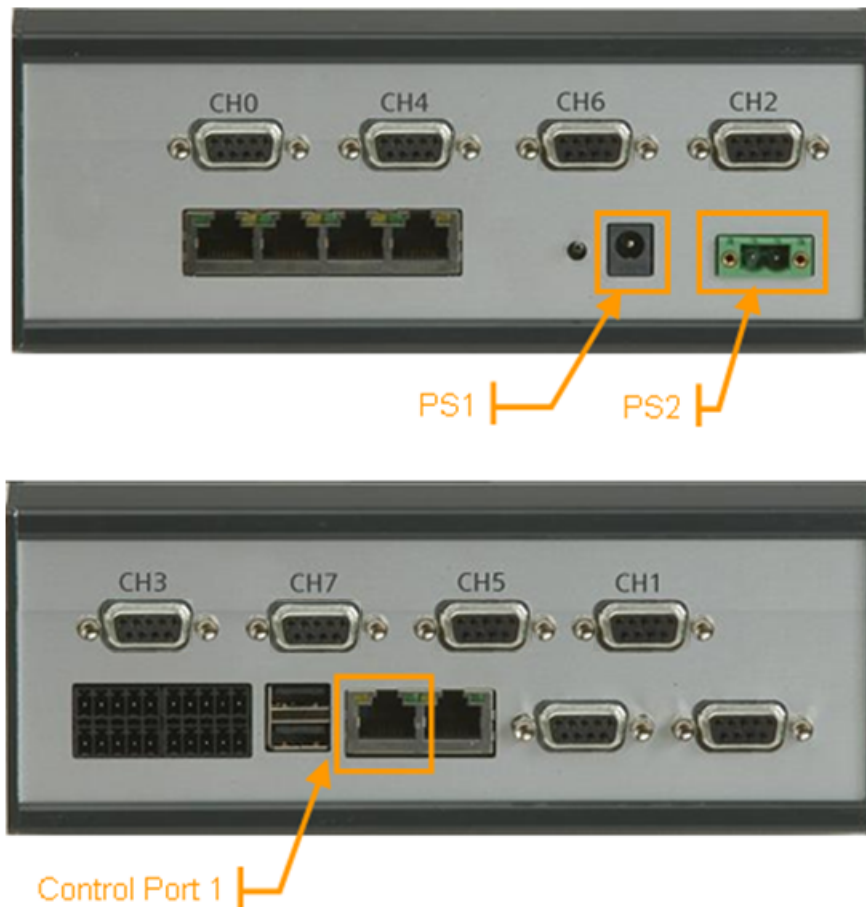
- IM-PBDP-2  
Two-channel interface module for simulating a maximum of 125 PROFIBUS slaves per channel.
- IM-PBDP-4  
Four-channel interface module for simulating a maximum of 125 PROFIBUS slaves per channel.
- IM-PBDP-8  
Eight-channel interface module for simulating a maximum of 125 PROFIBUS slaves per channel.

The interface module is always equipped with eight PROFIBUS DP connectors: four on the top and four on the bottom of the interface module. Depending on the variant only the first two, the first four or all eight connectors are active (labeled as CH0 through CH7).

Plug the connecting cable to your PC into the RJ45 socket labeled *Control Port 1*. A crossed or uncrossed LAN cable (twisted pair) can be used as the cable, because the connection on the interface module supports auto-crossover.

You need a 24V – 1300mA DC power supply to operate the interface module. This power supply is not supplied with the interface module. The interface module has a connection (PS1) for a power supply with a round connector. Alternatively you can also use a suitable power supply, for example, the 24V power supply of a SIMATIC S7-300, and connect it to power supply connector PS2 with a 2-pin plug-in terminal block.





### Supported PROFIBUS DP configurations

Typical PROFIBUS DP configurations that are supported by SIMIT are listed below. The left column of each figure (see below) shows the configuration of the automation system, and the right column shows the layout with SIMIT. The following examples of configurations with non-redundant controllers are shown:

- A controller with a PROFIBUS DP subnet,
- Two controllers, each with a single PROFIBUS DP subnet,
- One controller with two PROFIBUS DP subnets and
- Two controllers with one shared PROFIBUS DP subnet.

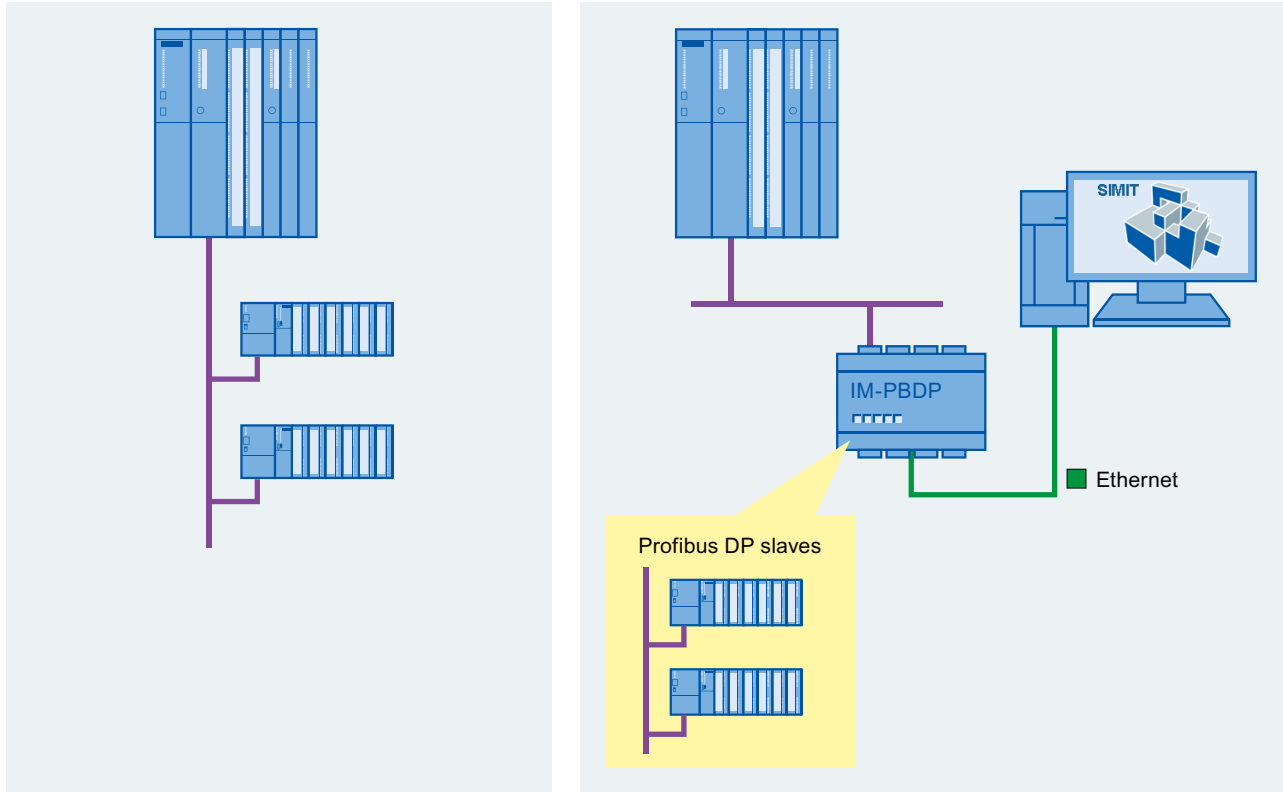
The following examples of redundant configurations are provided:

- A redundant controller with a redundant PROFIBUS DP subnet,
- two redundant controllers, each with a redundant PROFIBUS DP subnet and
- two redundant controllers with a one-sided PROFIBUS DP subnet.

For all configurations, a mixed configuration of real and simulated PROFIBUS DP slaves is possible.

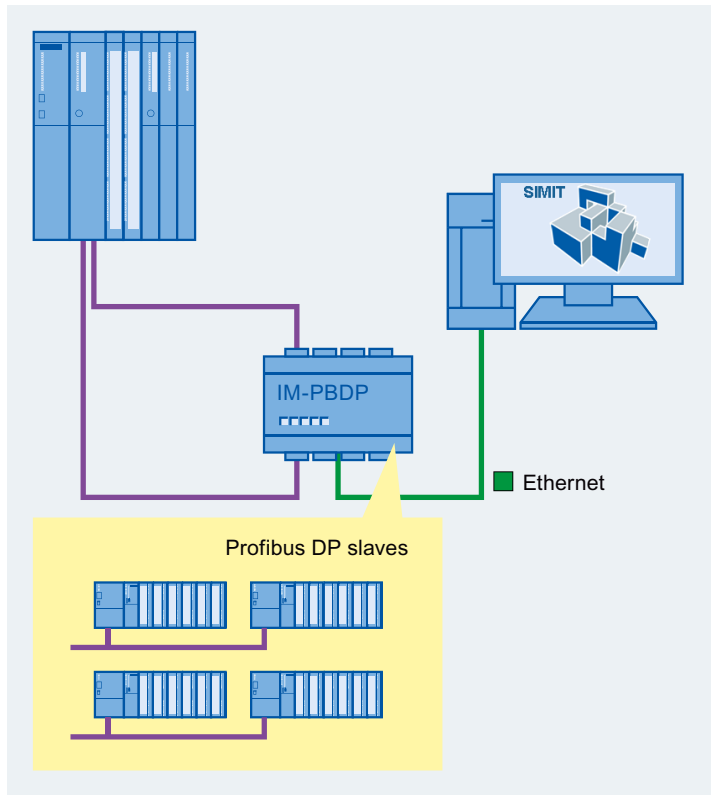
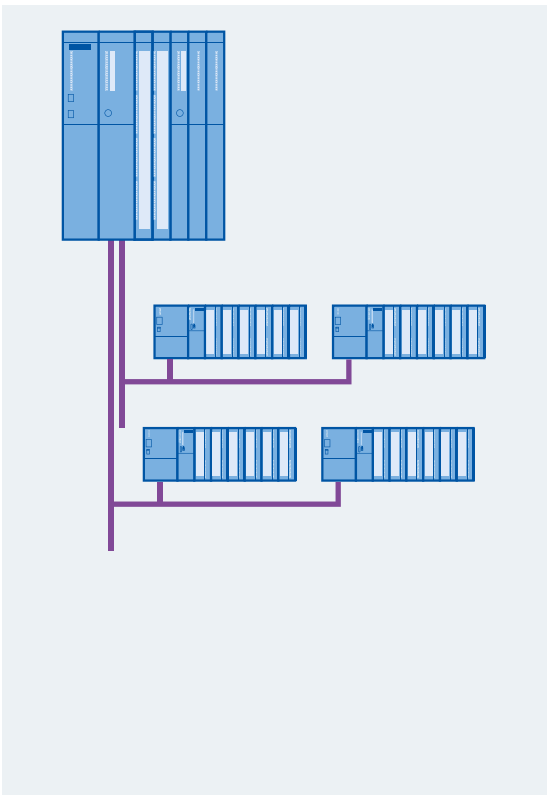
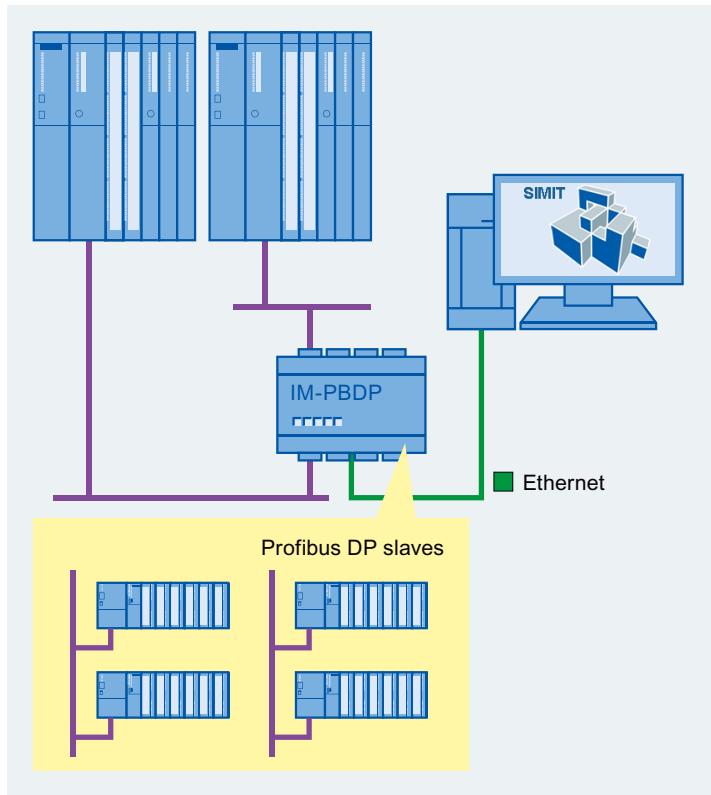
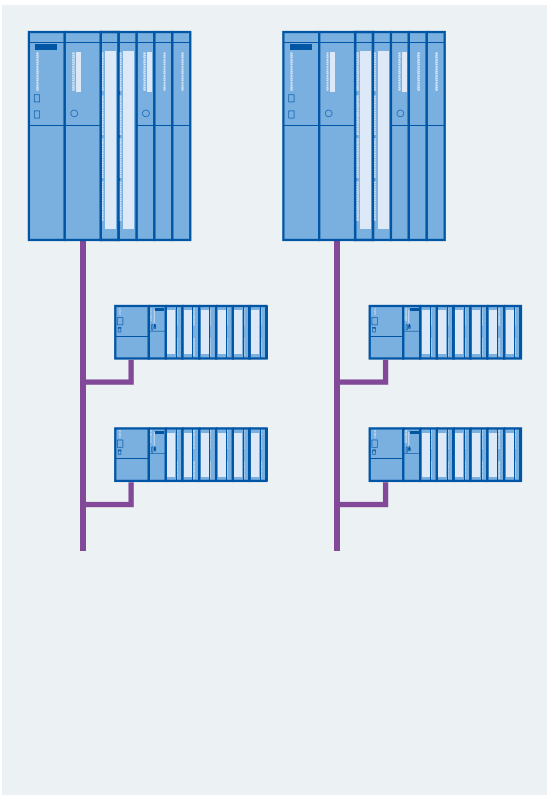
Bus couplers can also be used, either as DP-DP couplers, as DP-PA couplers, or in redundant configurations as a Y-link.

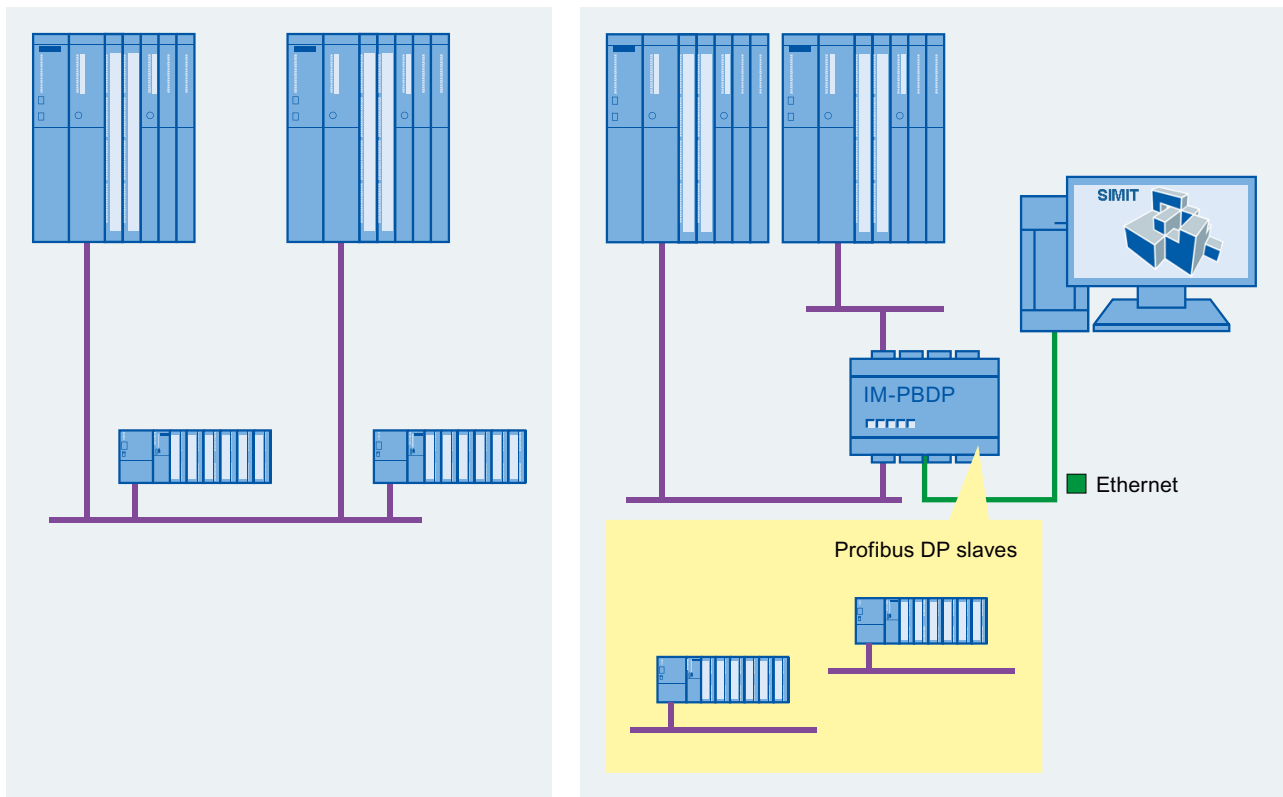
Fail-safe I/Os can also be simulated in redundant H/F systems.



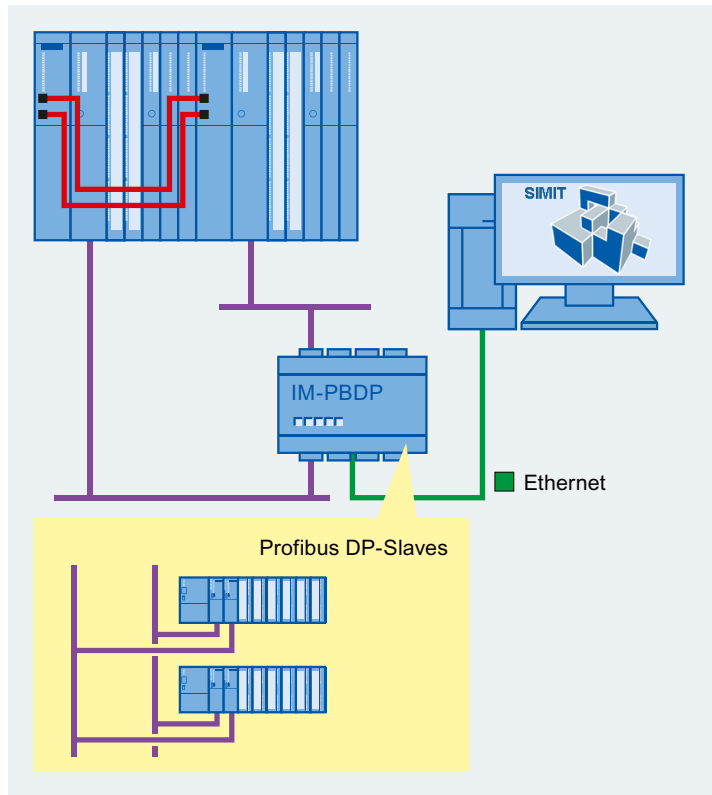
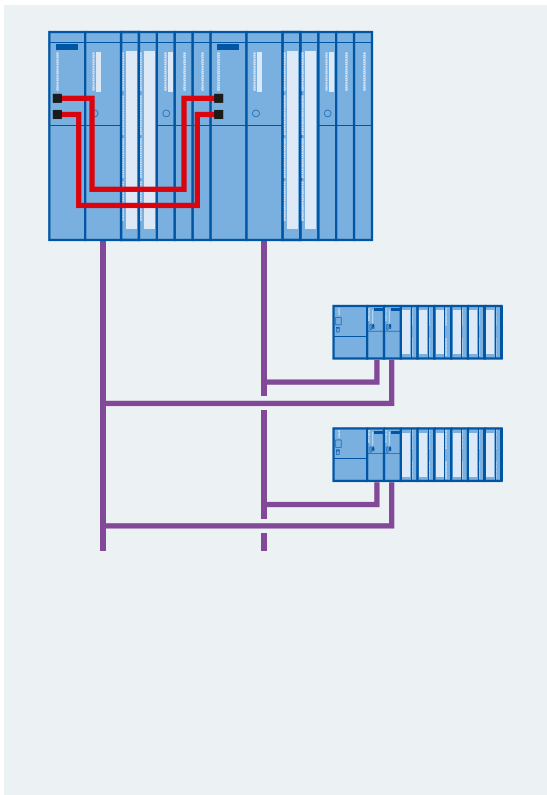
**Note**

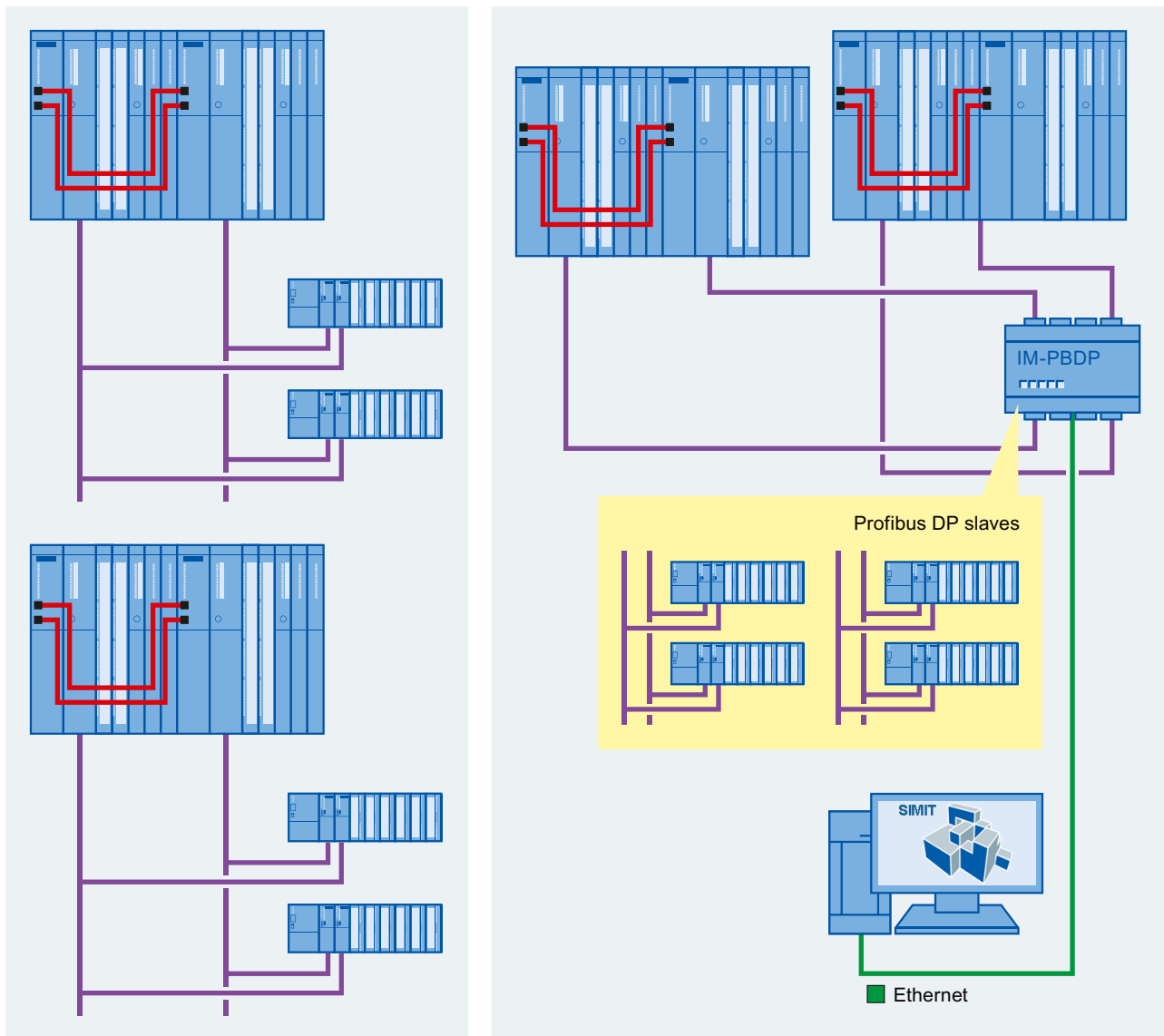
In this configuration, only one channel of a multi-channel module is used.

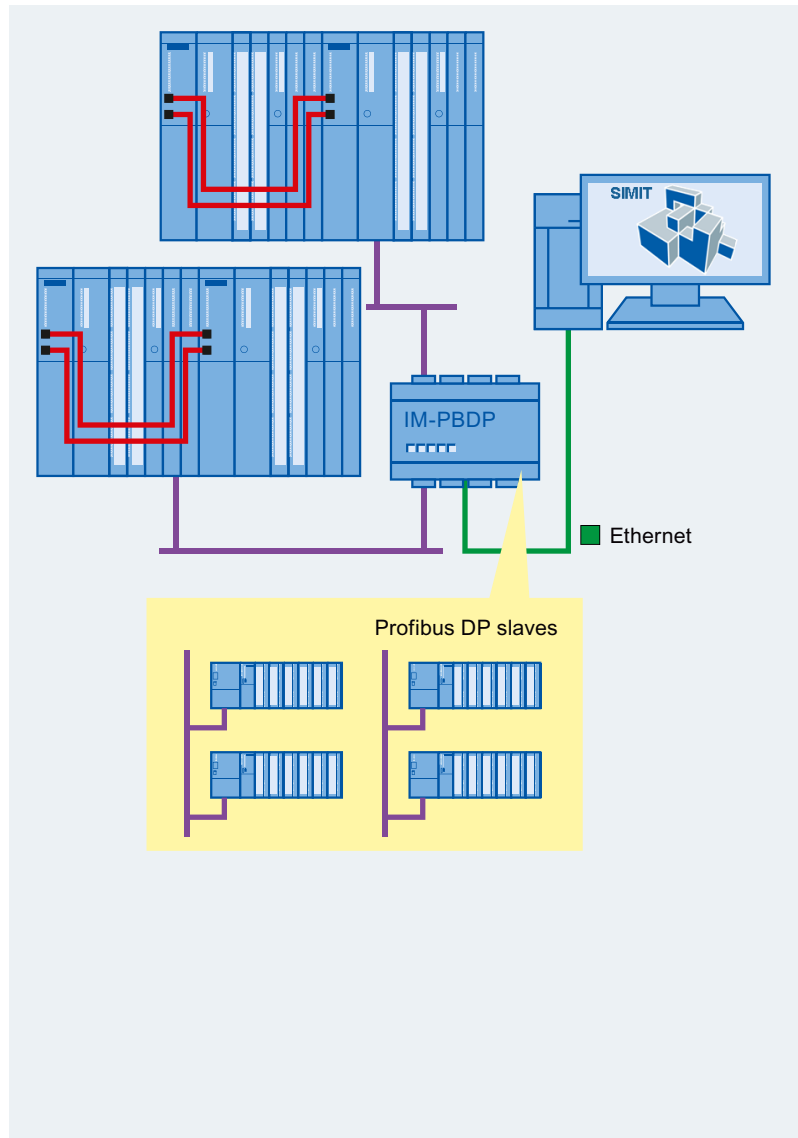
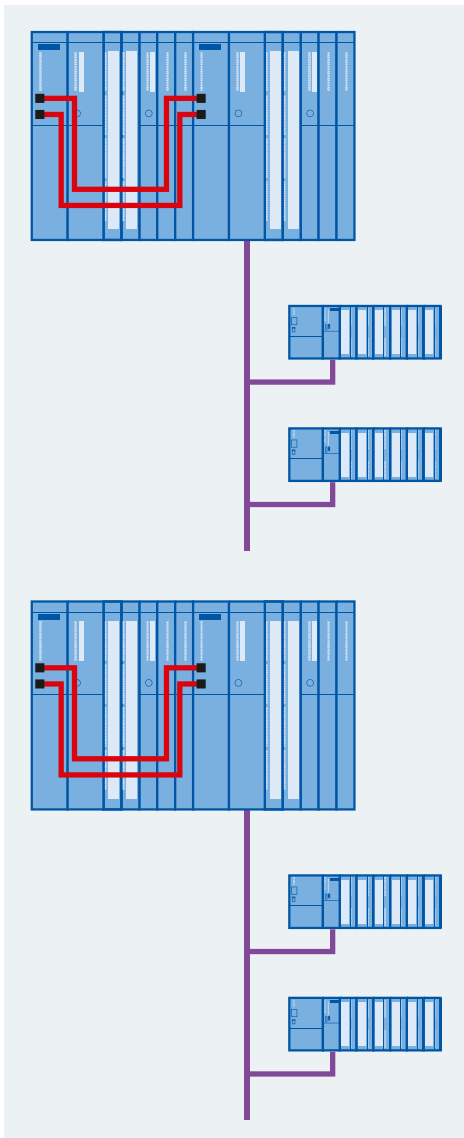




For each of the two PLCs in this configuration, corresponding system data blocks are created with HW Config. These system data blocks must be imported to configure the two channels of the interface module (you can find additional information in the section: Configuring the PROFIBUS DP coupling (Page 88)).




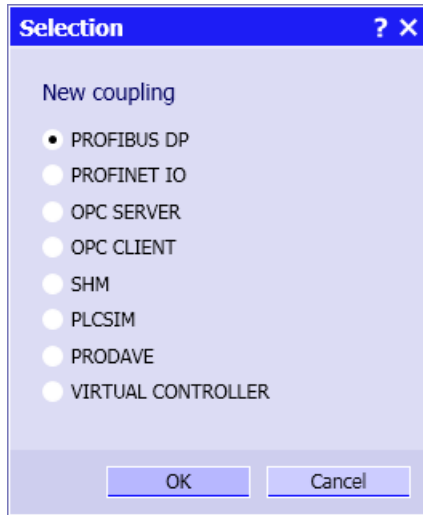




### 2.3.2.2 Configuring the PROFIBUS DP coupling

#### Creating a coupling

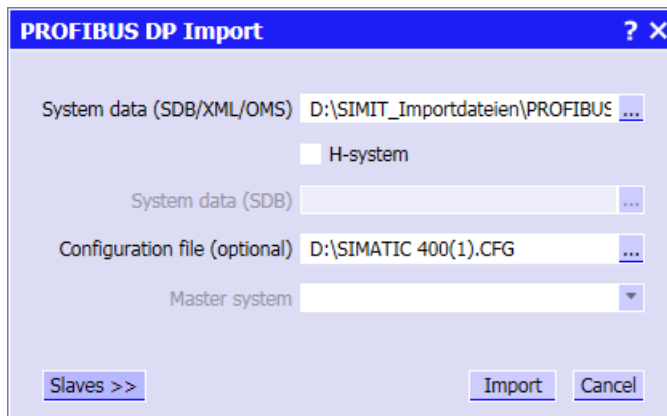
You can create a new PROFIBUS DP coupling either by selecting the command "Couplings > Add new coupling" in the portal view or by double-clicking *New coupling* (  *New coupling* ) in the "Coupling" folder of the project tree and then selecting *PROFIBUS DP* in the selection dialog that appears.



You may use the default name *PROFIBUS* as name of the coupling or assign an arbitrary name.

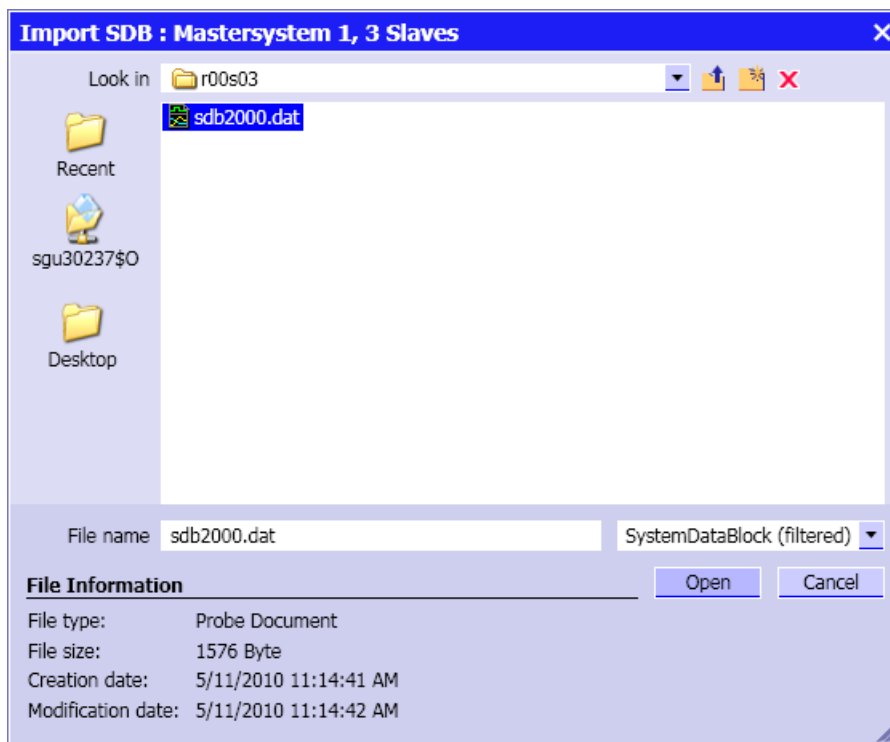
#### Importing system data blocks

After having created a coupling, select the system data block (sdb file) in the import dialog.



For each PROFIBUS DP master system a set of system data blocks is created (sdb files). To select the relevant system data block file set the file selection to *System data block (filtered)*. For the selected sdb file, the file selection dialog displays the number of the master system and the number of slaves in addition to general file information.





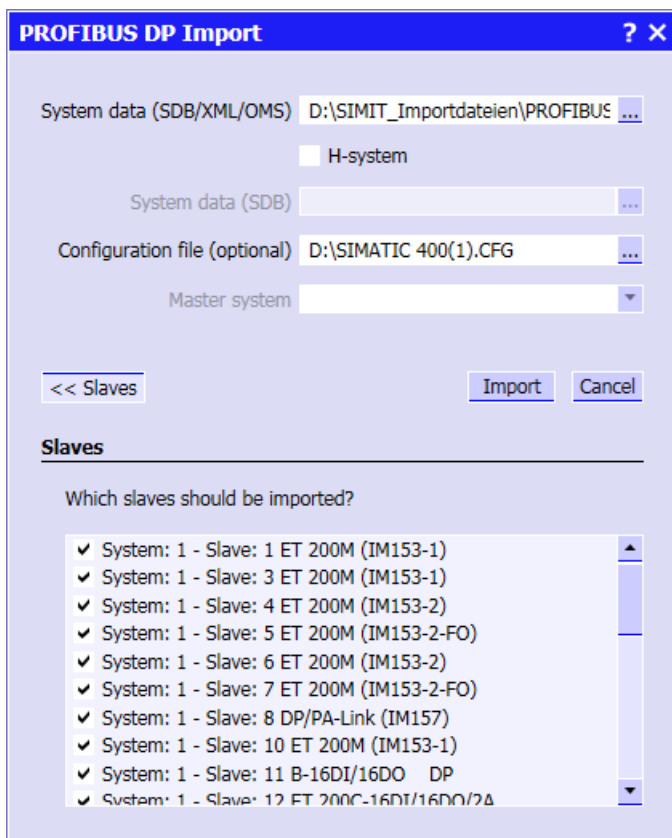
---

**Note**

System data blocks are newly created when the configuration is compiled in HW Config. After each change in HW Config, the updated sdb files need to be re-imported into the PROFIBUS DP coupling. Otherwise, the controller would contain different information about the PROFIBUS DP configuration than the coupling in SIMIT. Bus errors in the communication between the controller and SIMIT will occur as a result.

---

After selecting an sdb file you can use the *Slaves>>* button to open a preview of all of the slaves to be imported. If you do not wish to simulate individual slaves but rather connect them as real slaves, uncheck them in the preview.

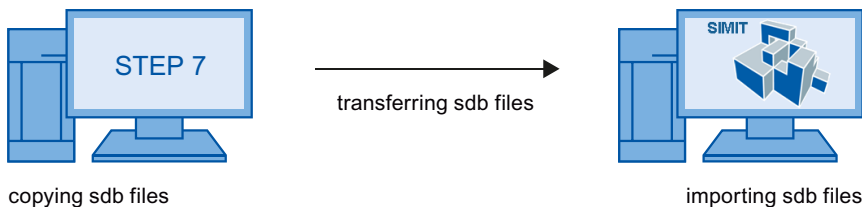


After a successful import the coupling editor will open with the imported signals in the work area. Finally, save the coupling using the menu command (Save).

You can open the import dialog as shown in the figure above again at any time using the symbol (Import hardware), for example, to import a modified PROFIBUS DP configuration.

### Copying system data blocks

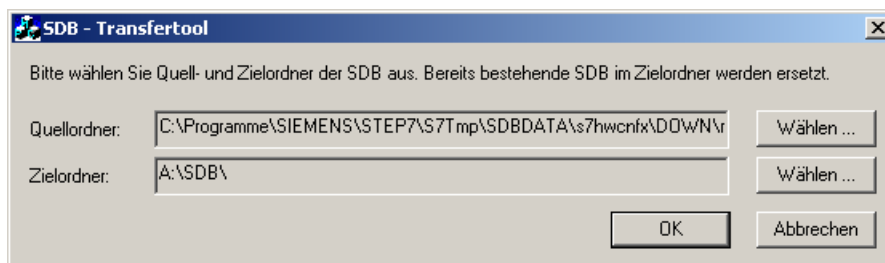
If SIMIT and the STEP 7 software that you use to edit your SIMATIC project are not installed on the same PC, you need to transfer the sdb files from the STEP 7 PC to the SIMIT PC.



The sdb files in the temporary folder on the SIMATIC PC should not be deleted after compiling the hardware configuration in HW Config. After installing your STEP 7 software, double-click the *UnlockHWConfig.exe* file, which can be found in the *\_tools\SIMATIC* folder on the SIMIT CD.

We provide a utility program to copy the sdb files. You can also find the *CopyHWConfig.exe* program in the *\_tools\SIMATIC* folder on the SIMIT CD. This program does not need to be

installed and can be started from any location. The figure below shows the dialog window of this program:



We suggest you use the temporary STEP 7 folder when selecting the source folder.

### Support for TIA Portal for transferring the hardware configuration

You can import a hardware configuration into SIMIT from TIA Portal V11 and V13.

The file format depends on the TIA version and the automation system used:

TIA V11:

- An XML file is generated when an S7-1200 is used
- An sdb file is generated when an S7-300 or S7-400 is used

TIA V13:

- An oms file is generated when an S7-1200 or S7-1500 is used
- An sdb file is generated when an S7-300 or S7-400 is used

The TIA Portal stores the configuration files in the file system under the folder *<Shared Documents>|TIAExport*.

---

#### Note

You cannot import system data from TIA V12 into SIMIT.

---

You can find information about importing symbol tables in the section: Importing PLC variable lists (Page 70)

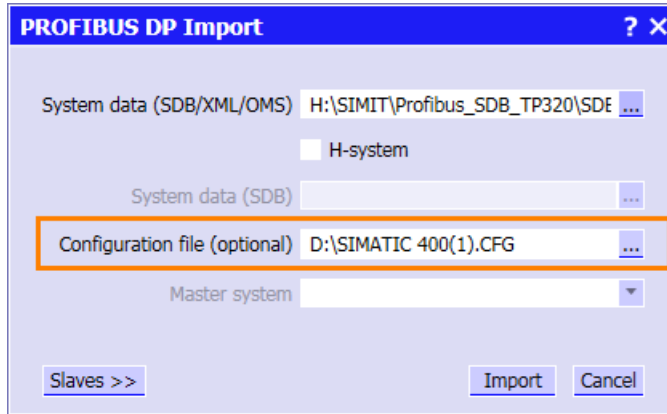
### Importing the configuration file

Usually, all information that is required for configuring the interface module can be read from the system data blocks. You only need the configuration file if your configuration uses one of the following HART modules of an ET200M station to correctly configure all signals for a module:

- 6ES7 331-7TF00-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7TF01-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7PF00-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7PF01-0AB0 AI8x16Bit HART/TC

- 6ES7 331-7PF10-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7PF11-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7SF00-0AB0 AI8x16Bit HART/TC

You can create the configuration file using the hardware configuration tool (HW Config) in Step 7 by exporting in .cfg format. The import dialog allows you to select the configuration file.



### Importing the signal properties

Import the signal properties by clicking the "📄" symbol in the coupling editor.

The "Import of signal properties" dialog box opens. You can make the import settings in this dialog box.

You can find additional information in the section: Import of signal properties (Page 66).

### Exporting the signal properties

Export the signal properties by clicking the "📄" symbol in the coupling editor.

The "Export of signal properties" dialog box opens. You can make the export settings in this dialog box.

You can find additional information on this in the section: Export of signal properties (Page 69).

## Assigning a channel of the interface module

In the property view, you assign the coupling of one of the available channels of the PROFIBUS DP interface modules that are configured in SIMIT. All available channels are offered in the *Hardware channel* selection box using the name of the respective interface module. After saving, a channel that was assigned in a coupling will no longer be available and will therefore no longer be offered for selection.

PROFIBUS		
Property	Value	
PROFIBUS		
System 1		
[1] ET 200M (IM153-1)		
[3] ET 200M (IM153-1)		
[4] ET 200M (IM153-2)		
[5] ET 200M (IM153-2-FO)		
[6] ET 200M (IM153-2)		
[7] ET 200M (IM153-2-FO)		
[8] DP/PA-Link (IM157)		
[10] ET 200M (IM153-1)		
Master address	2	
Hardware channel	Not assigned	
Baud rate	Not assigned	
H-system	IM_PBDP_8[0]	
F-system	IM_PBDP_8[1]	
	IM_PBDP_8[2]	
	IM_PBDP_8[3]	
	IM_PBDP_8[4]	
	IM_PBDP_8[5]	
	IM_PBDP_8[6]	
	IM_PBDP_8[7]	

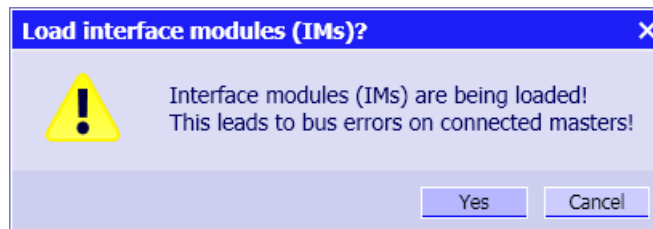
### Note

For PROFIBUS and PROFINET together, a maximum of 30 channels can be used, which can be distributed to the simulation units as desired.

## Loading the interface module

If you have configured a coupling or have changed its configuration, the configuration must be loaded into the interface modules before starting the simulation. The interface modules do not behave in general like real field devices in their communication with the controller (which means with the PROFIBUS DP masters in the controller), until they have been loaded with the correct PROFIBUS DP configuration for the PLC.

If you start the simulation for a project that contains a PROFIBUS DP coupling, a check is automatically performed during startup to see whether the interface modules have already been loaded with the correct configuration data. If not, you can use the dialog below to load the interface modules or to cancel the start of the simulation. The simulation can only be started if the interface modules have been loaded with the correct configuration data.



This dialog also appears when starting the simulation if your PC has been switched off since the last loading.

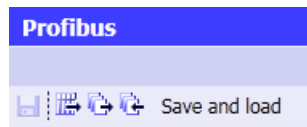
---

**Note**

If you switch off the DP interface modules, i.e. disconnect the supply voltage, or access the configuration via an application other than SIMIT, SIMIT can no longer ensure that the loaded configuration matches the SIMIT project. This may lead to unpredictable behavior.

---

You can also initiate the loading of the interface modules manually at any time. To do this click the *Save and Load* command in the toolbar of the PROFIBUS DP coupling editor.



---

**Note**

While the interface modules are being loaded, their communication with the connected PROFIBUS DP masters is interrupted for a short time at least. Automation systems could therefore go into the "Stop" state.

---

### Configuration in RUN (CiR)

CiR allows for certain changes to the hardware configuration of a plant to be performed during ongoing operation.

SIMIT does not support this function. You need to change your hardware configuration in HW Config and re-import the modified system data blocks into the PROFIBUS DP coupling. Then the interface modules need to be reloaded.

#### 2.3.2.3 Editing the PROFIBUS DP coupling

##### Editing signals

The coupling signals are displayed in the coupling editor in an input section and an output section. As usual in SIMIT, input signals are inputs of the PLC, output signals are outputs of the PLC. Signals are displayed in the coupling editor window with the following properties:

- Symbol name
- Address
- Data type
- Master system number
- Slave number
- Module slot number
- Comment

- Scaling
- Lower
- Upper

All properties of the currently selected signal are visible in the property view. Here you can, for example, set the scaling properties such as the type of *Scaling*, *Lower Scale Value* and *Upper Scale Value* for analog signals.

The screenshot shows the SIMATIC Manager interface. At the top, there is a 'Save and load' button. Below it, the 'Inputs' section is expanded, showing a table of input signals. The 'Increments' signal is selected, and its properties are displayed in the 'Outputs' section.

Default	Symbol name	Address	Data type	Syste	Slave	Slot	Comment	Scaling
<input type="checkbox"/>	CloseDoorManually	I34.2	BOOL	1	4	6	Tuer Hand zu	
<input type="checkbox"/>		I34.3	BOOL	1	4	6		
<input type="checkbox"/>	CabinFlush	I34.4	BOOL	1	4	6	Kabine buendig	
<input type="checkbox"/>		I34.5	BOOL	1	4	6		
<input type="checkbox"/>	CabinNearDoor	I34.6	BOOL	1	4	6	Kabine in Tuerbereich	
<input type="checkbox"/>		I34.7	BOOL	1	4	6		
900.0	Increments	IW40	WORD	1	5	4	Schachtzaehler	Unipolar
0		IW42	WORD	1	5	4		No scaling

Property	Value
Scaling	Unipolar
Lower scale value	0
Upper scale value	100
Unit	
Lower raw value	0
Upper raw value	27648

### Note

Only signals of the WORD type can be scaled.

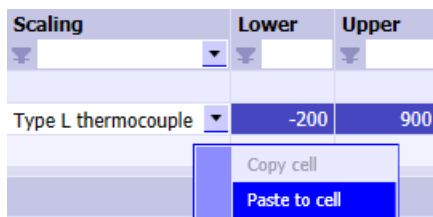
### Copying and inserting scaling values

It is now easy to transfer scaling from one signal to another or to a selection of signals. Select the cell you want to copy and then choose the *Copy cell* item in the shortcut menu.

The screenshot shows the 'Scaling' property view for a 'Type L thermocouple' signal. The 'Lower' and 'Upper' scale values are set to -200 and 900, respectively. A context menu is open over the 'Lower' cell, showing 'Copy cell' and 'Paste to cell' options.

Scaling	Lower	Upper
Type L thermocouple	-200	900

You can then select one or more target signals and choose *Paste to cell* in the shortcut menu for the selection.



The scaling type and the low and high value can be transferred in this way. This information has to be transferred column by column, however. In addition, you can only paste to cells that are enabled for editing. If the lower and upper values are not editable because no scaling has been set, for example, you need to set the scaling type first or transfer it from an existing signal. Only then can you transfer the associated values.

### Providing default signal values

The first column *Default* is used to provide default values for inputs, for example, to set binary inputs in the controller.

Save and load

▼ Inputs [Reset filter](#)

Default	Symbol name	Address	Data type	Syste	Slave	Slot	Comment	Scaling
<input type="checkbox"/>	CloseDoorManually	I34.2	BOOL	1	4	6	Tuer Hand zu	
<input type="checkbox"/>		I34.3	BOOL	1	4	6		
<input type="checkbox"/>	CabinFlush	I34.4	BOOL	1	4	6	Kabine buendig	
<input type="checkbox"/>		I34.5	BOOL	1	4	6		
<input type="checkbox"/>	CabinNearDoor	I34.6	BOOL	1	4	6	Kabine in Tuerbereich	
<input type="checkbox"/>		I34.7	BOOL	1	4	6		
▶ 900.0	Increments	IW40	WORD	1	5	4	Schachtzaehler	Unipolar
0		IW42	WORD	1	5	4		No scaling

► Outputs [Reset filter](#)

**Increments**

Property	Value
General	
Scaling	Unipolar
Limits	
Lower scale value	0
Upper scale value	100
Connection	
Unit	
Lower raw value	0
Upper raw value	27648

### Merging and splitting signals

When the system data blocks are imported, the signals belonging to the imported PROFIBUS DP configuration are created automatically in the coupling. This takes into account whether they are binary signals with the data width bit or signals with the data width byte, word (2 bytes), double word (4 bytes), etc.

The data width can be changed, for example 8 consecutive binary signals can be combined into a byte signal. Just select all signals to be merged and choose *Merge* in the shortcut menu.



**PROFIBUS\***

Save and load

Inputs Reset filter

Outputs Reset filter

Symbol name	Address	Data type	System	Slave	Slot
StaircaseIndicatorFloor0	Q10.0	BOOL	1	1	5
StaircaseIndicatorFloor1	Q10.1	BOOL	1	1	5
StaircaseIndicatorFloor2	Q10.2	BOOL	1	1	5
StaircaseIndicatorFloor3	Q10.3	BOOL	1	1	5
StaircaseIndicatorFloor4	Q10.4	BOOL	1	1	5
InOperation	Q10.5	BOOL	1	1	5
Direction	Q10.6	BOOL	1	1	5
V0	Q10.7	BOOL	1	1	5
CabinIndicatorFloor0	QW570	WORD	1	4	4
CabinIndicatorFloor1	QW572	WORD	1	4	4
CabinIndicatorFloor2	QW574	WORD	1	4	4

Merge

Similarly, signals can be also be split. Just select the signal to be split and choose *Split* in the shortcut menu.

**PROFIBUS\***

Save and load

Inputs Reset filter

Outputs Reset filter

Symbol name	Address	Data type	System	Slave	Slot
QB10	QB10	BYTE	1	1	5
CabinIndicatorFloor0	QW570	WORD	1	4	4
CabinIndicatorFloor1	QW572	WORD	1	4	4
CabinIndicatorFloor2	QW574	WORD	1	4	4

Split

The table below shows how signals can be converted. To merge signals, read the table from left to right; to split signals, read the table from right to left.

Number of signals	Data type	Data width		Number of signals	Data type	Data width
8	BOOL	1 bit	↔	1	BYTE	1 byte
2	BYTE	1 byte	↔	1	WORD (INT)	2 bytes
2	WORD (INT)	2 bytes	↔	1	DWORD (DINT, REAL)	4 bytes

#### Note

During a conversion, possibly existing symbol names of the signals are lost.

Signals can only be converted if they have been created in the PROFIBUS DP coupling by importing the system data blocks. Address ranges cannot be created or removed by changing the data width of signals, as the entire available address range is determined by the PROFIBUS DP configuration in SIMATIC Manager.

---

### Note

Signals that are not on the same module as well as fail-safe signals cannot be merged.

---

## Properties of the PROFIBUS DP coupling

After opening the PROFIBUS DP coupling the left side of the property view will show a tree structure of slaves and their modules.

Profibus		
Profibus	Property	Value
System 1	Cycle	2
[4] ET 200M (IM153-2)	Mnemonic	I/Q
[4] 6ES7 323-1BH00-0AA0	H-System	No
[5] 6ES7 323-1BH00-0AA0	F-System	No
[6] 6ES7 323-1BH00-0AA0		
[5] ET 200M (IM153-2)		
[4] 6ES7 331-7KB**-0AB0		
[5] 6ES7 332-5HB**-0AB0		
[6] ET 200M (IM153-2)		
[4] 6ES7 321-7BH0*-0AB0		
[5] 6ES7 331-7KB**-0AB0		

You can define the following properties of the coupling:

- Time slice**  
 Here you set the cycle in which the coupling exchanges data. The assignment of absolute cycle times to the eight available time slices affects the entire project in the project manager. Time slice 2 is the default, corresponding to a cycle of 100ms.

---

### Note

The time slice with the lowest cycle time always has the highest priority, regardless of the numbering.

---

- Mnemonics**  
 Here, you can select whether the international (*I/Q*) or German (*E/A*) syntax should be used for the designation of inputs and outputs.

The properties *H system* and *F system* are for informational purposes only. They indicate whether the system is redundant and/or fail-safe.

## Activating and deactivating slaves

The right side will show the properties for selected slaves. If you want to deactivate the selected slave, select the "Deactivate slave" check box. Then the slave is not active when simulation starts and is not simulated.

PROFIBUS			Property	Value
▼ PROFIBUS			Slave	ET 200M (IM153-1)
▼ System 1			PROFIBUS ID	801D
▼ [1] ET 200M (IM153-1)			PROFIBUS address	1
[4] 6ES7 332-5HF00-0AB0	AO8		Fail-safe	No
[5] 6ES7 322-1*H01-0AA0	DO16		Deactivate slave	<input type="checkbox"/>
▼ [3] ET 200M (IM153-1)				

You can also set this property of a slave after the simulation has started. This allows you to test the reaction of the PLC when a slave fails or recovers.

## Pulling/Plugging modules

The module properties are also shown on the right side of the property view. Modules can be deactivated using the *Pull module* property.

PROFIBUS			Property	Value
▼ PROFIBUS			Module	6ES7 322-1*H01-0AA0 DO16
▼ System 1			Slot	5
▼ [1] ET 200M (IM153-1)			Output address range	AB9 - AB10
[4] 6ES7 332-5HF00-0AB0	AO8		Fail-safe	No
[5] 6ES7 322-1*H01-0AA0	DO16		Pull module	<input type="checkbox"/>
▼ [3] ET 200M (IM153-1)				
[4] 6ES7 331-7PF00-0AB0	AI8RTD			

In order to test the reaction of the controller to pulled modules, you can also set this property after the simulation has been started.

## Modules with hardware interrupts

SIMATIC PLCs use hardware interrupts to monitor binary or analog I/O signals: When an analog value is exceeded or undershot or when a binary value is set or reset, the cyclical operation of the controller is interrupted to allow the control program to react appropriately to this interrupt. Whether and under what conditions a PROFIBUS DP slave triggers hardware interrupts is determined in its configuration in the SIMATIC project (HW Config).

Hardware interrupts for the modular I/O systems ET200M and ET200S can be generated with SIMIT. The table below lists the interface modules and bus modules of the ET200M for which

hardware interrupts are supported in SIMIT. The next table after that lists the interface modules and modules of the ET200S for which hardware interrupts are supported in SIMIT.

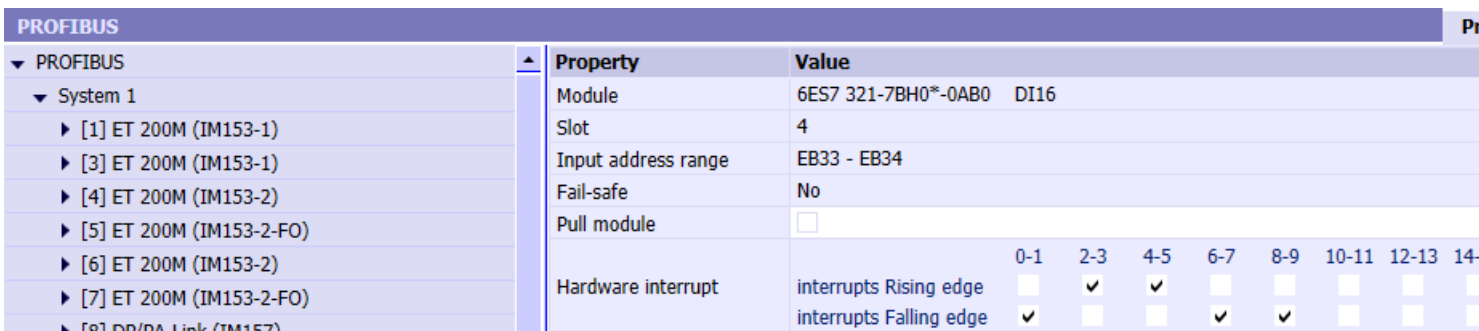
Table 2-7 Interface modules and modules within ET200M that support hardware interrupts

ET200M (IM153-1)		ET200M (IM153-2)		ET200M (IM153-3)	
6ES7 153-1AA01-0XB0		6ES7 153-2AA00-0XB0		6ES7 153-3AA00-0XB0	
6ES7 153-1AA02-0XB0		6ES7 153-2AA01-0XB0		6ES7 153-3AA01-0XB0	
6ES7 153-1AA03-0XB0		6ES7 153-2AA02-0XB0			
6ES7 153-1AA82-0XB0		6ES7 153-2BA00-0XB0			
6ES7 153-1AA83-0XB0		6ES7 153-2BA01-0XB0			
DI16	AI2	AI4	AI8		
6ES7 321-7BH00-0AB0	6ES7 331-7KB01-0AB0	6ES7 331-7RD00-0AB0	6ES7 331-7KF01-0AB0		
6ES7 321-7BH01-0AB0	6ES7 331-7KB02-0AB0		6ES7 331-7KF02-0AB0		
6ES7 321-7BH80-0AB0	6ES7 331-7KB81-0AB0		6ES7 331-7HF00-0AB0		
	6ES7 331-7KB82-0AB0		6ES7 331-7HF01-0AB0		
			6ES7 331-7NF00-0AB0		
			6ES7 331-7SF00-0AB0		

Table 2-8 Interface modules and modules within ET200S that support hardware interrupts

ET200S (IM151-1 HF)		
6ES7 151-1BA00-0AB0		
6ES7 151-1BA01-0AB0		
6ES7 151-1BA02-0AB0		
DI2	DI4	AI2
6ES7 313-4BB00-0AB0	6ES7 313-4CD00-0AB0	6ES7 313-4LB02-0AB0
6ES7 313-4BB01-0AB0	6ES7 313-4BD00-0AB0	
	6ES7 313-4BD01-0AB0	

To generate hardware interrupts in SIMIT, you must enable the required interrupts in the module properties. For binary signals, hardware interrupts can be generated on the rising or falling edge of the binary signal by settings in the module properties as shown in the figure below, for example:



The merging of several channels for one hardware interrupt is a typical SIMATIC feature and depends on how the corresponding module can be configured in the hardware manager.

In the case of analog signals an interrupt can be generated on exceeding or undershooting an adjustable limit. The limit is set as a percentage relative to the value range of the signal. 100% thus corresponds to a raw value of 27648. With scaled analog signals, this means the *upper scaling value* of the scale. If no percentage is entered, no hardware interrupt is triggered for this analog channel. The figure below shows a hardware interrupt configured to trigger by violating the high limit of 90% or a low limit of 10%.

PROFIBUS		Property	Value
▼ PROFIBUS		Module	6ES7 331-7**0*-0AB0 AI8
▼ System 1		Slot	5
▼ [1] ET 200M (IM153-1)		Input address range	EB538 - EB553
[4] 6ES7 332-5HF00-0AB0 AO8		Fail-safe	No
[5] 6ES7 322-1*H01-0AA0 DO16		Pull module	<input type="checkbox"/>
▶ [3] ET 200M (IM153-1)			
▼ [4] ET 200M (IM153-2)		Hardware interrupt	interrupts High limit violation
[4] 6ES7 332-5HF00-0AB0 AO8			0-1 % 2-3 % 4-5 % 6-7 %
[5] 6ES7 331-7**0*-0AB0 AI8			interrupts Low limit violation   % % % % %

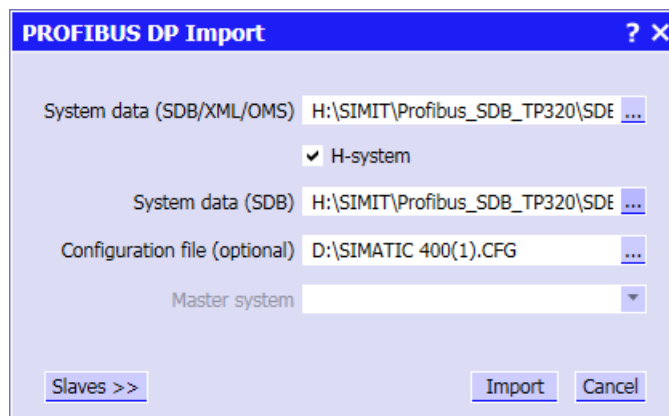
### 2.3.2.4 Redundant and fail-safe systems

#### Redundant configurations (H systems)

If your SIMATIC configuration contains a redundant PROFIBUS DP, this system can also be simulated redundantly.

In a redundant configuration, two system data block files are created when the hardware configuration is compiled in HW Config. These files must be read into SIMIT to configure the PROFIBUS DP coupling. To do this, select the "H-system" check box in the "PROFIBUS DP Import" dialog box.

One sdb file must be imported for each channel of the DP interface module.



In a redundantly connected system, all necessary redundant diagnostic signals are generated by the DP interface module so that you can also perform master/standby switchovers on your controller.

In SIMIT, only the signals from the first redundant PROFIBUS DP channel (channel 0) are used as coupling signals, regardless of which of the two buses is active. Bus switchovers are handled by SIMIT: The assignment of the coupling signals to the active channel is performed automatically. The redundant signals from the second channel are not listed. However, the signal list indicates that these signals are assigned to two master systems (1 and 2). The *H system* property tells you whether or not a slave is redundant.

**Profibus**

Save and load

▼ Inputs Reset filter

Default	Symbol name	Address	Data type	System	Slave	Slot	Comment
<input type="checkbox"/>		I0.0	BOOL	1 / 2	1	4	
<input type="checkbox"/>		I0.1	BOOL	1 / 2	1	4	
<input type="checkbox"/>		I0.2	BOOL	1 / 2	1	4	
<input type="checkbox"/>		I0.3	BOOL	1 / 2	1	4	

▼ Outputs Reset filter

Symbol name	Address	Data type	System	Slave	Slot	Comment
	Q0.0	BOOL	1 / 2	1	5	
	Q0.1	BOOL	1 / 2	1	5	
	Q0.2	BOOL	1 / 2	1	5	
	Q0.3	BOOL	1 / 2	1	5	

**Profibus**

Property	Value
▼ Profibus	
▼ System 1	
▶ [1] ET 200M (IM153-2) [H]	
▶ [3] ET 200M (IM153-2) [H]	
▼ System 2	
▶ [1] ET 200M (IM153-2) [H]	
▶ [3] ET 200M (IM153-2) [H]	
Master address	2
Hardware channel	IM_PBDP_8[0]
IP address	193.25.37.102
Baud rate	1.5 MBaud
H-system	Yes
F-system	No

DP/DP couplers can also be included as a Y link in redundant systems. The coupling signals of bus nodes that are connected behind the Y link are also available in SIMIT in the same way as the coupling signals from nodes that are connected directly to the redundant bus.

## Fail-safe configurations (F systems)

When a fail-safe configuration is imported, the associated quality signals are automatically created for all binary fail-safe signals and listed accordingly in the properties of the fail-safe signal.

**Profibus**

Save and load

▼ Inputs Reset filter

Default	Symbol name	Address	Data type	System	Slave	Slot	Comment
<input type="checkbox"/>		I24.0	BOOL	1	4	4	
<input type="checkbox"/>		I24.1	BOOL	1	4	4	
<input type="checkbox"/>		I24.2	BOOL	1	4	4	
<input type="checkbox"/>		I24.3	BOOL	1	4	4	

▼ Outputs Reset filter

Symbol name	Address	Data type	System	Slave	Slot	Comment
	Q56.0	BOOL	1	7	4	
	Q56.1	BOOL	1	7	4	
	Q56.2	BOOL	1	7	4	
	Q56.3	BOOL	1	7	4	

**I24.0**

General	Property	Value
Scaling	Symbol name	
Limits	Address	I24.0
Connection	Data type	BOOL
	Comment	
	Quality bit [I27.0]	<input checked="" type="checkbox"/>

All quality signals set to "1" are valid by default. If you wish to deliberately set a signal to "invalid" to test the reaction of your control program to a signal fault, for example, you only need to set the corresponding quality signal to zero. Just uncheck the *Quality bit* property.

### Note

After loading the interface module with a fail-safe configuration, the connected SIMATIC CPU needs to be restarted.

## Redundant and fail-safe configurations (H/F systems)

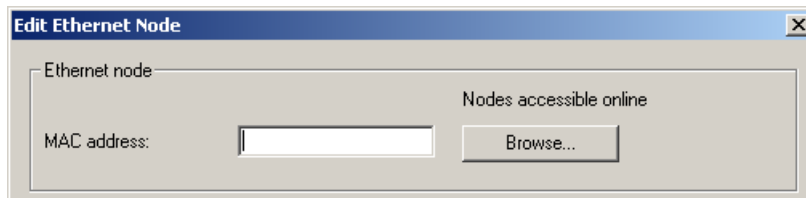
Redundant, fail-safe systems, also known as H/F systems, can also be simulated. The redundant, fail-safe signals and their quality signals are handled as described above.

Mixed configurations, which means fail-safe and non fail-safe bus nodes, as well as redundant and non-redundant bus nodes, can also be simulated.

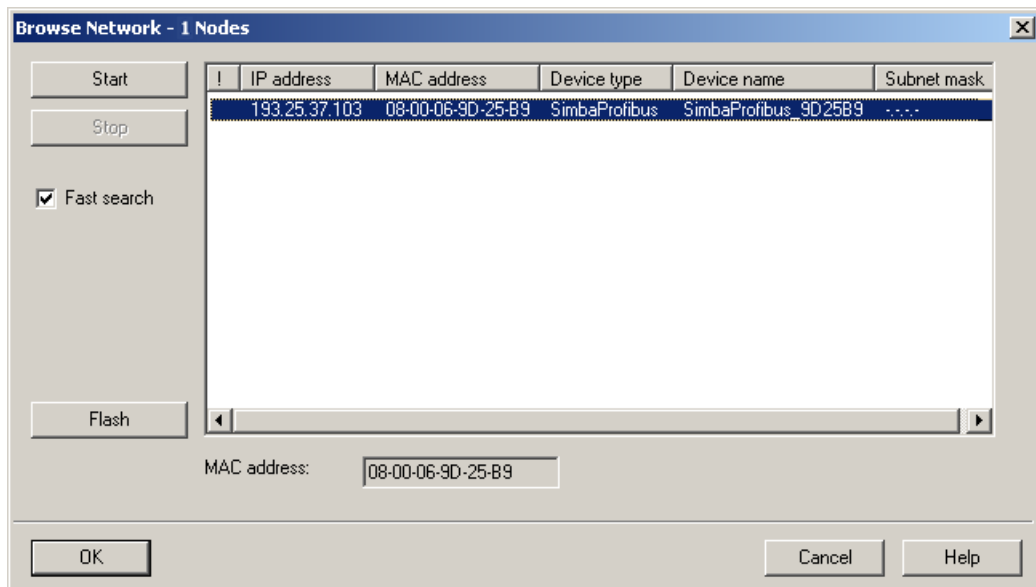
### 2.3.2.5 Configuring the PROFIBUS DP interface module

To access a PROFIBUS DP interface module with SIMIT via a network, you first need to assign a suitable IP address to the interface module. To do so, you need a PC with SIMATIC Manager installed and must – at least temporarily – connect the interface module to this PC. To configure the interface module, it does not matter whether you are using a direct connection to the PC or a switch or hub.

Configure the programming device/PC interface in SIMATIC Manager to use the network adapter to which the interface module is connected. Then choose *Target system / Edit Ethernet node* from the menu in SIMATIC Manager. In this dialog, click *Browse* in the *Nodes accessible online* section.



As shown in the figure below, a dialog appears that displays the nodes that were found. Select the entry for the interface module. It displays the device name *SIMIT IM-PBDP-x*. If you then click the *Flash* button, the LEDs on top of the interface module will flash briefly.



Confirm this dialog by clicking *OK*. In the previous dialog, you now see the interface module data. You can accept the settings for IP address and subnet mask or change them if necessary. In any case, confirm the dialog by clicking *Assign IP configuration*.



Now connect the interface module to your SIMIT PC again and enter the IP address of the interface module in the SIMIT *IM configuration* option dialog. Assign the interface module an arbitrary name so that it can be distinguished from other interface modules that may exist. When the *Update* button is clicked, the type and the status of the interface module are determined. The interface module is now ready for use with SIMIT.

IP address	Name	Type	Status
193.25.37.103	PBDP-Box1	Profibus 4 channels	Simba Profibus Rev. 4.004.5
*			

### 2.3.3 PROFINET IO coupling

#### 2.3.3.1 Functioning of the PROFINET IO coupling

The PROFINET IO coupling makes it possible for SIMIT to communicate with one or more PROFINET IO controllers.

SIMIT simulates the behavior of the PROFINET IO devices on the bus and performs data exchange between the PROFINET IO controllers and SIMIT. Special PROFINET IO interface modules are required for communication with the PROFINET IO controllers.

---

#### Note

SIMIT simulates the behavior of the PROFINET IO devices. In principle, it does not matter which PROFINET IO controller is connected. However, configuring the PROFINET IO coupling assumes that the hardware configuration to be simulated has been created with HW Config (STEP 7 software).

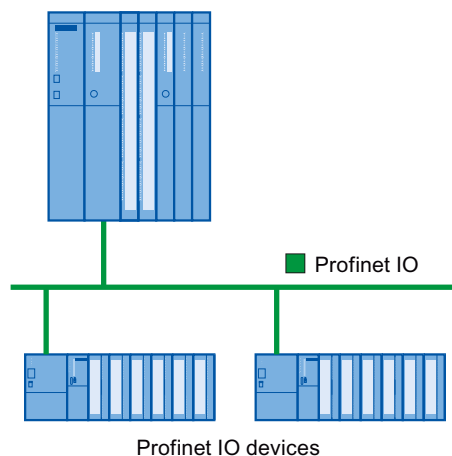
---

#### How the PROFINET IO coupling works

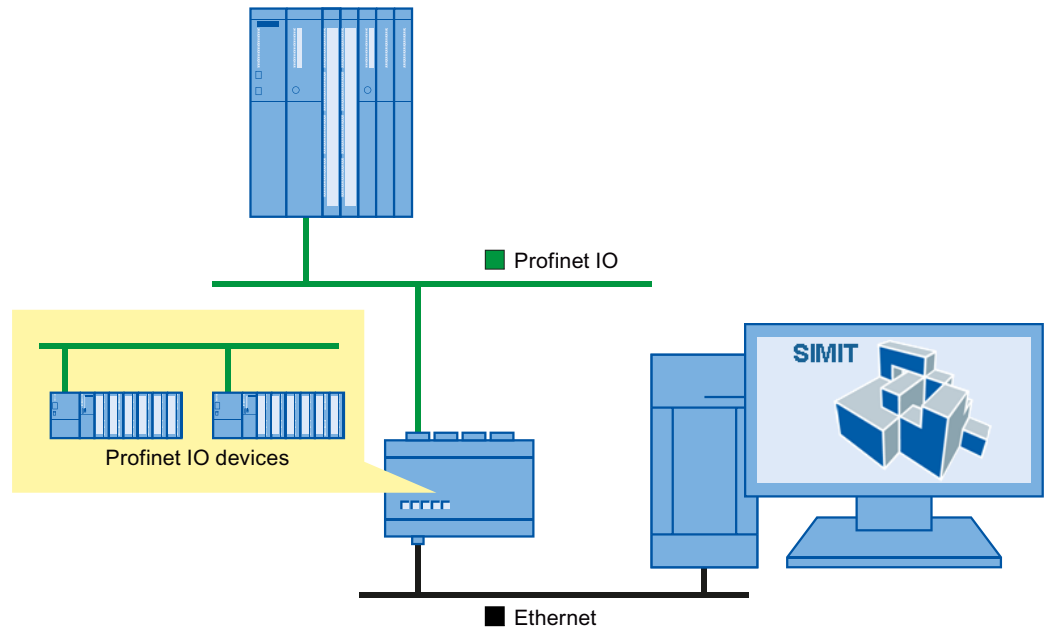
To illustrate how the SIMIT PROFINET IO coupling works, the figure below incorporates an automation configuration consisting of a non-redundant PLC and a PROFINET IO system. The coupling between SIMIT and the PLC then occurs using a special PROFINET IO interface module, whereby the PROFINET IO controller is no longer connected to the PROFINET IO devices but to the interface module instead.

The interface module simulates the configured PROFINET IO devices, which means it communicates with the PROFINET IO master of the PLC in the same way as the PROFINET IO devices. Otherwise, communication via the PROFINET IO would not be error-free.

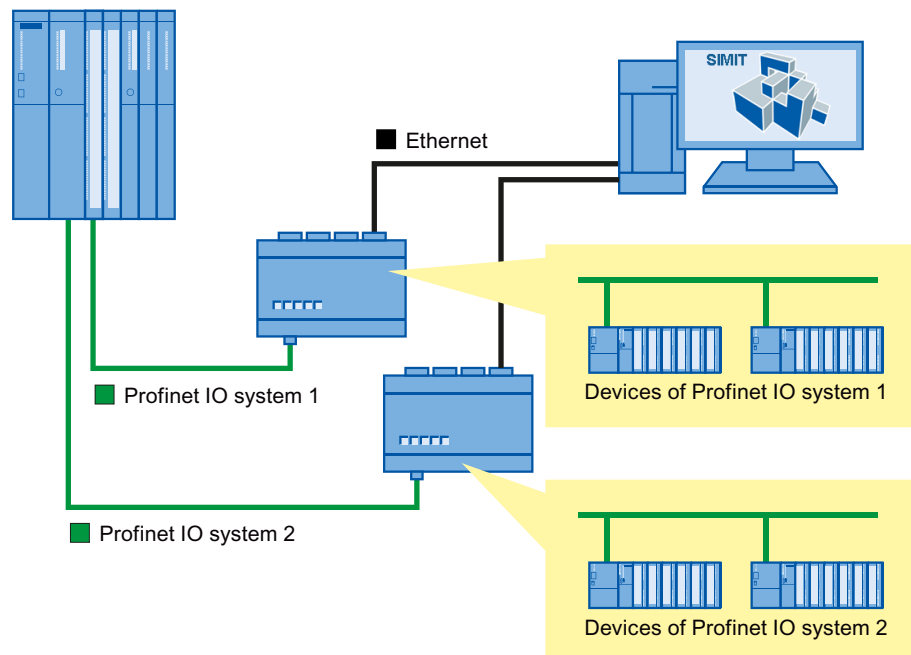
PLC with one Profinet IO controller



PLC with one Profinet IO controller



If there are several PROFINET IO systems in the plant configuration, an IM-PNIO interface module is required for each system. Each interface module has to be connected to one PROFINET IO controller and to the SIMIT PC. Each interface module then emulates the PROFINET IO devices that are configured in this PROFINET IO system.



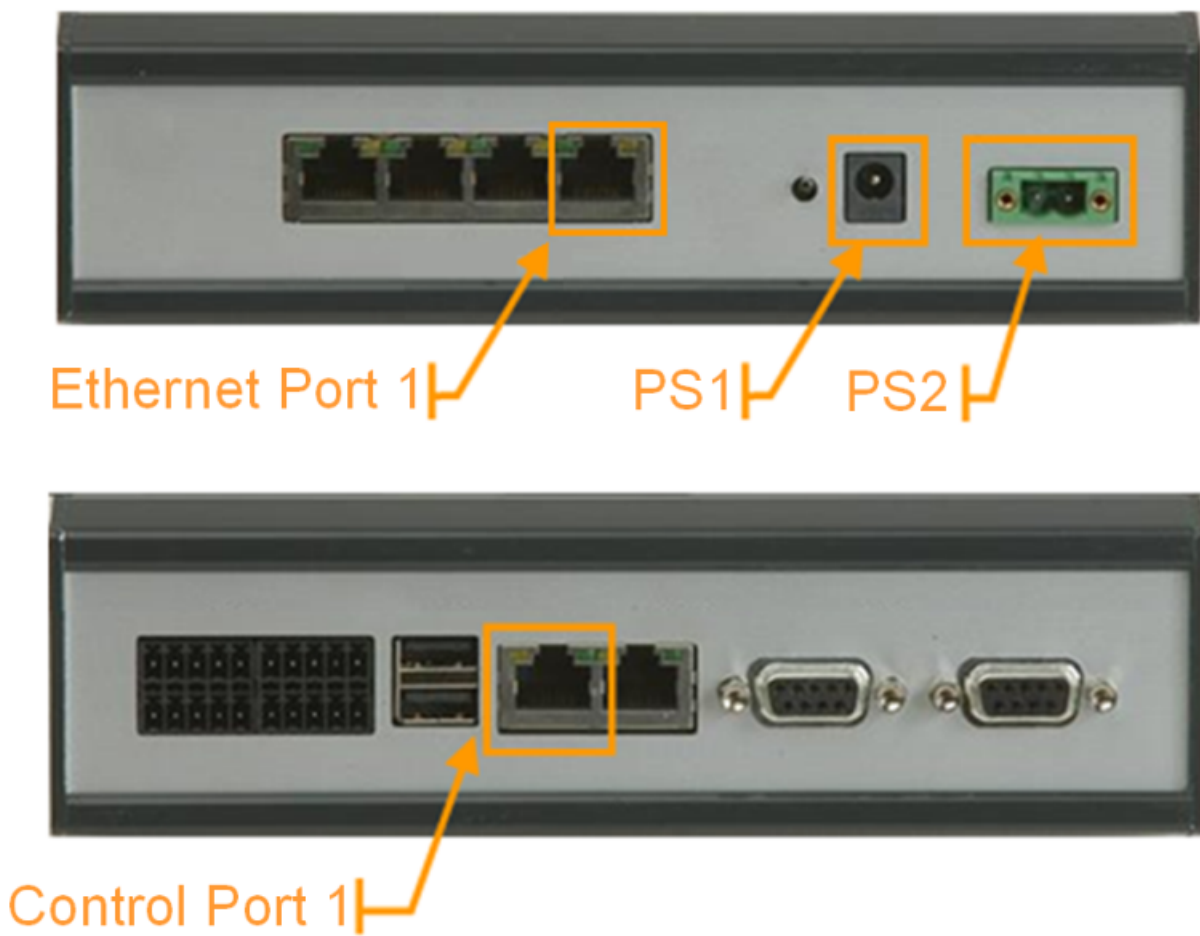
### The PROFINET IO interface module

The PROFINET IO interface module is available in different versions:

- IM-PNIO-128  
Interface module for simulating a maximum of 128 PROFINET IO devices in a PROFINET with RT protocol
- IM-PNIO-256  
Interface module for simulating a maximum of 256 PROFINET IO devices in a PROFINET with RT protocol

The interface module is always equipped with four PROFINET IO connectors but only the first one labeled *Ethernet Port 1* is usable.

You need a 24V – 1300mA DC power supply to operate the interface module. This power supply is not supplied with the interface module. The interface module has a connection (PS1) for a power supply with a round connector. Alternatively you can also use a suitable power supply, for example, the 24V power supply of a SIMATIC S7-300, and connect it to power supply connector PS2 with a 2-pin plug-in terminal block.



Plug the connecting cable to your PC into the RJ45 socket labeled *Control Port 1*. A crossed or uncrossed LAN cable (twisted pair) can be used as the cable, because the connection on the interface module supports auto-crossover.

---

**Note**

The network consisting of the IM-PNIO interface module and the SIMIT PC must not include additional nodes, because otherwise problems may occur in the communication between the interface module and SIMIT. This means the IM-PNIO should be the only device that is connected to this network adapter.

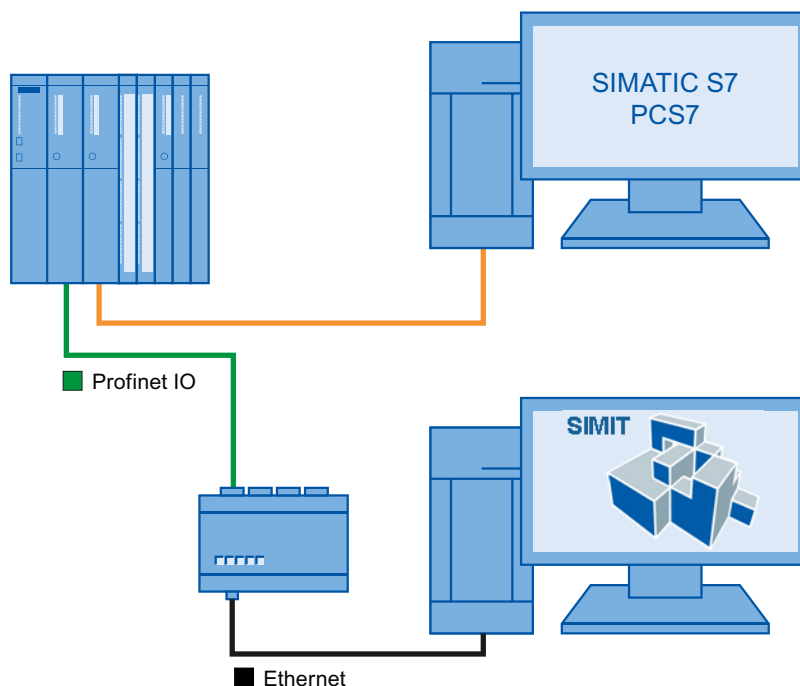
Assign an IP address to this network adapter that is unique across all networks that are configured in the SIMIT PC.

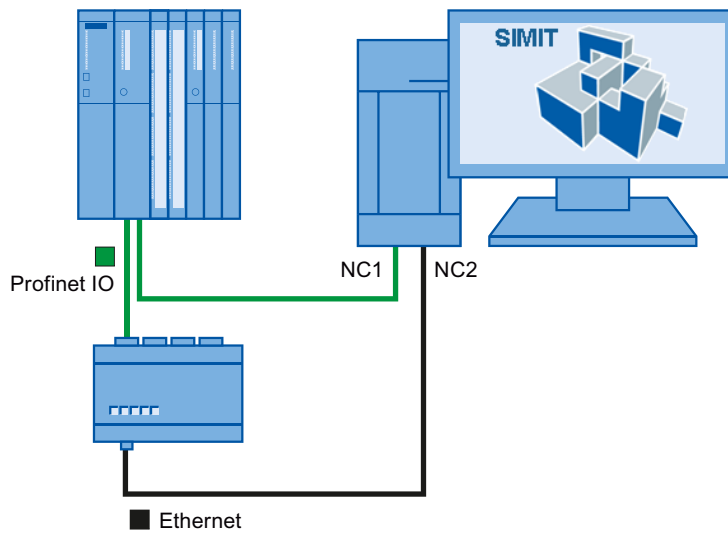
If you connect more than one interface module to the SIMIT PC, you must establish a connection to a separate network adapter in the PC for each interface module or you must use a network switch.

---

If you use a separate PC to configure and monitor your automation system using the SIMATIC or PCS 7 software, you need a connection from this PC to the automation system; it will not be connected to the SIMIT PC.

If you do not have a second communication processor in the automation system, the configuration shown in the figure at the very bottom is the result. The second network adapter (NC2) in the SIMIT PC is now connected to PROFINET via the connection with the PROFINET IO controller, and its IP address is thus also within the address range of the PROFINET.





---

**Note**


The Ethernet port and the Control port of the interface module must not be located in the same subnet. This is the reason why two network adapters are integrated into the PC in the example above.

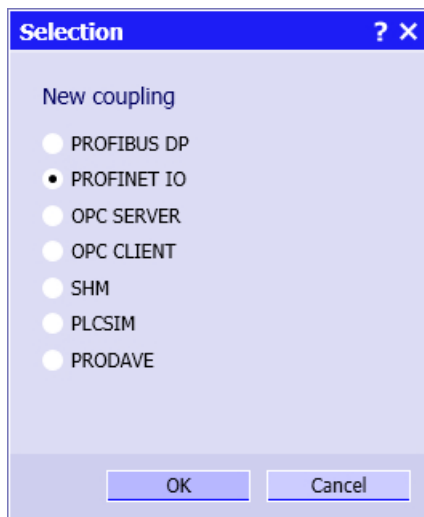
---

In all configurations, a mixed configuration of real and emulated PROFINET IO devices is possible. To do this, the real devices that are present are connected via a PROFINET switch to the PROFINET connection between the interface module and the SIMIT PC.

### 2.3.3.2 Configuring the PROFINET IO Coupling

#### Creating a coupling

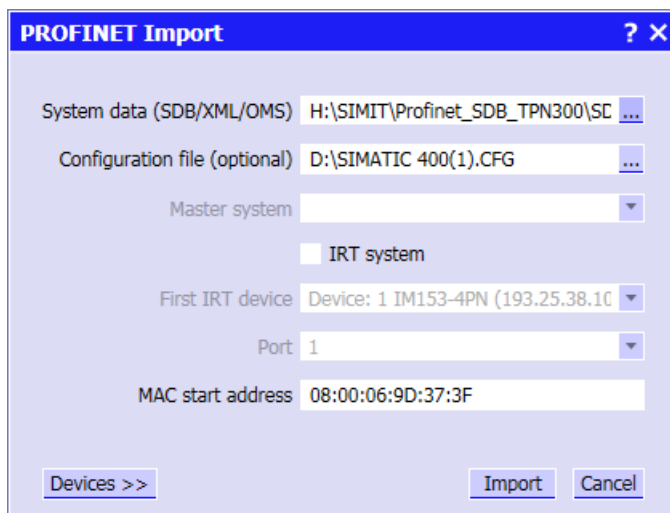
You can create a new PROFINET IO coupling either by selecting the command "Couplings > Add new coupling" in the portal view or by double-clicking *New coupling* (  *New coupling* ) in the "Coupling" folder of the project tree and then selecting *PROFINET IO* in the selection dialog that appears.



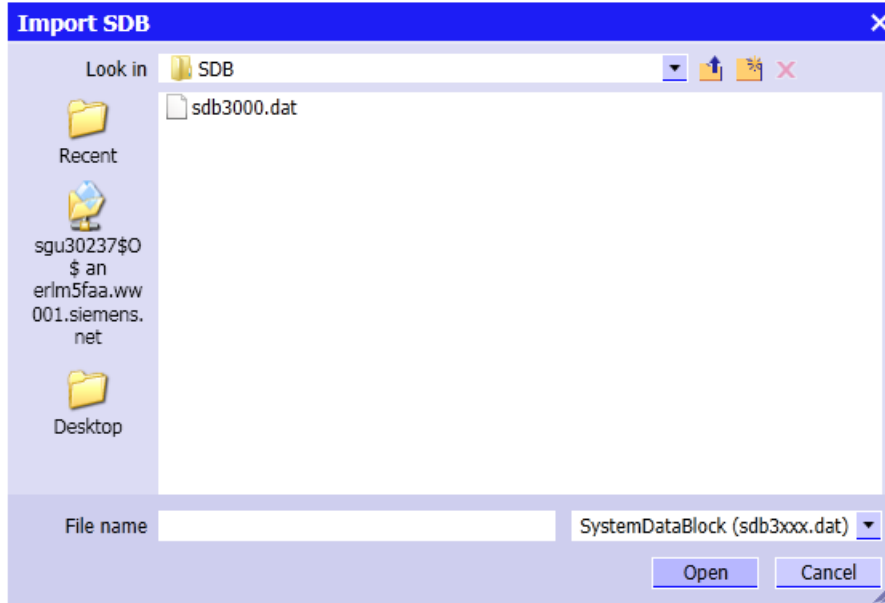
You may use the default name *PROFINET* as name of the coupling or assign an arbitrary name.

#### Importing system data blocks

After having created a coupling, select the system data block (sdb file) in the import dialog.



For each PROFINET IO system a set of system data blocks is created (sdb files). To select the relevant system data block file set the file selection box to *System data block (sdb3xxx.dat)*. The available sdb files are displayed in the file selection dialog.

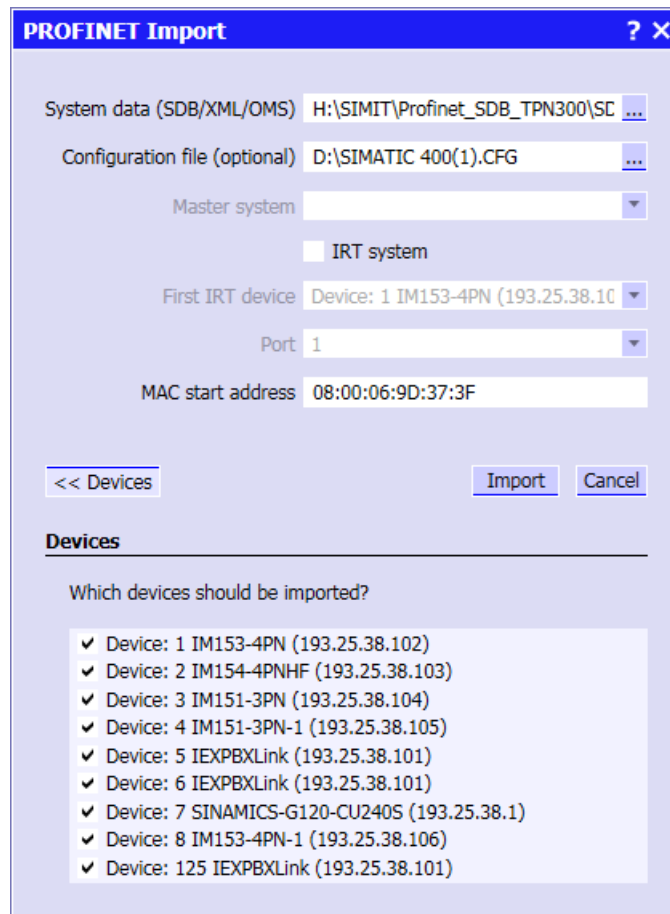



**Note**


System data blocks are newly created when the configuration is compiled in HW Config. After each change in HW Config, the updated sdb files need to be re-imported into the PROFINET IO coupling. Otherwise, the controller would contain different information about the PROFINET IO configuration than the coupling in SIMATIC. Bus errors in the communication between the controller and SIMATIC will occur as a result.

After selecting an sdb file you can use the *Devices>>* button to open a preview of all of the devices to be imported. If you do not wish to simulate individual devices but rather connect them as real devices, uncheck them in the preview.



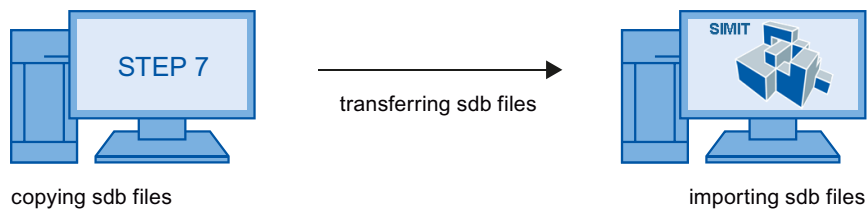


After a successful import the coupling editor will open with the imported signals in the work area. Finally, save the coupling using the menu command  (Save).

You can open the import dialog as shown in the figure at the top again at any time using the  symbol (Import hardware), for example, to import a modified PROFINET IO configuration.

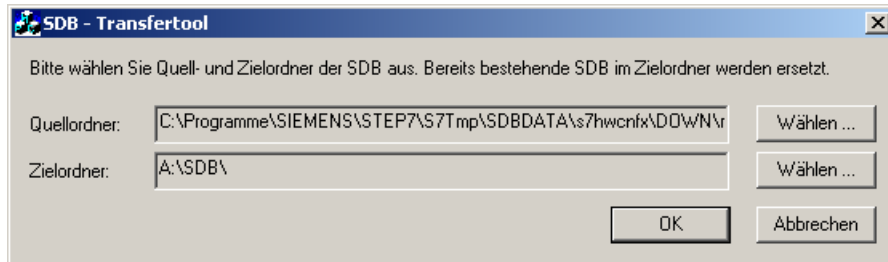
### Copying system data blocks

If SIMIT and the STEP 7 software that you use to edit your SIMATIC project are not installed on the same PC, you need to transfer the sdb files from the STEP 7 PC to the SIMIT PC.



The sdb files in the temporary folder on the SIMATIC PC should not be deleted after compiling the hardware configuration in HW Config. After installing your STEP 7 software, double-click the *UnlockHWConfig.exe* file, which can be found in the *\_tools/SIMATIC* folder on the SIMIT CD.

We provide a utility program to copy the sdb files. You can also find the *CopyHWConfig.exe* program in the *\_tools/SIMATIC* folder on the SIMIT CD. This program does not need to be installed and can be started from any location. The figure below shows the dialog window of this program:



We suggest you use the temporary STEP 7 folder when selecting the source folder.

### Support for TIA Portal for transferring the hardware configuration

You can import a hardware configuration into SIMIT from TIA Portal V11 and V13.

The file format depends on the TIA version and the automation system used:

TIA V11:

- An XML file is generated when an S7-1200 is used
- An sdb file is generated when an S7-300 or S7-400 is used

TIA V13:

- An oms file is generated when an S7-1200 or S7-1500 is used
- An sdb file is generated when an S7-300 or S7-400 is used

The TIA Portal stores the configuration files in the file system under the folder *<Shared Documents>|TIAExport*.

---

#### Note

You cannot import system data from TIA V12 into SIMIT.

---

You can find information about importing symbol tables in the section: Importing PLC variable lists (Page 70)

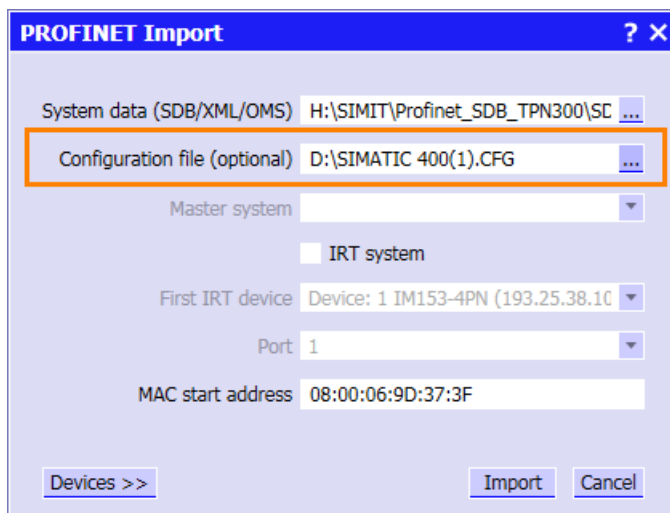
### Importing the configuration file

Usually, all information that is required for configuring the interface module can be read from the system data blocks. You only need the configuration file if your configuration uses one of the following HART modules of an ET200M station to correctly configure all signals for a module:

- 6ES7 331-7TF00-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7TF01-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7PF00-0AB0 AI8x16Bit HART/TC

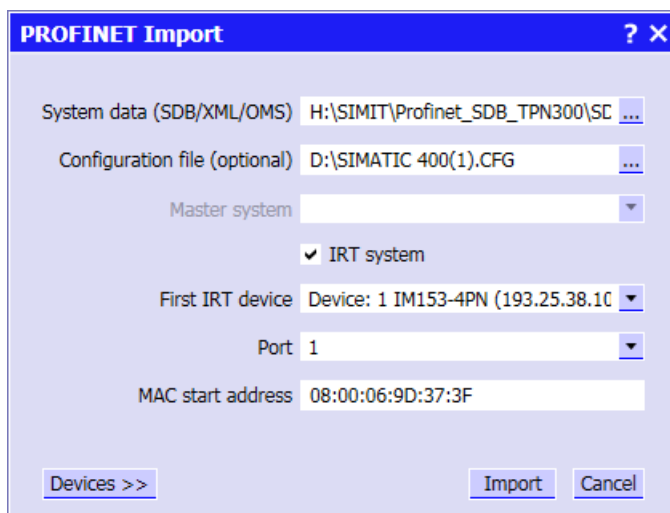
- 6ES7 331-7PF01-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7PF10-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7PF11-0AB0 AI8x16Bit HART/TC
- 6ES7 331-7SF00-0AB0 AI8x16Bit HART/TC

You can create the configuration file using the hardware configuration tool (HW Config) in Step 7 by exporting in .cfg format. The import dialog allows you to select the configuration file.



### Configuring the coupling for IRT systems

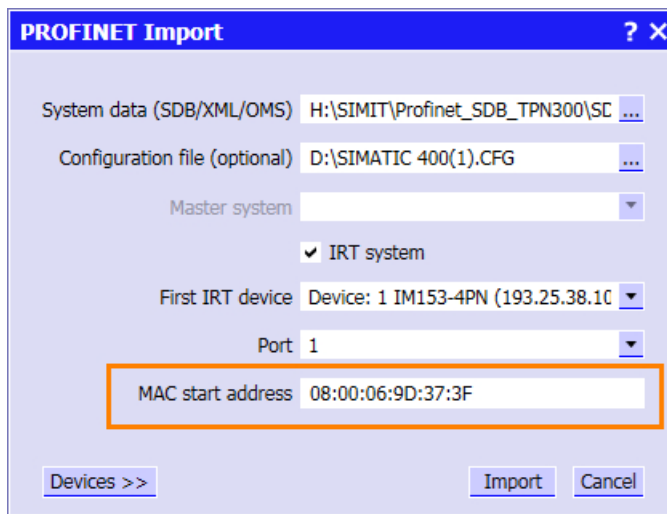
The PROFINET IO coupling also supports the IRT protocol. When importing the system data blocks the option *IRT system* must be checked and the IRT device that is the first device in the network configuration (from the point of view of the PROFINET controller) must be selected (*First IRT device*).



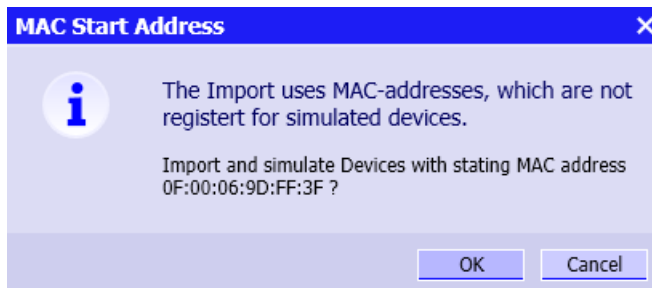
### Assigning a MAC start address

All PROFINET devices modeled in SIMIT are assigned their own MAC address during import of system data. This MAC address starts with the MAC start address, which is 08:00:06:9D:37:3F by default.

If you are only using a single PROFINET interface module you may leave this setting unchanged. If you are using several PROFINET interface modules that are located in the same subnet, however, the simulated devices must not have the same MAC addresses. In this case, you need to specify different MAC start addresses when importing system data for different PROFINET couplings within a project. MAC addresses reserved for simulation are located in the range from 08:00:06:9D:37:3F to 08:00:06:9D:38:3F.



You can assign any MAC addresses. Note, however, that address conflicts with other network nodes can occur outside of the range specified above. This is why you are required to confirm those MAC addresses in this case.



#### Note

SIMIT checks only whether the MAC start address is within the reserved range. SIMIT does not check whether all devices beginning with this address that are imported are completely within the reserved range.

The coupling is assigned the MAC start address selected during the import, and this address is displayed in the property view. This start address will also be used as the default for a new import.

PROFINET	
Property	Value
Time slice	2
Mnemonic	I/Q
F-system	No
IRT system	No
MAC start address	08:00:06:9D:37:3F

### Note

If you want to change the default MAC address, you need write access to the "cfg" folder below the installation folder of the SU software.

## Importing the signal properties

Import the signal properties by clicking the "🔗" symbol in the coupling editor. The "Import of signal properties" dialog box opens. You can make the import settings in this dialog box.

You can find additional information in the section: Import of signal properties (Page 66).

## Exporting the signal properties

Export the signal properties by clicking the "🔗" symbol in the coupling editor. The "Export of signal properties" dialog box opens. You can make the export settings in this dialog box.

You can find additional information on this in the section: Export of signal properties (Page 69).

## Assigning a channel of the interface module

In the property view, you assign one of the available channels of the PROFINET IO interface modules that are configured in SIMATIC to the coupling. All available channels are offered in the *Hardware channel* selection box using the name of the respective interface module. After saving, a channel that was assigned in a coupling will no longer be available and will therefore no longer be offered for selection.

Profinet	
Property	Value
Hardware Channel	PNIO-Box1[0]
IP-Address	Not assigned
F-System	PNIO-Box1[0]

---

**Note**

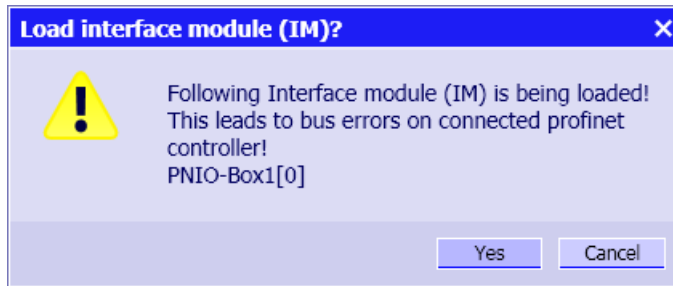
For PROFIBUS and PROFINET together, a maximum of 30 channels can be used, which can be distributed to the simulation units as desired.

---

**Loading the interface module**

If you have configured a coupling or have changed its configuration, the configuration must be loaded into the interface modules before starting the simulation. The interface modules do not behave in general like the real field devices in their communication with the controller (which means with the PROFINET IO masters in the controller) until they have been loaded with the correct PROFINET IO configuration for the PLC.

If you start the simulation for a project that contains a PROFINET IO coupling, a check is automatically performed during startup to see whether the interface modules have already been loaded with the correct configuration data. If not, you can use the dialog below to load the interface modules or to cancel the start of the simulation. The simulation can only be started if the interface modules have been loaded with the correct configuration data.



This dialog also appears when starting the simulation if your PC has been switched off since the last loading.

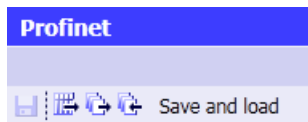
---

**Note**

If you switch off the interface modules, i.e. disconnect the supply voltage, or access the configuration via an application other than SIMIT, SIMIT can no longer ensure that the loaded configuration matches the SIMIT project. This may lead to unpredictable behavior.

---

You can also initiate the loading of the interface modules manually at any time. To do this click the *Save and Load* command in the toolbar of the PROFINET IO coupling editor.



**Note**

While the interface modules are being loaded, their communication with the connected PROFINET IO controllers will be interrupted for a short time at least. Automation systems could therefore go into the "Stop" state.

---

**Configuration in RUN (CiR)**

CiR allows for certain changes to the hardware configuration of a plant to be performed during ongoing operation.

SIMIT does not support this function. After each hardware change, you must reimport the modified system data blocks into SIMIT and then also reload the interface module(s).

**2.3.3.3 Editing the PROFINET IO coupling****Editing signals**

The coupling signals are displayed in the coupling editor in an input section and an output section. As usual in SIMIT, input signals are inputs of the PLC, output signals are outputs of the PLC. Signals are displayed in the coupling editor window with the following properties:

- Symbol name
- Address
- Data type
- IO system number
- Device number
- Module slot number
- Comment
- Scaling
- Lower
- Upper

All properties of the currently selected signal are visible in the property view. Here you can, for example, set the scaling properties such as the type of *Scaling*, *Lower Scale Value* and *Upper Scale Value* for analog signals.

The screenshot shows the SIMATIC Manager interface. At the top, there is a 'Save and load' button. Below it, the 'Inputs' section is expanded, showing a table with columns: Default, Symbol name, Address, Data type, System, Device, Slot, Comment, Scaling, and Location. The 'Increments' signal is selected, showing a scaling type of 'Unipolar'. Below the table, the 'Outputs' section is also visible. A detailed view of the 'Increments' property is shown below, with a 'Scaling' dropdown set to 'Unipolar', and 'Lower scale value' set to 0 and 'Upper scale value' set to 100. Other properties like 'Unit', 'Lower raw value', and 'Upper raw value' are also listed.

Default	Symbol name	Address	Data type	System	Device	Slot	Comment	Scaling	Location
<input type="checkbox"/>	CabinNearDoor	E2.6	BOOL	100	1	2	Kabine in Tuerbereich		
<input type="checkbox"/>		E2.7	BOOL	100	1	2			
▶ 900.0	Increments	EW512	WORD	100	1	4	Schachtzaehler	Unipolar	
0		EW514	WORD	100	1	4		No scaling	
0		EW516	WORD	100	1	4		No scaling	

Property	Value
Scaling	Unipolar
Lower scale value	0
Upper scale value	100
Unit	
Lower raw value	0
Upper raw value	27648

**Note**

Only signals of the WORD type can be scaled.

**Copying and inserting scaling values**

You can copy scaling values from one signal to another or to a selection of signals. Select the cell you want to copy and then choose the *Copy cell* item in the shortcut menu.

This screenshot shows a table with columns 'Scaling', 'Lower', and 'Upper'. The 'Type L thermocouple' row has values '-200' and '900'. A context menu is open over the 'Lower' cell, with 'Copy cell' selected.

Scaling	Lower	Upper
<input type="text"/>	<input type="text"/>	<input type="text"/>
Type L thermocouple	-200	900

You can then select one or more target signals and choose *Paste to cell* in the shortcut menu for the selection.

This screenshot shows the same table as above. A context menu is open over the 'Upper' cell, with 'Paste to cell' selected.

Scaling	Lower	Upper
<input type="text"/>	<input type="text"/>	<input type="text"/>
Type L thermocouple	-200	900

The scaling type and the low and high value can be transferred in this way. This information has to be transferred column by column, however. In addition, you can only paste to cells that are enabled for editing. If the lower and upper values are not editable because no scaling has been set, for example, you need to set the scaling type first or transfer it from an existing signal. Only then can you transfer the associated values.



## Providing default signal values

The first column *Default* is used to provide default values for inputs, for example, to set binary inputs in the controller.

▼ Inputs <span>Reset Filter</span>							
Default	Symbol Name	Address ▲	Data Type ▼	System ▲	Device ▲	Slot ▲	Comment
<input checked="" type="checkbox"/>	DoorClosed	I34.0	BOOL	100	3	2	Sensor Tuer zu
<input type="checkbox"/>	OpenDoorManually	I34.1	BOOL	100	3	2	Tuer Hand auf
<input type="checkbox"/>	CloseDoorManually	I34.2	BOOL	100	3	2	Tuer Hand zu
<input type="checkbox"/>		I34.3	BOOL	100	3	2	
<input type="checkbox"/>	CabinFlush	I34.4	BOOL	100	3	2	Kabine buendig
<input type="checkbox"/>		I34.5	BOOL	100	3	2	
<input type="checkbox"/>	CabinNearDoor	I34.6	BOOL	100	3	2	Kabine in Tuerbereich
<input type="checkbox"/>		I34.7	BOOL	100	3	2	
▶ 900	<b>Increments</b>	<b>IW40</b>	<b>WORD</b>	<b>100</b>	<b>3</b>	<b>4</b>	<b>Schachtzaehler</b>
0		IW42	WORD	100	3	4	

## Merging and splitting signals

When the system data block file is imported, the signals belonging to the imported PROFINET IO configuration are created automatically in the coupling. This takes into account whether they are binary signals with the data width bit or signals with the data width byte, word (2 bytes), double word (4 bytes), etc.

In certain cases it may be desirable to change the data width, for example, to merge eight successive binary signals into a single byte signal. Just select all signals to be merged and choose *Merge* in the shortcut menu.

▼ Outputs <span>Reset Filter</span>						
Symbol Name	Address ▲	Data Type ▼	System ▲	Device ▲	Slot ▲	
	Q6.3	BOOL	100	2	2	
CabinIndicatorUp	Q34.0	BOOL	100	3	3	
CabinIndicatorDown	Q34.1	BOOL	100	3	3	
FaultIndicator	Q34.2	BOOL	100	3	3	
	Q34.3	BOOL	100	3	3	
	Q34.4	BOOL	100	3	3	
	Q34.5	BOOL	100	3	3	
	Q34.6	BOOL	100	3	3	
	Q34.7	BOOL	100	3	3	
Display	QW40	WORD	100	3	5	

Similarly, signals can be also be split. Just select the signal to be split and choose *Split* in the shortcut menu.

The screenshot shows a table titled 'Outputs' with a 'Reset Filter' button. The table has columns for Symbol Name, Address, Data Type, System, Device, and Slot. A 'Split' button is visible over the 'System' column for the 'Display' signal.

Symbol Name	Address	Data Type	System	Device	Slot
	Q6.3	BOOL	100	2	2
	QB34	BYTE	100	3	3
Display	QW40	WORD	100	3	5
	QW42	WORD	100	3	5

The table below shows how signals can be converted. To merge signals, read the table from left to right; to split signals, read the table from right to left.

Number of signals	Data type	Data width		Number of signals	Data type	Data width
8	BOOL	1 bit	↔	1	BYTE	1 byte
2	BYTE	1 byte	↔	1	WORD (INT)	2 bytes
2	WORD (INT)	2 bytes	↔	1	DWORD (DINT, REAL)	4 bytes

**Note**

During a conversion, possibly existing symbol names of the signals are lost.

Signals can only be converted if they have been created in the PROFINET IO coupling by importing the system blocks. Address ranges cannot be created or removed by changing the data width of signals, as the entire available address range is determined by the PROFINET IO configuration in SIMATIC Manager.

**Note**

Signals that are not on the same module as well as fail-safe signals cannot be merged.

## Fail-safe signals

When a fail-safe configuration is imported, the associated quality signals are automatically created for all binary fail-safe signals and listed accordingly in the properties of the fail-safe signal.

**Profinet**

Save and load

▼ Inputs Reset filter

Default	Symbol name	Address	Data type	System	Device	Slot	Comment
<input type="checkbox"/>		I0.0	BOOL	100	1	2	
<input type="checkbox"/>		I0.1	BOOL	100	1	2	
<input type="checkbox"/>		I0.2	BOOL	100	1	2	
<input type="checkbox"/>		I0.3	BOOL	100	1	2	

▼ Outputs Reset filter

Symbol name	Address	Data type	System	Device	Slot	Comment
	Q0.0	BOOL	100	1	2	
	Q0.1	BOOL	100	1	2	
	Q0.2	BOOL	100	1	2	
	Q0.4	BOOL	100	1	2	

**I0.0**

General	Property	Value
Scaling	Symbol name	
Limits	Address	I0.0
Connection	Data type	BOOL
	Comment	
	Quality bit [I1.0]	<input checked="" type="checkbox"/>

All quality signals set to "1" are valid by default. If you wish to deliberately set a signal to "invalid" to test the reaction of your control program to a signal fault, for example, you only need to set the corresponding quality signal to zero. Just uncheck the *Quality bit* property.

### Note

After loading the interface module with a fail-safe configuration, the connected SIMATIC CPU needs to be restarted.

## Properties of the PROFINET IO coupling

After opening the PROFINET IO coupling the left side of the property view will show a tree structure of devices and their modules.

PROFINET		
Property	Value	
PROFINET		
System 100		
[1] im153-4pn	Time slice	2
[2] im154-4pnhf	Mnemonic	I/Q
[3] im151-3pn	F-system	No
[4] im151-3pn-1	IRT system	No
	MAC start address	08:00:06:9D:37:3F

You can define the following properties of the coupling:

- Time slice**  
 Here you can set the cycle in which the coupling exchanges data. Time slice 2 (100ms) is the default.
- Mnemonics**  
 The mnemonics can be toggled between I/O and I/Q syntax for inputs and outputs.

The property *F system* is for informational purposes only. It indicates whether the system is fail-safe.

## Activating and deactivating devices

The right side will show the properties for selected devices. The *Deactivate device* property can be used to deactivate the selected device. The device is then inactive when the simulation is started and will not be simulated.

PROFINET		
Property	Value	
PROFINET		
System 100		
[1] im153-4pn	Device	im151-3pn-1
[2] im154-4pnhf	Device type	ET 200S
[3] im151-3pn	PROFINET ID	002A0301
[4] im151-3pn-1	PROFINET address	4
[5] iexpbmlink	Fail-safe	No
[6] iexpbmlink	About	GSDML-V2.25-Siemens-ET200S-201:
	Deactivate device	<input type="checkbox"/>

You may also set this property of a device after the simulation has been started. A device can thus still be deactivated after the simulation has been started. This allows you to test the PLC reaction when a device fails or comes back online.

## Pulling/Plugging modules

The module properties are shown on the right side of the property view. Modules can be deactivated using the *Pull module* property.

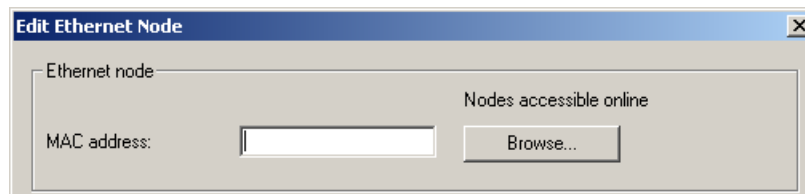
PROFINET		
▼ PROFINET	<b>Property</b>	<b>Value</b>
▼ System 100	Module	SM 323 DI8/DO8xDC24V/0.5A (6ES7
▼ [1] im153-4pn	Slot	1
[0] ! IM153-4 PN V1.0 (6ES7 153-4AA	Input address range	EB0
[1] SM 323 DI8/DO8xDC24V/0.5A (6E	Output address range	AB0
[2] SM 321 DI16xDC24V (6ES7 321-1B	Fail-safe	No
[3] SM 322 DO16xDC24V/0.5A (6ES7	Pull module	<input type="checkbox"/>

In order to test the reaction of the controller to pulled modules, you can also set this property after the simulation has been started.

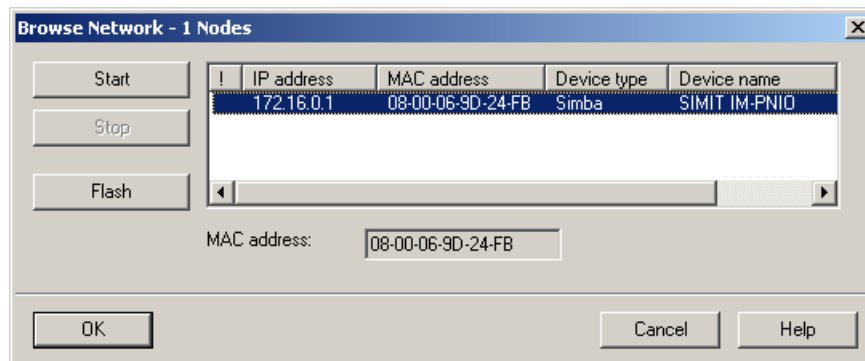
### 2.3.3.4 Configuring the PROFINET IO interface module

To access a PROFINET IO interface module with SIMIT via a network, you first need to assign a suitable IP address to the interface module. To do so, you need a PC with SIMATIC Manager installed and must – at least temporarily – connect the interface module to this PC. To configure the interface module, it does not matter whether you are using a direct connection to the PC or a switch or hub.

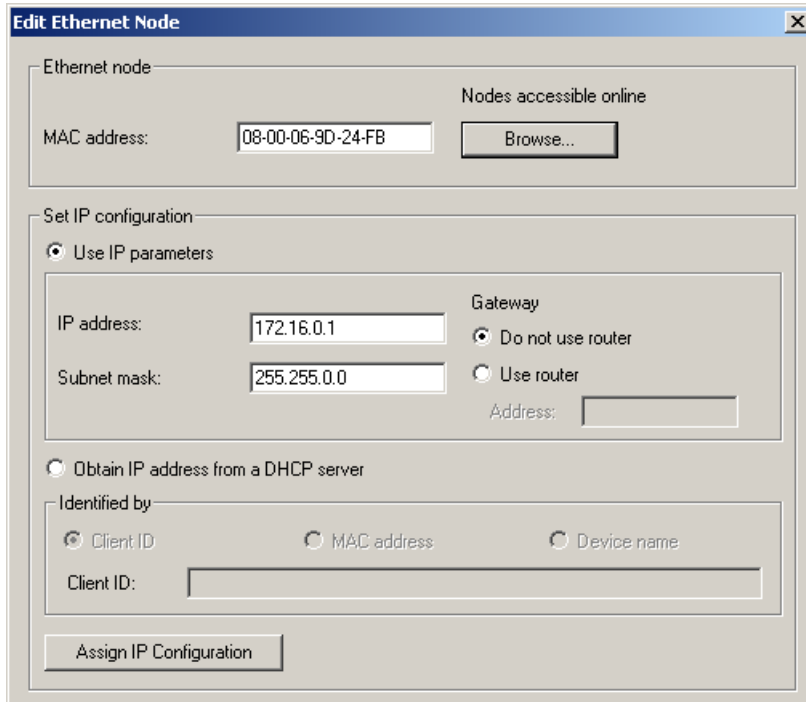
Configure the programming device/PC interface in SIMATIC Manager to use the network adapter to which the interface module is connected. Then choose *Target system | Edit Ethernet node* from the menu in SIMATIC Manager. In this dialog, click *Browse* in the *Nodes accessible online* section.



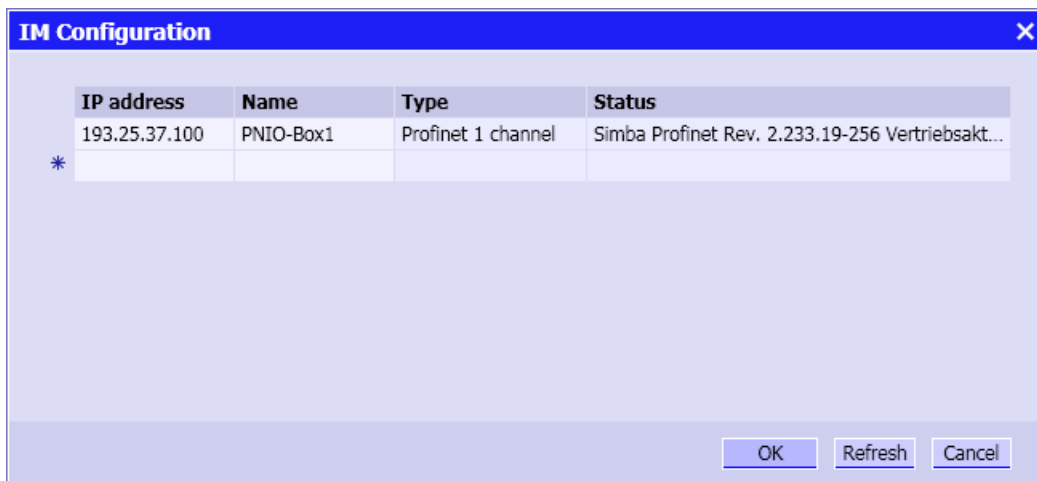
As shown in the figure below, a dialog appears that displays the nodes that were found. Select the entry for the interface module. It displays the device name *SIMIT IM-PNIO*. If you then click the *Flash* button, the LEDs on top of the interface module will flash briefly.



Confirm this dialog by clicking *OK*. In the previous dialog, you now see the interface module data. You can accept the settings for IP address and subnet mask or change them if necessary. In any case, confirm the dialog by clicking *Assign IP configuration*.



Now connect the interface module to your SIMATIC PC again and enter the IP address of the interface module in the SIMATIC *IM configuration* option dialog. Assign any name to the interface module so that it can be distinguished from other interface modules that may exist. When the *Update* button is clicked, the type and status of the interface module are determined. The interface module is now ready for use with SIMATIC.



### 2.3.3.5 Data record communication

Data record communication is possible in the PROFINET IO coupling. This type of communication is used for components of type *ASM475*, *RF170C* and *SIWAREX-U1* or *SIWAREX-U2*.

## 2.3.4 PRODAVE coupling

### 2.3.4.1 Functioning of the PRODAVE coupling

You can use the PRODAVE coupling to write and read inputs and outputs within the process image of a SIMATIC controller with SIMIT.

The PRODAVE coupling allows you to make a connection from SIMIT to a SIMATIC controller via the MPI interface. For Ethernet-capable controllers, the connection can also be made via Ethernet. An Ethernet connection performs somewhat better than a connection via MPI. It is important in all cases to configure the programming device interface in the SIMATIC Manager for the selected type of connection.

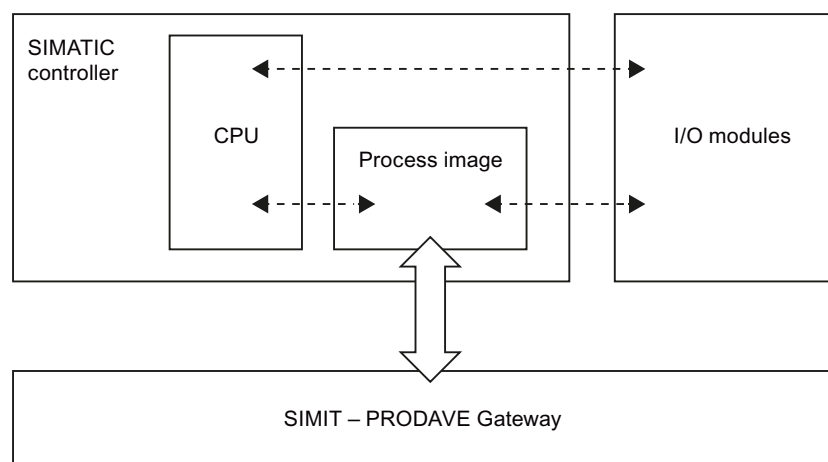
---

#### Note

To use the PRODAVE coupling you also need version 6.2 of the SIMATIC PRODAVE software. This software is not included in the SIMIT product package. You may also need additional hardware, such as MPI adapters or connecting cables. These also are not included in the SIMIT product package.

---

The PRODAVE coupling does not simulate any SIMATIC I/Os. SIMIT directly addresses the process image of a SIMATIC controller via the PRODAVE coupling. The addresses that are actually addressable depend on the size of the CPU process image and on what was defined in your hardware configuration.



---

**Note**

You must not operate any I/O modules on your controller that access the same addresses ranges as the PRODAVE coupling, because this would lead to competing accesses to the process image.

---

**Note**

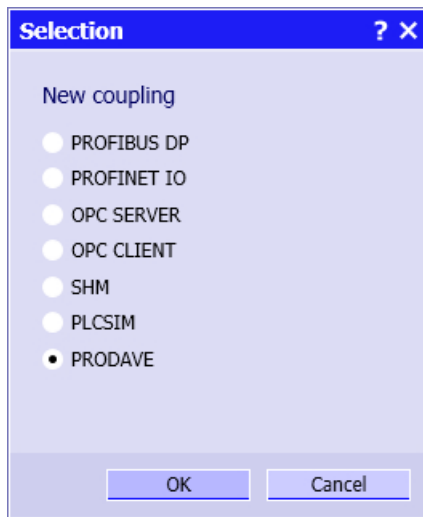
If you have configured I/O modules in your hardware configuration, you can still access their addresses ranges from the PRODAVE coupling, provided that these configured modules are not really plugged in. This will cause access errors in the CPU, which you can catch by creating the corresponding OBs in the control program. Because the PRODAVE coupling only has access to the process image, you cannot access any simulated signals that are addressed in the CPU as I/O signals (*PIW, PQW*). Some SIMATIC controllers, such as CPU 313C, are hard-wired with I/O modules whose address ranges are immutable. These address ranges cannot be accessed via the PRODAVE coupling.

---

### 2.3.4.2 Configuring the PRODAVE coupling

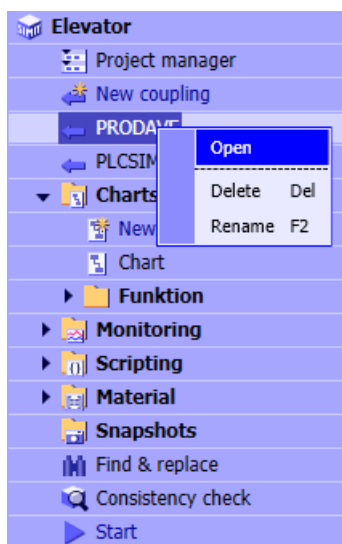
#### Creating a PRODAVE coupling

To add a PRODAVE coupling to your project, create a coupling in the "Coupling" folder of the project tree using the "New Coupling" command. Select *PRODAVE* as the coupling type.



In the SIMIT project you may assign an arbitrary name to your PRODAVE coupling. To accept the default name, press Enter. The coupling editor then opens. By double-clicking the coupling or by using the shortcut menu you can open the editor at any time later.





### Configuring I/O signals in the PRODAVE coupling

The input and output signals of the PRODAVE coupling can either be edited manually in the coupling window, or configured by importing the symbol table from your SIMATIC project.

PRODAVE				
Inputs				
Default	Symbol Name	Address	Data Type	Comment
<input type="checkbox"/>	StaircaseButtonFloor0	I32.0	BOOL	Ruf Treppenhaus EG
<input type="checkbox"/>	StaircaseButtonFloor1	I32.1	BOOL	Ruf Treppenhaus 1.OG
<input type="checkbox"/>	StaircaseButtonFloor2	I32.2	BOOL	Ruf Treppenhaus 2.OG
<input type="checkbox"/>	StaircaseButtonFloor3	I32.3	BOOL	Ruf Treppenhaus 3.OG
<input type="checkbox"/>	StaircaseButtonFloor4	I32.4	BOOL	Ruf Treppenhaus 4.OG
<input type="checkbox"/>	Mainswitch	I32.5	BOOL	Betrieb Ein
<input type="checkbox"/>	Reset	I32.7	BOOL	Reset-Taster
Outputs				
	Symbol Name	Address	Data Type	Comment
	StaircaseIndicatorFloor0	Q32.0	BOOL	Rueckmeldung Treppenhaus Ruf EG
	StaircaseIndicatorFloor1	Q32.1	BOOL	Rueckmeldung Treppenhaus Ruf 1.OG
	StaircaseIndicatorFloor2	Q32.2	BOOL	Rueckmeldung Treppenhaus Ruf 2.OG
	StaircaseIndicatorFloor3	Q32.3	BOOL	Rueckmeldung Treppenhaus Ruf 3.OG
	StaircaseIndicatorFloor4	Q32.4	BOOL	Rueckmeldung Treppenhaus Ruf 4.OG
	InOperation	Q32.5	BOOL	Rueckmeldung Betrieb

## Properties of the PRODAVE coupling

After the coupling has been opened, the coupling editor appears in the work area. The following properties can be defined in the property view:

- **Time slice**

Here you can set the cycle in which the coupling exchanges data. The assignment of absolute cycle times to the eight available time slices is done for the entire project in the project manager. Time slice 2 is the default, corresponding to a cycle of 100ms.

---

### Note

The time slice with the lowest cycle time always has the highest priority, regardless of the numbering.

---

- **Mnemonics**

Here, you can select whether the international (*I/Q*) or German (*E/A*) syntax should be used for the designation of inputs and outputs.

- **CPU slot**

Enter the slot number of your SIMATIC CPU here.

- **Access mode**

Here, you select whether to you want to use MPI (cable or adapter) or IP (Ethernet) access.

- **MPI address / IP address**

Here, you enter the of MPI number of the SIMATIC CPU or its IP address.

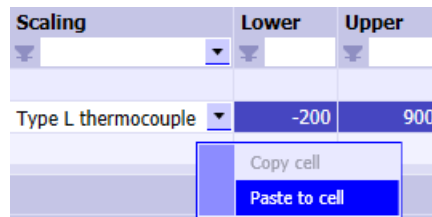
PRODAVE	
Property	Value
Time slice	2
Mnemonic	I/Q
CPU slot	2
Access mode	MPI
MPI address	2

## Copying and inserting scaling values

You can copy scaling values from one signal to another or to a selection of signals. Select the cell you want to copy and then choose the *Copy cell* item in the shortcut menu.

Scaling	Lower	Upper
▼	▼	▼
Type L thermocouple	-200	900
	Copy cell	
	Paste to cell	

You can then select one or more target signals and choose *Paste to cell* in the shortcut menu for the selection.



The scaling type and the low and high value can be transferred in this way. This information has to be transferred column by column, however. In addition, you can only paste to cells that are enabled for editing. If the lower and upper values therefore cannot be edited because no scaling has been set, for example, you need to set the scaling type first or transfer it from an existing signal. Only then can you transfer the associated values.

### Importing the signal properties

Import the signal properties by clicking the "📄" symbol in the coupling editor. The "Import of signal properties" dialog box opens. You can make the import settings in this dialog box.

You can find additional information in the section: Import of signal properties (Page 66).

### Exporting the signal properties

Export the signal properties by clicking the "📄" symbol in the coupling editor. The "Export of signal properties" dialog box opens. You can make the export settings in this dialog box.

You can find additional information on this in the section: Export of signal properties (Page 69).

## 2.3.5 PLCSIM coupling

### 2.3.5.1 How the PLCSIM coupling works

---

#### Note

The PLCSIM coupling is only included in the variants SIMIT PROFESSIONAL and SIMIT ULTIMATE.

---

You can link SIMIT to PLCSIM to write and read inputs and outputs of PLCSIM from SIMIT.

---

#### Note

To use the PLCSIM coupling, version 5.4 SP5 of the PLCSIM software is required in addition. This software is not included in the SIMIT product package.

---

PLCSIM can be launched from SIMATIC Manager along with a STEP 7 program and can be used independently of SIMIT. PLCSIM will then simulate a SIMATIC controller. In order to simulate the controller behavior as realistically as possible, we recommend that you always load the STEP 7 program along with the system data blocks (SDBs) into PLCSIM.

**Note**

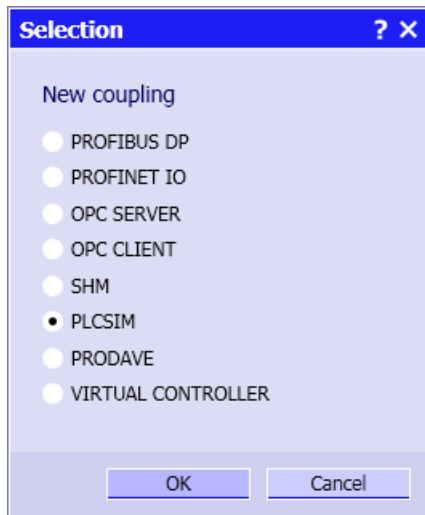
The hardware configuration is also determined through the system data. In the SIMIT coupling with PLCSIM, data exchange is only possible for I/O signals that are defined by the configured hardware.

The PLCSIM coupling allows SIMIT to communicate with PLCSIM using a software interface provided by PLCSIM (Prosim). Both SIMIT and PLCSIM need to be installed on the same computer in order to use the PLCSIM coupling. PLCSIM must be running before a SIMIT simulation containing a PLCSIM coupling is launched. While the simulation is running, PLCSIM must not be closed as this would terminate the connection, and it may not be possible to reestablish the connection.

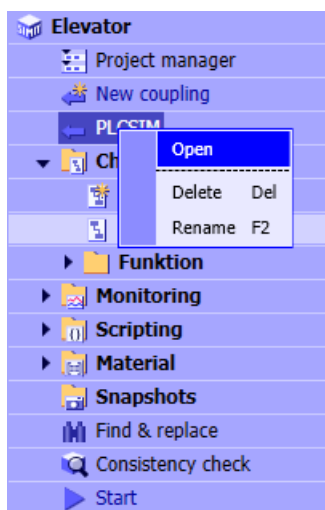
### 2.3.5.2 Configuring the PLCSIM coupling

#### Creating a PLCSIM coupling

To add a PLCSIM coupling to your project, create a coupling using the *New coupling* command in the "Couplings" folder of the project tree or select "Couplings > Add new coupling" in the portal view. Then select the *PLCSIM* coupling type.



In the SIMIT project you may assign an arbitrary name to your PLCSIM coupling. To accept the default name, press Enter. The coupling editor then opens. By double-clicking the coupling or by using the shortcut menu you can open the editor at any time later.



### Configuring I/O signals in the PLCSIM coupling

The input and output signals of the coupling can either be entered manually in the PLCSIM coupling editor or configured by importing the symbol table from your SIMATIC project.

PLCSIM				
Inputs				
Default	Symbol name	Address	Data type	Comment
<input type="checkbox"/>	StaircaseButtonFloor0	I32.0	BOOL	Ruf Treppenhaus EG
<input type="checkbox"/>	StaircaseButtonFloor1	I32.1	BOOL	Ruf Treppenhaus 1.OG
<input type="checkbox"/>	StaircaseButtonFloor2	I32.2	BOOL	Ruf Treppenhaus 2.OG
<input type="checkbox"/>	StaircaseButtonFloor3	I32.3	BOOL	Ruf Treppenhaus 3.OG
<input type="checkbox"/>	StaircaseButtonFloor4	I32.4	BOOL	Ruf Treppenhaus 4.OG
<input type="checkbox"/>	Mainswitch	I32.5	BOOL	Betrieb Ein
<input type="checkbox"/>	Reset	I32.7	BOOL	Reset-Taster
Outputs				
	Symbol name	Address	Data type	Comment
	StaircaseIndicatorFloor0	Q32.0	BOOL	Rueckmeldung Treppenhaus Ruf EG
	StaircaseIndicatorFloor1	Q32.1	BOOL	Rueckmeldung Treppenhaus Ruf 1.OG
	StaircaseIndicatorFloor2	Q32.2	BOOL	Rueckmeldung Treppenhaus Ruf 2.OG
	StaircaseIndicatorFloor3	Q32.3	BOOL	Rueckmeldung Treppenhaus Ruf 3.OG
	StaircaseIndicatorFloor4	Q32.4	BOOL	Rueckmeldung Treppenhaus Ruf 4.OG
	InOperation	Q32.5	BOOL	Rueckmeldung Betrieb
	Direction	Q32.6	BOOL	Antrieb Richtung 1= Auf, 0 = Ab

## Properties of the PLCSIM coupling

When the coupling is opened, the editor appears. The following properties can be defined in the property view:

- **Time slice**

Here you can set the cycle in which the coupling exchanges data. The assignment of absolute cycle times to the eight available time slices is done for the entire project in the project manager. Time slice 2 is the default, corresponding to a cycle of 100ms.

---

### Note

The time slice with the lowest cycle time always has the highest priority, regardless of the numbering.

---

- **PLCSIM number**

You may create a maximum of eight PLCSIM couplings in a project to connect to corresponding instances of PLCSIM. Here you specify the number of the PLCSIM instance with which the coupling should communicate. You see the corresponding number in the title bar of the PLCSIM window, for example, *S7-PLCSIM2* for the second PLCSIM instance. If you set the PLCSIM number to *Not assigned*, no connection to PLCSIM is established when starting the simulation and no data exchange will occur.

- **Mnemonics**

Here, you can select whether the international (*I/Q*) or German (*E/A*) syntax should be used for the designation of inputs and outputs.

PLCSIM	
Property	Value
Time slice	2
PLCSIM number	1
Mnemonic	I/Q

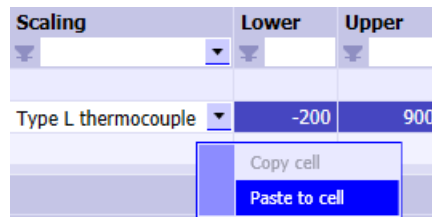
## Copying and inserting scaling values

You can copy scaling values from one signal to another or to a selection of signals. Select the cell you want to copy and then choose the *Copy cell* item in the shortcut menu.

Scaling	Lower	Upper
Type L thermocouple	-200	900

Copy cell  
 Paste to cell

You can then select one or more target signals and choose *Paste to cell* in the shortcut menu for the selection.



The scaling type and the low and high value can be transferred in this way. This information has to be transferred column by column, however. In addition, you can only paste to cells that are enabled for editing. If the lower and upper values therefore cannot be edited because no scaling has been set, for example, you need to set the scaling type first or transfer it from an existing signal. Only then can you transfer the associated values.

### Importing the signal properties

Import the signal properties by clicking the "📄" symbol in the coupling editor. The "Import of signal properties" dialog box opens. You can make the import settings in this dialog box.

You can find additional information in the section: Import of signal properties (Page 66).

### Exporting the signal properties

Export the signal properties by clicking the "📄" symbol in the coupling editor. The "Export of signal properties" dialog box opens. You can make the export settings in this dialog box.

You can find additional information in the section: Export of signal properties (Page 69).

## 2.3.6 "Virtual controller" coupling

### 2.3.6.1 How the Virtual Controller coupling works

---

#### Note

The SIMIT Virtual Controller (VC) is a separate product and therefore not a SIMIT component. Version 3.0 of SIMIT VC must be installed for the coupling to work.

---

SIMIT VC simulates the behavior of one or more SIMATIC PLCs. SIMIT VC has its own user interface for configuration and engineering. You can find more information on this topic in the SIMIT VC manuals:

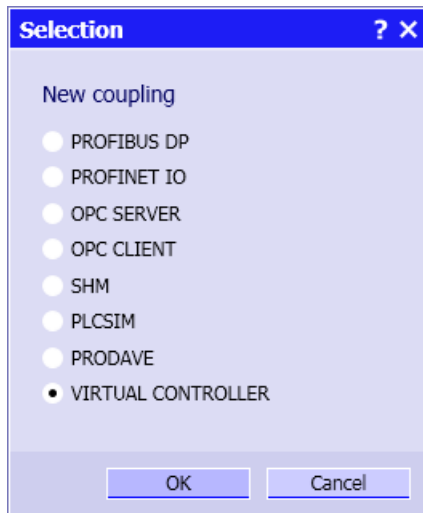
- SIMIT Virtual Controller (VC) - User Manual
- SIMIT Virtual Controller (VC) - Reference Manual

A Virtual Controller coupling must be created in SIMIT in order to exchanges signals with SIMIT VC. The range of signals is then automatically copied from the Emulation Manager of SIMIT VC.

### 2.3.6.2 Configuring the Virtual Controller coupling

#### Creating a Virtual Controller coupling

To add a Virtual Controller coupling to your project, create a coupling using the *New coupling* command in the "Couplings" folder of the project tree or select "Couplings > Add new coupling" in the portal view. Then select *VIRTUAL CONTROLLER* as the coupling type.



The Virtual Controller coupling automatically attempts to find a VC project on the local machine that bears the same name as the current SIMIT project. If such a VC project exists in the VC coupling, a coupling is created for each controller configured there and the corresponding range of signals is imported.

If you subsequently make changes in the Emulation Manager, you can apply these changes by selecting "Update" from the shortcut menu of the coupling folder.

#### Configuring I/O signals in the Virtual Controller coupling

Configuration of the Virtual Controller is performed in the Emulation Manager. Most signal properties are therefore cannot be changed in the coupling. Only a limitation of physical signals and an implicit interconnection can be entered in the properties view.

If the configuration is updated by the Emulation Manger, these entries are retained as far as possible.



## Properties of the Virtual Controller coupling

When the coupling is opened, the editor appears. The following properties can be defined in the property view:

- **Time slice**

Here you can set the cycle in which the coupling exchanges data. The assignment of absolute cycle times to the 8 available time slices is done for the entire project in the project manager. Time slice 2 is the default, corresponding to a cycle of 100ms.

The time slice can only be set for all Virtual Controller couplings together from the shortcut menu of the folder coupling.


---

**Note**

The time slice with the lowest cycle time always has the highest priority, regardless of the numbering.

---


## Importing the signal properties

Import the signal properties by clicking the "" symbol in the coupling editor.

The "Import of signal properties" dialog box opens. You can make the import settings in this dialog box.

You can find additional information in the section: Import of signal properties (Page 66).

## Exporting the signal properties

Export the signal properties by clicking the "" symbol in the coupling editor.

The "Export of signal properties" dialog box opens. You can make the export settings in this dialog box.

You can find additional information on this in the section: Export of signal properties (Page 69).

## 2.4 OPC coupling

### 2.4.1 Functioning of the OPC coupling

---

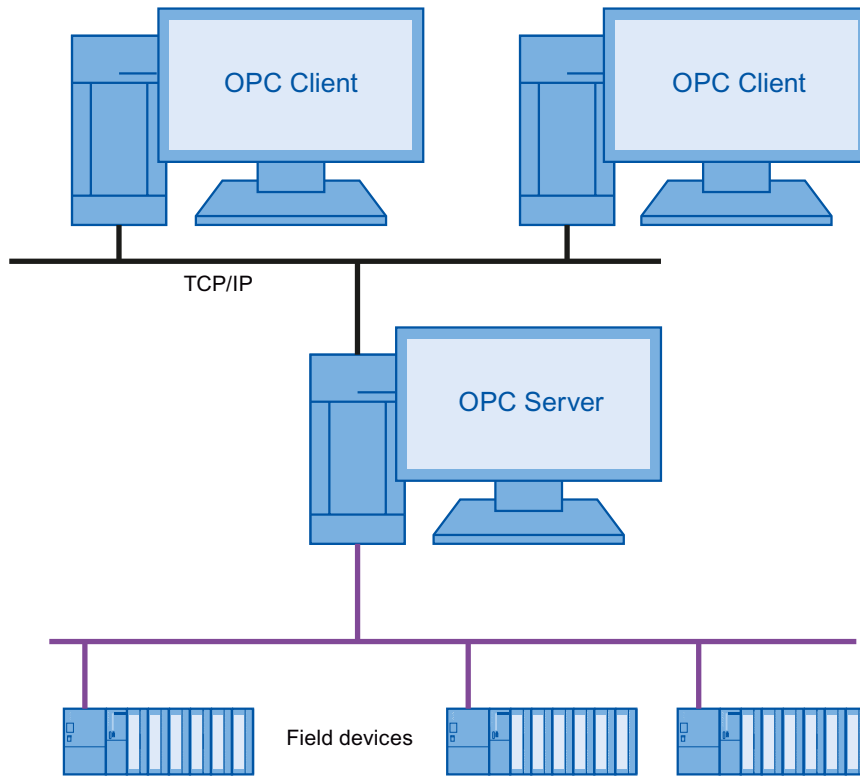
**Note**

The OPC coupling is only included in the variants SIMIT PROFESSIONAL and SIMIT ULTIMATE.

---

OPC is a well-established standard for communication in automation engineering and is maintained and supported by the OPC Foundation ([www.opcfoundation.org](http://www.opcfoundation.org)).

In the OPC Classic Architecture an OPC DA Server is connected to a subordinate hardware layer whose signals are made available via the local network to one or more OPC DA Clients.



The OPC Server and OPC Client may be running locally on the same computer or on different computers. The OPC protocol can also be used for completely device-independent data exchange between programs that support OPC.

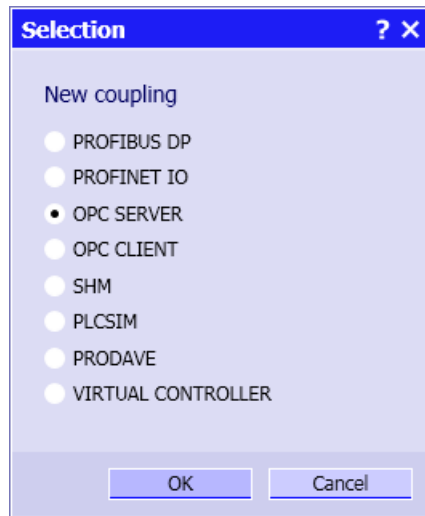
With the OPC coupling, SIMIT can be used as an OPC Server and as an OPC Client. In both cases, SIMIT supports the OPC standard version 3.0 in "Data Access" mode (OPC DA 3.0).

Within a SIMIT project, only one single OPC Server coupling can be used. The OPC Client coupling can be used in several instances, however. This allows for you to connect a single SIMIT simulation to several OPC Servers. You can also use one single OPC Server coupling and several OPC Client couplings simultaneously within a SIMIT project.

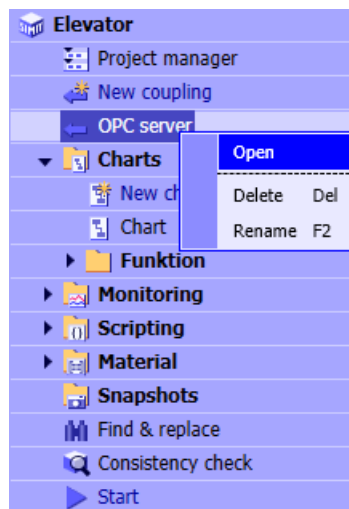
## 2.4.2 The OPC Server coupling

### 2.4.2.1 Adding an OPC Server coupling

To add an OPC server coupling to your project, create a coupling using the *New coupling* command in the "Couplings" folder of the project tree or select "Couplings > Add new coupling" in the portal view. Then select the *OPC server* coupling type.



In the SIMIT project you may assign an arbitrary name to your OPC Server coupling. To accept the default name, press the enter key. The coupling editor then opens. By double-clicking the coupling in the project tree or by using the shortcut menu you can open the editor at any time later.



### 2.4.2.2 Configuring I/O signals in the OPC server coupling

#### Overview

The input and output signals of the OPC server coupling can either be edited manually in the editor or they can be configured by importing a signal table.

OPC Server			
Inputs			
Default	Name	Type	Comment
<input type="checkbox"/>	CabinButtonFloor0	binary	Anwahl Innenkabine EG
<input type="checkbox"/>	CabinButtonFloor1	binary	Anwahl Innenkabine 1.OG
<input type="checkbox"/>	CabinButtonFloor2	binary	Anwahl Innenkabine 2.OG
<input type="checkbox"/>	CabinButtonFloor3	binary	Anwahl Innenkabine 3.OG
<input type="checkbox"/>	CabinButtonFloor4	binary	Anwahl Innenkabine 4.OG
<input type="checkbox"/>	CabinFlush	binary	Kabine buendig
<input type="checkbox"/>	CabinNearDoor	binary	Kabine in Tuerbereich
Outputs			
Name	Type	Comment	
CabinIndicatorDown	binary	Innenanzeige Ab	
CabinIndicatorFloor0	binary	Rueckmeldung Anwahl EG	
CabinIndicatorFloor1	binary	Rueckmeldung Anwahl 1.OG	
CabinIndicatorFloor2	binary	Rueckmeldung Anwahl 2.OG	
CabinIndicatorFloor3	binary	Rueckmeldung Anwahl 3.OG	
CabinIndicatorFloor4	binary	Rueckmeldung Anwahl 4.OG	
CabinIndicatorUp	binary	Innenanzeige Auf	

#### Importing the signal properties

Import the signal properties by clicking the "📥" symbol in the coupling editor.

The "Import of signal properties" dialog box opens. You can make the import settings in this dialog box.

You can find additional information in the section: Import of signal properties (Page 66).

#### Exporting the signal properties

Export the signal properties by clicking the "📤" symbol in the coupling editor.

The "Export of signal properties" dialog box opens. You can make the export settings in this dialog box.

You can find additional information in the section: Export of signal properties (Page 69).

### 2.4.2.3 Properties of the OPC server coupling

When the coupling is opened, the editor appears. The following properties can be defined in the property view:

- **Time slice**

Here you can set the cycle in which the coupling exchanges data. The assignment of absolute cycle times to the eight available time slices is done for the entire project in the project manager. Time slice 2 is the default, corresponding to a cycle of 100ms.

---

**Note**

The time slice with the lowest cycle time always has the highest priority, regardless of the numbering.

The OPC server updates its data every 100ms. Cycle times below this value should therefore not be set.

---

- **Server name**

Display of the SIMIT OPC server name is for your information. The server name is always "SIMIT OPC DA Server" and cannot be changed. Server name or ProgID are required to configure an OPC client for access to this OPC server.

- **ProgID**

Display of the SIMIT OPC server ProgID is for your information only. The ProgID is always "Simit.OpcDaServer.8" and cannot be changed. ProgID or server name are required to configure an OPC client to access this OPC server.

- **Use 64-bit integers**

Integer values in SIMIT have a data width of 64 bits (8 bytes). They are therefore also stored in the OPC server coupling in this format.

If an OPC client that is connected to this OPC server cannot handle integer values with 8 byte data width, deselect this option. SIMIT then transmits all integer values with a data width of only 32 bits (4 bytes). Note that this may lead to loss of data if a number cannot be represented with 32 bits.

OPC server	
Property	Value
Time slice	2
Server name	SIMIT OPC DA Server
ProgID	Simit.OpcDaServer.8
Use 64-bit integer	<input checked="" type="checkbox"/>

### 2.4.2.4 Special features of the OPC server coupling

#### Used data types

In the OPC server, SIMIT creates signals with data types as listed in the table below.

Table 2-9 Assignment of data types

SIMIT	OPC
Binary	BOOL

Integer	I8, if the option "use 64-bit integers" is checked, otherwise I4.
Analog	R8

As a consequence, the OPC server will only provide data types BOOL, R8 and I8 or I4.

### "Quality" of the signals

When the simulation is running, SIMIT communicates all OPC signals with quality "Good, non-specific", which means with the value 0xC0. This value cannot be changed.

If the simulation is terminated, the OPC server remains active as long as at least one OPC Client is still connected. In this case the signal quality will be set to "bad, out of service", which means to the value 0x1C. The quality of a signal can be read by any connected OPC Client, but is not displayed in the OPC server coupling.

### OPC Server status

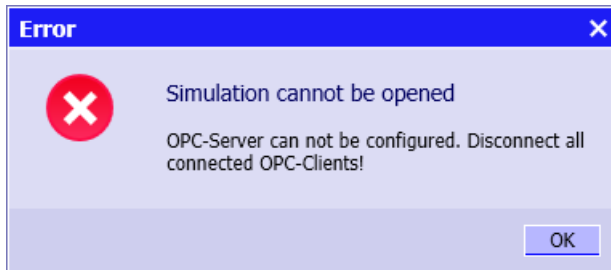
A connected OPC Client can query the server status. When the simulation is running, the server of the OPC coupling will return the status value 1 (Running); when the simulation is not running, the value is 4 (Suspended).

### Changing the configuration of the OPC server

When the simulation is started, the SIMIT OPC server is automatically configured with the signals that are configured in the coupling. From that point forward, OPC Clients can connect to the server and access the signals.

After the simulation is closed the OPC server remains active as long as a minimum of one OPC Client is still connected to it. The server status will change to OPC\_STATUS\_SUSPENDED, the signal values are no longer updated, and the quality of the signals is set to "bad, out of service", which means to the value 0x1C.

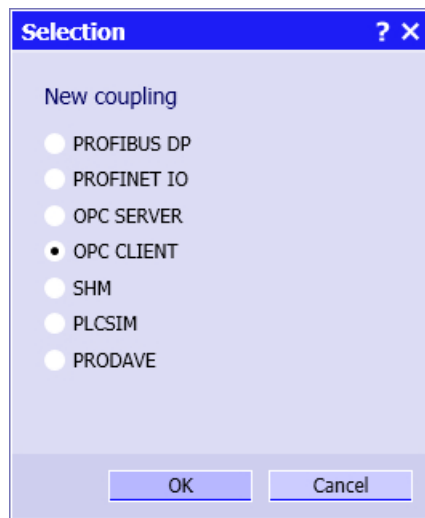
When the simulation is started again, the OPC server is automatically checked for modifications in its configuration. If the configuration has changed and there are still clients connected, the new configuration cannot be transferred. To avoid an inconsistent simulation state, an error message is issued in this case and the simulation is not started.



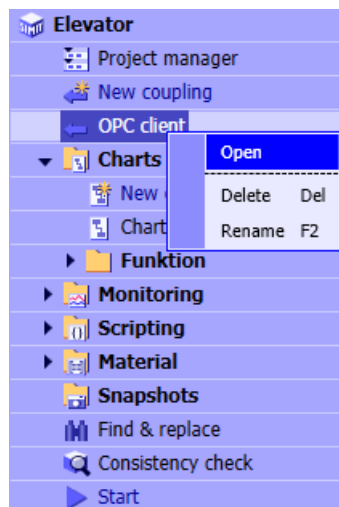
## 2.4.3 The OPC Client coupling

### 2.4.3.1 Adding an OPC Client coupling

To add an OPC client coupling to your project, create a coupling using the *New coupling* command in the "Couplings" folder of the project tree or select "Couplings > Add new coupling" in the portal view. Then select the *OPC client* coupling type.



You may assign an arbitrary name to the OPC client coupling in your SIMIT project. To accept the default name, press the enter key. The coupling editor then opens. By double-clicking the coupling in the project tree or by using the shortcut menu you can open the editor at any time later.



### 2.4.3.2 Configuring I/O signals in the OPC client coupling

#### Overview

The input and output signals of the OPC client coupling can be edited manually in the editor, configured by importing a signal table or by querying an accessible OPC server.

OPC Client				
Browse				
▼ Inputs <span>Reset Filter</span>				
Default	Name ▲	Type	Multiplier	Comment
<input type="checkbox"/>	CabinButtonFloor0	binary	1	Anwahl Innenkabine EG
<input type="checkbox"/>	CabinButtonFloor1	binary	1	Anwahl Innenkabine 1.OG
<input type="checkbox"/>	CabinButtonFloor2	binary	1	Anwahl Innenkabine 2.OG
<input type="checkbox"/>	CabinButtonFloor3	binary	1	Anwahl Innenkabine 3.OG
<input type="checkbox"/>	CabinButtonFloor4	binary	1	Anwahl Innenkabine 4.OG
<input type="checkbox"/>	CabinFlush	binary	1	Kabine buendig
<input type="checkbox"/>	CabinNearDoor	binary	1	Kabine in Tuerbereich
▼ Outputs <span>Reset Filter</span>				
Name ▲	Type	Multiplier	Comment	
CabinIndicatorDown	binary	1	Innenanzeige Ab	
CabinIndicatorFloor0	binary	1	Rueckmeldung Anwahl EG	
CabinIndicatorFloor1	binary	1	Rueckmeldung Anwahl 1.OG	
CabinIndicatorFloor2	binary	1	Rueckmeldung Anwahl 2.OG	
CabinIndicatorFloor3	binary	1	Rueckmeldung Anwahl 3.OG	
CabinIndicatorFloor4	binary	1	Rueckmeldung Anwahl 4.OG	
CabinIndicatorUp	binary	1	Innenanzeige Auf	

#### Importing the signal properties

Import the signal properties by clicking the "📄" symbol in the coupling editor.

The "Import of signal properties" dialog box opens. You can make the import settings in this dialog box.

You can find additional information in the section: Import of signal properties (Page 66).

#### Exporting the signal properties

Export the signal properties by clicking the "📄" symbol in the coupling editor.

The "Export of signal properties" dialog box opens. You can make the export settings in this dialog box.

You can find additional information in the section: Export of signal properties (Page 69).



## Configuration by querying the OPC server

SIMIT provides an easy way to transfer all signals from an OPC server to your OPC client coupling.

In the OPC client coupling properties, first select the OPC server from which signals are to be transferred. Then click the *Browse* button. All signals from the OPC server that can be mapped to a SIMIT data type are then transferred to the coupling.

Table 2-10 Assignment of data types

OPC data type	SIMIT data type
BOOL, UI1	binary
I1, I2, I4, I8, UI2, UI4, UI8	integer
R4, R8	analog

In case not all signals are required in the simulation, you can manually delete any signals from the coupling signal table later.

Signals that are provided by an OPC server can be readable, writable, or both. Depending on this, they are treated as inputs or outputs in SIMIT.

Table 2-11 Assignment of OPC access types

OPC	SIMIT
readable	Output
writable	Input
readWriteable	Input

### 2.4.3.3 Properties of the OPC client coupling

When the coupling is opened, the editor appears. The property window is used define the *time slice*, *host name*, *ProgID* and *status display*.

OPC client	
Property	Value
Time slice	2
Host name	localhost
ProgID	Not assigned
Status display	is_active

The meaning of these properties is as follows:

- **Time slice**

Here you can set the cycle in which the coupling exchanges data. The assignment of absolute cycle times to the eight available time slices is done for the entire project in the project manager. Time slice 2 is the default, corresponding to a cycle of 100ms.

---

**Note**

The time slice with the lowest cycle time always has the highest priority, regardless of the numbering.

---

- **Host name**

Here you can specify the computer on which the OPC server is running with which this OPC client should communicate. You can provide either the computer name or its IP address. The default value is the local computer (localhost).

- **ProgID**

Enter the ProgID of the OPC server with which the OPC client should connect on the computer that is specified by the host name.

If you use a local connection, which means the host name is "localhost", you can select from a list of all ProgIDs of the OPC servers that can be accessed locally. Then select the OPC server with which this coupling should communicate.

If you leave the ProgID *not assigned* there will be no connection established to any OPC server when starting the simulation. In this case, the coupling signals are not updated cyclically.

- **Status display**

You can query the status of an accessed OPC server when the simulation is running. Each OPC client coupling provides a dedicated integer output signal for that purpose whose name you can specify here. The default signal name is *is\_active*. You may use this signal like any other output signal of any coupling in your SIMIT project to evaluate the server status.

When the simulation is running, the property view of the OPC client coupling shows the current status of the addressed OPC server. The table below shows the meaning of the possible status values.

OPC client	
Property	Value
Time slice	2
Host name	localhost
ProgID	Not assigned
Status display [is_active]	

Tip: It depends on the implementation of the server which of these status values are provided by an OPC server.

Table 2-12 Meaning of an OPC server status values

Status value <sup>1</sup>	Meaning <sup>2</sup>
0	No connection to the OPC server
1	The server is running normally. This is the usual state for a server.
2	A vendor specific fatal error has occurred within the server. The server is no longer functioning. The recovery procedure from this situation is vendor specific. An error code of E_FAIL should generally be returned from any other server method.

Status value <sup>1</sup>	Meaning <sup>2</sup>
3	The server is running but has no configuration information loaded and thus cannot function normally. Note this state implies that the server needs configuration information in order to function. Servers which do not require configuration information should not return this state.
4	The server has been temporarily suspended via some vendor specific method and is not getting or sending data. Note that Quality will be returned as OPC_QUALITY_OUT_OF_SERVICE.
5	The server is in Test Mode. The outputs are disconnected from the real hardware but the server will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. Quality will generally be returned normally.
6	The server is running properly but is having difficulty accessing data from its data sources. This may be due to communication problems, or some other problem preventing the underlying device, control system, etc. from returning valid data. It may be complete failure, meaning that no data is available, or a partial failure, meaning that some data is still available. It is expected that items affected by the fault will individually return with a BAD quality indication for the items.

<sup>1</sup> Information concerning status values 1 through 6 is taken from the specification "Data Access Custom Interface Standard, Version 3.00".

<sup>2</sup> The description of status values 1 through 6 is taken from the OPC specification

#### 2.4.3.4 Signal properties within the coupling

##### Cycle multiplier

Like any SIMIT coupling, the OPC client coupling is executed cyclically. Set the cycle time using the time slice in the properties of the coupling.

Because by definition the OPC coupling uses software layers (COM, DCOM) which may influence the performance of the signal exchange, you may use a *multiplier* for each signal to adapt the simulation to existing options. A multiplier of  $n$  causes the corresponding signal to no longer be updated in every cycle, but only in every  $n$ th cycle. In order to reduce the amount of bandwidth required for communication between OPC server and OPC client you may use a small time slice multiplier value for fast-changing signals and a large value for slowly changing signals.

The default setting of the multiplier is 1, which means each signal is updated in each cycle.

##### Mapping of data types

SIMIT only has binary, integer and analog signals, with integer and analog signals having a data width of 8 bytes.

An OPC Server may provide signals in various other data types. The table below shows how OPC data types are assigned to SIMIT data types.

Table 2-13 Assignment of data types

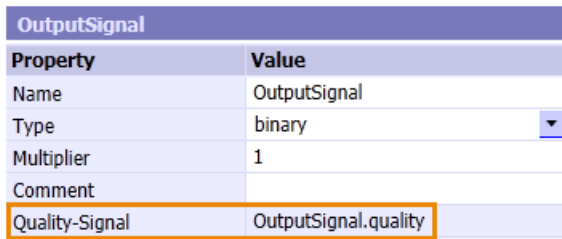
OPC	SIMIT
BOOL, UI1	binary

I1, I2, I4, I8, UI1, UI2, UI4, UI8	integer
R4, R8	analog

For output signals within the coupling there is no loss of data due to conversion, because both integer and analog signals in SIMIT have the maximum data width of 8 bytes.

### "Quality" of the signals

In addition to the wanted signal, an OPC server will communicate information about its quality. You can access this information within the simulation, because SIMIT automatically places a quality signal as integer signal for each output signal in the OPC client coupling. The name of this signal consists of the name of the output signal and the suffix *.quality*.



OutputSignal	
Property	Value
Name	OutputSignal
Type	binary
Multiplier	1
Comment	
Quality-Signal	OutputSignal.quality

Within the property view of a signal in the OPC client coupling you can observe the value of its corresponding quality signal when the simulation is running. You may also use this quality signal like any other (integer) signal in your simulation.

Input signals, which means signals that are communicated from the SIMIT OPC client to the OPC server, are always assigned the quality "good, non-specific", which means the value 0xC0 (192 decimal). This value cannot be changed.

---

#### Note

The quality signals are only updated when there is a connection to the OPC server. Therefore, check the status display in the properties of the OPC client coupling.

---

**OPC Client**

Browse

**Inputs** Reset Filter

**Outputs** Reset Filter

	Name	Type	Multiplikator	Comment
	4667 SimulatedData.Random	integer	1	
	SimulatedData.Signal	binary	1	
	0.58778525 SimulatedData.Sine	analog	1	

**SimulatedData.Sine**

Property	Value
Name	SimulatedData.Sine
Type	analog
Multiplier	1
Comment	
Quality-Signal [SimulatedData.Sine.quality]	192

### 2.4.3.5 Special features of the OPC client coupling

#### Read back of input signals

Signals that are transferred from SIMIT to an OPC server do not need to be accepted and applied by the OPC server in all cases. Input signals can be read back to make the value that is actually in effect in the OPC server available to your simulation model. Note that this is possible only if the signal is declared as “readWritable” on the OPC server.

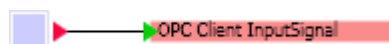
To read back an input signal, set the corresponding option in its properties.

**Increments**

Property	Value
Name	Increments
Type	binary
Multiplier	1
Comment	
Signal which can be read back	<input checked="" type="checkbox"/>

You can then use the input signal in an output connector to read the value that is actually in effect on the OPC server.

Value that is sent from SIMIT to the OPC server:



Value that is read back by the OPC server:



## Establishing a connection to the OPC server

A connection to the OPC server is established automatically upon starting the simulation. If no connection could be established with a timeout of five seconds, the simulation is started anyway and the connection is established in background. If an already established connection fails later, SIMIT will cyclically try to reestablish the connection. In such cases you will receive corresponding notifications from the messaging system.

Messages			
Mask acknowledged messages			
Simulation time	Category	Source	Message
00:00:00:000	SYSTEM MESSAGE	Gateway OPC Client	OPC Server not ready

The current connection status is indicated by the value of the status display signal.

After a connection has been established to the OPC server, SIMIT will try to register all signals listed in the OPC client coupling on the OPC server. It may happen that a signal is unknown to the OPC server or does not have the correct data type. In this case you will receive a corresponding notification from the messaging system.

Messages			
Mask acknowledged messages			
Simulation time	Category	Source	Message
00:00:00:000	SYSTEM WARNING	Gateway OPC Client	Unknown OpcItem: InputSignal
00:00:00:000	SYSTEM WARNING	Gateway OPC Client	Unknown OpcItem: OutputSignal

## 2.4.4 DCOM configuration

### 2.4.4.1 Overview

As long as both the server and the client are located on the same computer, an OPC connection is a local connection. It does not require any special system settings.

If communication between two computers is to be established using the local network, some marginal conditions must be observed. In this case, DCOM mechanisms are used which will only work if configured correctly.

Because communication between different computers is usually controlled by a firewall, the necessary settings have to be made there as well.

---

**Note**

All settings concerning access rights on your Windows computer heavily depend on the version of your operating system and the software that is already installed on your computer. This means we can only provide you with hints about how to configure DCOM, but not detailed instructions with respect to your specific environment.

---

**Note**

Apply changes to your computer settings only if you are aware of their consequences. Make sure that you do not inadvertently remove the protection of your computer.

---

#### 2.4.4.2 Firewall

When using a firewall you need to enable TCP port 135 for DCOM for incoming queries. Furthermore all relevant OPC Servers and OPC Clients as well as *Microsoft Management Console* and *OPCEnum* must not be blocked.

#### 2.4.4.3 Domain and user

All relevant computers need to belong to the same work group or domain. Use the same user name and password on all computers.

#### 2.4.4.4 Granting access rights

When establishing an OPC connection via DCOM, a computer will access the resources of another computer and launch certain processes on it. This needs to be explicitly permitted because such actions might possibly be dangerous to your computer.

Access rights as well as launch rights and activation rights must be properly set. Required rights can be granted to all users. It is more secure, however, to create a dedicated user group. This allows you to individually decide who should be a member of this group and is assigned these enhanced access rights. In addition to general settings concerning COM security you need to set the rights for both the OPC server and the utility program *OPCEnum.exe*.

For configuration, use the *dcomcnfg.exe* program in Windows 7.

Details concerning DCOM configuration can be found in the document *Using OPC via DCOM with Windows XP Service Pack 2*, for example, released by the OPC Foundation and available on the Internet at <http://www.opcfoundation.org>.

## 2.5 Shared Memory coupling

### 2.5.1 How the SHM coupling works

---

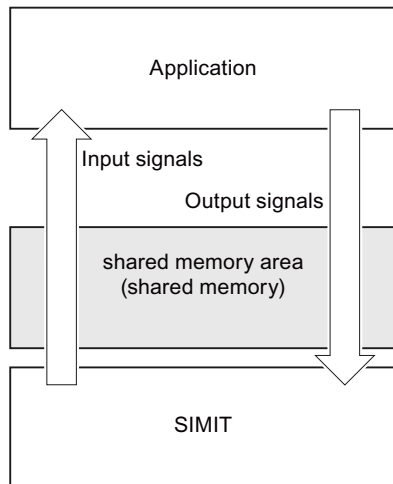
#### Note

The Shared Memory coupling is only included in the variant SIMIT ULTIMATE.

---

The Shared Memory coupling, or SHM coupling for short, can be used to exchange signals from SIMIT with any other application via a shared memory area (SHM). This coupling thus provides a universal, high-performance signal interface to SIMIT.

In this coupling, as is usual in SIMIT couplings, input signals are signals written by SIMIT to the memory area and output signals are signals read by SIMIT from the memory area.



#### 2.5.1.1 Accessing the memory area

If several mutually independent processes access the same memory area, this access has to be synchronized to ensure the consistency of the values. For that reason a mutex is used in the SHM coupling as a synchronization object.

In each simulation time slice, SIMIT writes and reads all the input and output signals defined in the SHM coupling and blocks the mutex while it does so. All other applications that access this memory area should do the same.

---

#### Note

Every application that is connected to SIMIT via an SHM coupling should keep the time for which the mutex is blocked as short as possible to avoid blocking access to the shared memory by SIMIT and other applications any longer than is necessary.

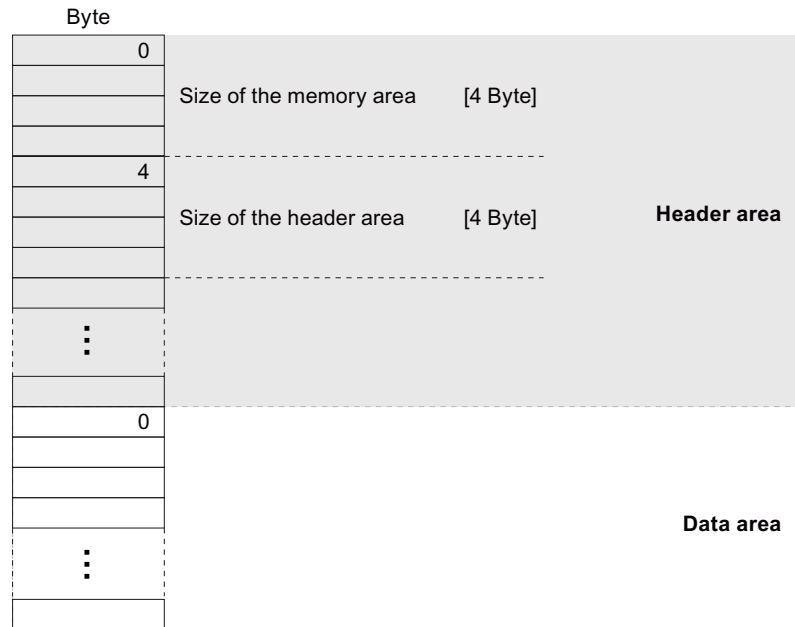
---



### 2.5.1.2 Structure of the memory area

#### Overview

The memory area is divided into a header area and a data area. The header area size is at least 8 bytes. The first four bytes of the header area contain the size of the entire memory area, the next four bytes the size of the header area. Both values are in little-endian format, as is usual in Microsoft Windows.



#### Structure of the data area

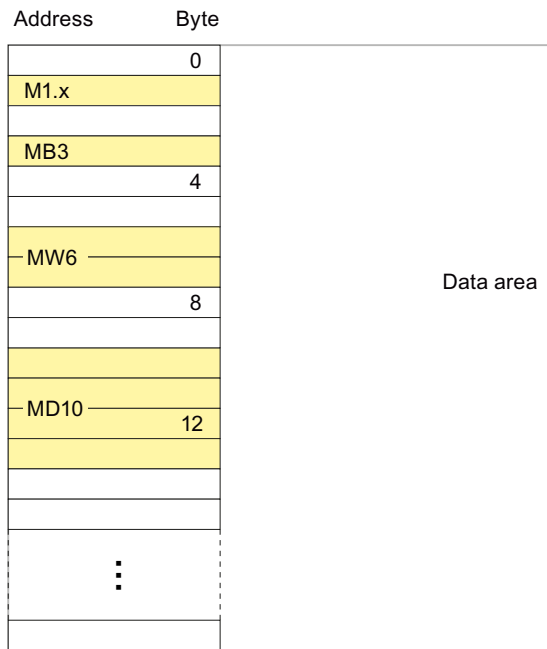
SIMIT addresses the data area in bytes similarly to the way in which SIMATIC automation systems address the I/O area: every signal from the SHM coupling is linked to a unique address in the data area. However, as a signal in an address in the data area can only be defined as an input signal or output signal, it is not permissible to assign the same address to both an input and an output signal.

The input and output signals are mapped to a shared memory area. This is why no overlapping between input and output signals is permitted for this coupling. Overlapping input and output signals are detected in the consistency check and displayed as inconsistencies.

Depending on its data type, a signal occupies one, two or four bytes under its address in the data area:

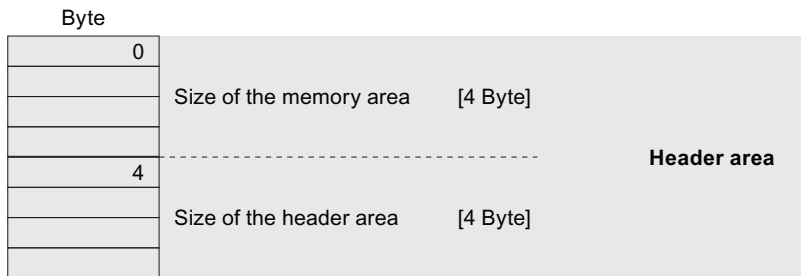
- One byte for BOOL and BYTE data types
- Two bytes for WORD and INT data types, and
- Four bytes for DWORD, DINT and REAL data types

An example of addressing for various data types is shown in the figure below.



**Structure of the header area**

The minimum header area size is 8 bytes. The first four bytes of the header area contain the size of the entire memory area, the next four bytes the size of the header area.



If SIMIT creates the shared memory area, SIMIT can optionally add additional variables and a list of signals to the header area. This information can then be used by applications to configure their access to the data area.

Byte			
0			
	Size of the memory area	[4 Byte]	
4			
	Size of the header area	[4 Byte]	
8	Version	[2 Byte]	
10			
	Scanning cycle in ms	[4 Byte]	<b>Header area</b>
14	Cycle counter	[2 Byte]	
16	Length of mutex name	[1 Byte]	
17			
⋮	Mutex name ( $m$ characters)	[ $m$ Byte]	
17+ $m$			
⋮	<b>Signal list</b>		

The version identifies the memory structure. As the structure defined here always has the version code zero, you will always find the value zero entered there. If the structure of the memory area is altered, the version is changed accordingly.

The predefined sampling time slice for the SHM coupling is entered in milliseconds (ms) as an integer value.

The value of the time slice counter is incremented by one in each cycle of the SHM coupling.

The length of the mutex name, which means the number  $m$  of characters in the mutex name, is stored in byte 16 of the header area. The mutex name is stored in the adjacent header area from byte 17 onwards.

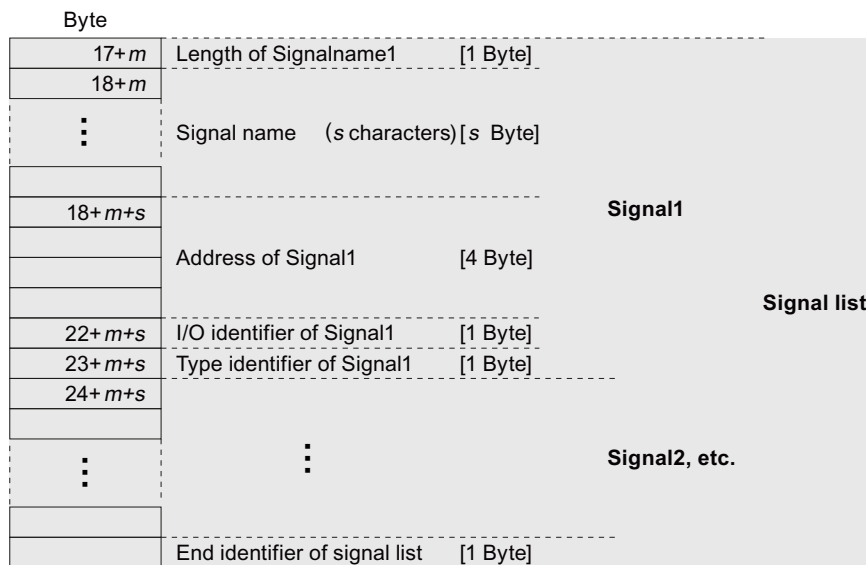
The signal list created from byte 17+ $m$  onwards in the header provides information about the signals in the data area. The following variables are specified for each signal:

- The length of the signal name, which means the number of characters in the signal name
- The signal name
- The signal address, which means the offset of the signal in the data area
- The I/O identifier, which means the identifier of whether the signal is an input signal (identifier 0) or an output signal (identifier 1), and
- The type identifier, to identify the data type

Table 2-14 Signal type identifiers

Type identifier	Meaning (data type)
0	BOOL, bit address 0
1	BOOL, bit address 1
...	...
7	BOOL, bit address 7
8	BYTE
9	WORD
10	INT
11	DWORD
12	DINT
13	REAL

The structure of the signal list in the header area is shown in the figure below.



The signal list ends with an end identifier of value zero.

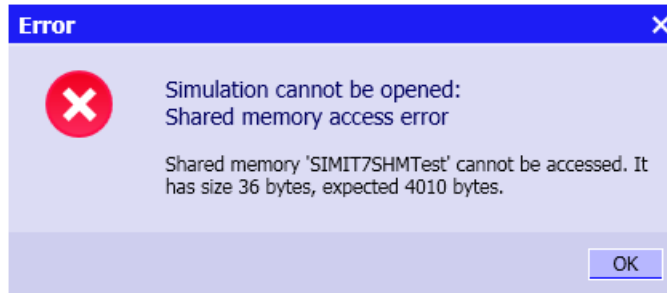
### 2.5.1.3 Creating the memory area

The shared memory area can be created either by SIMIT or by another application that is connected to SIMIT via the memory area. The way in which SIMIT behaves when a simulation containing an SHM coupling is started is adapted accordingly.

If the shared memory area was created by an application, meaning that it already exists when the simulation is started in SIMIT, then SIMIT opens the memory area. However, SIMIT only connects to the memory area if the size of the data area corresponds to the size of the address area defined by the input/output signals in the SHM coupling. Otherwise, an error message (see below) is displayed and SIMIT does not connect to the memory area.

If the shared memory area does not exist when SIMIT is started, it is created by SIMIT. SIMIT enters the size of the entire memory area and the header area into the header area. The size of the data area is determined by the highest address of the signals defined in the SHM

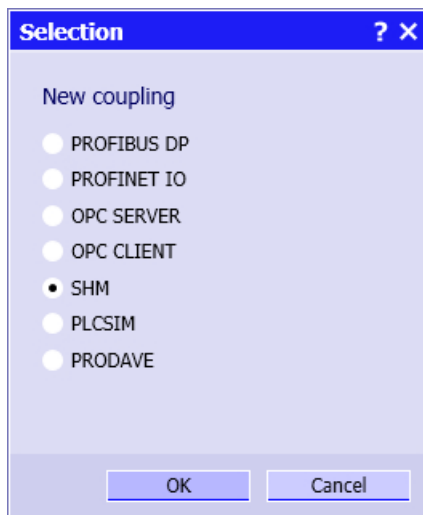
coupling. Optionally, other sizes and a list of the signals can be entered in the header area by SIMIT. You can find additional information on this in the section: Structure of the memory area (Page 153).



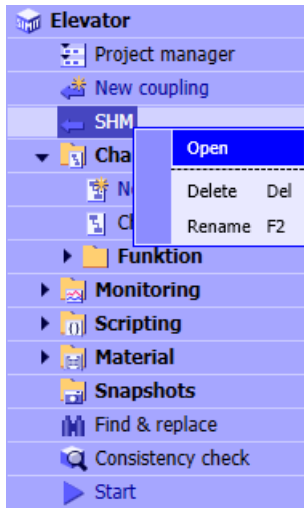
## 2.5.2 Configuring the SHM coupling

### 2.5.2.1 Creating an SHM coupling

To add a SHM coupling to your simulation project, create it using the *New coupling* command in the "Couplings" folder of the project tree or select "Couplings > Add new coupling" in the portal view. Then select *SHM* as the coupling type.



You can choose any name for the SHM coupling in your SIMIT project. To accept the default name, press Enter. The coupling editor then opens. You can also open the coupling editor at a later date by double-clicking the coupling in the project tree or via the shortcut menu.



---

**Note**

A maximum of 8 SHM couplings can be created in a project.

---

### 2.5.2.2 Configuring the signals in the SHM coupling

The input and output signals of the SHM coupling can either be edited manually in the editor of the SHM coupling or they can be configured by importing a signal list. The figure below shows the open SHM coupling editor as an example:

The screenshot shows the SHM coupling editor interface. It has a blue header bar with the text 'SHM'. Below the header, there are two main sections: 'Inputs' and 'Outputs'. Each section has a 'Reset Filter' button. The 'Inputs' section contains a table with columns: Default, Symbol Name, Address, Data Type, and Comment. The 'Outputs' section contains a table with columns: Symbol Name, Address, Data Type, and Comment.

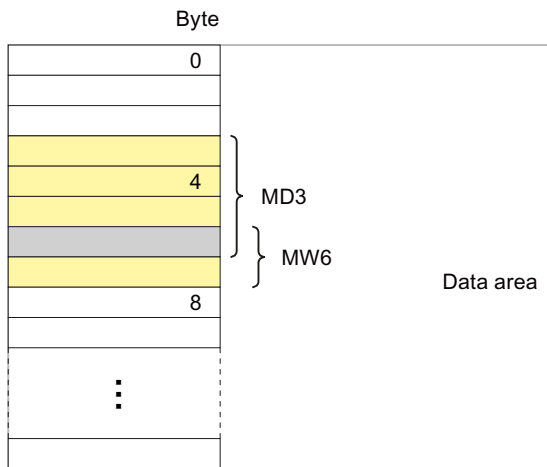
Default	Symbol Name	Address	Data Type	Comment
<input type="checkbox"/>	NK112_open	M0.0	BOOL	Valve NK112 open, RMT 1
<input type="checkbox"/>	NK113_open	M0.1	BOOL	Valve NK113 open, RMT 1
<input type="checkbox"/>	NK114_open	M0.2	BOOL	Valve NK114 open, RMT 1
<input type="checkbox"/>	NK115_open	M0.3	BOOL	Valve NK115 open, RMT 2
<input type="checkbox"/>	NK116_open	M0.4	BOOL	Valve NK116 open, RMT 2
<input type="checkbox"/>	NK117_open	M0.5	BOOL	Valve NK117 open, RMT 2
<input type="checkbox"/>	NK118_open	M0.6	BOOL	Valve NK118 open, RMT 2
<input type="checkbox"/>	NK311_open	M0.7	BOOL	Valve NK311 open, Reactor 1
<input type="checkbox"/>	NK312_open	M1.0	BOOL	Valve NK312 open, Reactor 1
<input type="checkbox"/>	NK313_open	M1.1	BOOL	Valve NK313 open, Reactor 1

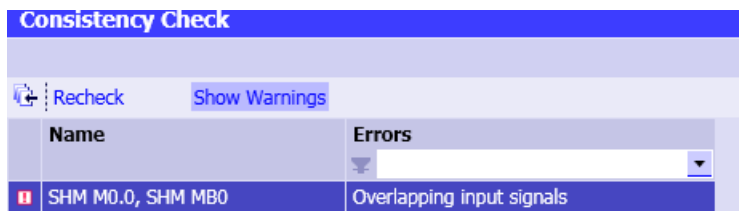
Symbol Name	Address	Data Type	Comment
NK111_open	MB32	BYTE	Valve NK111 open, RMT 1
NK111_copen	M4.0	BOOL	Valve RMT 1 NK111 open
NK112_copen	M4.1	BOOL	Valve RMT 1 NK112 open
NK113_copen	M4.2	BOOL	Valve RMT 1 NK113 open
NK114_copen	M4.3	BOOL	Valve RMT 1 NK114 open
NK115_copen	M4.4	BOOL	Valve RMT 2 NK115 open

SIMIT reads the output signals from and writes the input signals to the shared memory area. Access takes place cyclically in the cycle defined for the coupling. One byte of the memory area must be uniquely assigned to either an input signal or an output signal. In addition, each input signal must be uniquely mapped in the data area. The data areas of different input signals must not overlap.

The figure below shows an example of two input signals, MD3 and MW6, which overlap in byte 6:



In the consistency check by SIMIT, overlapping signals are reported as an error. The figure below shows a typical error message when a binary signal (M0.0 in this example) is in the data area of another signal (MB0 in this example).



**Note**

As the smallest addressable unit of the memory area is a byte, a byte can only be assigned to either an input signal or an output signal. Therefore binary signals with the same byte address can only be either all binary input signals or all binary output signals.

In addition, write access to the memory area by SIMIT is in bytes rather than in bits. A bit in a byte of the memory area for which no binary input signal is defined in the SHM coupling is set to zero during write access to the memory area by SIMIT.

The address designation of a signal starts with "M" for memory and contains the data type and the address. Based on the notation for addresses in SIMATIC automation systems, the data types listed in the table below can be used.

Table 2-15 Definition of data types

Data type	Variable	Notation	Value range
BOOL	1 bit	M<byte>.<bit>	True/False
BYTE	1 byte (8 bits)	MB<byte>	0 ... 255 or -128 ... +127
WORD	2 bytes	MW<byte>	0 ... 65,535
INT	2 bytes	MW<byte>	-32,768 ... 32,767
DWORD	4 bytes	MD<byte>	0 ... 4,294,967,295
DINT	4 bytes	MD<byte>	-2,147,483,648 ... 2,147,483,647
REAL	4 bytes	MD<byte>	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$



The following applies to the mapping of integer signals in SIMIT simulation projects to data types in the memory area:

The least significant byte (LSB) of the signal is used for the BYTE data type. The WORD and DWORD data types are unsigned and the INT and DINT data types are signed. The values are limited to the value ranges specified in the table above.

The structure of a floating-point number is governed by the "IEEE Standard for Binary Floating Point Arithmetic" (ANSI/IEEE Std 754-1985).

### 2.5.2.3 Signal properties

The properties of a signal are shown in the individual columns of the editor window and in the property view of the coupling editor.

NK112_open		
General	Property	Value
Connection	Symbol name	NK112_open
	Address	M32.1
	Data type	BOOL
	Comment	Valve NK112 open, RMT 1

The following signal properties are defined:

- **Symbol name**  
The signal is identified by this name in SIMIT.
- **Address**  
Signal values are stored in the data area of the shared memory area under this byte address. The address zero corresponds to the first byte after the header area.
- **Data type**  
A signal data type determines the space it occupies in the data area and how the values stored there should be interpreted: as a logical value, as a signed or unsigned integer or as a floating-point number.
- **Comment**  
The comment is used to document the signal. It is not evaluated by SIMIT.

### 2.5.2.4 Properties of the SHM coupling

The properties of the SHM coupling can be defined in the property view of the coupling editor:

SHM	
Property	Value
Time slice	2
Shared memory name	SIMITSHM
Mutex name	SIMITSHMMutex
Signal description in header	<input type="checkbox"/>
Header size	8
Big/Little Endian	little

- **Time slice**  
Like any SIMIT coupling, the SHM coupling is processed in fixed time slices. Here you specify which of the eight possible time slices of the SIMIT project the coupling should be assigned to.

---

**Note**

The time slice with the lowest cycle time always has the highest priority, regardless of the numbering.

---

- **Shared Memory name**  
Here you enter the name with which the shared memory area can be addressed.
- **Mutex name**  
Here you enter the name of the mutex for synchronizing access to the shared memory area.
- **Signal description in the header**  
This option allows you to choose whether SIMIT should create an extended header area. You can find additional information on this in the section: Structure of the header area (Page 154).
- **Header size**  
Here you enter the size of the header area in bytes. The value may be freely chosen, but must be at least 8 bytes. When SIMIT creates the shared memory area it creates a header area of the specified size.

---

**Note**

If you activate the "Signal description in header" option, the size of the header area is determined by SIMIT based on the input/output signals in the coupling. The "Header size" property is not editable in this case.

---

- **Big/Little Endian**  
This property determines the byte order in which values of the WORD, INT, DWORD and DINT data types are encoded in the data area.

Table 2-16 Byte order

Setting	Byte order
Big endian	The most significant byte is stored first, which means at the lowest memory address.
Little endian	The least significant byte is stored first, which means at the lowest memory address.

### 2.5.2.5 Importing and exporting signals

SIMIT can store coupling content in txt format; this format contains SIMATIC addresses and information about scaling. Because this kind of information does not exist in SHM couplings, this format is only suited to a limited extent. However, you can still export import signal tables in txt format; only relevant information is available or taken into consideration in the SHM coupling.

You can find additional information on this in the section: The signal table (Page 64).

SIMATIC symbol tables can also be imported in asc, seq and xlsx format as well as txt format. Especially in this case, ensure that addresses for inputs and outputs in the SHM coupling are not allowed to overlap.

---

**Note**

If you open the signal table for editing in Excel, all cells must be formatted as "text" so that Excel does not make any unwanted format conversions.


---

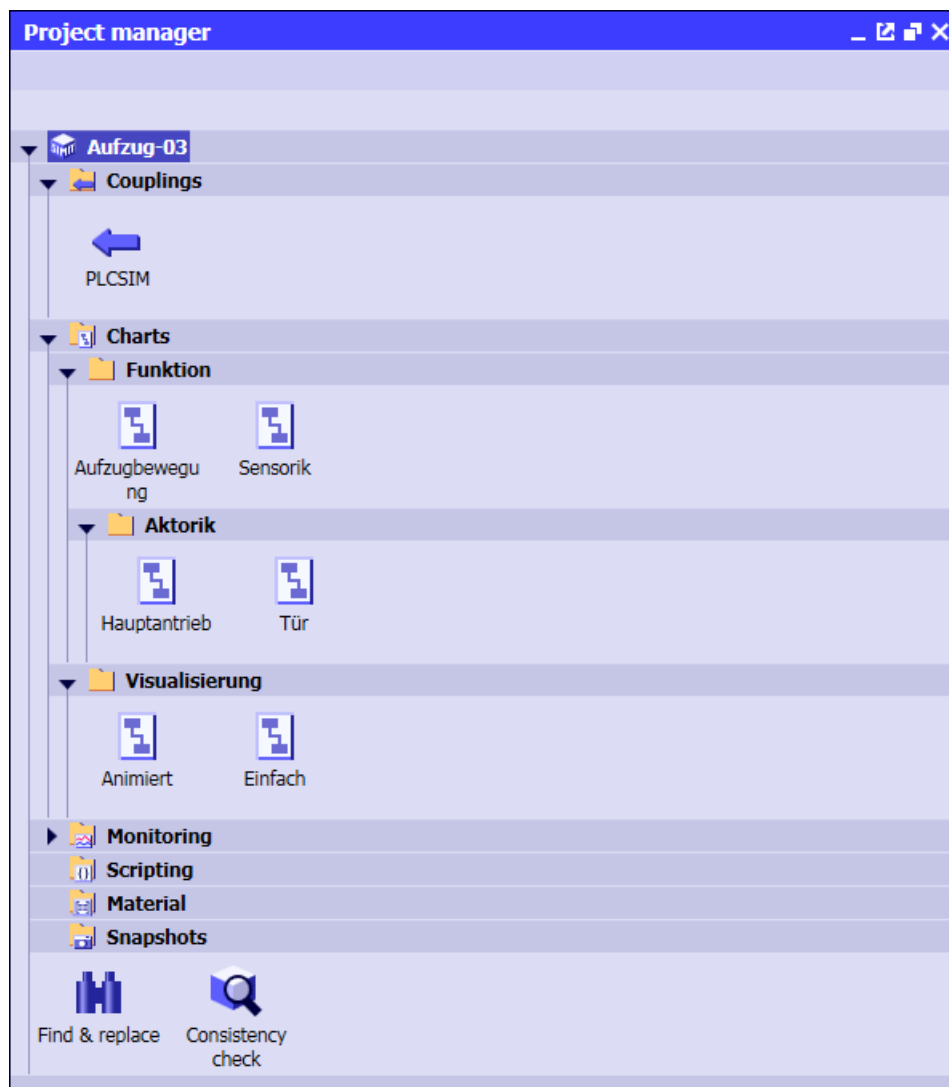


## Simulation model

### 3.1 The Project Manager

#### 3.1.1 View and functions of the Project Manager

Double-click the  Project manager node in the project tree to open the Project Manager. An alternative view of the project compared to the project tree opens in the "Project Manager" editor window:



Just like in the project tree, the Project Manager allows you to move charts and other project elements as well as entire folders via drag-and-drop or to use commands in the shortcut menu

### 3.1 The Project Manager

to copy, paste, delete, or rename them. In the Project Manager, you can also double-click project elements such as charts to open them for editing.

The property view of the Project manager provides access to the properties of your simulation project:

- The project location,
- The project version
- The "*Read-only*" property
- The "*Default scale*" if you use the CONTEC library and
- The eight different time slices (Time slice 1 to Time slice 8)
- The operating modes "Synchronous" and "Asynchronous".  
You can find more information on this in the section: Synchronous and asynchronous operating modes (Page 29).

The "*Project location*" property indicates which folder your project is located in. The 8 time slices, which you assign to components, controls and couplings in your project, can be assigned any value in milliseconds, provided it is at least 10 ms.

You can find a description of the "*Default scale*" property in the section: Scalability (Page 569).

You can find a description of the "Project version" property in the section: Versioning (Page 166).

You can find a description of the "*read-only*" property in the section: Write protection (Page 168).

#### 3.1.2 Versioning

The version number is automatically included in the simulation project. It consists of several elements:

- License number (for example AB12345)
- Time stamp (resolution is approx. 38 seconds)
- Major version (0 - 127)
- Minor version (0 - 999)

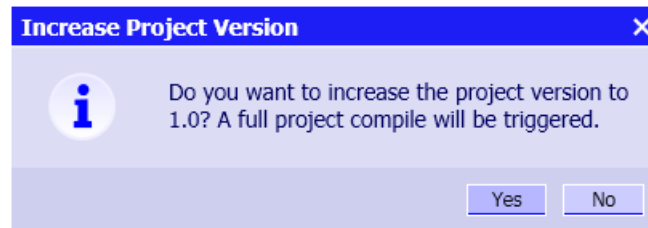
If the simulation project was changed, the version number is updated automatically the next time the simulation is started.

Property	Value
Project Location	E:\SIMIT-Projekte\Examples\Examples.simit
Project Version	AA12320-926737-0.2 <span style="float: right;">...</span>

If the simulation project has been changed but has not been started since then, an asterisk appears after the version number.

Property	Value
Project Location	E:\SIMIT-Projekte\Examples\Examples.simit
Project Version	AA12320-926737-0.2 (*)

The minor version is incremented up to 999, then it returns to zero and the major version is increased by one. If you want to increment the major version before then to document a particular project status, for example, click the "..." button. After a prompt, the project is compiled and the new version number entered.

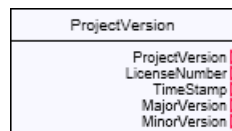


### Note

When you create a new simulation project or open or retrieve a simulation project that was created with an older version of SIMIT, the version is set to *unversioned*. The version is set automatically once the simulation is started.

### Viewing the versioning with the "ProjectVersion" component type

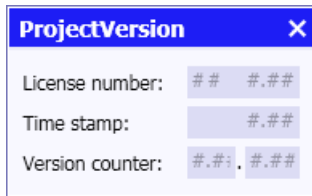
The project version can also be determined using the *ProjectVersion* component type. This component type is located in the standard library in the *Misc* subfolder. It has the following symbol:



At the outputs of a component of this type the following integer values are available:

- **ProjectVersion**  
The complete version number coded in numerical form.
- **LicenseNumber**  
The license number coded in numerical form.
- **TimeStamp**  
The time stamp.
- **MajorVersion**  
The major version.
- **MinorVersion**  
The minor version.

The individual version numbers are displayed in readable form in the component operating window.



When the Automatic Control Interface function module is used, the project version can be accessed via the `_ProjectVersion` system variable. This allows the project version to be output in the log file, for example, as shown in the figure below.

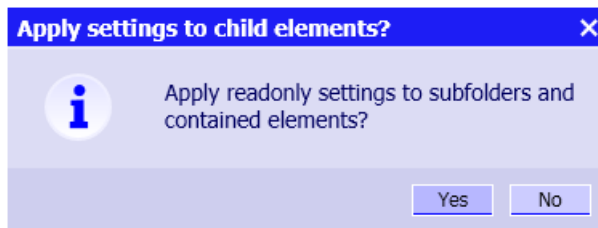


### 3.1.3 Write protection

You can protect your entire simulation projects or parts of them against accidental changes. All protectable elements of a project now have an additional *Read-only* attribute.

#### Setting and removing write protection

When you set or remove the write protection for a folder, you can choose to apply this action to all subfolders and the elements they contain. A prompt occurs for confirmation:



The write protection for individual objects, such as charts in lower-level folders, can still be changed subsequently on an individual basis so that it is not permanently tied to the folder status.

To set or remove the write protection for an element of your simulation project, select this element. The property view of the project manager allows you to define write protection as desired:

Diagram	
Property	Value
Readonly	<input checked="" type="checkbox"/>



### The effects of write protection

The effect of the write protection varies according to which element of a project it is applied to:

- **Chart**  
Read-only charts cannot be deleted or renamed. They can be opened in the editor but not modified. Objects in the chart (components, controls and graphics) can be selected and copied to other charts. Inputs can be controlled while the simulation is running.
- **Chart folders**  
Read-only folders cannot be deleted or renamed. New charts or subfolders cannot be created in read-only folders, not even by moving or copying charts or folders.
- **Couplings**  
Read-only couplings cannot be deleted or renamed. They can be opened in the editor but not modified.
- **Snapshots**  
Read-only snapshots can be loaded when a simulation is running but not deleted or renamed.
- **Snapshot folders**  
Read-only folders cannot be deleted or renamed. Charts or snapshots cannot be created in read-only folders, not even by moving or copying snapshots or folders.  
If the uppermost snapshot folder in the project tree is read-only, no more snapshots can be created, because they are automatically stored in the uppermost snapshot folder. The corresponding icon in the toolbar is disabled in this case.
- **Trend**  
Read-only trends cannot be deleted or renamed. They can be opened in the editor but not modified.
- **Archive**  
A read-only archive can be opened in the editor but not modified.
- **Script**  
Read-only scripts cannot be deleted or renamed. They can be opened in the editor but not modified.
- **Script folders**  
Read-only folders cannot be deleted or renamed. New scripts or subfolders cannot be created in read-only folders, not even by moving or copying scripts or folders.
- **Project Manager**  
If the Project Manager is read-only, project properties such as cycle times cannot be modified. However, all actions relating to project elements are available in both the Project Manager and the project tree, subject to their write protection settings.
- **Complete project**  
Read-only projects cannot be renamed and couplings cannot be created.  
Note that write protection for a project does not automatically mean that all components of the project are protected from being changed. However, when you set write protection for a project, you can cause this setting to be inherited by all elements of the project.

Read-only elements such as charts are opened in the editor with a white title bar to indicate their special status.

## 3.2 The chart editor

You can find additional information about the chart editor in the following sections:

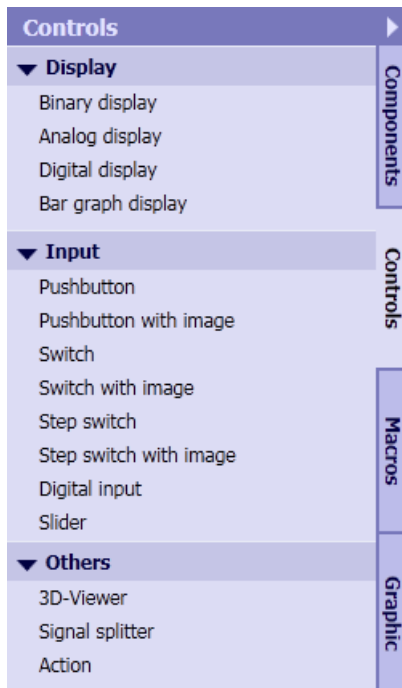
- Creating simulations (Page 26)
- Connecting connectors (Page 301)
- Setting inputs (Page 303)
- Properties of components (Page 304)
- Properties of controls (Page 311)
- Visualizing simulations (Page 34)

## 3.3 The task cards

### 3.3.1 The "Controls" task card

Controls are provided in the "Controls" task card. There are three panes of controls:

- Controls for displaying signal values in the *Display* pane
- Controls for entering signal values in the *Input* pane
- Other controls in the *Others* pane



Controls can be found in the library under their names.

You can find additional information on this in the section:

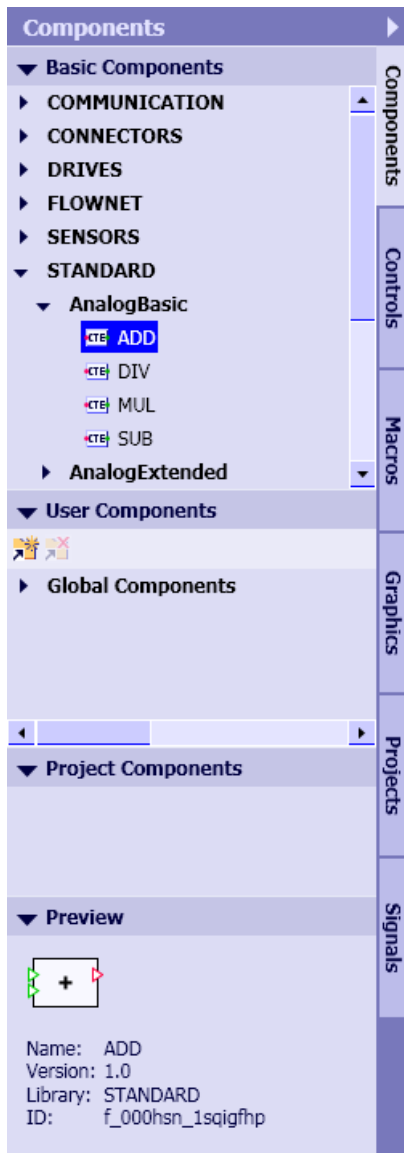
- Controls (Page 427)
- Visualizing signals (Page 38)

### 3.3.2 The "Components" task card



The "Components" task card of SIMIT consist of three panes:

- *Basic components*
- *User components*
- *Project components*

and one pane for the preview.



The *basic components* pane contains the component types from the SIMIT basic library. This basic library is created when you install SIMIT: You cannot modify these component types, nor can you add further component types to the basic library. You can, however, copy component types from the basic library to the other two panes. You can find additional information on this in the section: The basic library (Page 297).

*User components* gives you access to your own libraries of component types, which means to component types that you have created yourself or have been made available to you by other people. There you can store component types in the fixed *Global components* directory as a global library that is available in your SIMIT installation and thus add to the work area of your SIMIT installation. You can also use the  command to open any library directories in this pane and access the component types stored in them. The  command will remove the selected directory from this pane.

You can find additional information about creating your own components in the "SIMIT - Component Type Editor" help.

In the *Project components* pane, you can connect component types to the open SIMIT project. When you archive the project, all component types in this pane are archived with the project as the project library and will be available in the project again after you have retrieved it.

You can move component types anywhere within the two *User components* and *Project components* panes or add them as copies. The component types from the basic components pane can only be copied to the other two panes.

You can open component types via their shortcut menu (*Open* command) in the editor.

### Updating the "Components" task card

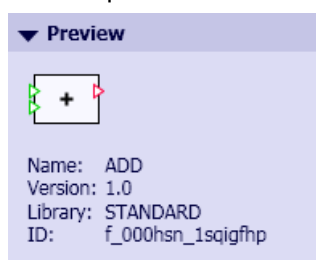
When you start SIMIT, the basic library, global library and project library are read in and made available in the corresponding panes in the "Components" task card. If library directories were open in the *User components* pane when you previously closed SIMIT, these will also be opened once more.

Now when you create custom component types with the CTE, you have to save them either in a library directory or under *Global components* so that you can access them in SIMIT. SIMIT updates the pane of the *User components* automatically when you save a component type with the component type editor in it.

### The preview for component types

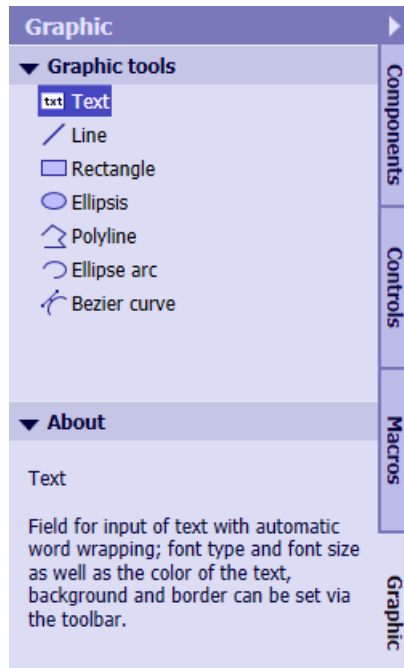
In the preview for the "Components" task card, the following information is displayed for a selected component type.

- **Symbol**  
The basic symbol for the component type
- **Name**  
The name entered in the component type
- **Version**  
The version information entered in the component type
- **Library**  
The library information entered in the component type
- **UID**  
The unique identifier that is automatically assigned to the component type



### 3.3.3 The "Graphic" task card

The "Graphic" task card provides you with graphic elements for use in charts.




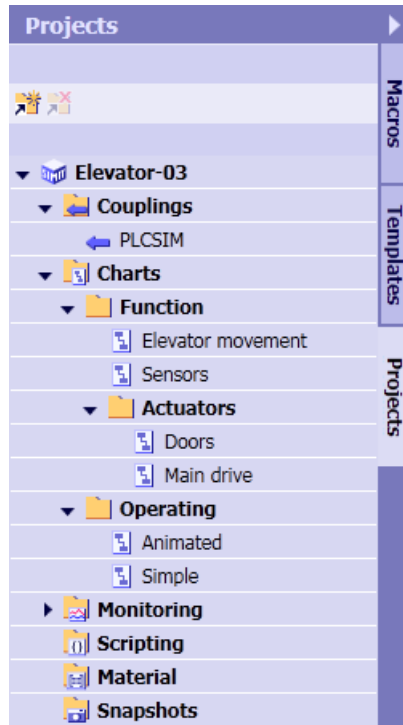
The various graphic elements are listed. Select a graphic element to display a short description of the graphic element under "About".

Drag-and-drop the desired graphic element into the chart editor to display it and edit it there. Symbols from the chart editor are available for editing.

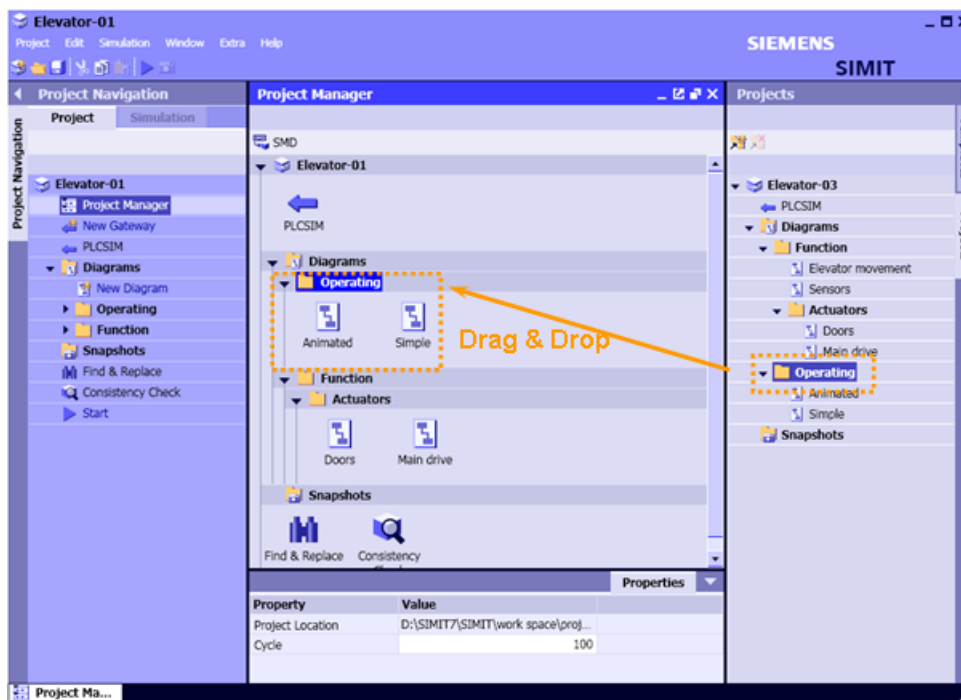
You can find additional information in the section: Visualization with graphics (Page 34).

### 3.3.4 The "Projects" task card


The Project Manager provides the "Projects" task card. This task card allows you to open projects other than the one you are currently editing to copy elements from these projects into your project. For example, retrieve each of the 2 projects into different project folders, and then open another project in SIMIT. Now open the Project Manager. Click  (Open project) in the "Projects" task card. In the file selection dialog that appears, navigate to the project folder of the open project and select the *\*.simit* project file there. The project is then opened in read-only mode in the "Projects" task card.



You can now drag-and-drop single elements such as charts as well as entire folders into the project manager and thus create a copy within the current project as shown in the figure below:



You may also open individual charts from a project in the chart editor, copy parts of these charts and paste them into charts in another project.

The "Projects" task card allows for several projects to be open simultaneously. This provides an easy and efficient way to build a project from parts of several "standard projects", for example. Use the  symbol to close projects in the Projects task card.

The Project Manager along with projects that may be open in the "Projects task" card is also available when a simulation is running. You may also open elements of a project from the Project Manager when a simulation is running.



### 3.3.5 The "Signals" task card

In the "Signals" task card, you can search for the signals in the open project and have them displayed.

Source	Name
ADD#1	X1
ADD#1	X2
ADD#1	X3
ADD#1	Y
ADD#2	X1
ADD#2	X2
ADD#2	Y

You can enter text of your choosing as a search criterion in the input boxes "Source" and "Name". Additional search criteria can be selected from the drop-down lists for "Origin", "Signal type" and "Data type". You can use the "Reset Filter" button to clear the settings and display all the signals of the open project under "Search Results".

▼ Info

**Origin:** Component

**Signal Type:** Input

**Data Type:** analog

When you select a signal, the origin, signal type and the data type are displayed under "About".

You can find additional information in the following sections:

- Dragging coupling signals from the Signals task card (Page 43)
- I/O signals for connection on charts (Page 57)
- Symbolic addressing (Page 74)

### 3.3.6 The "Macros" task card

A macro component allows you to combine frequently recurring functional parts of your simulation model, for example, parts of a chart. Using the macro then provides simple access to this function: You no longer need to copy the function from one chart and paste it to another, rather you can use the macro component similarly to a standard component and drag it from the task card, assign parameters to it and link it to other components or macro components.

Macro components are provided in the "Macros" task card in the chart editor. This task card is divided into the *Basic Macros*, *User Macros* and *Project Macros* panes. The name, version, library and UID of a selected macro component are displayed under *Info*.



The *Basic Macros* pane contains five macros which may be used as signal generators. They can be configured with respect to time slice duration and amplitude and yield different signal patterns:

- Random
- Sawtooth
- Sine
- Square
- Triangle

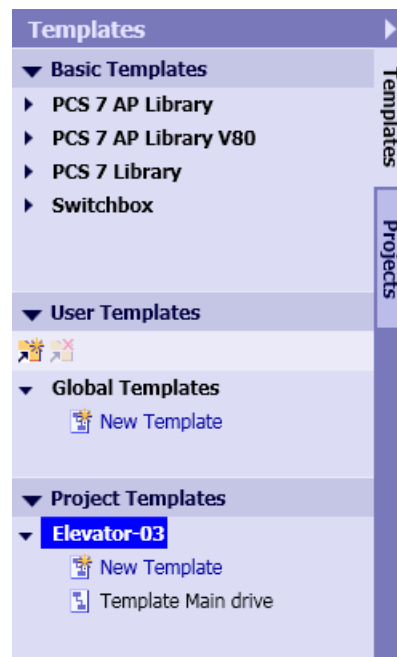
You can create your own macro components in either the *User Macros* or the *Project Macros* pane. Macro components created in the *User Macros* pane are stored in the SIMIT work area and are therefore available for all projects. In *User Macros* you can also open folders containing

macro components using the 📁 button. The 🗑 button closes a selected folder, which means that it removes it from the pane again.

Macro components in the *Project macros* pane are saved in the project folder. They are only available while this project is open. All macro components stored in *Project Macros* are archived with the project. Therefore they are available in the project again when you retrieve the project.

### 3.3.7 The "Templates" task card

In the Project Manager templates are made available via the "Templates" task card.



This "Templates" task card is divided into three panes:

- *Basic templates*
- *User templates*
- *Project templates*

*Basic templates* are templates provided by SIMIT. In the *PCS 7 Library* folder you will find templates matching the process tag types (templates) from the PCS 7 library. Templates that are adapted to the APL library (Advanced Process Library) of PCS 7 V7 and V8.0 can be found in the template folders *PCS 7 AP Library* and *PCS 7 AP Library V80*.

You may store templates for your SIMIT installation under *User templates* in the *Global templates* folder. User libraries may be loaded using 📁 and closed using 🗑.

The templates stored under *Project templates* are saved in the project and are archived with the project. Thus under *Project templates* you may store templates for the project that is currently open. After opening or retrieving a project these templates are available again.


### 3.4 The Macro Component Editor (MCE)

You may drag-and-drop *Basic templates* to copy them into *User templates* or *Project templates*. Templates in *User templates* and *Project templates* can be moved or copied between these two panes freely using drag-and-drop.

You can find additional information in the section: Templates (Page 218).

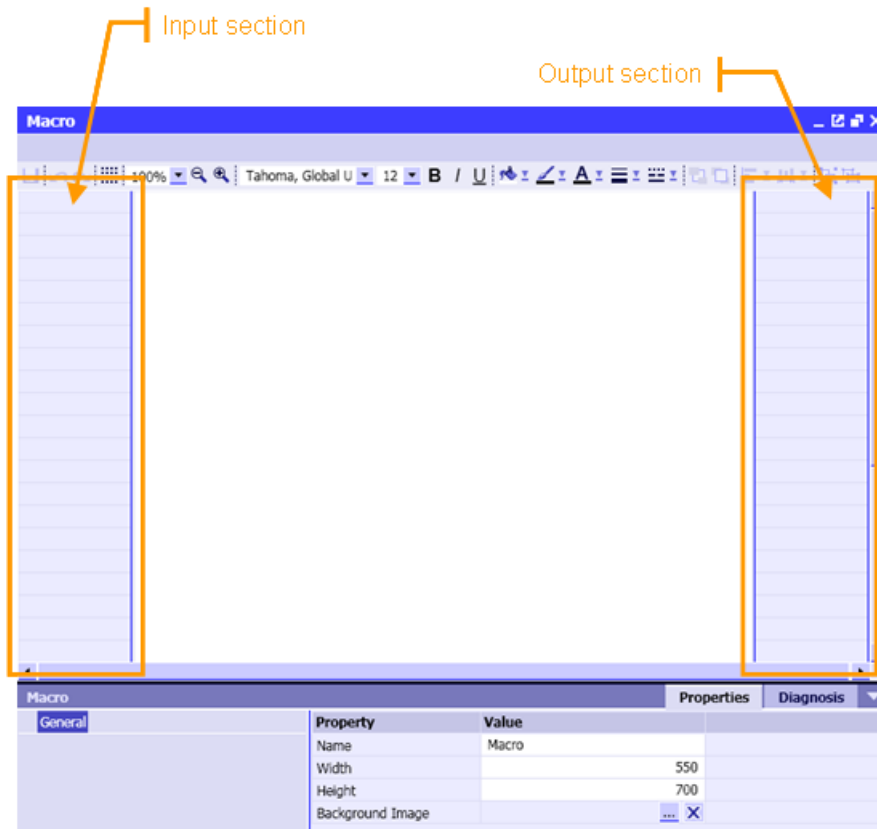
## 3.4 The Macro Component Editor (MCE)

### 3.4.1 The macro editor

To create a new macro component, double-click  **New Macro** in the *User Macros* or *Project Macros* pane. A new macro component is created in this area, and you may accept the suggested name or change it.



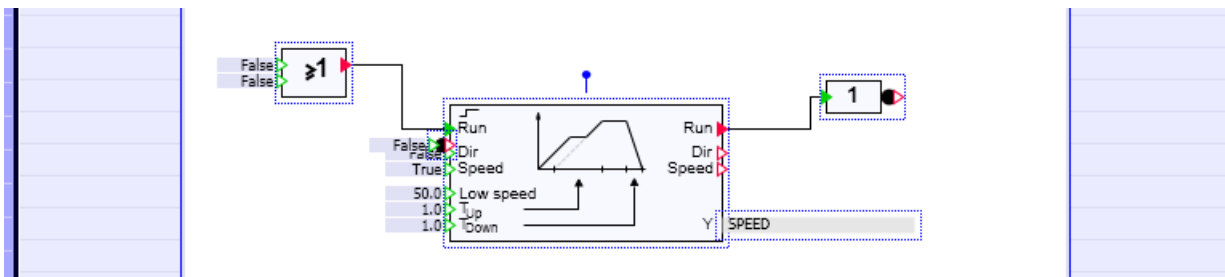
The macro editor opens after you confirm the default name or change the name by pressing the Enter key.



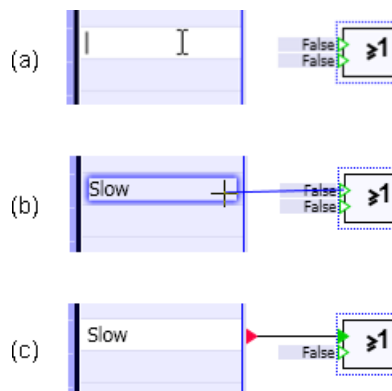
A macro chart differs from a standard chart by the two border sections on the left and right side. The inputs and outputs of the macro are assigned to cells in these sidebars: Inputs in the left sidebar, outputs in the right sidebar. As with charts you may use all of the components from the library to model certain behavior. You may also use static graphics in macro charts. You may only use the global connector from among the connector components. The other connectors cannot be used in macro components. You can find additional information on this in the section: Topological connectors of macro components (Page 184)

Controls and animations are also not available in macro components because macro components only package functional parts of the simulation. Macro components cannot be hierarchically structured, which means you cannot insert any macro components into a macro chart.

If you have created a chart already and wish to build a macro component from it, you can copy the relevant parts of the chart and paste them into the macro chart. Open the *Main Drive* chart from the sample project Elevator-03, for example, and copy its entire chart contents using the menu command *Edit > Copy*. Then open the macro chart and insert a copy there with *Edit > Paste*. Only usable components are pasted into the macro chart.



Now you need to provide the macro component with connectors, which means with inputs and outputs. To do this, double-click in a cell of the sidebar. The cell opens, and you may enter a name for the connector as shown in the figure below under (a). Enter a name, confirm your input using the *ENTER* key and connect the macro connector to a component connector, for example, by dragging the component connector onto the macro connector as shown in the figure below under (b). The connection is then shown as usual by means of a connecting line (see figure below under (c)).



You may also create a connector of a macro component by first clicking a component connector to open the connection, and then clicking the desired cell in the macro component sidebar to

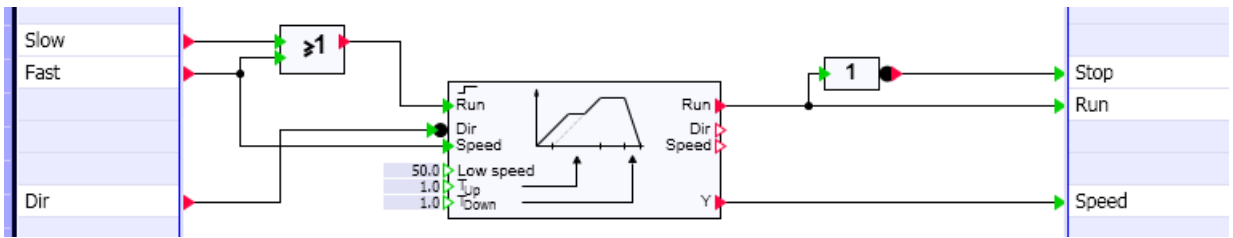
3.4 The Macro Component Editor (MCE)

close the connection. The name of the component connector is used as the default in the sidebar.


**Note**

Every connector must have a unique name. The name must contain only letters, digits and the underscore character, and must start with a letter. The name is case sensitive.

Proceed similarly for all additional connectors. Your macro should then look similar to what is shown in the figure below:



Save and close the macro chart.

Macro components containing errors are identified by means of the  overlay as shown in the figure below:

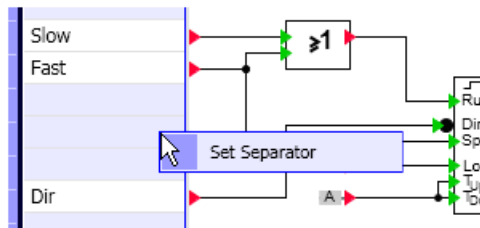


Macro components with errors can be opened and edited in the editor, but you cannot use them on charts. You can find an explanation of the errors in the editor properties view on the *Diagnostics* tab.

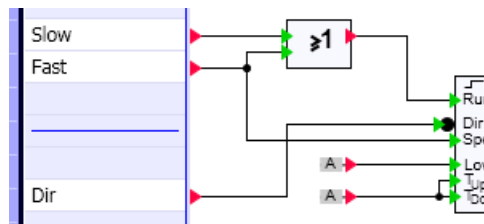
Macro	Properties	Diagnostics
<b>Description</b>		
	The macro does not have any connections.	

### 3.4.2 Separating connectors of a macro component

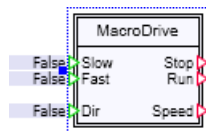
The connectors on the symbol of the macro components are set one after the other in the connector grid. You may use dividing lines to visually offset connectors from one another. To do this, open the example macro component in the macro editor. To offset the two *Slow* and *Fast* inputs from the *Dir* input by one grid unit, for example, use the shortcut menu in one of the cells between them to place a dividing line in that cell, as shown in the figure below.



Dividing lines are shown in the cell as a line.



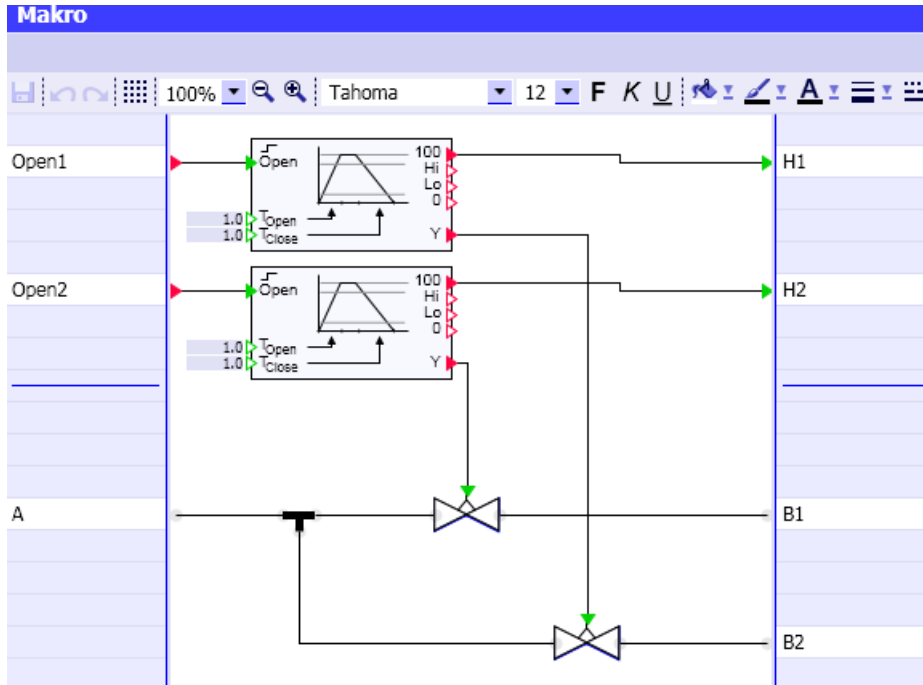
The connectors are now offset from one another accordingly in the macro component symbol. In the figure below, you can see that the outputs have also been offset from one another using a dividing line:



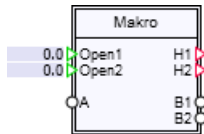
Dividing lines can be deleted using the corresponding command in the shortcut menu of the cell.

### 3.4.3 Topological connectors of macro components

You can also use components from the FLOWNET and CONTEC expansion libraries in macro components. Open topological connectors of a model created from FLOWNET or CONTEC components are mapped to topological connectors in the sidebars of the macro component. The topological connectors can be arranged on the right or left side of the macro component as preferred.









If macro components of this type are used on charts, the topological connectors of the macro component can be connected to topological connectors of other components or macro components. The topological structures defined in the macro components are then merely substructures of the complete topological structure of the flow or transport network.





### 3.4.4 Default settings for macro component inputs

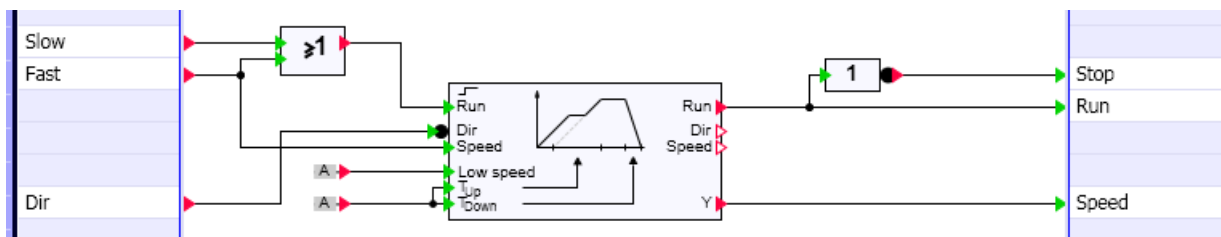
Default settings can be applied to individual macro component inputs. To do this, open the *Input* tab in the properties window of the macro component editor and enter the corresponding values.


Macro		
General	Name	Value/Signal
Input	Open1  	123 True 
Output	Open2  	123 False 


### 3.4.5 Defining parameters of macro components


Parameters of a macro component can only be built from parameters of the components that are used within the macro component.

Here is an example of how to set input values of a drive as parameters.




As you can see in the component property view, parameters of components are provided with an additional switch  in the macro editor.

AConst#1		
General	Name	Value
Input	Constant	0.0 
Output		
Parameter		
State		

If the switch is activated for a parameter () , this parameter then becomes a parameter of the macro component. It is elevated from the component level to macro component level.

The field that appears on the left side of the switch allows you to enter a name with which the parameter is displayed in the macro component as shown in the figure. You can enter an appropriate default value for the macro component.

AConst#1		
General	Name	Value
Input	Constant	30.0 SpeedSlow 
Output		
Parameter		
State		

## 3.4 The Macro Component Editor (MCE)

When the macro component is used on a chart, you can now see the parameters with their default values in the properties window.

MacroDrive#1		
General	Name	Value
Input	SpeedSlow	30.0
Output	TimeUpDown	0.0
Parameter		
State		

## 3.4.6 Properties of macro components

You can set the properties of a macro component in its properties window.

Property	Value
Name	Macro
Abbreviation	Macro
Version	
Library	
Password	
Width	550
Height	700

Macro components have the following properties:

- **Name**

The name of a macro component corresponds to the file name under which a macro component is saved in the file system (work area or any directory in the file system). The macro component is displayed with this name in the *Macros* task card.

---

**Note**

If macro components have been imported from an earlier SIMIT version, the file name may initially differ from the macro name.

---

- **Abbreviation**

The abbreviation is displayed in the header in the instance of a macro component.

- **Version**

The version of a macro component can be freely assigned. It appears in the Info palette in the *Macros* task card and in the tooltip for the macro component on the chart.

- **Library family**

The library family of a macro component can be freely assigned. It appears in the Info palette in the *Macros* task card and in the tooltip for the macro component on the chart.

- **Password**

Macro components can be password protected. If a macro component is password protected, it can be used on charts but cannot be opened without entering the correct password. The last password entered to open password protected macros is saved until the project is closed. Ensure that no unauthorized access to password-protected macros can occur!

---

**Note**

Keep the passwords for your macro components in a safe place, otherwise you will not be able to open even your own password-protected macro components.

---

- **Width**

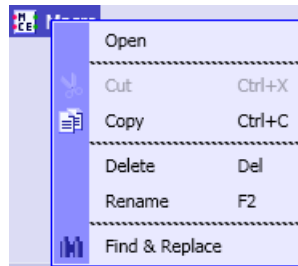
The width of the symbol.

- **Height**

The height of the symbol.

### 3.4.7 Find and Replace in macro components

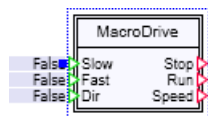
The shortcut menu for a macro component includes the command *Find & replace* which can be used to replace components within a macro component.



This command opens the Find & replace editor with a preset focus on the macro component.

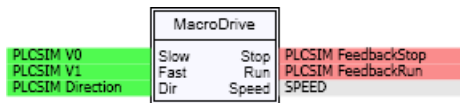
### 3.4.8 Using macro components

Like other components, macro components can be added on a chart with drag and drop. It is displayed on the chart as a rectangle with a double frame. The name of the macro component is displayed in the symbol header. The inputs and outputs are arranged on the left and right hand sides of the component in the sequence specified in the macro chart.



You can use a macro component on a chart like any other component. It can be connected with components and controls as well as with other macro components.

### 3.5 Migrating projects from previous versions of SIMIT



Double click the macro component to open the macro chart in read-only mode, which means you cannot make any changes to the macro chart.

## 3.5 Migrating projects from previous versions of SIMIT

### 3.5.1 Overview

Projects from SIMIT 7 can be opened and retrieved unchanged in SIMIT 8.

---

#### Note

Retrieving or opening a SIMIT 7 project in SIMIT 8 may take a very long time the first time it is done.

---

Projects from SIMIT 5 can be used in SIMIT 8 without having to perform any adaptations. Compared to SIMIT 5, SIMIT 8 represents a fully new development and, in addition to its fully redesigned user interface, offers an extended range of functions. These changes as well as reworked file formats of SIMIT projects make SIMIT 5 projects incompatible with SIMIT 8 projects. However, projects created with SIMIT 5 can be used in SIMIT 8 after migration.

---

#### Note

##### Migrating projects from SIMIT 5.4 SP1

These projects can be migrated only if they have been exported from SIMIT 5.4 SP1.

---

## 3.5.2 Migrating SIMIT projects

### 3.5.2.1 Aspects of project migration

SIMIT 5 projects use various objects such as charts, operating screens, components, couplings, etc. The following aspects must be considered when migrating a SIMIT 5 project:

- **Component types**

User-defined component types that were created with SIMIT 5 can be migrated with the component type editor (CTE) in SIMIT 8. You can find more information on this topic in the help "SIMIT - Component Type Editor > Migration of component types".

Component types that are part of the SIMIT 5 basic library require no migration because SIMIT 8 provides equivalent component types and controls in its basic library.
- **Charts and operating screens**

The charts and operating screens from SIMIT 5 are replaced with uniform charts in SIMIT 8 and are converted into charts during the migration. You can find additional information on this in the section: Migrating charts and operating screens (Page 192).
- **Macro components**

In SIMIT 5, macro components are encapsulated charts, whose connectors are defined by means of macro connectors. Macro components are created with the chart editor. In SIMIT 8, macro components are encapsulated charts that are created using a macro editor. Definition of a macro component connectors with this macro editor no longer requires macro connectors.

Macro components are migrated using the same mechanisms as for the migration of charts. You can find additional information on this in the section: Migrating macro components (Page 204).
- **Templates**

In SIMIT 5, a template is a chart or an operating screen that contains a placeholder instead of values for signals, parameters, etc. Templates are created using the chart editor or the operating screen editor. In SIMIT 8, templates are charts with placeholders. They are created with a template editor.

As with the migration of macro components, the migration of templates is based on the migration of charts. You can find additional information on this in the section: Migrating templates (Page 212).
- **Couplings**

Couplings need to be newly created in SIMIT 8. With a PROFIBUS DP coupling or a PROFINET IO coupling, the hardware configuration needs to be imported from the SIMATIC project – just like in SIMIT 5.

In SIMIT 8, the data types for coupling signals are more sophisticated than in SIMIT 5. In addition, signals in SIMIT 8 are also assigned to a source. You can find additional information on this in the section: Migrating couplings (Page 202).
- **Signal groups**

Signal groups are omitted in SIMIT 8. Migration of signal groups is not possible. Signal groups were replaced with a powerful archive in SIMIT 8.
- **Snapshots**

Snapshots cannot be migrated. Snapshots should be recreated in SIMIT 8 if necessary.
- **Scripts**

You can find additional information in the section: Migrating scripts (Page 214).

### 3.5.2.2 Step-by-step migration of projects

To migrate a SIMIT project, please follow these steps:

- *Step 1:*  
As needed, migrate all user-defined components that are used in the project. You can find more information on this topic in the help "SIMIT - Component Type Editor > Migration of component types".
- *Step 2:*  
Migrate the macro components that are used in the project. You can find additional information on this in the section: Migrating macro components (Page 204).
- *Step 3:*  
Export the project from the previous SIMIT version.

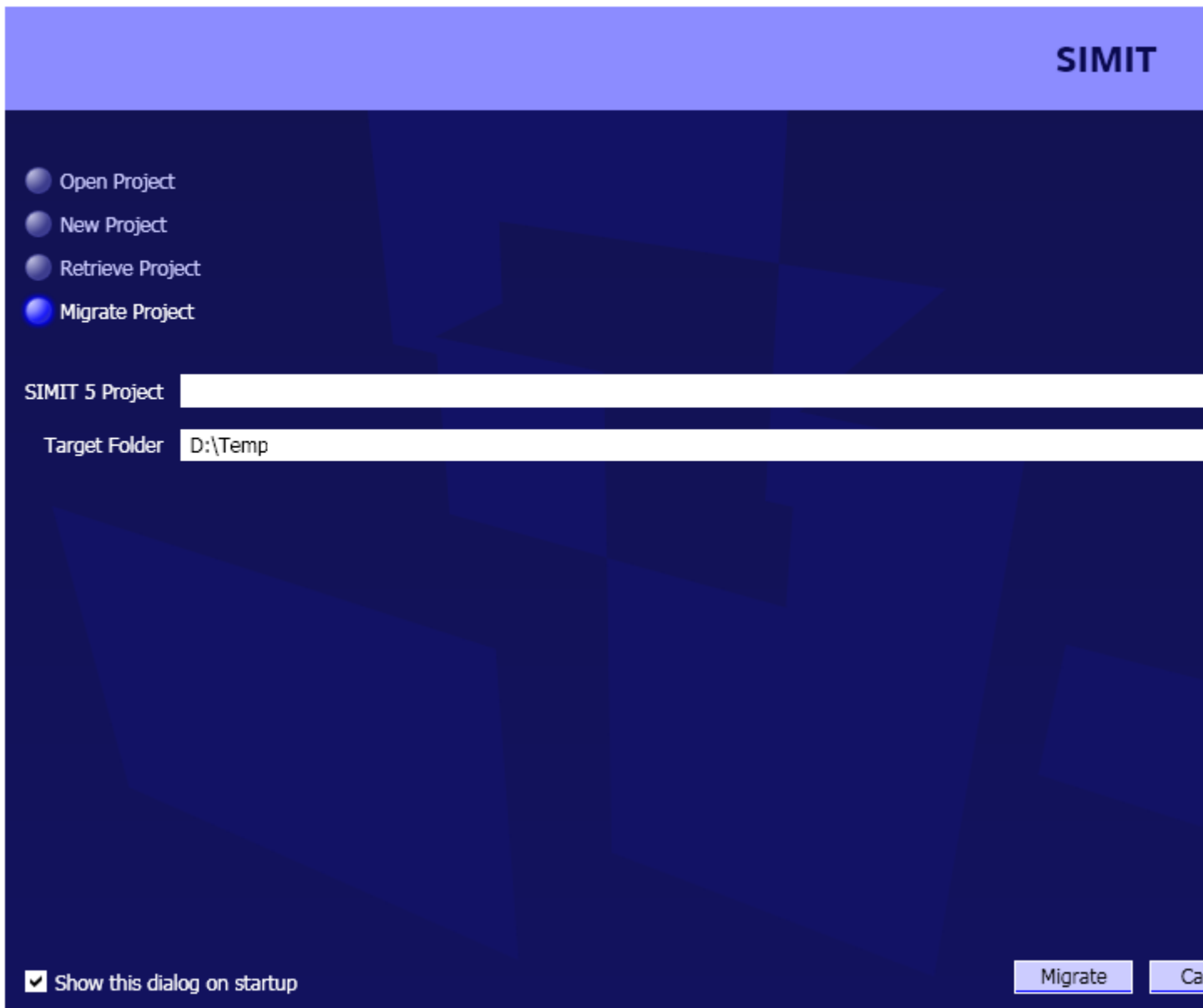
---

**Note**

Only SIMIT projects which have been exported with SIMIT 5.4 SP1 are migrated.

---

- **Step 4:**  
Import the exported SIMIT project using the menu command *Project / Migrate* in SIMIT 8 in order to migrate it to SIMIT 8.  
This menu command opens the dialog shown below. Now select the SIMIT project to be migrated and specify a target folder for the migrated SIMIT 8 project.



- **Step 5:**  
Configure the couplings in the SIMIT 8 project. You can find additional information on this in the section: Migrating couplings (Page 202).
- **Step 6:**  
Edit the project in SIMIT 8. Inconsistencies within the migrated project will be shown by the consistency check. A description of possible inconsistencies and instructions for resolving them can be found in section: Consistency check (Page 273).

### 3.5 Migrating projects from previous versions of SIMIT

Templates are not bound to any particular project; they can be migrated independently of any projects. You can find additional information on this in the section: Migrating templates (Page 212).

#### 3.5.2.3 Log file

When a project is migrated, a *migration.txt* log file is created in the folder %ProgramData%/Siemens/Automation/SIMIT/8.0/log. This file contains specific information about the places you might need to rework in the migrated project. After reading a file for migration into SIMIT 8, open this file in a text editor and search within it for the term *warning*.

### 3.5.3 Migrating charts and operating screens

#### 3.5.3.1 Overview

In SIMIT 8, there is no longer a distinction between charts and operating screens. This means that both charts and operating screens from SIMIT 5 are converted into charts. The migration takes place as part of a project migration. You can find additional information on this in the section: Step-by-step migration of projects (Page 190). Some specifics regarding the migration of charts and operating screens are explained below.

---

#### Note

After loading a project for migration in SIMIT, always start the consistency check. All warnings and errors that result from the check require corresponding post processing of the migrated project.

---

#### 3.5.3.2 Migrating charts with sheets

In SIMIT 5, a chart can consist of multiple sheets; in SIMIT 8, there are no sheets in charts. A chart with sheets is migrated to a corresponding number of charts. The name of the charts corresponds to the original chart name to which a sequence number is appended. For a chart called Chart with 3 sheets, three charts called *Chart*, *Chart2* and *Chart3* are generated.

#### 3.5.3.3 Migrating component names

Components are not renamed during migration, which means the component names remain unchanged. If this leads to ambiguous names in the migrated project, the consistency check will show the corresponding errors.



In SIMIT 5, components are identified by a name and an optional suffix. In SIMIT 8, this suffix is omitted. Component names are migrated according to the schema illustrated in the table "Migration of component names".

Table 3-1 Migrating component names

SIMIT 5	SIMIT 8
Name, no suffix	Name
Name and suffix	Name/suffix
No name, suffix	/suffix

In SIMIT 8, unique component names are mandatory. The consistency check will reveal any such ambiguities.

### 3.5.3.4 Migrating signal names

A component name and a connector name are concatenated using a slash '/' in the names of signals for connecting components and/or operator controls in SIMIT 5 according to *component name/connector name*. In SIMIT 8, a source and a name must always be specified for signals. The source is specified using the name of a component, a control or a coupling. The name is specified using the name of the connector on the component or the name of the signal in the coupling.

During migration the name of a SIMIT 5 signal is split in such a way that the part from the beginning up to the last slash is interpreted as the component name, which means as the source, and the remainder after the last slash is interpreted as the signal name.

### 3.5.3.5 Migrating data types

The data types *logical* and *double* in SIMIT 5 correspond to *binary* and *analog* in SIMIT 8. The *integer* data type is new in SIMIT 8. Some of the connectors of some components in the SIMIT 8 standard library therefore no longer have *analog* connector type; they now have the *integer* type. This holds true for all converter component types in the *Conv* directory and for the *Multiplexer* and *SimulationTime* component types of the standard library.

All components that are connected to status words or control words now have the new connection type *integer* at these connectors. This holds true for the drive components *PROFIDrive* and *SIMOCODEpro*. When these connections are directly connected to input and output connectors, migration is consistent because the corresponding coupling signals also have the *integer* data type. Inconsistencies can occur, if such connectors of drive components are not directly connected to input or output connectors.

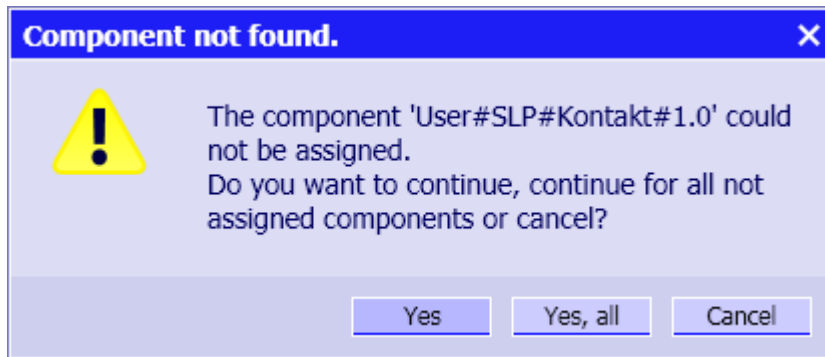
If migration leads to connections between components with inconsistent data types, these connections are deleted from the migrated project. For coupling signals that are mapped in SIMIT 8 to the data type *integer*, proceed as described in the following section: Migrating data types of coupling signals (Page 204).

### 3.5.3.6 Specifics when migrating components

#### Overview

During migration, components of types available in SIMIT 5 are mapped to components of types that are available in SIMIT 8. This mapping is provided for the SIMIT standard library, including some specifics as shown in the sections to follow.

If your SIMIT 5 project contains user-defined components that have no assignment in SIMIT 8, an error message will appear when the project is read into SIMIT 8 for migration.



You may then cancel the migration, ignore this case and continue or ignore all cases of this sort and continue.

#### Global connectors

SIMIT 8 does not distinguish between global input and output connectors (*Global Input*, *Global Output*); there is just one global connector *Connector*. During migration the global input and output connectors are mapped to this connector with appropriate connections.

#### Chart connectors

SIMIT 8 does not include the SIMIT 5 chart connectors *Diagram Input* and *Diagram Output*. During migration, they are replaced by global connectors. The parameter assignment of the chart connectors using *Connector name* and *Chart name* in SIMIT 5 is converted to a name for the global connector in SIMIT 8 according to the scheme shown in the table "Migration of chart connectors" table. *Diagram* in this context stands for the name of the chart to which the chart connector belongs.

Table 3-2 Migration of chart connectors

SIMIT 5	SIMIT 8
Diagram output (connector name, chart name)	Chart / chart name / connector name
Chart input (connector name, chart name)	Chart name/chart/connector name

## Characteristic

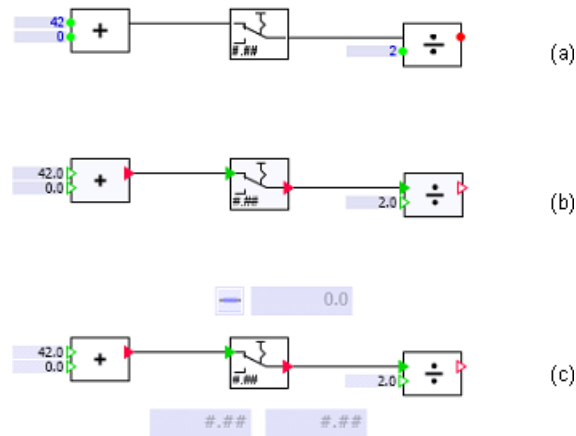
The characteristic component is transferred to SIMIT 8 during the migration without the configured interpolation points. To transfer the interpolation points, use SIMIT 5 to export the interpolation points to a .csv file, convert the .csv file into a tab-separated .txt file and import the .txt file into the corresponding component in SIMIT 8 after migration.

## Components in the Misc subfolder

In the SIMIT 5 standard library, the *Misc* subfolder contains a few component types that allow for operator controls in operating screens to be functionally integrated into charts.

Because operator controls can be directly connected as controls in SIMIT 8, such component types are no longer necessary. They therefore have no equivalent in the SIMIT 8 basic library, which means components of this type no longer exist in migrated projects. This holds true for components of type *Button*, *Display*, *Indicator*, *Input*, *Pushbutton*, *Slider* and *Stepper*. In the migrated project just place an appropriate control in your chart for any of these components, if necessary.

The case is similar for component types *ASwitch* and *BSwitch*. By way of example, the figure below shows the application of the *ASwitch* under (a). Because SIMIT 8 allows a signal connection to be terminated at any place using the *signal splitter* control, these component types are no longer necessary and thus no longer exist in the standard library. In order to retain this functionality in migrated projects, *ASwitch* and *BSwitch* are replaced by components that copy the input signal unchanged to the output as can be seen under (b) in the figure. If manual setting of the value is still required at this location in the migrated project, it can be implemented using additional controls as shown by way of example under (c) in the figure.



### 3.5.3.7 Migrating graphics and animation

#### Overview

When migrating a project, graphics and their parameter assignments are converted if possible. Animations need to be recreated on the graphics objects in the migrated project.

### Migrating rectangles and ellipses

Rectangles and ellipses are transferred along with all of their properties into the migrated project. There are no differences except when a rectangle in the SIMIT 5 project that is to be migrated has rounded corners. Rounding-off is not included in the migration.

---

#### Note

Rectangles and ellipses in SIMIT 5 are automatically converted into (Bezier) curves by a rotation. Therefore, after the migration, a rotated rectangle in SIMIT 5 will be a curve rather than a rotated rectangle.

---

### Migrating lines and curves

Lines and curves are transferred into the migrated project along with all of their properties.

### Migrating text

Text is transferred along with its font type and size into the migrated project. Distortions, which means horizontal and vertical scaling of a single-line text, are not transferred into the migrated project. The migration of text is illustrated in the figure below: The text lines on the left are converted to the texts on the right.

Text line, Arial 12 pt

Text line, Arial 12 pt

Text line, Times 24 pt

Text line, Times 24 pt

Text line, Times 24 pt

Text line, Times 24 pt

---

#### Note

Text that was rotated in SIMIT 5 by an angle that is not a multiple of 90° is not shown correctly in the migrated project.

---

### Operating symbol of components

The operating icons of components are not present in the migrated operating screens.

## Migrating animations

Graphic objects can be much more intuitively animated in SIMIT 8, because the action that is to be performed is interactively specified by the user. Animations can now be defined for each individual element within a group, which considerably expands the possibilities for animation. Animations in SIMIT 5 projects are not migrated to SIMIT 8. Animations must be recreated in a migrated project, if necessary.

### 3.5.3.8 Migrating operator controls

#### Overview

SIMIT 5 operator controls are replaced in SIMIT 8 by corresponding controls according to the table "Migration of operator controls". Despite the differing representation of operator controls and corresponding controls, their proportions are retained as far as possible during migration. The options for parameter assignment of controls are adjusted to their specified color scheme, and thus differ for operator controls. Parameters are retained as far as possible during the migration.

Table 3-3 Migrating operator controls

Operator control in SIMIT 5	Control in SIMIT 8
Bar graph display	Bar graph display
Digital display	Digital display
Binary display	Binary display
Speedometer	Analog display
Image display	Animation You can find more information on this in the section: Image display (Page 198).
Digital input	Digital input
Digital input with fixation	Digital input and signal splitter
Slider	Slider
Pushbutton	Pushbutton
Switch	Switch
Switch with fixation	Switch and signal splitter
Switch with image	Switch with image or Step switch with image
Pushbutton with image	Pushbutton with image

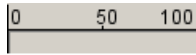
#### Note

In the SIMIT 5 project that is to be migrated, if you have linked controls with components that are not migrated, which means they are not contained in the migrated project, then the signal that is linked to this control is also not contained in the project. The consistency check displays such controls as *Control with unknown signal*.

The operator controls and their equivalent controls are contrasted below, and any special aspects of migrating an operator control are explained.

### Bar graph display

Operator control in SIMIT 5



Control in SIMIT 8



### Digital display

Operator control in SIMIT 5



Control in SIMIT 8



In controls, it is no longer possible to specify two signals from which a single numerical value is calculated. Only the *Display signal (low byte)* from the operator control is migrated. If required, you can replicate the calculation using standard components.

### Binary display

Operator control in SIMIT 5



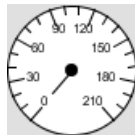
Control in SIMIT 8



If desired, the 3D effect of the Binary display from SIMIT 5 can be replicated using an animation. To do this, replace the control in the migrated project with an image alternation animation.

### Speedometer

Operator control in SIMIT 5



Control in SIMIT 8



### Image display

There is no equivalent control for the image display. During migration the image display is replaced by an animation without loss of functionality. An image display that is linked to a binary signal is replaced by an image alternation animation. An image sequence animation is used if the image display is linked to an analog signal.

### Digital input

Operator control in SIMIT 5



Control in SIMIT 8



## Digital input with override

Operator control in SIMIT 5



Controls in SIMIT 8



## Slider

Operator control in SIMIT 5



Control in SIMIT 8



The numerical value in the control is no longer displayed above the slider. If the numerical value is to be displayed, that can be configured in the control. To do this, set the *Show value* parameter in the view properties of the control.

### Note

If the slider is moved in SIMIT 5, the new value from the slider does not take effect in the simulation until the slider is released. With controls in SIMIT 8, even the intermediate values of the slider are effective in the simulation. Please verify whether this behavior is acceptable.

The behavior of a SIMIT 5 slider can be replicated in SIMIT 8 as follows: Select the *Show value* option in the slider view properties. A field then appears next to the slider, which displays the current value of the slider while the simulation is running. When the simulation is running you may also enter a value directly in this field, which will take effect in the simulation immediately after being entered.

## Pushbutton

Operator control in SIMIT 5



Control in SIMIT 8



## Switch

Operator control in SIMIT 5



Control in SIMIT 8

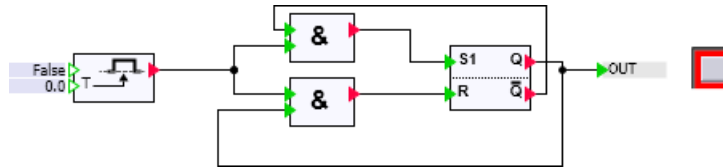


The switch in SIMIT 8 is represented as toggling. Its switch state therefore no longer needs to be represented using two colors. The color settings of the operator control consequently have no equivalent in the control.

If it is necessary to replicate the operator control color change, this can be implemented using a replacement switch as shown in the figure below. You can find an archived SIMIT 8 project on the SIMIT software CD in the folder *Sample Projects\Migration*, which contains this

3.5 Migrating projects from previous versions of SIMIT

replacement switch as a macro component. If required, you can copy this macro component to your migrated projects.



Switch with override

Operator control in SIMIT 5



Control in SIMIT 8



Switch

Operator control in SIMIT 5



Control in SIMIT 8



This operator control is replaced by a Switch control of the appropriate size.

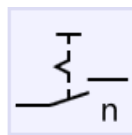
If you want the exact appearance of a SIMIT 5 switch, you can use a switch with image as shown in the figure below. The SIMIT 8 project on the SIMIT software CD in the folder *Sample Projects\Migration* also provides this control for use in your migrated SIMIT 8 projects.



Switch with Image#1		
General	Property	Value
Connector	Adapt to Image Size	<input checked="" type="checkbox"/>
<b>View</b>	Image (off)	S2_OFF ... X
	Image (on)	S2_ON ... X

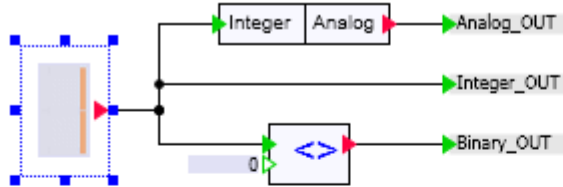
Switch with image

Control in SIMIT 8





When the operator control is linked to a binary signal, the *Switch with image* control is used in the migration. If it is linked to an analog signal, it is replaced with a *Step switch with image* control. If the operator control is linked to both an analog and a binary signal, no corresponding control can be assigned. If required, this behavior must be re-implemented in the migrated project as shown by way of example in the figure below. You will also find this replacement switch in the sample project *Migration.simarc* on the SIMIT software CD.



Stepping Switch with Image#1		
General	Property	Value
Connector	Switch-Over	Left-Right
View	Adapt to Image Size	<input type="checkbox"/>
	Images	<div style="border: 1px solid gray; padding: 2px;"> <span>...</span> <span>X</span> <span>↑</span> <span>↓</span> </div> 7seg1 7seg2 7seg3

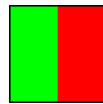
The areas in the image for toggling values via mouse click are pre-defined in the operator control, but they are configurable for the controls in SIMIT 8 as shown in the figure below.

#### Operator control in SIMIT 5

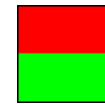


#### Control in SIMIT 8

Switch  
Left – Right



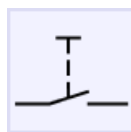
Switch  
Top – Bottom



Check whether the type of switchover in the migrated project meets your expectations and adjust it as needed.

#### Pushbutton with image

##### Control in SIMIT 8



### 3.5.3.9 Rule-based implicit connections

In SIMIT 5 projects, implicit connections can be created based on rules. The rules that are used are stored in the configuration files. This rule-based mechanism does not exist in SIMIT 8. If required, implicit connections must be created in the migrated project using the easy-to-use mechanism that exists in SIMIT 8.

## 3.5.4 Migrating couplings

Couplings are not migrated together with the project, but must be created from scratch in SIMIT 8. Therefore, create new couplings in your SIMIT 8 project and check the coupling signals for inconsistencies as shown in the figure below.

### 3.5.4.1 Migrating the names of coupling signals

Both symbolic names and absolute addresses can be used as coupling signal names in SIMIT 5. In SIMIT 8, coupling signals are specified using a source and a name. The coupling name is used for the source; the symbolic name or the absolute address is used for the signal name.

#### Adding the source name

When adding a coupling in SIMIT 8, the source name is automatically assigned to the coupling signals. When migrating charts and operating screens, the source name is missing from the coupling signals on the resulting charts. You can easily add the source names by using the *Find & Replace* function. Search for *Signal* and select the *Find and replace as regular expression* option.

Use the Find and Replace dialogs as shown in the figure below if you did not use a prefix in the coupling in SIMIT 5.

?	Hit	Replacement	
<input checked="" type="checkbox"/>	Q0.0	PLCSIM	Q0.0
<input checked="" type="checkbox"/>	I0.0	PLCSIM	I0.0

Refer to the figure below for the corresponding dialogs, if you did use a prefix in the coupling in SIMIT 5. In both examples *PLCSIM* was used as both the coupling name and the prefix.

**Find & Replace**

**Search** | **Replace**

▼ **Search Criteria**

Search for: Signal [v] [ ] PLCSIM(.\*  
 Search in: Full diagrams folder of project "Migration"  
 Replace with: [ ] PLCSIM [ ] \$1 [ Search]

**Options**

Include Subfolders  Case Sensitive Search and replace regular expression [v]

▼ **Search Results**

? Hit		Replacement	
<input checked="" type="checkbox"/>		PLCSIMQ0.0	PLCSIM Q0.0
<input checked="" type="checkbox"/>		PLCSIMIO.0	PLCSIM IO.0

## Symbolic and absolute addressing

In SIMIT 8, a coupling signal that was entered for a symbolic name can only be accessed using this symbolic name. If you used absolute addresses to access signals in SIMIT 5, even though the signals were assigned symbolic names in the coupling, in SIMIT 8 you must either delete the symbolic names in the coupling signal or change the access to these symbolic names.

---

### Note

If an input connector or output connector in SIMIT 8 is assigned a signal that cannot be found in any coupling, the consistency check will only classify this as a warning. It will therefore not prevent the simulation from being started.

To avoid problems, manually call the consistency check once after migration to actually see all of the warnings.

---

### 3.5.4.2 Migrating data types of coupling signals

In SIMIT 5, all non-binary coupling signals were treated as analog signals and were mapped to the data type double. In SIMIT 8, the non-binary coupling signals are further differentiated according to the types of signals in SIMATIC. The mapping to an analog or integer data type occurs as shown in the table below.

Table 3-4 Data types of the coupling signals

Signal type in SIMATIC	Data type in SIMIT 8
BOOL	Binary
BYTE	Integer
WORD	Integer, if the signal is <b>not</b> normalized Analog, if the signal is normalized
INT	Integer
DWORD	Integer
DINT	Integer
REAL	Analog

The effects of this enhancement are visible in migrated projects for coupling signals that are mapped to the Integer data type. On charts resulting from the migration of charts and operating screens, these coupling signals are usually connected to analog inputs and outputs of components and controls.

The consistency check will display such inconsistent signals as *Connector with signal of wrong type*. Resolve these inconsistencies by using appropriate converter components, for example, by connecting an *Analog2Integer* component in front of integer inputs and by connecting an *Integer2Analog* component after integer outputs.

### 3.5.4.3 Migrating data record communication

In SIMIT 8, data record communication via PROFIBUS DP has become more flexible and easier to use. In SIMIT 5, if data record signals were created in the coupling for this kind of communication, the special *Unit* connector establishes a component connection to the associated coupling in SIMIT 8.

This only affects components of type *SIWAREXU1* and *SIWAREXU2*. For details with respect to converting data record communication for components of these types, refer to the description of the basic library or to the online help for these two components.

## 3.5.5 Migrating macro components

### 3.5.5.1 Overview

Macro components from SIMIT 5 can be migrated to SIMIT 8 by being transferred to charts and then exported in the form of a project. The project can then be migrated to SIMIT 8. In SIMIT 8, the charts are then transferred back to macro components.

In order to correctly migrate charts in which macro components were used, the macro components must first be migrated to SIMIT 8 and placed in the global macros. These macro

components must be arranged in a hierarchy that exactly matches the folder structure in SIMIT 5.

### 3.5.5.2 Step-by-step migration of macro components

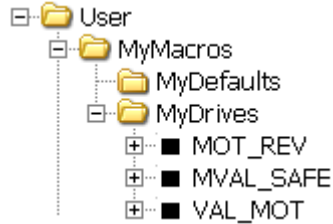
Please follow these steps to migrate your macro components:

- *Step 1:*

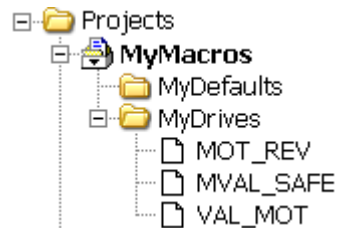
Create a new project in SIMIT 5, and create a new chart in this project for each macro component.

If you placed your macro components in the SIMIT 5 *User* section using a directory structure, create the charts using the same directory structure and assign the names of the macro components to the charts as shown by way of example in the figures below.

Macro components in the library

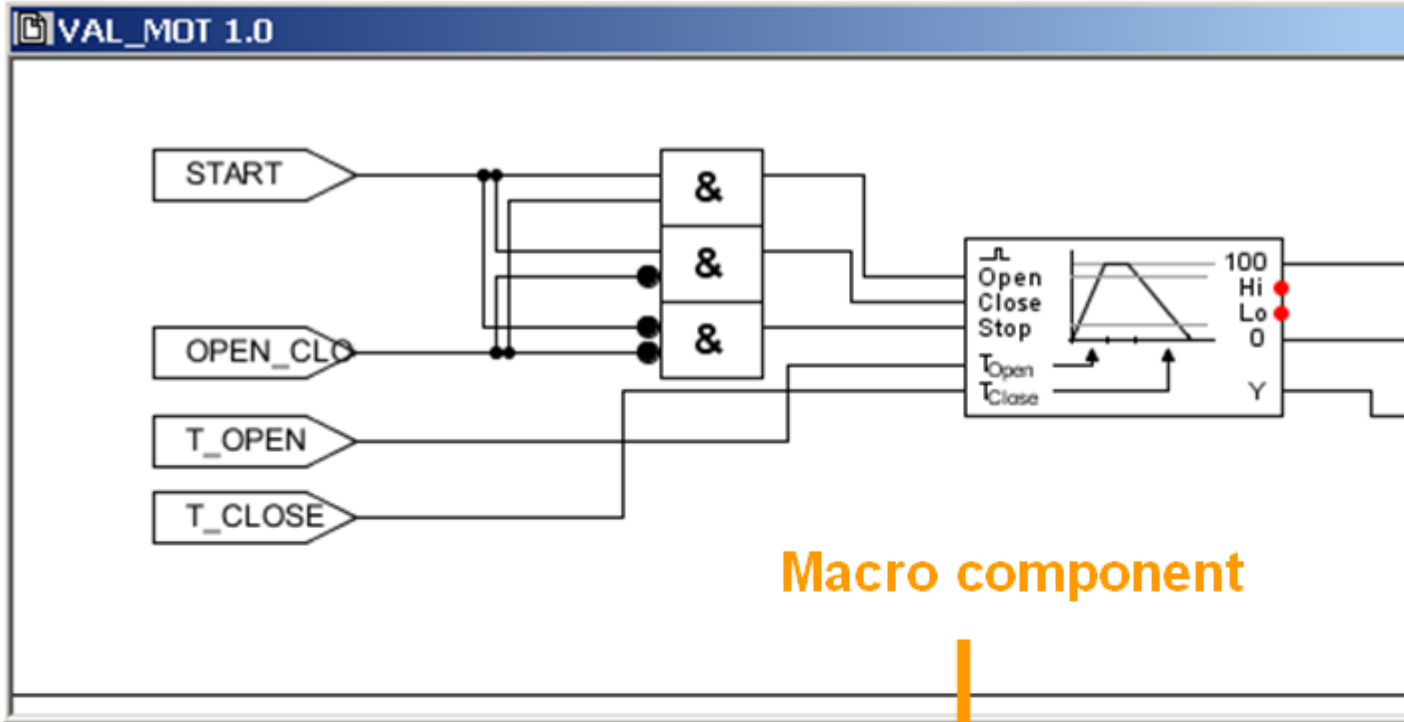


Project with charts for migrating the macro components



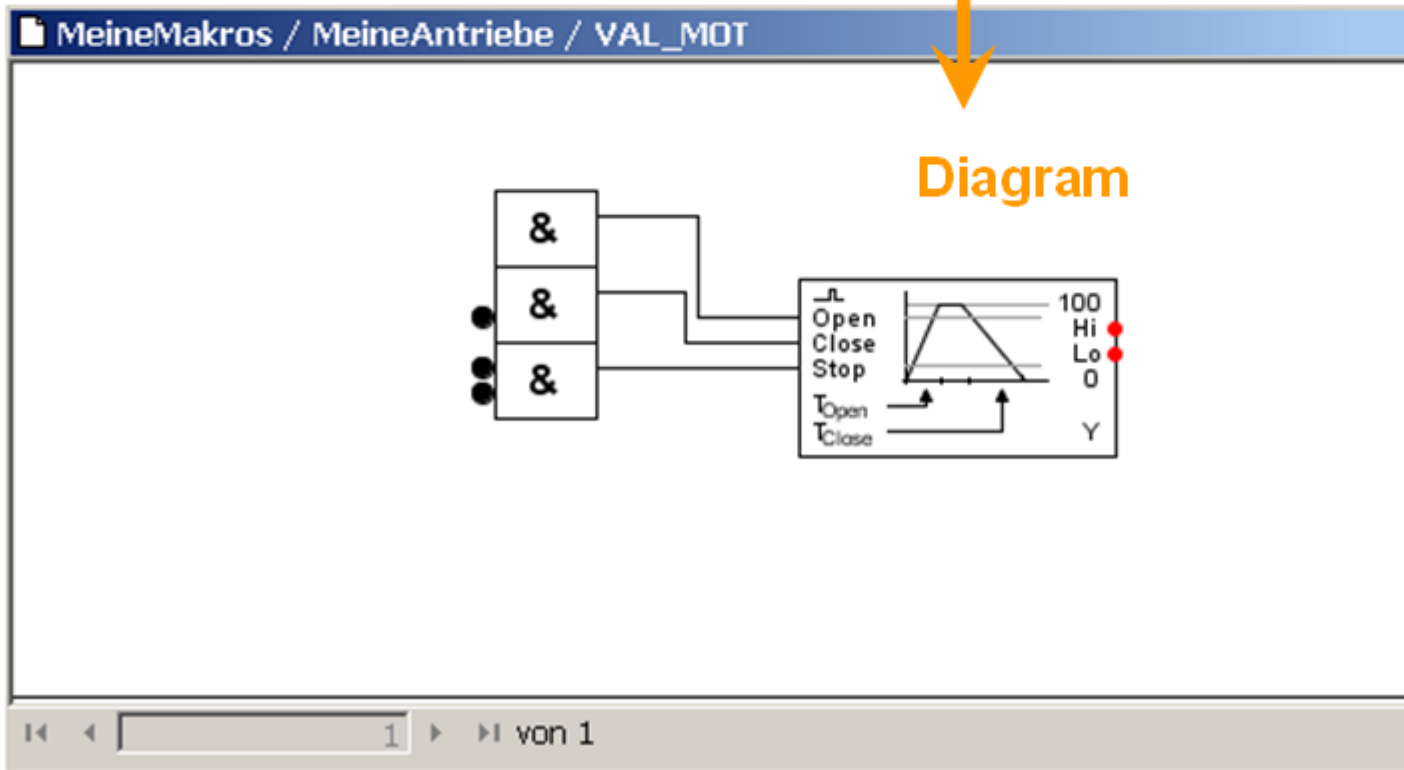
- *Step 2:*

From each macro component copy its content onto the corresponding chart. You will see that the macro connectors and therefore the connections with them are not transferred into the charts. By way of example, the figure below shows this procedure for a macro component (VAL\_MOT).



Macro component

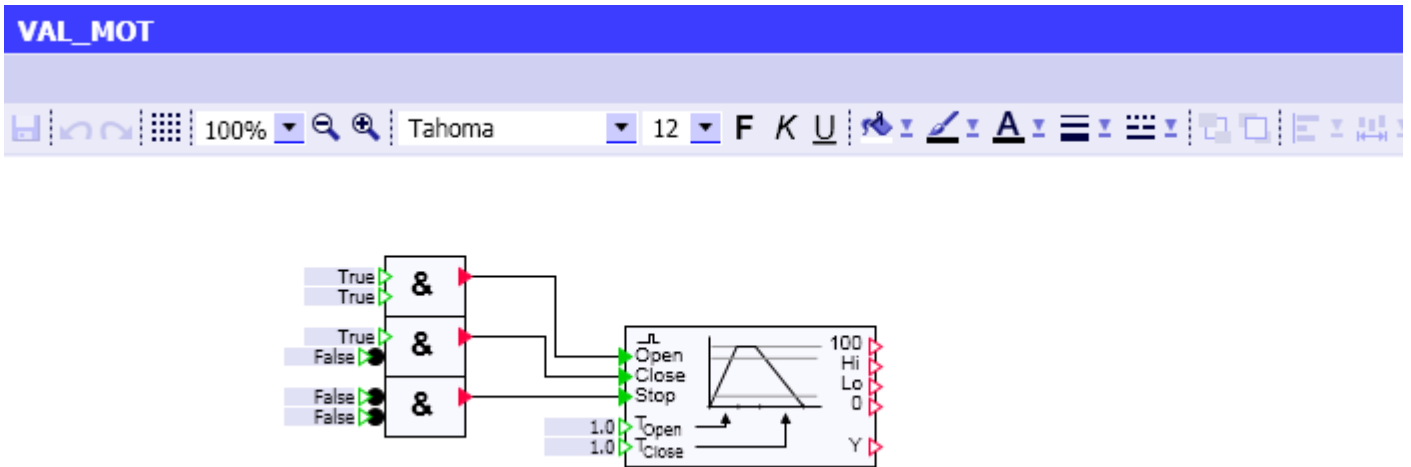
Copy



Diagram

3.5 Migrating projects from previous versions of SIMIT

- *Step 3:*  
Then export the SIMIT 5 project.
- *Step 4:*  
Load the exported project into SIMIT 8 using the menu command *Migrate project*. You can find additional information on this in the section: Step-by-step migration of projects (Page 190). The SIMIT 8 project now contains the migrated charts in the form of charts. The figure below shows the chart for the example macro component VAL\_MOT.



- *Step 5:*  
Under "Global macros" in the "User macros" area of the "Macros" task card, create the same folder hierarchy as the one in the user area of SIMIT 5. Create a new editable macro component for each chart using the same name as in SIMIT 5 and copy the content of the chart into the macro component that is open in the editor.
- *Step 6:*  
Edit each macro component by defining its connectors in the sidebars and by establishing connections to the connectors as in the original macro component. Use the same names for the connectors as in SIMIT 5.

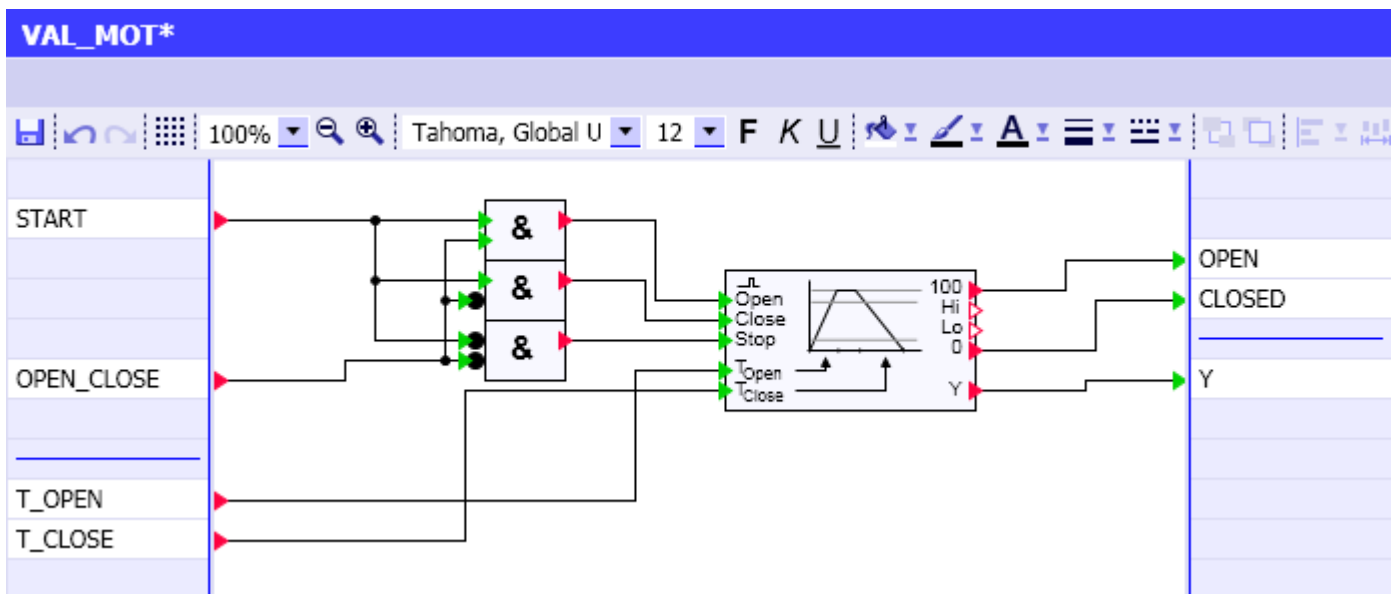
**Note**

In SIMIT 8, the connector names of a macro component cannot contain any special characters, for example the plus sign (+).

If special characters were used in the connector names of macro components in SIMIT 5, this will lead to incompatibilities when migrating projects that contain such macro components. Modify your macro components and the projects that use them accordingly in SIMIT 5 prior to migration by removing the special characters from the connector names.

When doing so, note that in SIMIT 8 the connector sequence on a macro component symbol is specified in the sidebars. In SIMIT 5, the connectors are arranged on the symbol in alphabetical order. You can also use dividing lines to offset the connectors from one another in SIMIT 8. The figure below shows the resulting example macro component VAL\_MOT.





Finally define the parameters for each macro component as required. By way of example, the figure below shows this for the parameter *Initial\_Value*. This parameter is defined with the name INITVAL as a parameter of the macro component.

MyMacros/MyDrives/VAL_MOT#6			
General	Name	Value	
Input	HI_Limit	95.0	
Output	Initial_Value	Closed <input type="button" value="v"/>	
Parameter	LO_Limit	5.0	
State			

If you used component inputs to assign parameters of a macro component in SIMIT 5, you must define these inputs using an *AConst*, *BConst* or *IConst* constant component as a parameter of the macro component in SIMIT 8. For details, refer to the standard library Manual.

Having migrated these macro components you can now migrate projects from SIMIT 5 that use these macro components.

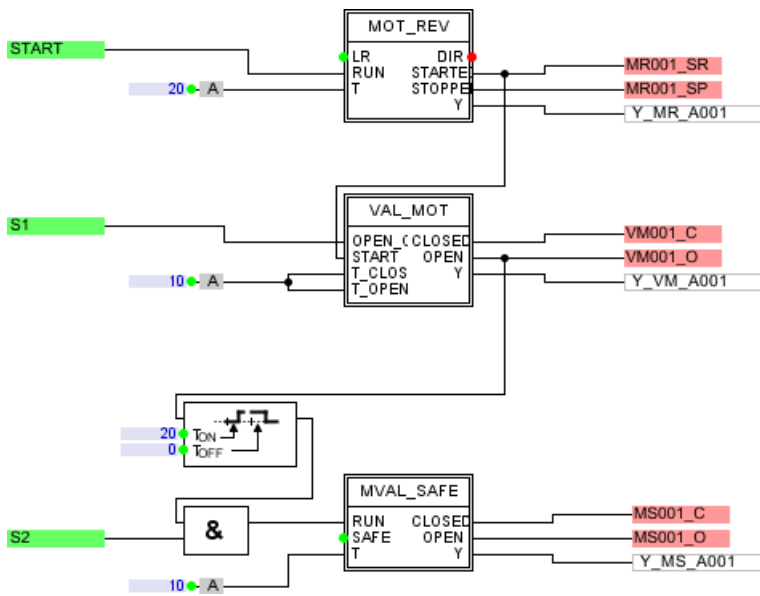
### 3.5.5.3 Migrating charts with macro components

In SIMIT 5, macro components that are used on charts are referenced using their library path, name and version. In order for it to also be possible to assign macro components in SIMIT 8 when migrating charts, macro components in SIMIT 8 need to be stored in the same directory structure as in SIMIT 5. You can find additional information on this in the section: Step-by-step migration of macro components (Page 206).

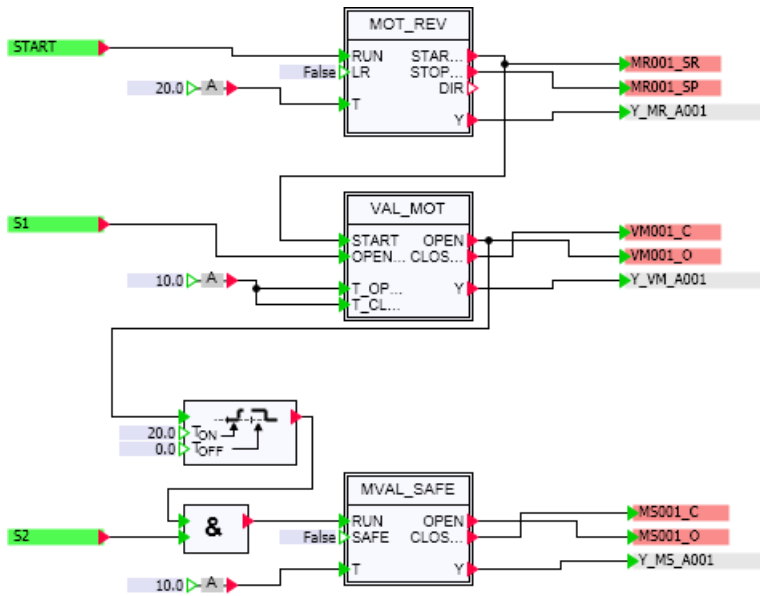
SIMIT 5 projects containing macro components that have been migrated in this way can then be migrated. You can find additional information on this in the section: Migrating SIMIT projects (Page 189).

The following figure shows a chart with macro components before migration:

3.5 Migrating projects from previous versions of SIMIT



The following figure shows a chart with macro components after migration:



Each macro component in the SIMIT 5 chart has been replaced with the corresponding macro component in SIMIT 8. As can be seen, the arrangement and width of the macro components are retained in the migration.

### 3.5.5.4 Problems arising during migration

#### Overview

The automatic migration of charts with macro components assumes that the macro components and their connectors can also be unambiguously assigned on the charts in SIMIT 8. Potential problem cases are described below.

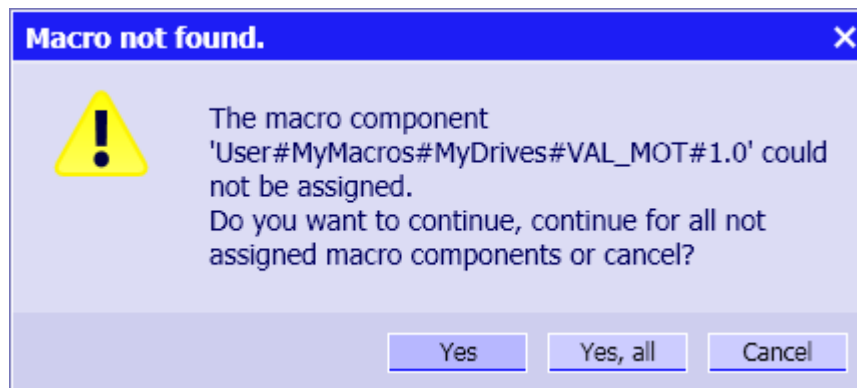
#### Incorrect connector name

If a macro component connector in SIMIT 8 has not been assigned the same name as in SIMIT 5, no connections can be established for this connector when migrating charts with this macro component. Hence there will be no connections on the migrated chart for this connector.

#### Missing macro component

If a macro component in SIMIT 8 has been assigned a different name than in SIMIT 5 or a macro component has not been placed in the same directory as in SIMIT 5, charts using this macro component cannot be completely migrated. The SIMIT 8 charts resulting from this migration will lack the corresponding macro component that was to be assigned, which means this macro component and its assigned connections are omitted. This is obviously also the case if a macro component used in a SIMIT 5 project does not exist at all in SIMIT 8 for the migration of projects.

Upon the first occurrence of this case when migrating charts, a dialog will appear as shown in the figure below. You may then cancel the migration, ignore this case and continue or ignore all cases of this sort and continue.



#### Versioned macros in SIMIT 5

In SIMIT 8, macro components are no longer versioned. Because macro components are assigned based on their name when migrating a project, this assignment can only be performed for a single version of a macro component.

In this way, all versions of a macro are automatically updated to the migrated version during the migration.

### 3.5 Migrating projects from previous versions of SIMIT

If you have used macro components on charts with the same name but different versions in your SIMIT 5 project and you want to maintain this distinction when migrating, you must first ensure that these macro components differ either in their name or their position within the directory hierarchy in SIMIT 5. To do so, assign different names to these macro components in SIMIT 5 or place them into different folders within the library; then adjust the charts by replacing the macro components. You can now migrate the macro components and charts as usual.

## 3.5.6 Migrating templates

### 3.5.6.1 Overview

Templates created with SIMIT 5 can be migrated to SIMIT 8 based on the migration of charts and operating screens.

### 3.5.6.2 Step-by-step migration of templates

To migrate your SIMIT 5 templates to SIMIT 8, follow these steps:

- *Step 1:*  
Create a project in SIMIT 5 and import the templates that are to be migrated into this project.
- *Step 2:*  
Export the SIMIT 5 project.

---

#### Note

Note that only SIMIT projects that were exported from SIMIT 5.4 **SP1** are migrated.

---

- *Step 3:*  
Load the exported project into SIMIT 8 using the menu command *Migrate project*. You can find additional information on this in the section: Step-by-step migration of projects (Page 190).  
The SIMIT 8 project now contains the migrated charts and operating screens in form of charts.  
Reworking the charts may be required.
- *Step 4:*  
Open the Project Manager and drag the charts into the *User Templates* section in the "Templates" task card.
- *Step 5:*  
Now open each template in the template editor to define placeholders, if required. If connectors for coupling signals were used on templates or if implicit connections were used, the migration does not provide a placeholder for the signal source. Enter appropriate placeholders here.  
Note in particular the information in the following section: Placeholders for enumeration parameters (Page 213)

### 3.5.6.3 Specifics when migrating templates

#### Overview

In addition to the specifics concerning migration of charts and operating screens, when migrating templates you must also take into account the specifics that are explained below.

#### Adapting profiles

Table profiles P1, P2 and P3 for table-driven instantiation of templates are provided unchanged in SIMIT 8 under the names in the table below.

Table 3-5 Names of the profiles

Profile in SIMIT 5	Profile in SIMIT 8
P1	Fixed placeholders "V1", "V2", etc.
P2	Placeholders defined in 1st row
P3	Fixed placeholders listed individually

Profiles P1ML, P2ML and P3ML are no longer available in SIMIT 8. Tables using any of these profiles need to be converted into a table format supported by SIMIT 8.

#### Placeholders for enumeration parameters

If parameters that are set using an enumeration are defined as placeholders in a template that is to be migrated, these parameters are set to the first value in the enumeration in the migrated template. In the migrated template, these parameters must be redefined as placeholders.

#### Placeholders for numerical parameters

If a parameter data type does not allow a text value, placeholders entered in that parameter cannot be migrated. In this case the parameter is set to its default value as defined in the component type and must be redefined as a placeholder.

#### Placeholders for input defaults

Placeholders that are entered in input defaults cannot be migrated, if the input data type does not allow text. In this case the inputs are set to the default value defined in the component type and must be redefined as placeholders.

### 3.5.7 Migrating scripts

In SIMIT 8, the Automatic Control Interface extension module has been fundamentally revised. Scripts that were created for SIMIT 5 need to be adapted accordingly for SIMIT 8. Please particularly note the following points:

- In SIMIT 8, there is no longer a `RESET` command.
- In SIMIT 8, there are no longer `ASSERT` and `CHECKBUFFER` commands.  
In SIMIT 8, the simulation is no longer logged and there is no longer the option for post-processing.  
Instead, the script mechanism in SIMIT 8 provides the `_printlog` command which a component can use to directly write to the log file of a running script. The component type needs to be adapted accordingly.
- Scripts in SIMIT 8 are type-safe. This means that some instructions that were still accepted in SIMIT 5 are no longer permitted in SIMIT 8:
  - A binary variable cannot be assigned a numerical value any more and vice versa.
  - The comparison operators `>`, `<`, `>=`, `<=` and the arithmetic operators `+`, `-`, `*`, `/` only operate on numerical values and not on binary values.
  - The comparison operators `==` and `!=` only operate on either numerical values or binary values; a binary value cannot be compared to a number.
  - The negation operator `!` only operates on binary values.
- Boolean constants are written as `"True"` or `"False"` and no longer as `"TRUE"` or `"FALSE"`.
- The decimal point always is a dot `'.'`.
- Variables within a `PRINTF` command are delimited with commas.

In order to keep the work involved in implementing a SIMIT 5 script as low as possible, some extensions to the new script syntax are accepted to maintain syntactical compatibility with SIMIT 5:

- When using an option dialog the mode may be omitted. In this case, a "Yes-No-Cancel" dialog is displayed like in SIMIT 5, which means

```
DIALOG "Ready?"
```

yields the same result as

```
DIALOG "Ready?" YESNOCANCEL
```

- An assignment can be written in the form

```
"Variable" = Value
```

but can also be written using the SET-VAR command:

```
SET-VAR "Variable" (Value)
```

Note that the value needs to be written in parentheses.

- Within an IF condition the THEN keyword may be omitted provided that the expression is written in parentheses, which means instead of

```
IF Expression THEN
```

```
    Block
```

```
ELSE ... or ENDIF
```

you may also write

```
IF (Expression)
```

```
    Block
```

```
ELSE ... or ENDIF
```

*3.5 Migrating projects from previous versions of SIMIT*



## Automatic model creation

SIMIT provides several automatic mechanisms to create and edit charts. The automation is based on files that have a format that can be read by SIMIT. The files contain the information needed to create or modify charts. This information is automatically converted into charts once the corresponding files are imported into SIMIT. The following automatisms are available:

- **Templates**  
Templates are patterns for charts in which placeholders are defined for parameters, defaults, etc. When a table is imported in which values are set for the placeholders, the charts are automatically generated from the templates. IEA files exported from PCS 7, CMT files and very general Excel spreadsheets are based on templates. You can find additional information on this in the section: Templates (Page 218).
- **Table import**  
Import of files in table format. These tables contain the replacements for the placeholders. Depending on their profile, additional information are available about the templates to be used and instances to be generated.  
You can find additional information on this in the section: The table import (Page 226).
- **IEA import**  
Import of a IEA file from PCS 7. An appropriate chart is created in SIMIT for each CFC based on a sample solution or a process tag type.  
You can find additional information on this in the section: The IEA import (Page 229)
- **CMT import**  
Import of control module types from PCS 7.  
You can find additional information on this in the section: The CMT import (Page 232).
- **Generic import**  
Charts cannot only be created using the graphical user interface but also based on a description that is written in XML syntax. This description is stored in a file. When importing this file the charts are automatically created as specified. Instantiation of templates can also be optionally performed via this import. You can find additional information on this in the section: Generic import (Page 235).
- **Create device level**  
Instead of an external file, this automatism uses information contained in the parameters of the components used in the simulation model. In this way, templates are automatically instantiated for certain types of components, for example, to create appropriate simulation of the control logic for process engineering devices.  
You can find additional information on this in the section: Generating the simulation of drives and sensors (Page 572).
- **Automatic parameter assignment**  
Parameters and input default values can be defined in a table and can be imported into the existing charts of the simulation project with the import of a table. You can find additional information on this in the section: Automatic parameter assignment (Page 243).

The automatisms are grouped under the menu command "Automatic modeling". When they are called from there, the new charts created on top-level project or, if you specify a folder hierarchy, relative to the "Charts" system folder.


## 4.1 Templates

If the automatisms are started from the shortcut menu of a chart subfolder, all new charts are created relative to this selected folder. Since the parameter import does not create new charts, it is not included in the shortcut menu.

## 4.1 Templates

You may place frequently recurring functions of a simulation model in a template. For example, you can prepare parts of a chart such that they can be used as a template for charts when creating new projects. You may use all of the same elements in templates that you use in charts: components, macro components, controls and graphics. In templates, unlike in charts, only placeholders are used for various elements of the components, controls, etc. contained in template. When instantiating a template, a chart is generated in which the placeholders are substituted by values, signal names, etc. SIMIT offers you two options for instantiating templates: You can manually drag-and-drop a template into your project or use a table-based import.

### 4.1.1 Creating templates

To create a template open the "Templates" task card in the Project Manager. Use  **New Template** to open the templates editor and create a new template. To open a template for editing just double-click the template in the "Templates" task card. Components, controls, macro components, graphics as well as templates, and projects are available in the template editor as resources, i.e. as task cards. This means you can edit templates like charts.

Templates in the *Basic* pane can only be opened for viewing in the templates editor, which means you cannot edit these templates. To indicate this the template is opened in the editor with a white title bar. If you wish to edit a basic template, first copy it to the *User templates* pane.

The difference between templates and charts is that placeholders are used in templates. To understand this more clearly, just create a new chart, for example, in the *global templates* folder. Then open the sample project *Elevator* from the "Projects" task card and the *Main drive* chart within this project. Now copy the entire content of this chart into the open template.

**Project navigation**

- Project
- Simulation
- Elevator-01
  - Project manager
  - Couplings
    - New coupling
    - PLCSIM
  - Charts
    - New chart
    - Function
    - Operating
    - Monitoring
    - Scripting
    - Material
    - Snapshots
    - Find & replace
    - Consistency check
    - Start

**MainDrive\***


100% | Tahoma | 12 | B / U

PLCSIM V0, PLCSIM V1, OR gate, PLCSIM Direction, Run, Dir, Speed, 30.0, 0.5, 0.5, Low speed, Up, Down, Y, Connector#1

**DriveP1#1**

General	Name	Value/signal
Input	Run	OR#1 OUT
Output	Dir	NOTc#1 OUT
Parameter	Speed	PLCSIM V1
State	SpeedLow	123 30.0
	T_Up	123 0.5
	T_Down	123 0.5
	MAN	123 False
	Setpoint	123 0.0

Portal view | MainDrive

If the properties of the PLCSIM V0 output connector you will now see that the input fields for both source (coupling) and signal name contain an additional symbol  that is used to define placeholders.

PLCSIM V0	
General	
Property	Value
Signal	PLCSIM  V0 
Display Gateway Name	<input checked="" type="checkbox"/>

Now convert the signal into a placeholder by clicking both symbols, and then enter *Coupling and Slow*, for example, as the placeholders.

4.1 Templates



Now set corresponding placeholders for the other input and output signals and for the global connector. Always use the same placeholder *Coupling* for input and output signals.

Using the placeholder symbol allows you to define the following as placeholders: inputs and parameters of components and macro components, signals in controls and animations, and component names. Close the template editor when finished.

For some controls you can also define the type, default setting and scaling as placeholders. The list in the table below shows which properties can be used as placeholders for controls.

Table 4-1 Placeholders for controls

Control	Name	Time slice	Type	Default setting	Scaling	Connector	Objective
Pushbutton	X	X	X				
Pushbutton with image	X	X	X				
Switch	X	X	X	X			
Switch with image	X	X	X	X			
Step switch	X	X		X			
Step switch with image	X	X		X			
Digital input	X	X		X			
Slider	X	X		X	X	X	
Bar graph display	X	X			X	X	
Binary display	X	X				X	
Analog display	X	X				X	
Digital display	X	X				X	
Action	X						X

In some cases the type and default setting of controls take the form of language-dependent lists in the user interface. If you define these properties as placeholders, you must specify the position of the corresponding term in the list when making the replacement.

Table 4-2 Parameter value for the NC/NO contact type

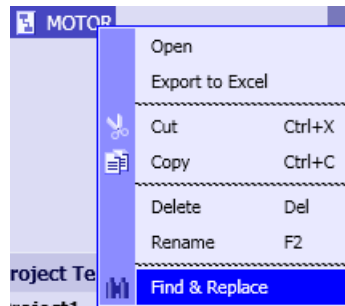
Type	Value
NC contact	0
NO contact	1

Table 4-3 Parameter value for the Off/On default setting

Default setting	Value
Off	0
On	1

#### 4.1.1.1 Find and replace in templates

You can replace components and macro components within a template using the find and replace feature. For this purpose, the shortcut menu for a template includes the command *Find & replace*.



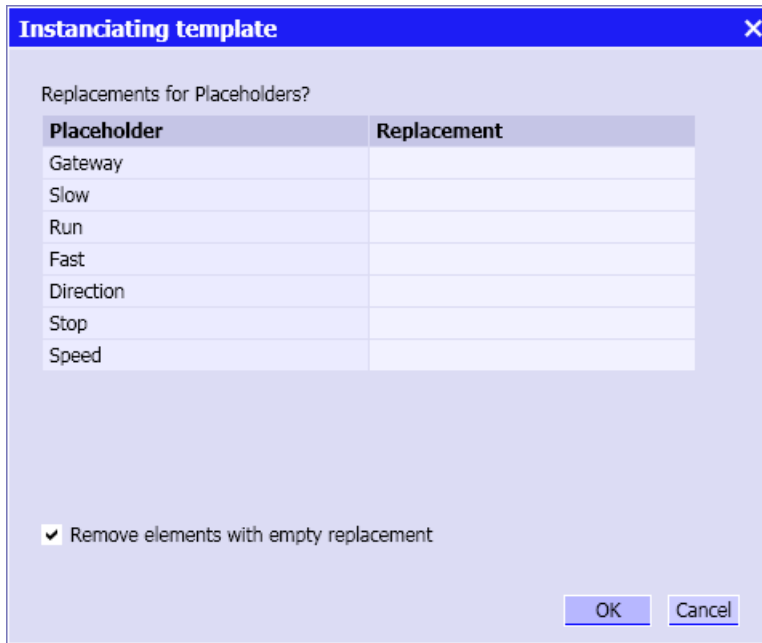
This command opens the Find & replace editor with a preset focus on the template.

#### 4.1.2 Instantiating templates by entering replacements

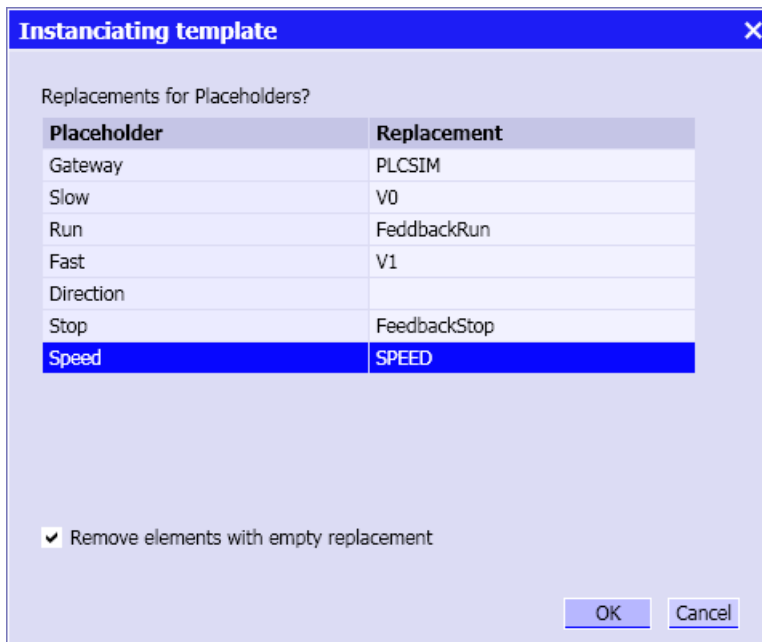
Templates are instantiated by replacing placeholders with appropriate values. Components that have a name not defined as a placeholder, receive the name they have obtained on the template, possibly supplemented by a sequential number, which makes the component name unique.

In the Project Manager, you may instantiate a template by dragging and dropping it from the task card into a project folder. Open the project manager and select the Templates task card. If you now drag the template that was created above, a dialog appears as shown in the figure below. A two-column list is displayed, which shows all of the placeholders that are contained in this template. Please enter the replacement for each placeholder in the right column.

4.1 Templates

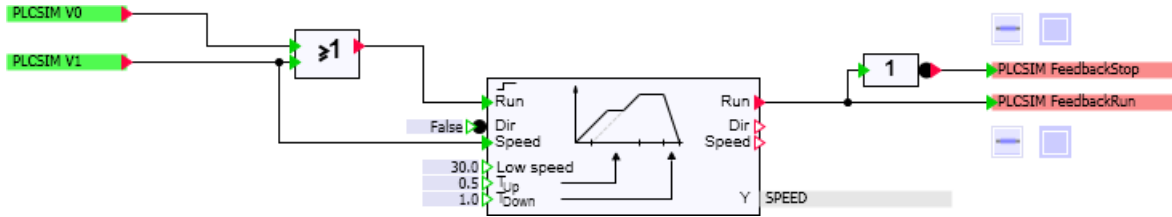


If the "Remove elements with empty replacement" option is checked, all elements are removed where a placeholder does not have a replacement in the table. Global connectors and I/O connectors, for example, are deleted if no signal is specified. By way of example, this is the case for the Direction placeholder in the figure below. You may thus create your templates in such a way that they cover several use cases, and may then instantiate the template for a specific use case by not replacing the placeholders that apply to the other use cases.



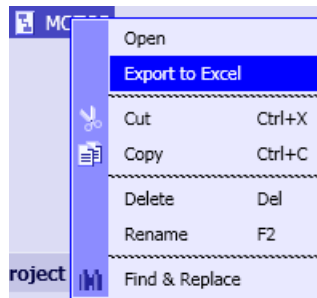
After confirming the dialog you will see a new chart with the same name as the template in the Project Manager. When you open the chart, you will see the instantiated simulation. It is identical to the simulation of the main drive, except for the missing output signal *Direction*. The output connector was not instantiated because it lacked a replacement. If you had not checked

"Remove elements with empty replacement", a connector without a signal value would have been generated in the instance for this signal, which means an empty connector would have been generated in the instance.



### 4.1.3 Creating tables from a template

If you want to create a table for a template so that this template can then be instantiated multiple times, SIMIT allows you to create a table which already contains a title row matching the template. In the shortcut menu for the template select the command *Export to Excel*.



The next steps will depend on whether or not Microsoft Excel is installed on your computer.

- Microsoft Excel installed**  
 A new workbook is directly opened in Excel, with the entries needed for the selected template already entered in the top row on the first sheet. Complete the table and save it in a location of your choice for use in the table import.
- Microsoft Excel not installed**  
 An Excel file in \*.xlsx format is created, with the entries needed for the selected template already entered in the top row on the first sheet. Save this file in a location of your choice for editing in Excel at a later date – possibly on another computer.

The first three columns of the generated tables contain the terms HIERARCHY, TEMPLATE and CHART. The subsequent columns contain the names of the placeholders used in the template. The name of the template is already entered in the second row, but you must enter all other data yourself.

	A	B	C	D	E
1	HIERARCHY	TEMPLATE	CHART	GATEWAY	Plan_ChName
2		MOTOR			
3					
4					

---

**Note**

Note that when this table is imported, the profile is set to *Placeholders defined in 1st row*.

---

### 4.1.4 Defining a folder hierarchy for templates

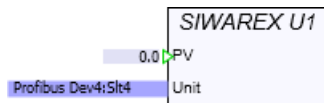
You can specify the folder in which to search for a template in your tables. For the *TEMPLATE* placeholder enter the complete folder hierarchy as shown in the *Templates* task card. Use the backslash '\' as the separator. To refer to a template that is not located in a subfolder, enter the name of the template preceded by a backslash.

If the template name is entered without separators, all folders and subfolders are searched for a template of this name as before. In this case, make sure that there are not two templates with the same name in different subfolders. In principle the three panes of the *Templates* task card are searched in the following order:

1. Project templates
2. User templates
3. Basic templates

### 4.1.5 Addressing a module via the I/O address

The identity of a module on PROFIBUS or PROFINET is determined by the slave or device number and the slot number. Accordingly, both values must be entered in the Unit connector if it is used to identify a device, such as a balance of type SIWAREX-U.



If such devices are used on templates, this information is not always available. Instead the module start address may be known. Therefore it is now permissible to identify a module on templates by its start address, for example, "IW512" or "AW512" instead of "Dev4:Sl4".

When the simulation is started it is necessary to know the address with the slave or device number and the slot number. Therefore when a template is instantiated, a start address is automatically converted to the address with the slave or device number and slot number. If no coupling with the appropriate address exists when the template is instantiated, the automatic conversion can also be carried out later by means of the *Extra | Assign coupling signals* command.

### 4.1.6 Indirect addressing

In the input and output connectors, an additional item of information can be included on templates, which allows access to signals located relative to a known address. This expression has the form  $[\$+n : RW]$ , where



$n$  is the offset to be added in bytes,

$R$  denotes an input or output, which means E, I, A or Q, and

$W$  denotes the data width, which means B, W or D.

So, for example, if the address of a control word EW560 is known, the associated status word AW560 can be specified as follows:

```
IW560[$+0:QW]
```

Or, if the control word EW560 has the symbolic name STW1 in the coupling, it can also be specified in the form STW1[\$+0:QW]

If the coupling already exists when the templates are instantiated, an attempt is made to replace the expression with the corresponding signal. If not, this expression is retained even after the signal to which it relates is replaced. For example, if the control word STW1 exists in a PLCSIM coupling with its address IW560 but no associated status word, then the connector

```
{ $Gateway } { $STW1 } [ $+0:QW ]
```

is replaced as follows:

```
PLCSIM IW560 [ $+0:QW ]
```

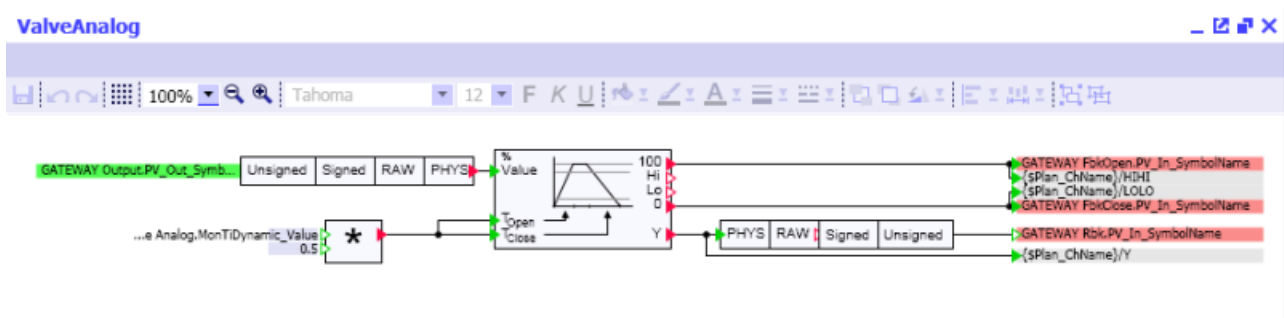
However, you can also start the complete replacement subsequently at any time by calling the *Extra | Assign coupling signals* function.

### Note

Indirect addressing is only supported by the following couplings: PROFIBUS DP, PROFINET IO, PRODAVE and PLCSIM.

## 4.1.7 Opening basic templates in the editor

Templates in the *Basic templates* pane can now be opened directly via the shortcut menu in the template editor. However, basic templates opened in this way can only be viewed, not edited. To indicate this the template is opened in the editor with a white title bar.



If you wish to edit a basic template, first copy it to the *User templates* pane.

## 4.2 The table import

Templates can be instantiated using tables. These tables contain the replacements for the placeholders and, depending on their profile, additional information about the templates to be used and the instances to be created. 3 profiles are available for the format of the tables:

- Profile 1 Profile with fixed placeholders "V1", "V2", etc.
- Profile 2 with placeholders in the first row
- Profile 3 with individually listed placeholders

In all tables, each row defines the instantiation of one template.

The following table formats are supported:

- **\*.xls**  
The Microsoft Excel format for Office 97 – 2003. Only the first sheet of the workbook is taken into account. This file format is only available if Microsoft Excel is installed on your computer.
- **\*.xlsx**  
The current Microsoft Excel format. Only the first sheet of the workbook is taken into account. This file format is available even if Microsoft Excel is not installed on your computer.
- **\*.txt**  
Tab-separated list in text format.

---

### Note

If you still want to use \*.csv files from projects in earlier versions of SIMIT, you must convert them to this tab-separated \*.txt format or to the \*.xls or \*.xlsx format.

---

### Note

When importing an .xls or .xlsx file that is still open in Excel, it is the last saved version of this file that is imported and not the currently open version.

---

### Importing to SIMIT

There are 3 ways to perform a table import into SIMIT:

- In the portal view, Select "Automatic model creation" > "Table import"
- From the Project view, select the menu command "Automatic model creation > Table import".
- Select "Automatic model creation" > "Table import" from the shortcut menu of a chart folder.

## Table profile

### Profile with fixed placeholders

A table with this profile cannot contain any header rows:

	A	B	C	D	E	F
1	TemplateA	Folder1	ChartA	3.0	I5.1	
2	TemplateB	Folder2	ChartB	1.0		Q4.2

The columns in the table are pre-defined:

- Column A defines the template to be used
- Column B defines the folder to be created within the project
- Column C defines the name of the chart to be instantiated
- Columns D through W: Placeholders *V1* through *V20*

Consequently, only the twenty placeholders *V1* through *V20* may be used in the templates. The folder can be specified as a folder hierarchy. The chart is instantiated within the specified folder.

#### Profile with placeholders in the first row

In this profile, the column headers define the meaning of each column.

	A	B	C	D	E	F
1	HIERARCHY	TEMPLATE	CHART	P1	P2	P3
2	H1	T1	C1	1.0	E0.0	
3	H2	T2	C2	3.0		A1.0

The first three columns have a pre-defined meaning and their headings are also pre-defined:

- Column A (*HIERARCHY*) defines the folder hierarchy to be created in the project
- Column B (*TEMPLATE*) defines the template to be used
- Column C (*CHART*) defines the name of the chart to be instantiated

Starting with column D, the table may contain any number of additional columns with replacements for the placeholders. The placeholder is given in the header of each column. In this table profile, there are no restrictions in the number of placeholders.

---

#### Note

Templates (*TEMPLATE*) may be specified with the full folder hierarchy. Use the backslash '\' as the separator. To refer to a template that is not located in a subfolder, enter the name of the template preceded by a backslash.

---

If the template name is entered without separators, all folders and subfolders are searched for a template with this name. Therefore, it is not permitted for two templates with the same name to exist in different subfolders. The three panes of the *Templates* task card are searched in the following order:

1. *Project templates*
2. *User templates*
3. *Basic templates*

#### Profile with placeholders listed individually

A table with this profile must not contain any header rows.

## 4.2 The table import

	A	B	C	D	E	F	G
1	TemplateA	FolderA	ChartA	I1	I0.0	Q1	Q0.0
2	TemplateB	FolderB	ChartB	P1	5.0	T1	Motor Off

The columns in the table are pre-defined:

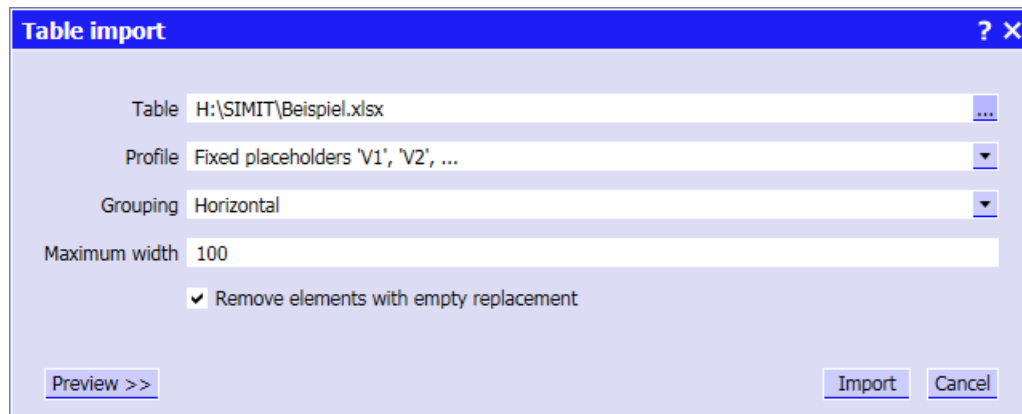
- Column A defines the template to be used
- Column B defines the folder to be created within the project
- Column C defines the name of the chart to be instantiated

The next 40 columns may contain a maximum of 20 replacements. For each replacement, one column contains the placeholder and the next column contains the replacement.

Consequently, only the 20 placeholders can be used in the templates. The placeholder names may be freely chosen. The folder can be specified as a folder hierarchy. The chart is instantiated within the specified folder.

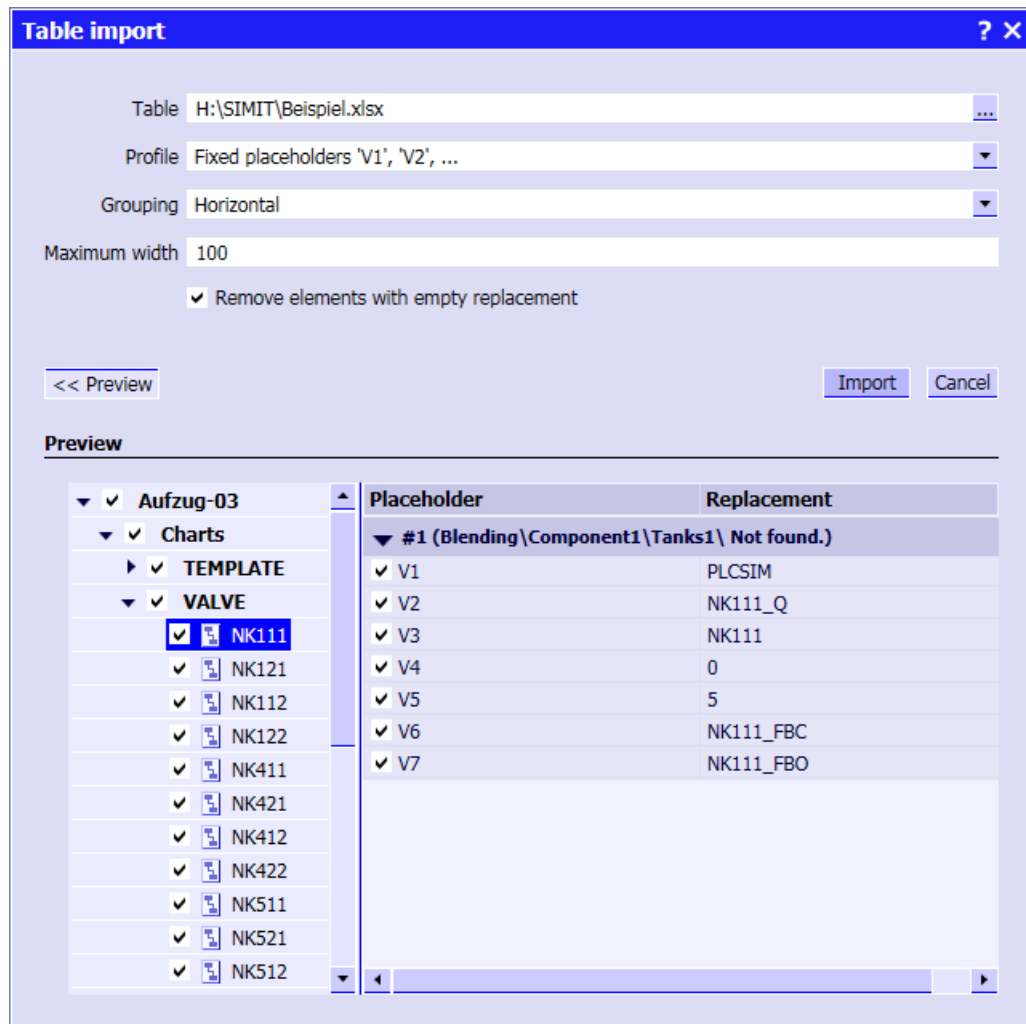
### Example of instantiation using a table

On the SIMIT software CD in the folder *Examples >SMD*, you will find an example of a table with profile 2 (placeholders defined in the first row). To instantiate the template according to table, select the menu command "Automatic model creation > Table import" from the project view. The following dialog box opens:



Select the file *Sample.txt* and the profile "Placeholders defined in 1st row" as the table. If according to the table several instances of a template are to be instantiated in a single chart, you must specify how they are to be grouped and also how much space is available on the chart. If the instances are to be arranged in a *horizontal* sequence, you must specify the *maximum width* of the chart. For instances that are to be arranged in a *vertical* sequence, you must specify the *maximum height* of the chart.

If you now use the *Preview>>* button to open the preview, on the left-hand side of the preview you will see the folder hierarchy and charts that are to be instantiated. For each chart, you can see and verify the list of replacements according to the table in the right half of the preview before the charts are actually instantiated. To open the list, select the corresponding chart. You can now decide whether to accept all information from the table or whether to deselect certain replacements.



If a chart is deselected in the left tree view, it is not instantiated. By selecting a folder, you can also select or deselect multiple charts within it. You can tell from the check box whether a chart or folder is selected () or deselected () or whether at least one check box in one of the subfolders is deselected () .

In the right table, you can also select and deselect individual replacements. Deselecting a replacement has the same effect as if the replacement were not present in the table.

## 4.3 The IEA import

If you use PCS 7 and work with process tag types or models, you can create a suitable chart in SIMIT for any CFC that is based on a process tag type or a model.

### 4.3 The IEA import

There are several templates supplied with SIMIT that match the process tag types from the PCS 7 libraries. You can find suitable templates in the directories of the basic templates:

- *PCS 7 Library* contains templates compatible with the PCS 7 standard library
- *PCS 7 AP Library* contains templates compatible with the PCS 7 V7.1 AP Library,
- *PCS 7 AP Library V80* contains templates compatible with the PCS 7 V8.0 AP Library.

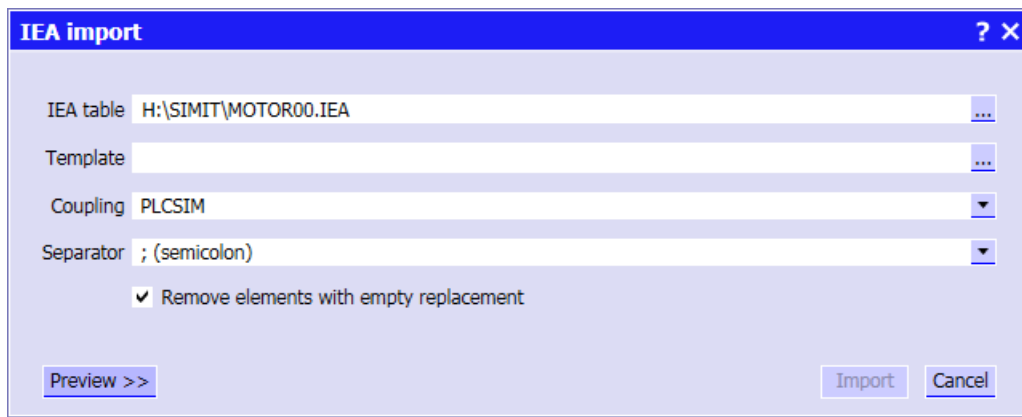
If you have used one of the standard templates from a PCS 7 library in your project, you can use these templates and create a matching simulation. The procedure is explained below using the example of a motor drive (MOTOR). On the SIMIT software CD, you will find two example IEA files from a PCS 7 project in the *\_Samples\SMD* folder.

#### Importing to SIMIT

There are 3 ways to perform an IEA import into SIMIT:

- In the portal view, Select "Automatic model creation" > "IEA import"
- From the Project view, select the menu command "Automatic model creation > IEA import".
- From the shortcut menu of chart folder, select "Automatic model creation > IEA import".

The following dialog box opens:

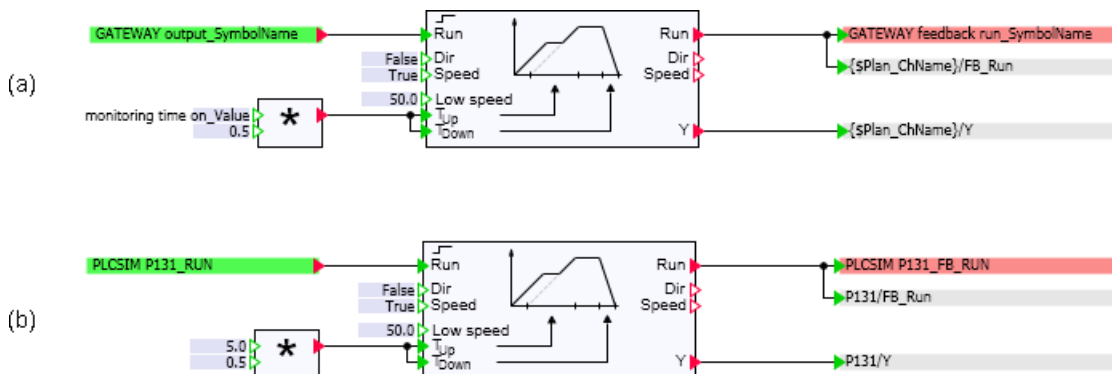


Provide a name for the coupling to be used for I/O signals.

In the preview, you will see that a chart is instantiated for each CFC. This chart has the same name as the CFC and is placed in a folder hierarchy that is identical to the folder hierarchy of the PCS 7 project (see figure below). You can decide whether to copy all information from the IEA file or whether to deselect certain replacements.



The following figure shows the template under (a) and the resulting instance under (b).



Then import the second file *VALVE00\_Exp.IEA* matching the VALVE template. This will create additional charts in the SIMIT project, within the folder hierarchy that was created during the first import.

When using modified or user-defined process tag types in your PCS 7 project, you can of course use these IEA files along with matching SIMIT templates. Just create matching

templates in the *User templates* folder, for example, or copy suitable basic templates there and modify the copies.

## 4.4 The CMT import

CMT stands for "Control Module Type" in PCS 7. Control module types are used to create CFC templates and they are provided with variable components. These CFC templates are then copied and individualized by instantiation. PCS 7 creates an exportable XML file for this, which includes all control module types as well as all instances in the project, their variable replacements and versions.

### Export from PCS 7

This functionality is used in PCS 7 for bulk data engineering or for exchanging configuration information with other configuration tools, such as COMOS.

---

### Note

The "XML transfer" function required by PCS 7 can be optionally installed during the SIMIT installation on the PCS 7 system. After that, the export can be performed directly from the PCS 7 project in the SIMATIC Manager (from the technological view) using the shortcut menu command "Export XML".

The "XML transfer" can be installed independent of SIMIT, if PCS 7 is installed on a computer other than the one on which SIMIT is installed.

---

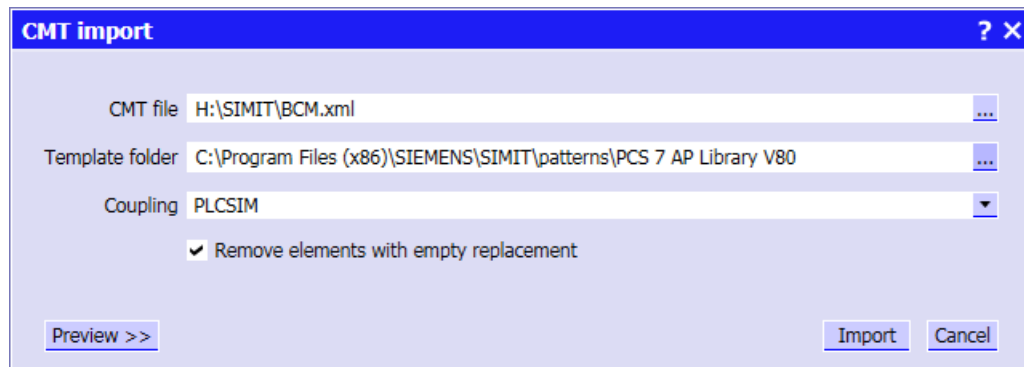
The exported XML file can now be imported into SIMIT to generate and test the instantiated control module types in the simulation.

### Importing to SIMIT

There are 3 options to perform a CMT import into SIMIT:

- In the portal view, Select "Automatic model creation" > "CMT import"
- From the Project view, select the menu command "Automatic model creation > CMT import".
- From the shortcut menu of chart folder, select "Automatic model creation > CMT import".

The "CMT Import" dialog box opens.





After the templates are instantiated, the charts can no longer be identified as "template instances" unlike in PCS 7. Therefore, it is not possible to subsequently switch versions or retain customizations. Synchronization is not possible.

If the project changes in PCS 7, the XML file must be exported again and imported once more into SIMIT. In this import, all the existing instances can be replaced with new charts. Individual customizations that were made after the last CMT import to the charts already instantiated are lost in the process.

### Template creation

The following options are available in the Template Editor to define variables and placeholders in the templates:

- A variable can be assigned to the value of a property field. For this, the "{\$}" symbol to the right of it must be enabled. {\$} means that the entire field content is interpreted as a variable name.
- If more than one variable or fixed and variable components are to be combined, you can also enter a variable name in the respective field using the notation {\$ variable name}.

This enables you to specify replacements from the XML file in the instance of a template.

### Versions and multiple connections in CMTs

The mapping of versions of a CMT is realized in SIMIT as follows:

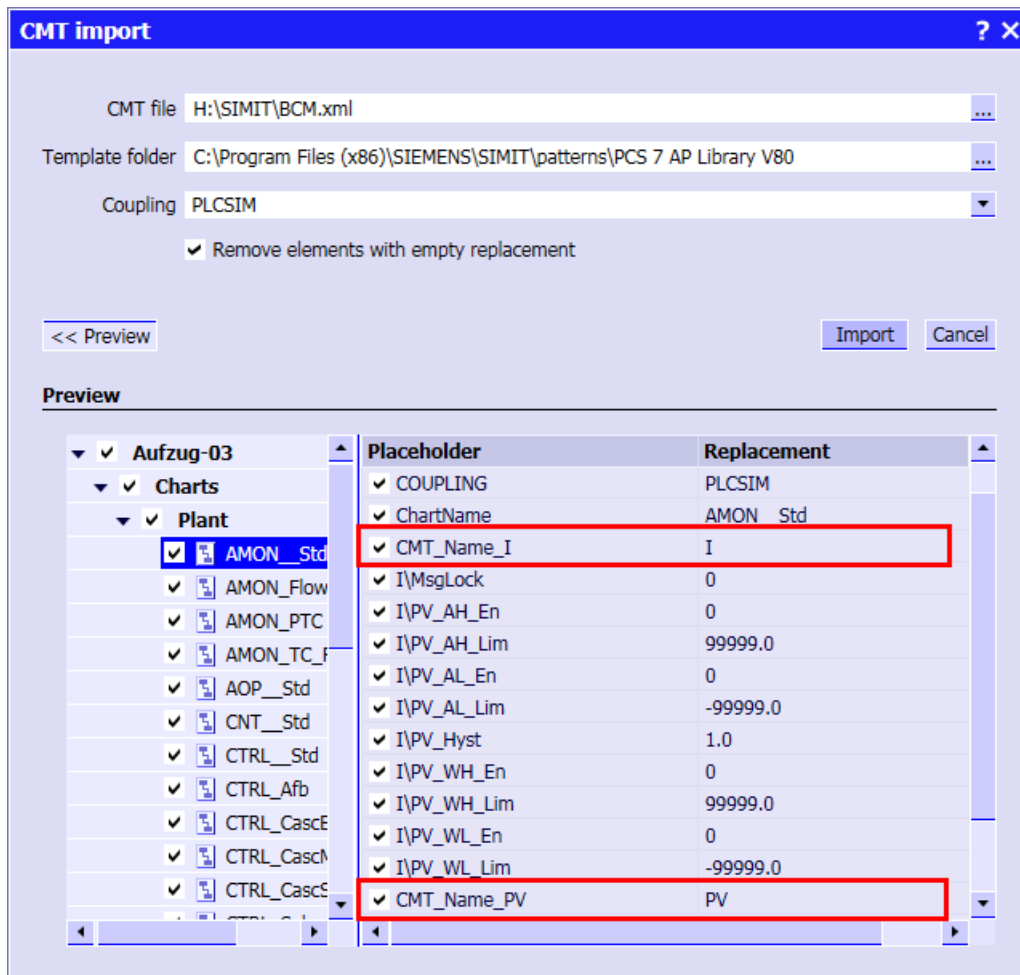
- The "Remove elements with empty replacement" check box is selected.
- The variable "CMT\_Name\_x" is only in the list of replacements available when this variant is present in the CMT.  
This applies to all variables and parameters derived from this CMT.

---

#### Note

The "x" in "CMT\_Name\_x" can be seen as the CMT name, and is "I" or "PV" in the figure below.

---



If a component is to be integrated in the simulation model in SIMIT with a dependency to an optional CMT in PCS 7, the name of the component must refer to this variable, for example.

If the CMT is not in the XML file, its variables, parameters and signals are also absent and all components that contain one of these replacements are automatically omitted in SIMIT as well. This makes it possible to reflect the version concept of the CMTs in SIMIT.

Multiple interconnections to CMTs in PCS 7 can be implemented in SIMIT with connectors. Because these versions can be used only as an alternative in PCS 7, the mechanism of the versions described above can be used here in SIMIT to interconnect multiple outputs from components in the template to one input.

These components are assigned to the versions and their outputs are linked to the input of another component with connectors. If one of the two CMTs is now omitted in the instance, the corresponding component is also omitted in SIMIT and the connector remains unconnected on a chart. Such connectors do not exist for SIMIT which means there is no double assignment that would otherwise lead to a consistency error.

## 4.5 Generic import

### 4.5.1 Importing the XML file

Importing an XML file is called a "Generic import". The XML file contains a description of charts based on an XML syntax. This allows existing templates to be instantiated, but also custom charts to be created without recourse to templates.

---

#### Note

In order to validate the XML file for correct syntax, run it through a validation with appropriate tools before importing.

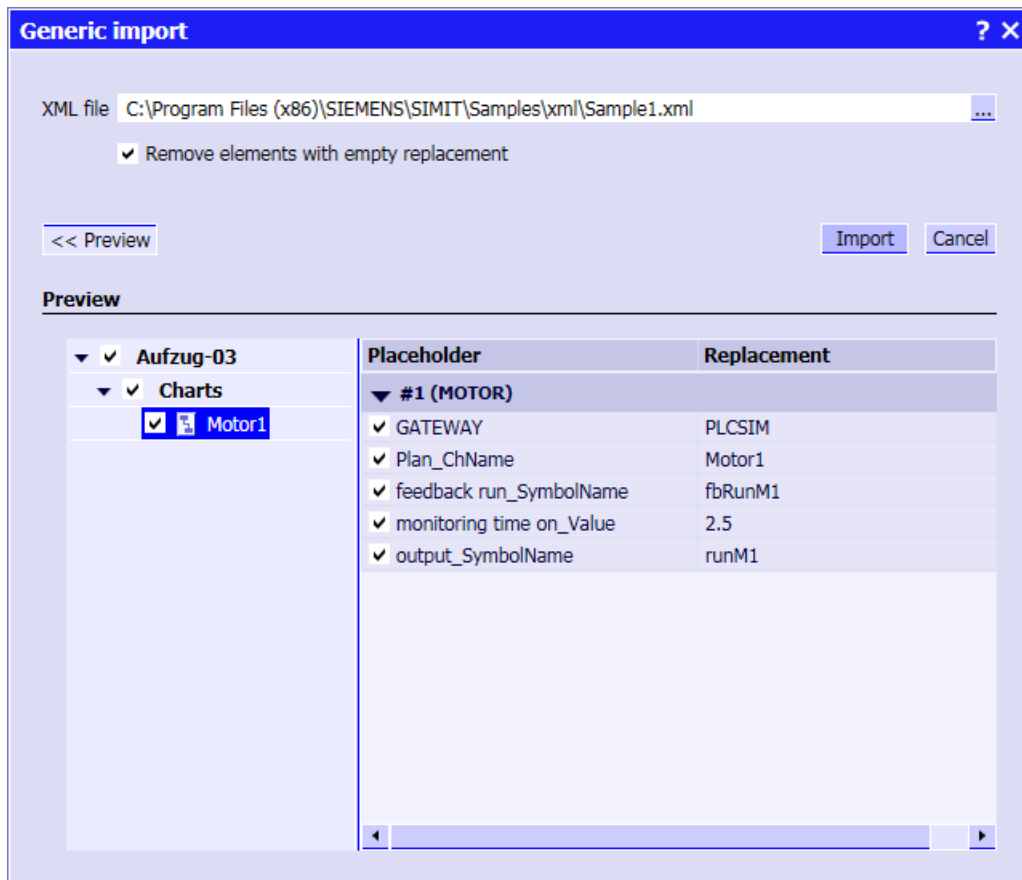
---

#### Importing to SIMIT

There are 3 ways to perform a generic import into SIMIT:

- In the portal view, Select "Automatic model creation" > "Generic import"
- From the project view, select the menu command "Automatic model creation > Generic import".
- From the shortcut menu of chart folder, select "Automatic model creation > Generic import".

The following dialog box opens:



Select the XML file to import and click the "Import" button.

## 4.5.2 The syntax

The file for import must be provided as a text file in XML format and must correspond to the following Document Type Definition:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE GENERIC [
<!ELEMENT GENERIC (FOLDER | DIAGRAM | TEMPLATE | BUILDUP)*>
<!ATTLIST GENERIC
  VERSION CDATA #REQUIRED>
<!ELEMENT FOLDER (FOLDER | DIAGRAM | TEMPLATE | BUILDUP)*>
<!ATTLIST FOLDER
  NAME CDATA #REQUIRED>
<!ELEMENT BUILDUP (TEMPLATE+)>
<!ATTLIST BUILDUP
  INSTANCE CDATA #REQUIRED
  ALIGNMENT (HOR|VER) "HOR"
  WIDTH CDATA #IMPLIED
  HEIGHT CDATA #IMPLIED>
<!ELEMENT TEMPLATE (SUBST*)>
<!ATTLIST TEMPLATE
```

```

        NAME CDATA #REQUIRED
        INSTANCE CDATA #IMPLIED>
<!ELEMENT SUBST EMPTY>
<!ATTLIST SUBST
        NAME CDATA #REQUIRED
        VALUE CDATA #REQUIRED>
<!ELEMENT DIAGRAM (COMP)*>
<!ATTLIST DIAGRAM
        NAME CDATA #REQUIRED
        WIDTH CDATA #IMPLIED
        HEIGHT CDATA #IMPLIED>
<!ELEMENT COMP (POS+, TRANSFORM?, PROP*, DEFAULT*, PORT*)>
<!ATTLIST COMP
        UID CDATA #REQUIRED
        NAME CDATA #REQUIRED
        REF CDATA #IMPLIED
        CYCLE CDATA #IMPLIED>
<!ELEMENT TRANSFORM EMPTY>
<!ATTLIST TRANSFORM
        SCALEX CDATA #IMPLIED
        SCALEY CDATA #IMPLIED
        ROTATION CDATA #IMPLIED>
<!ELEMENT POS EMPTY>
<!ATTLIST POS
        X CDATA #REQUIRED
        Y CDATA #REQUIRED>
<!ELEMENT PROP EMPTY>
<!ATTLIST PROP
        NAME CDATA #REQUIRED
        VALUE CDATA #REQUIRED>
<!ELEMENT PORT (POS?, CONNECTION*)>
<!ATTLIST PORT
        NAME CDATA #REQUIRED>
<!ELEMENT DEFAULT EMPTY>
<!ATTLIST DEFAULT
        NAME CDATA #REQUIRED
        VALUE CDATA #REQUIRED>
<!ELEMENT CONNECTION EMPTY>
<!ATTLIST CONNECTION
        TYPE (LINE|IMPLICIT) "LINE"
        SOURCE CDATA #REQUIRED
        NAME CDATA #REQUIRED>
]>

```

This definition can be found in the *Generic.dtd* file in the *\_Samples/XML* folder on your SIMIT installation CD.

## 4.5 Generic import

A valid XML file starts with the element **GENERIC**. The version number 8.0 should be entered as the attribute for this element.

Element	Attribute	Description
GENERIC		<i>Folder</i>
	VERSION	In the document form described here, "8.0" must be entered as the version.

The generic import interface can also be used to create an arbitrarily nested folder hierarchy. The specified folder hierarchy in the import file is relative to the folder whose shortcut menu you have selected to perform the import. This is the "Charts" system folder when the "Automatic model creation" menu is used. If the import file contains no **FOLDER** elements, the templates are created directly in the selected folder.

Element	Attribute	Description
FOLDER		<i>Generic import interface</i>
	NAME	The name of a folder to be created. A deeper folder hierarchy can be achieved by nesting several <b>FOLDER</b> tags.

You can choose to instantiate individual templates (*TEMPLATE*) or combine several templates in a chart (*BUILDUP*). In the template enter the name of the template (*NAME*) and the name of the chart to be created (*INSTANCE*) as attributes.

Element	Attribute	Description
TEMPLATE		<i>Template</i>
	NAME	Template name
	INSTANCE	Name of the chart to be created

If you want to combine several templates to create a single chart, the template elements must be enclosed by a *BUILDUP* element. In this case the name of the chart to be created should be entered in the *BUILDUP* element and not in each individual template element:

Element	Attribute	Description
BUILDUP		<i>Grouped templates</i>
	INSTANCE	Name of the chart to be created
	ALIGNMENT	HOR: Horizontal alignment (side by side) VER: Vertical alignment (one below the other)
	WIDTH	Maximum width of the chart in pixels (for horizontal alignment)
	HEIGHT	Maximum height of the chart in pixels (for vertical alignment)

Within a template element any number of replacements for placeholders can be specified using the *SUBST* element.

Element	Attribute	Description
SUBST		<i>Substitution</i>
	NAME	Name of the placeholder
	VALUE	Value to be replaced

Charts can also be created without using templates. To do this, use the *DIAGRAM* element:

Element	Attribute	Description
DIAGRAM		<i>Chart</i>
	NAME	Name of the chart to be created
	WIDTH	Width of the chart to be created in pixels
	HEIGHT	Height of the chart to be created in pixels

Components on a chart are described with the *COMP* element:

Element	Attribute	Description
COMP		<i>Component</i>
	UID	The unique identifier of the component type
	NAME	The instance name of the component
	REF	A freely selectable reference name for the component
	CYCLE	The time slice to which this component is to be assigned (1 to 8).

The component type is identified by means of the unique *UID*. For the component type to be instantiated, it must be present in a library, which means it must exist in the *Components* task card.

You also need to specify the name (*NAME*) to be given to the component on the chart. If this name already exists when it is imported into the project, SIMIT automatically adds a sequential number to make it unique.

A reference name (*REF*) only needs to be specified if the name of the component is not unique within the import file and a connection to this component needs to be specified. This can happen with connectors, for example, which occur repeatedly with the same name.

The time slice to which this component is to be assigned can be specified with the *CYCLE* element. If this attribute is not specified, a time slice value of 2 is entered.

The position of a component is specified by the *POS* element and its attributes X and Y. The top left corner of the component is specified. The zero point of the chart is in the top left corner of the chart. Positions must be positive and located within the dimensions of the chart:

Element	Attribute	Description
POS		<i>Position</i>
	X	The X position of the component in pixels
	Y	The Y position of the component in pixels

A component can also be scaled and rotated with the *TRANSFORM* element:

Element	Attribute	Description
TRANSFORM		<i>Transformation (scaling, rotation)</i>
	SCALEX	Scaling in X direction (Default: 1).
	SCALEY	Scaling in Y direction (Default: 1).
	ROTATION	Angle of rotation in degrees by which the component is to be rotated counterclockwise. The center of rotation is the geometric center of the component.

## 4.5 Generic import

The component parameters can be specified by means of the *PROP* element and its *NAME* attributes:

Element	Attribute	Description
PROP		<i>Parameter</i>
	NAME	Parameter name
	VALUE	Parameter value

Component connectors can be joined together. In some cases this takes place automatically if they are superimposed on the chart and have the same connection type. In other cases the connection has to be specified in the import file. Start by specifying the name of the connector using the *PORT* element:

Element	Attribute	Description
PORT		<i>Component connector</i>
	NAME	Connector name

The connection is then described by the *CONNECTION* element:

Element	Attribute	Description
CONNECTION		<i>Connection</i>
	TYPE	LINE: Connection using signal line IMPLICIT: Implicit connection
	SOURCE	The name of the component with which the connection is to be established. If the component has the REF attribute, its value must be used here.
	NAME	The name of the connector with which the connection is to be established.

The connection of directional signals should always be defined starting from the input, as the input can only be connected to exactly one output of another component. Therefore the connector to be specified in the connection definition is always the output of a component. As topological connections are directionless, with this type of connection it makes no difference which of the two connectors to be connected is specified as the connector in the connection definition.

The *SOURCE* attribute contains the name component or coupling that is to be connected. Similarly the *NAME* attribute contains the name of the connector or signal. For connectors of *IMPLICIT* type, the *SOURCE* attribute must be assigned the name of the component; reference names (*REF*) are not allowed here for implicit connections.

Default connector settings are defined by means of the *DEFAULT* element and its *NAME* and *VALUE* attributes.

Element	Attribute	Description
DEFAULT		<i>Default</i>
	NAME	Connector name
	VALUE	Default

If no default settings are specified, the values defined in the component type apply for the connectors.



In accordance with the usual XML conventions, comments are also permitted in the import file:  
 <!-- Comment -->

## 4.5.3 Examples

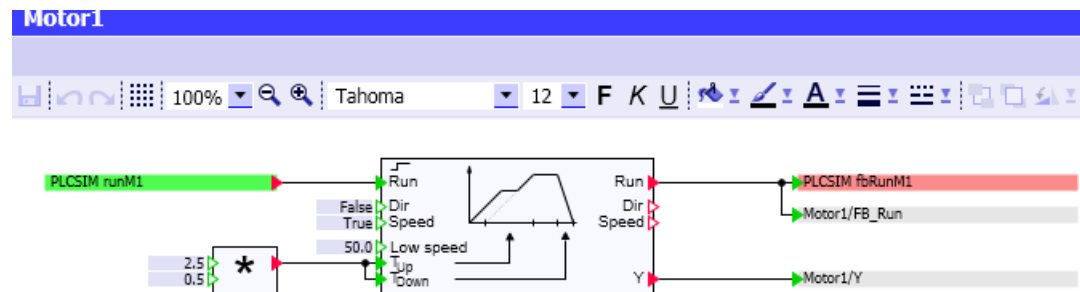
### 4.5.3.1 Example of a single template instantiation

These examples can be found in the *\_Sample/XML* folder on your SIMIT installation CD.

XML file:

```
<GENERIC VERSION="7.1">
  <TEMPLATE NAME="MOTOR" INSTANCE="Motor1">
    <SUBST NAME="GATEWAY" VALUE="PLCSIM"/>
    <SUBST NAME="Plan_ChName" VALUE="Motor1"/>
    <SUBST NAME="feedback run_SymbolName" VALUE="fbRunM1"/>
    <SUBST NAME="monitoring time on_Value" VALUE="2.5"/>
    <SUBST NAME="output_SymbolName" VALUE="runM1"/>
  </TEMPLATE>
</GENERIC>
```

Chart:



### 4.5.3.2 Example of a grouped template instantiation

These examples can be found in the *\_Sample/XML* folder on your SIMIT installation CD.

XML file:

```
<GENERIC VERSION="7.1">
  <BUILDUP INSTANCE="Motor1" DIR="VER" HEIGHT="1500">
    <TEMPLATE NAME="MOTOR">
      <SUBST NAME="GATEWAY" VALUE="PLCSIM"/>
      <SUBST NAME="Plan_ChName" VALUE="Motor1"/>
      <SUBST NAME="feedback run_SymbolName" VALUE="fbRunM1"/>
      <SUBST NAME="monitoring time on_Value" VALUE="2.5"/>
      <SUBST NAME="output_SymbolName" VALUE="runM1"/>
    </TEMPLATE>
    <TEMPLATE NAME="MOTOR">
      <SUBST NAME="GATEWAY" VALUE="PLCSIM"/>
      <SUBST NAME="Plan_ChName" VALUE="Motor2"/>
      <SUBST NAME="feedback run_SymbolName" VALUE="fbRunM2"/>
    </TEMPLATE>
  </BUILDUP>
</GENERIC>
```

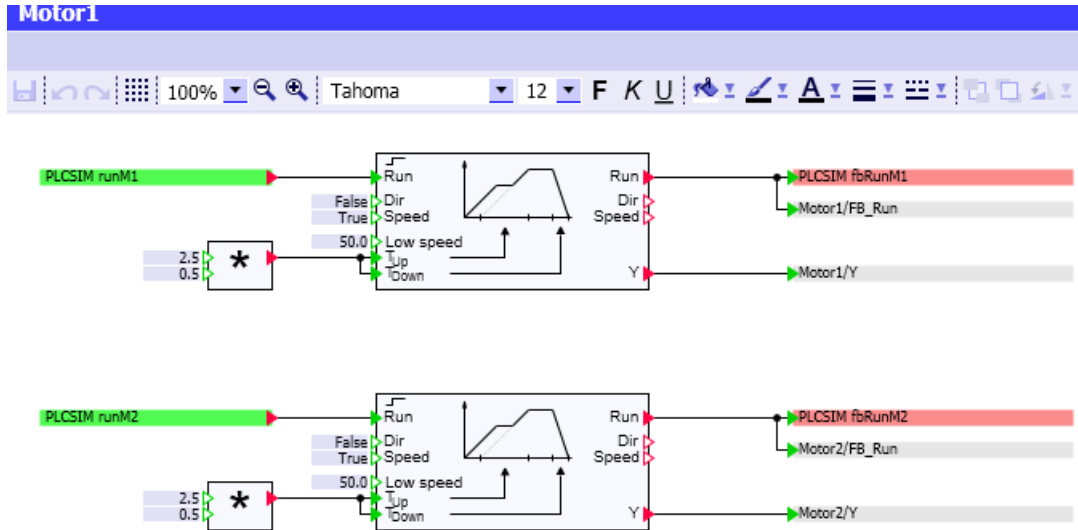
## 4.5 Generic import

```

        <SUBST NAME="monitoring time on_Value" VALUE="2.5"/>
        <SUBST NAME="output_SymbolName" VALUE="runM2"/>
    </TEMPLATE>
</BUILDUP>
</GENERIC>

```

Chart:



## 4.5.3.3 Example of chart creation

These examples can be found in the *\_Sample/XML* folder on your SIMIT installation CD.

XML file:

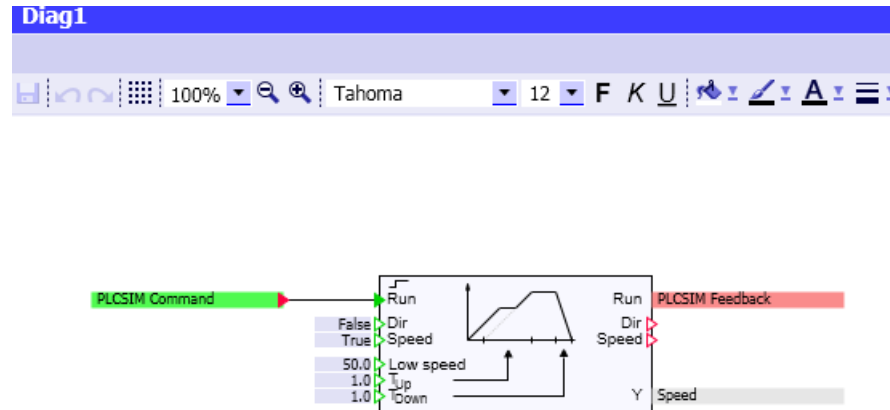
```

<GENERIC VERSION="7.1">
  <FOLDER NAME="Motors">
    <DIAGRAM NAME="Diag1">
      <COMP ID="f_000hsn_20cyz1u8" NAME="Drive1">
        <POS X="230" Y="100"/>
        <PORT NAME="Run">
          <CONNECTION SOURCE="PLCSIM/Command" NAME="Y"/>
        </PORT>
      </COMP>
      <COMP ID="f_000hsn_1zisl8r" NAME="PLCSIM/Feedback">
        <POS X="400" Y="110"/>
        <TRANSFORM SCALEX="2"/>
      </COMP>
      <COMP ID="f_000hsn_1zislnd3" NAME="PLCSIM/Command">
        <POS X="50" Y="110"/>
        <TRANSFORM SCALEX="2"/>
      </COMP>
      <COMP ID="f_000hsn_21b4dv0t" NAME="Speed">
        <POS X="400" Y="170"/>
      </COMP>
    </DIAGRAM>
  </FOLDER>

```

&lt;/GENERIC&gt;

Chart:



## 4.6 Automatic parameter assignment

### Introduction

With automatic parameter assignment, you can automatically copy values for the following variables from tables and insert them in existing charts:

- Parameters (except for parameters of the "dimension", "characteristic" and "text" type)
- Default input settings
- Default control settings (corresponds to input *X* of the control)

### Table format

The table for automatic parameter assignment must have the following 3 columns:

- The first column contains the name of the component to be modified
- The second column contains the name of the connector or parameter
- The third column contains the value to be applied

Each row of the table describes the replacement for a parameter. The figure below shows an example of a table opened in Excel.

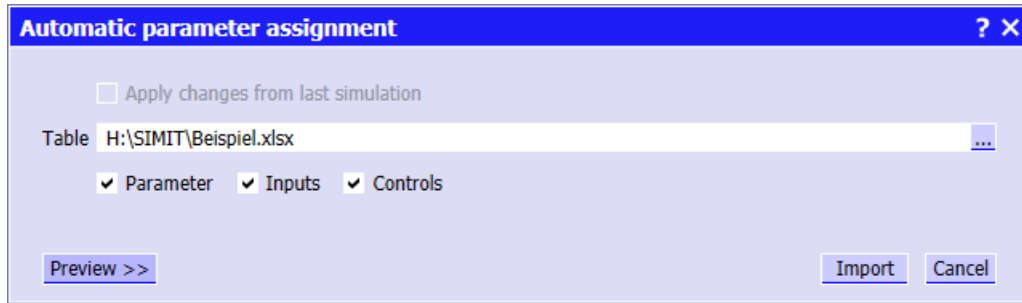
	A	B	C
1	Slider#1	X	42
2	DriveV2#1	Initial_Value	1
3	DriveV2#1	HI_Limit	90
4	DriveV2#1	T_Close	2.5
5	DriveV2#1	Open	True

### Importing the table

There are 2 ways to perform automatic parameter assignment:

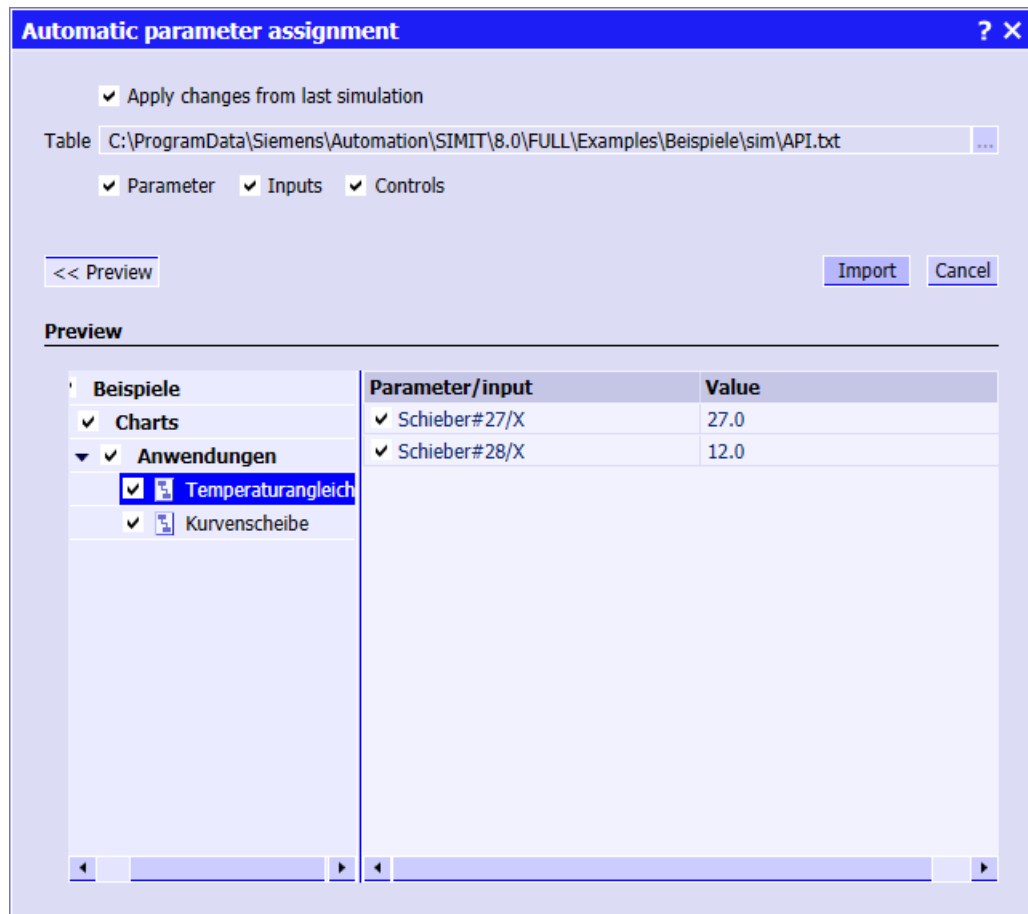
- In the portal view, Select "Automatic model creation" > "Automatic parameter assignment"
- From the project view, select the menu command "Automatic model creation > Automatic parameter assignment".

The following dialog box opens:

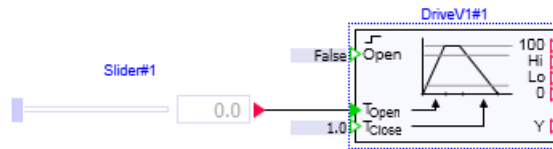


Select the table to be imported. You can also specify the variables (parameters, inputs, controls) whose values are to be copied from the table. You can import an Excel file in \*.xls or \*.xlsx format or a tab-separated text file.

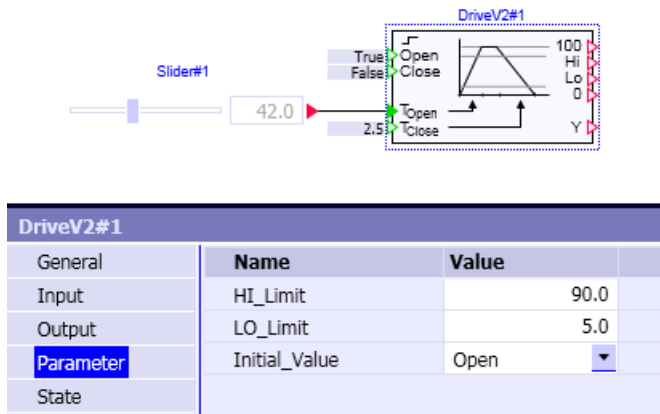
In the import dialog preview shown below, you can see which replacements are to be made for the parameter table from the example of a parameter table in the figure at the very top.



The figure below shows the original chart, while the changes resulting from automatic parameter import are shown in second figure.



DriveV1#1		
General	Name	Value
Input	HI_Limit	95.0
Output	LO_Limit	5.0
Parameter	Initial_Value	Closed
State		



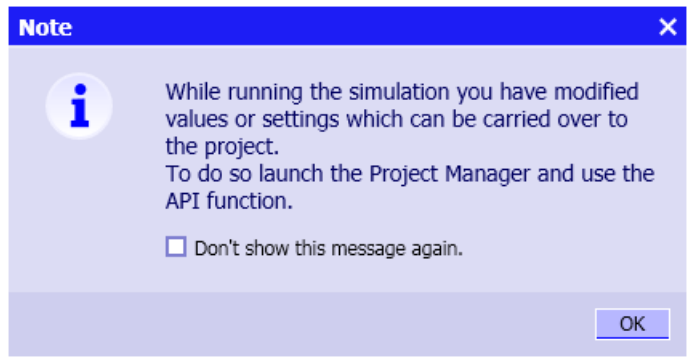
**Note**

If you want to specify parameters with enumeration types, enter a numerical value specifying the desired position of the term within the enumeration. The numbers start at 0.

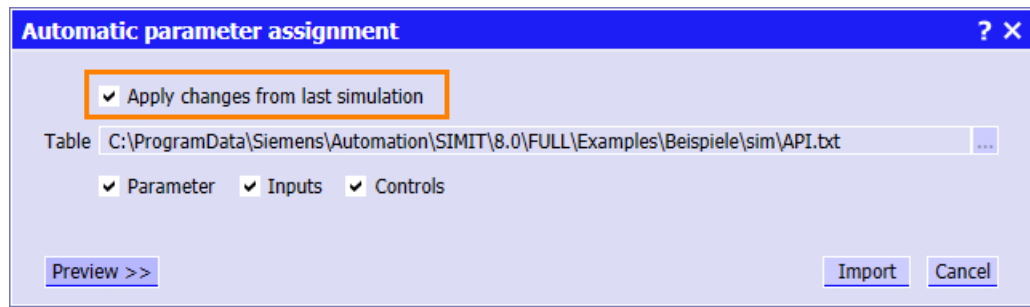
**Applying parameter changes made while the simulation is running**

With the automatic parameter assignment, you can also apply changes in the simulation project that were made while simulation was running after the simulation is completed. During the simulation, SIMIT records which default input settings and parameters are changed and which controls are used. Once the simulation has ended, the latest value settings are made available as a list and can be applied to the simulation project by means of automatic parameter import.

When the simulation ends, a corresponding message is displayed. If you do not need to see this message again, you can choose for it not to be displayed in the future.



Each time the simulation is run, the list of changes is automatically stored in the SIMIT work area. If you select the *Apply changes from last simulation* option in the import dialog, the corresponding file is selected.







## Diagnostics & visualization

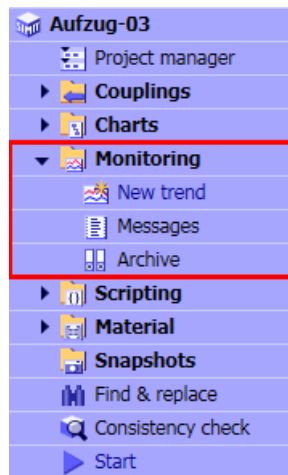
### 5.1 Trend and Messaging Editor (TME)

#### 5.1.1 The functions of the Trend and Messaging Editor

The Trend and Messaging Editor supports the analysis of processes and signal values in the simulation with the following functions:

- **New trend**  
A trend displays the course of signals graphically and can also be printed. You can find additional information on this in the section: Trends (Page 255).
- **Messages**  
The messaging system generates messages for events in the simulation. The messages can be evaluated in the message editor. You can find additional information on this in the section: The Messaging System (Page 250).
- **Archive**  
The archive allows you to record signals in a simulation run. You can find additional information on this in the section: The Archive (Page 253).

The functions are located in the "Monitoring" folder in the project tree .



## 5.1.2 The Messaging System

### 5.1.2.1 Overview

Messages provide information on the occurrence of certain events within the simulation. Depending on the triggering event they may be classified as follows:

- Messages sent by the message component**  
 A designated message component sends a message. This component is part of the standard library and can be linked with binary signals on charts. The message category and message text can be specified individually for any instance of the message component.
- Messages sent by other components**  
 These messages are defined in the component types and have a fixed message category and message text that are stored in the component type.
- System messages**  
 System messages are generated by SIMIT itself and are used to inform you about unexpected events, for example, problems when establishing a coupling connection. SIMIT system messages are always of the *SYSTEM* category.

All messages that are created after simulation starts are buffered in SIMIT and are available in the message editor. Existing messages are deleted when the simulation is restarted.

### 5.1.2.2 The message editor

#### Overview

You can open the message editor by double-clicking the *Messages* entry under *Monitoring* in the project tree. In this editor you see the messages along with the simulation time, a category, the name of the component that triggered the message, and a message text. Simulation time is displayed in the format hh:mm:ss:msec.


Messages			
Mask acknowledged messages			
Simulation time	Category	Source	Message
00:00:03:200	ERROR	DIV#1	DIV: division by zero
00:00:10:100	ERROR	DIV#2	DIV: division by zero
00:00:16:800	ERROR	DIV#1	(DIV: division by zero)

You may open the message editor when the simulation is running or stopped. If the simulation is running, new messages are added as soon as they are issued. After stopping the simulation, all messages of the last simulation run remain available.

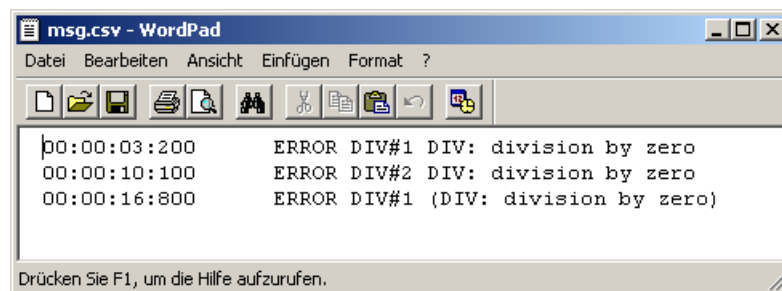
You may filter the displayed messages according to various criteria. You can select the category that is to be filtered. All other criteria are used to search for a text that is contained in the messages that are to be filtered. By way of example, the figure below shows a filter that has been set for messages with the source *DIV#1*.

Messages			
Mask acknowledged messages			
Simulation time	Category	Source	Message
00:00:03:200	ERROR	DIV#1	DIV: division by zero
00:00:16:800	ERROR	DIV#1	(DIV: division by zero)


## Exporting messages

Use the  command in the message editor toolbar to export all messages into a text file (\*.txt). The resulting file will contain four columns with the following content: Simulation time, category, name of issuing component and message text. The columns are tab-delimited.

This allows you to archive messages that were issued during a simulation, for example, or to further process them in other applications such as an editor.



## Deleting messages

Use the  command to delete all messages in the message editor.

## Hiding message pairs

A message component generates messages that are triggered by the binary signal to which it is connected. When the signal changes from 0 to 1, a message will be generated as an "incoming" message; when it changes from 1 to 0, an "outgoing" message is generated. "Outgoing" messages are indicated by placing the message text in parentheses.

A pair of messages consists of an "incoming" message and a subsequent "outgoing" message for the same event. You can use the "Message pair" button to hide and show message pairs. If you hide message pairs, the only messages that remain visible are those that have "come in" but have not "gone out" yet.

### 5.1.2.3 Message – the message component

#### Symbol



#### Function

The *Message* component type is to be found in the standard library *Misc* folder. A component of this type will generate a message when the binary value at its input changes. In the case of an "outgoing" message, the message text that is output by the component is placed in parentheses.

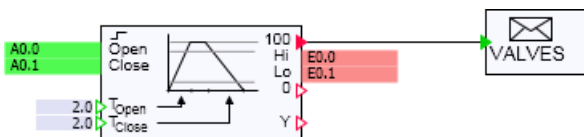
You may specify both the message category and text as parameters in the component property view.

Message#1		
General	Name	Value
Input	Category	WARNING
Output	Message	torque switch-off
Parameter		
State		

#### Note

Messages in the SYSTEM category are system messages from SIMIT. Messages in the ERROR category are messages from components in the basic library. When choosing category names, avoid using SYSTEM and ERROR. When filtering messages by category, this will allow to see just the messages from your own categories.

The following figure shows an example of how you can use the message component in your simulation. The component is linked to the 100 output of a valve drive simulation. As a result, messages are output when the drive has opened the valve ("incoming" message), and when the drive closes it again ("outgoing" message).



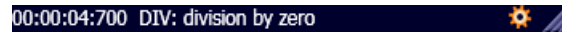
### 5.1.2.4 Component-specific messages


Some components from the basic library generate messages to notify you about illegal operations such as division by zero or illegal parameter assignments. For example a component of type SQRT will output the message "*SQRT: invalid argument!*" when the input value has become negative.

Messages from standard library components are always assigned to the *ERROR* category, and their message text is of the form "Component type: Error message". All messages are created as "incoming" or "outgoing".

### 5.1.2.5 Messages in the status bar

The most recent message is shown to the right of the SIMIT interface status bar, along with its simulation time and message text:



00:00:04:700 DIV: division by zero 

### 5.1.2.6 Limitations of the message system

The message system can process up to 1000 messages per second. If this value is exceeded, messages are lost. If this occurs, you receive a corresponding message:



00:00:02:600 SYSTEM ControlSystem Warning TME : Message system overload

The maximum number of messages that can be stored is limited to 4096. If this limit is reached, you receive a final message indicating this:



00:02:58:600 SYSTEM TME Message buffer full

## 5.1.3 The Archive

### 5.1.3.1 Overview

The archive is used to record signals from a simulation run, which means from simulation start until simulation stop. When the simulation is run again, which means the next time the simulation starts, the archive is deleted and the signals that are to be archived will once again be recorded until the simulation stops.

You can record the following signals in the archive:

- Input and output signals of components
- States of components
- Input and output signals of the couplings

Archived signals can be visualized in charts in form of trends.

### 5.1.3.2 Configuring the archive

Double-click the *Archive* entry within the project tree *Monitoring* folder to open the archive editor and define the signals that are to be archived. In the table, you can now manually enter the signals that are to be archived or simply drag them into the table from the *Signals* task card.

The screenshot shows the 'Archive\*' dialog box. On the left is a table with columns 'Source', 'Name', and 'Dead band'. The table contains two rows: 'ADD#1' with 'Y' and '0', and 'INT#1' with 'LL' and '0.05'. The 'INT#1' row is selected. On the right is the 'Signals' task card, which has dropdown menus for 'Origin', 'Signal Type', and 'Data Type', all set to 'All'. Below these is a 'Reset Filter' button and a 'Search Results' table. The 'Search Results' table has columns 'Source' and 'Name' and lists several signals: 'INT#1' (LL), 'INT#1' (LLR), 'INT#1' (SET), 'INT#1' (SP), and 'INT#1' (T).

Source	Name	Dead band
ADD#1	Y	0
INT#1	LL	0.05

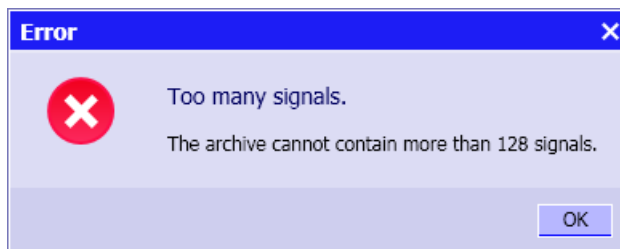
Source	Name
INT#1	LL
INT#1	LLR
INT#1	SET
INT#1	SP
INT#1	T

The ability to define a deadband is provided so that you may limit the quantity of data that arises from recording the signals. The deadband specifies how much a signal value must change before the value is recorded. A signal value is only recorded if the value differs from the last recorded value by more than the deadband. A deadband can only be specified for analog and integer signals. The value of a binary signal is recorded every time it changes.

#### Note

If a deadband of 0 is used, a value is recorded in every simulation time slice for analog and integer signals that change continuously.

A maximum of 128 signals can be entered in the archive. If you exceed this limit, an error message as shown in the figure below is displayed:



The archive will reserve memory for each signal. Memory size is computed such that, given a base cycle of 100 msec, a signal whose value is recorded in every cycle can be recorded for about 2.5 hours. Less frequent signal changes will therefore lead to longer recording times. The archive memory is organized in form of a circular buffer.

## 5.1.4 Trends

### 5.1.4.1 Overview

Trends allow you to graphically display the change of signal values over time. You can display the following signals in trends:

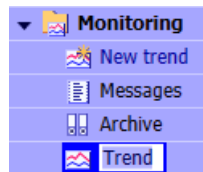
- Input and output signals of components
- States of components
- Input and output signals of the couplings

The signals may be either archived or not archived. Archived signals can be displayed with their complete trend history as recorded in the archive.

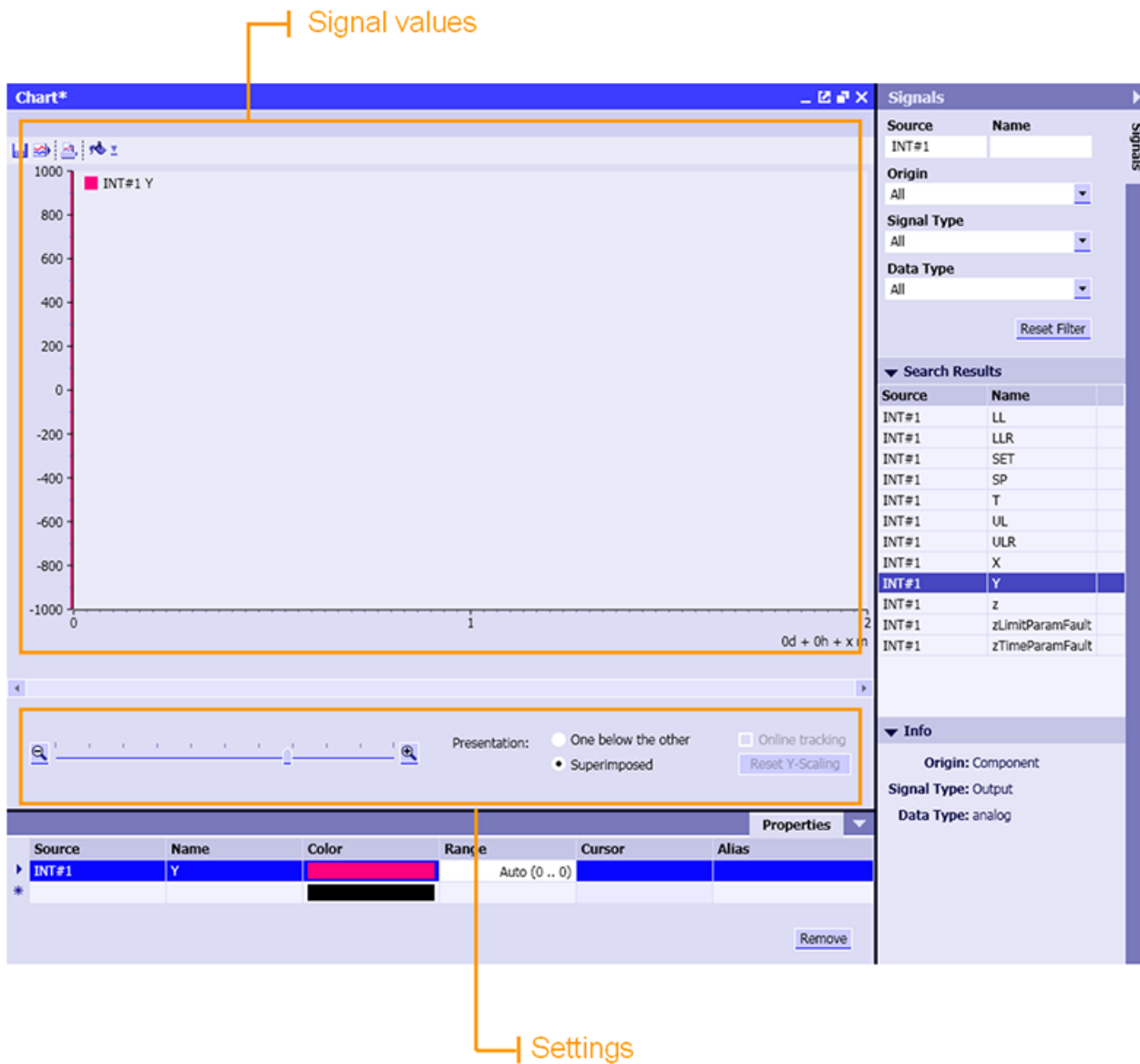
Signals that are not archived will only be displayed for the time span during which the trend is open. The trend will reserve memory for each signal that is not archived. Its size is computed such that, given a base cycle of 100 msec, a signal whose value changes in every cycle can be recorded for about 35 minutes. Less frequent signal changes will therefore lead to longer recording times. The memory is organized in form of a circular buffer.

### 5.1.4.2 Adding and configuring trends

Trends are stored in the project tree *Monitoring* folder. Use the *New trend* command to create a new trend. In your project you may add any number of trends and configure them, even if the simulation is running already.



Trends are opened in the trend editor within the work area. The trends of the signals are displayed in the upper section of the trend window, and settings for the trend can be made in the lower section.



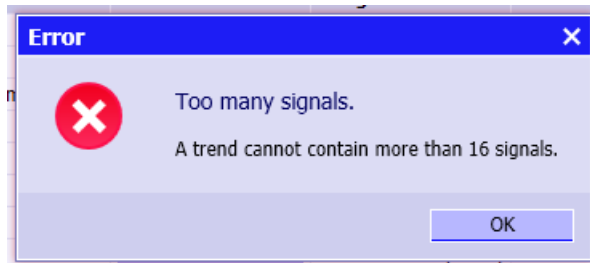
### Configuring trend signals

Use the table in the property view to specify which signals should be displayed in the trend. You can manually enter signals into the table with their source and name or simply drag the signals into the table from the *Signals* task card.

Properties					
Source	Name	Color	Range	Cursor	Alias
PTn#1	Y	<span style="display:inline-block; width:15px; height:15px; background-color:red;"></span>	Auto (0 .. 0)		X1-B25-K456
PTn#2	Y	<span style="display:inline-block; width:15px; height:15px; background-color:blue;"></span>	(0 .. 2000)		
Compare#1	OUT	<span style="display:inline-block; width:15px; height:15px; background-color:red;"></span>	Binary		
Compare#2	OUT	<span style="display:inline-block; width:15px; height:15px; background-color:green;"></span>	Binary		
*		<span style="display:inline-block; width:15px; height:15px; background-color:black;"></span>			



You may enter a maximum number of 16 signals into a trend. A dialog will appear if you attempt to enter more than 16 signals into a trend.



### Assigning parameters to signals within a trend

You may specify the following parameters for each signal:

- The color in which the signal trend should be displayed
- The value range of the signal
- An alias for the signal

The color in which the trend for the signal should be displayed can be set using a color selector.

Analog and integer signals are displayed within a certain value range. An automatic mode *Auto (0..0)* is set by default, in which the value range is continuously adapted in accordance with the actually occurring signal values. This mode spares you the effort of individually specifying a value range and ensures that all signal trends are displayed in maximum resolution.

---

#### Note

When using automatic range control, it is more difficult to compare several different signal profiles because the scaling continuously changes.

---

You may also directly enter a fixed range by entering the start value and end value separated by two dots, for example: *0..2000*.

If you want to label a signal in the trend with a name that differs from the signal name (which consists of the source and name), you can assign an alias to the signal. This alias is used to label the trend.

---

#### Note

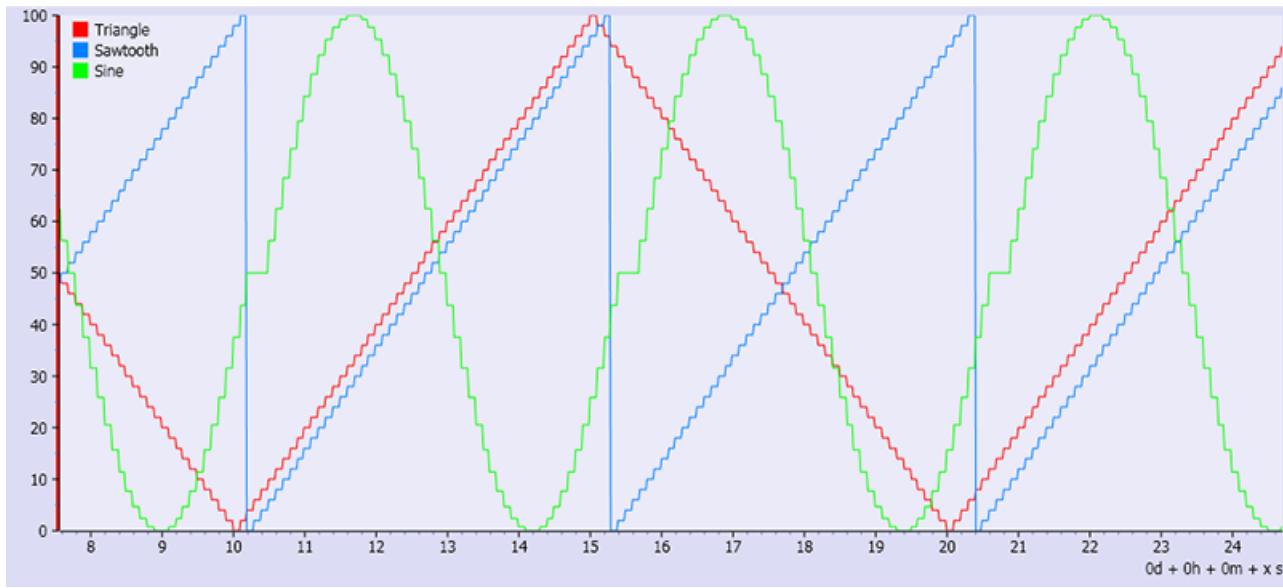
You may create any number of trends in a project, but the total number of signals contained in all open trends must not exceed 128. The opening of trends is canceled with an error message in this case.

---

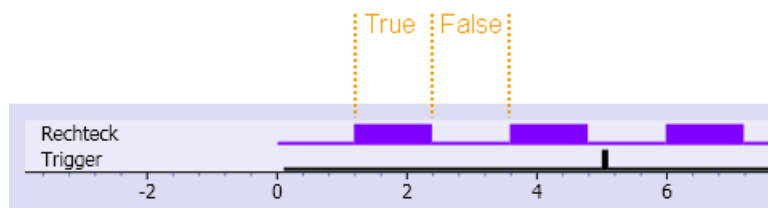
### 5.1.4.3 Displaying signal trends

#### Overview

Analog and integer signal trends are displayed using trends. The horizontal axis (X axis) shows the simulation time, the vertical axis (Y axis) shows the signal values in these charts.



Binary signals are displayed below the analog and integer signal trends. Because binary signals can only have the values 0 (false) and 1 (true), a bar graph is used to plot the trend of a binary signal over the simulation time. A line thus represents a value of 0 (false) and a bar represents the value 1 (true), as shown in the figure below for the binary signal *Rectangle*.



For archived signals, the signal trend is always based on the signal values in the archive. Because the recording of signals in the archive starts when the simulation is started, the trend of archived signals always shows all times since the last time the simulation was started, provided the memory limit was not exceeded.

Signals that are not archived are only plotted while the trend is open. This means you can only see the trend of such signals starting with the time the trend was opened. Consequently, when you stop a running simulation, you will see only the signal trend of non-archived signals in trends that were already open while the simulation was running. When a trend is closed, all of the recorded signals are discarded so that the signal trends of non-archived signals can no longer be displayed after reopening the trend.

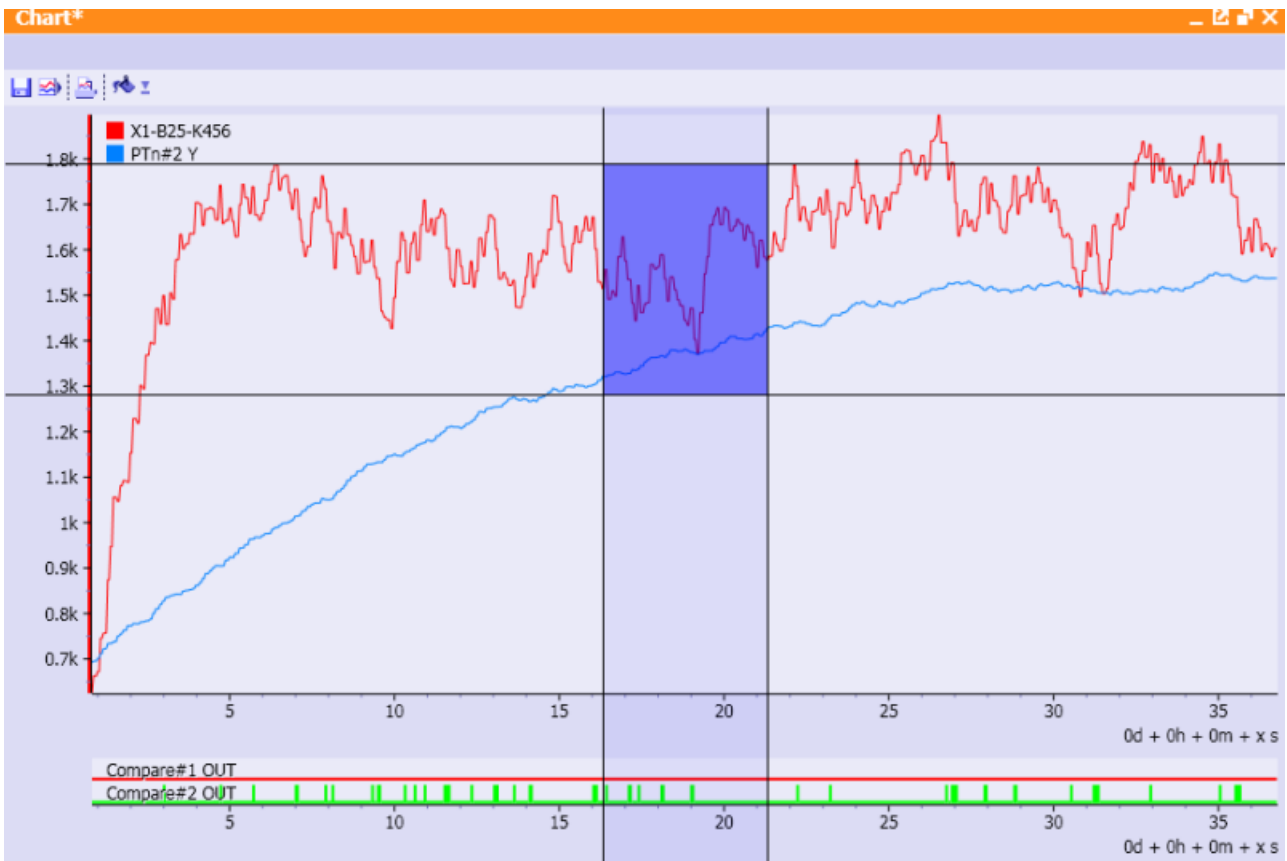
While the simulation is running the signal properties also display the current signal value.

Source	Name	Color	Range	Dead band	Current	Alias	Archived
Ramp#2	Y	<span style="background-color: red; width: 20px; height: 10px; display: inline-block;"></span>	Auto (0 .. 100)	0	40.0	Triangle	<input type="checkbox"/>
Ramp#4	Y	<span style="background-color: blue; width: 20px; height: 10px; display: inline-block;"></span>	Auto (0 .. 100)	0	34.0	Sawtooth	<input type="checkbox"/>
MUL#3	Y	<span style="background-color: green; width: 20px; height: 10px; display: inline-block;"></span>	Auto (-49.9013364...	0	-18.40622763423...	Sine	<input type="checkbox"/>

The dead band of archived signals is taken into account when displaying the signals in the trend. The signal property view displays the dead band values while the simulation is running. For non-archived signals, all values that are calculated in the simulation are recorded in trends, which is equivalent to a dead band value of zero.

### Zooming in on sections of the trend

When you turn off continuous updating of the trend and move the cursor into the trend chart, a crosshair is shown. To zoom in on a specific area, select the area with the left mouse button pressed. The selected area is then zoomed in the trend chart.





Use the *Reset Y scaling* button to reset the Y axis to the defined ranges.

Presentation:  One below the other  Online tracking  
 Superimposed

You can find additional information on this in the section: Trend settings (Page 260).

## Trend settings

### Overview

Within a trend you may specify

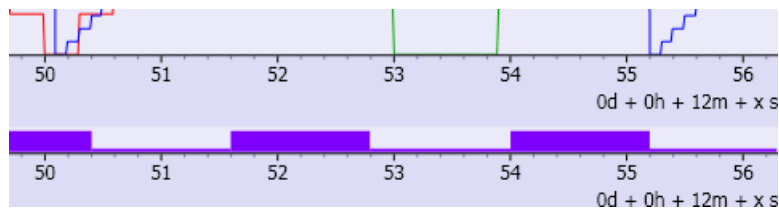
- The scale of the time axis,
- Whether or not to constantly update the trends
- Whether to draw trends in superimposed mode or one below the other.

## Time axis

The scale of the time axis can be set with a slider:



The time at which the visualization starts is shown on the right hand side, below the time axis. Add the corresponding number of days (d), hours (h), minutes (m) or seconds (s) according to the measured values on the X axis to this start time. By way of example, the figure below show a section of the time axis ranging from 12 minutes and 50 seconds to 12 minutes and 56 seconds.



## Cyclic update

When you open a trend and start the simulation, you will see the trend values in a constantly updated sequence. The trends track the signals as they change. In order to obtain a static view of the current trends, switch off the updating. To do so, deselect the update option:

Online tracking

If there is no simulation running, you see a static view of the recorded signals.

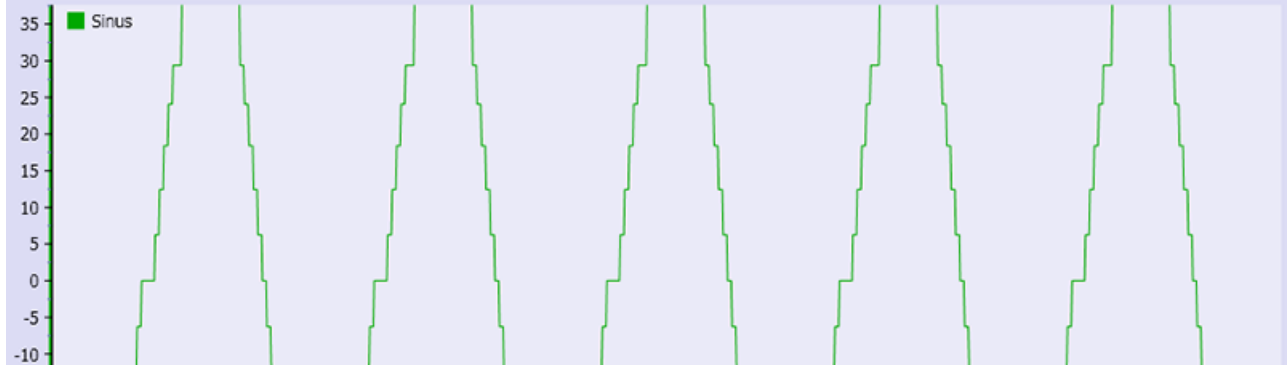
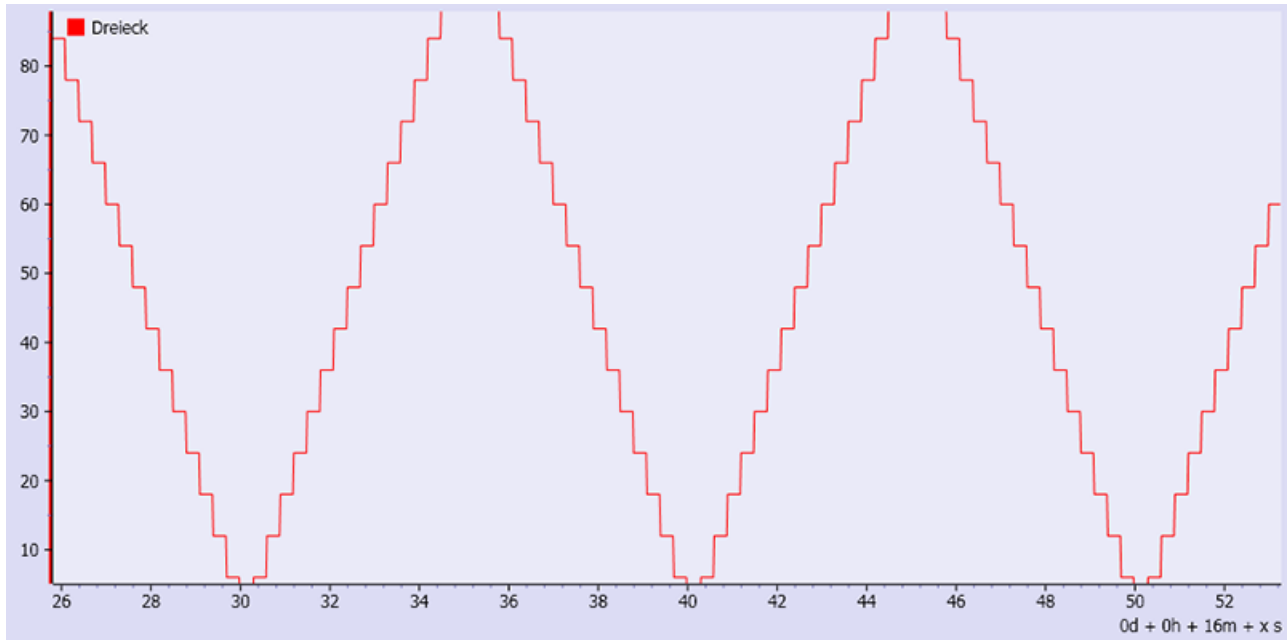
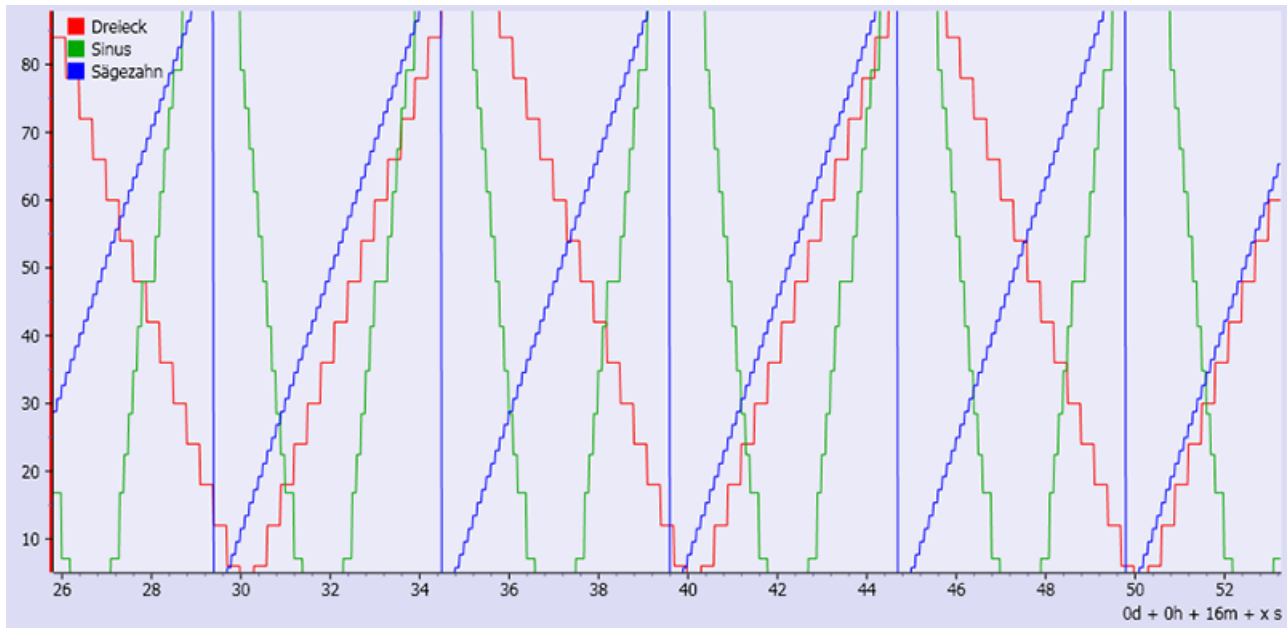
## Displaying trends in superimposed mode or one below the other

You can display all analog and integer signals of a trend in one single chart in *superimposed* mode or you may display all signals *one below the other*.

In superimposed display mode, an appropriately labeled Y axis can only be shown for one of the signals. You can specify which Y axis to display by clicking on the signal name. The Y axis is highlighted in the signal color so that you can see which Y axis is being displayed.

The following two figures show a graph with superimposed signals (top figure) and a graph with signals one below the other (bottom figure):

5.1 Trend and Messaging Editor (TME)



The trends for binary signals are always displayed in separate trends, one below the other, with a single common time axis.

#### 5.1.4.4 Features of the trend editor

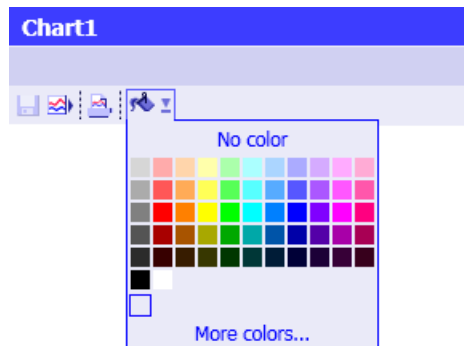
##### Overview

You can use the trend editor to define the trend background colors or to print and export trends. The corresponding commands are available in the toolbar of the trend editor:




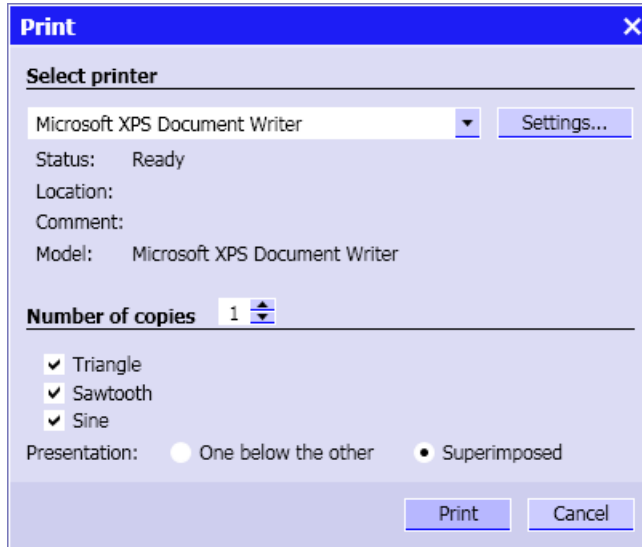
##### Background color of trends

You can freely choose the background color of a trend. To do this, use the color selection in the toolbar of the trend editor:




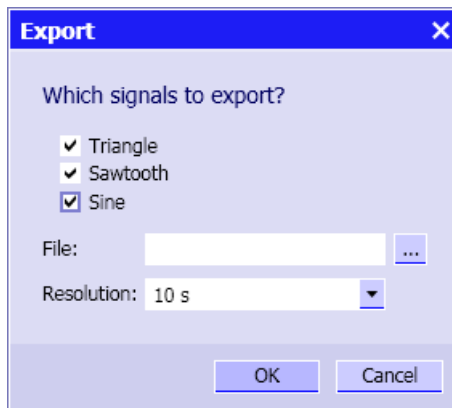
### Printing trends

Use the  command in the trend editor toolbar to print a trend. The dialog shown in the figure below appears. In this dialog, you can specify which trend signals should be printed by selecting the signals. You can also select how the signal trends should be displayed: one below the other or superimposed.



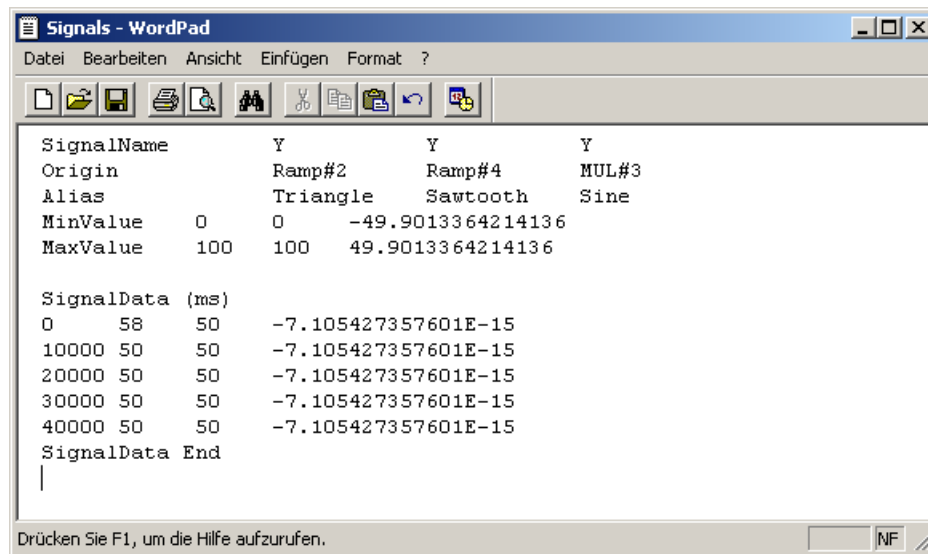
### Exporting trends

Use the  command in the trend editor toolbar to export the signals that are recorded in a trend into a text (\*.txt) file. In the export dialog, you can specify which signals are to be exported.



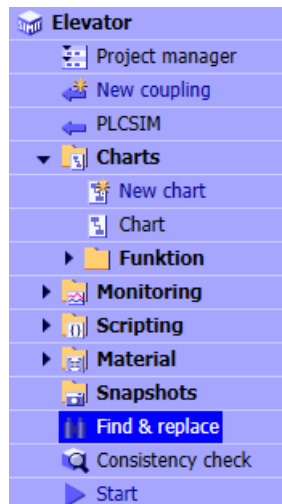
The numeric values of the selected signals at equidistant points in time are output to the file. The time resolution can be selected in the export dialog as 100 msec, 1 s, 10 s or 60 s. The generated file will contain the points in time and the values for each signal, with one column per signal. The individual columns are separated with the tabulator character. If there are several signal values for an analog or integer signal within a time period, the arithmetic mean between the minimum and maximum values that occurred is used.






## 5.2 Find & replace

Use the *Find & replace* entry in the project tree to find elements within your project and to perform replacements.



In the signal properties of components and controls, use the  symbol to find references to a signal in the project.

## 5.2.1 Find

### 5.2.1.1 Finding with the Find & replace editor

The *Find & replace* editor opens in the work area. The "Components", "Macros" and "Signals" task cards are available in this editor. The following elements can be selected, for which you can search in your project:

- Signals
- Connectors
- Instances of components and macro components
- Components and macro components by type
- Graphic text

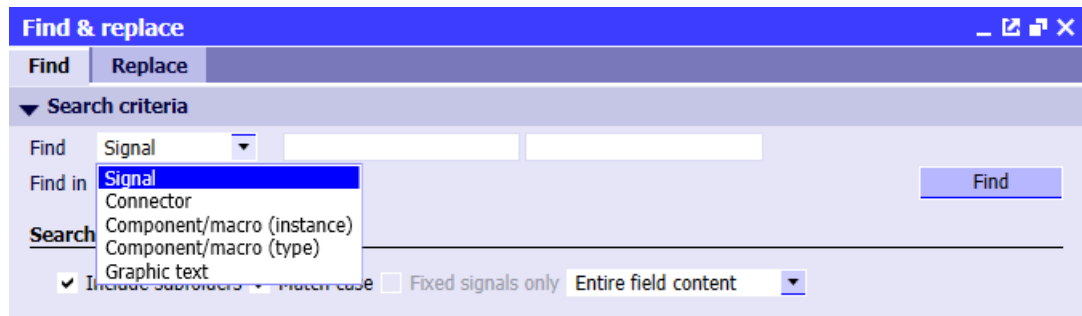
---

#### Note

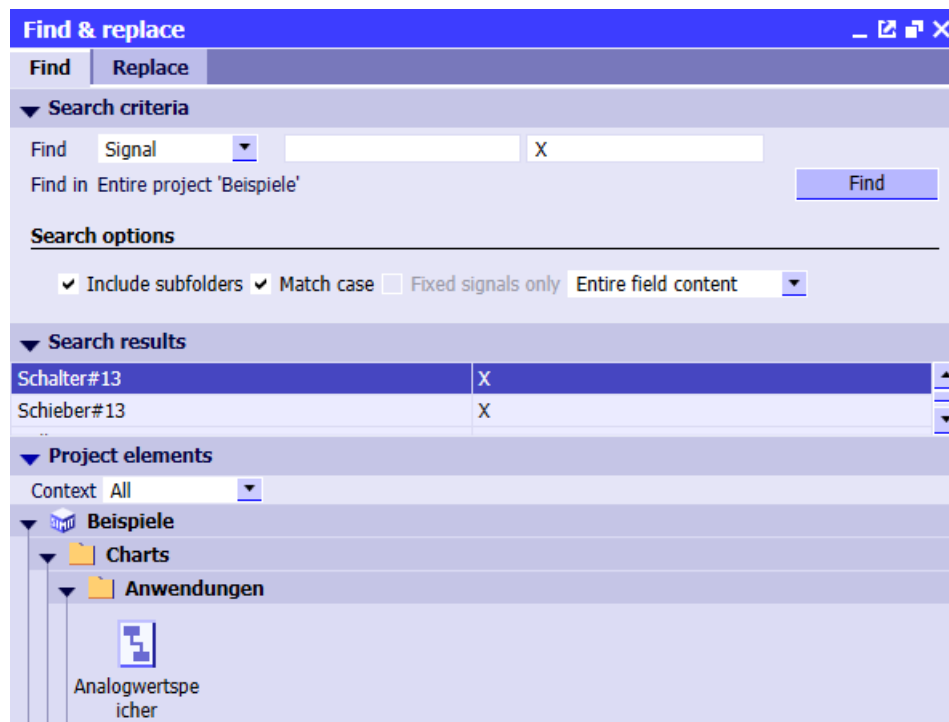
The search only works with stored data only, therefore charts and couplings must be saved before searching.

---

Select search for a signal and enter *V* as the signal name in the input field on the right side. Now click the *Search* button to start the search.

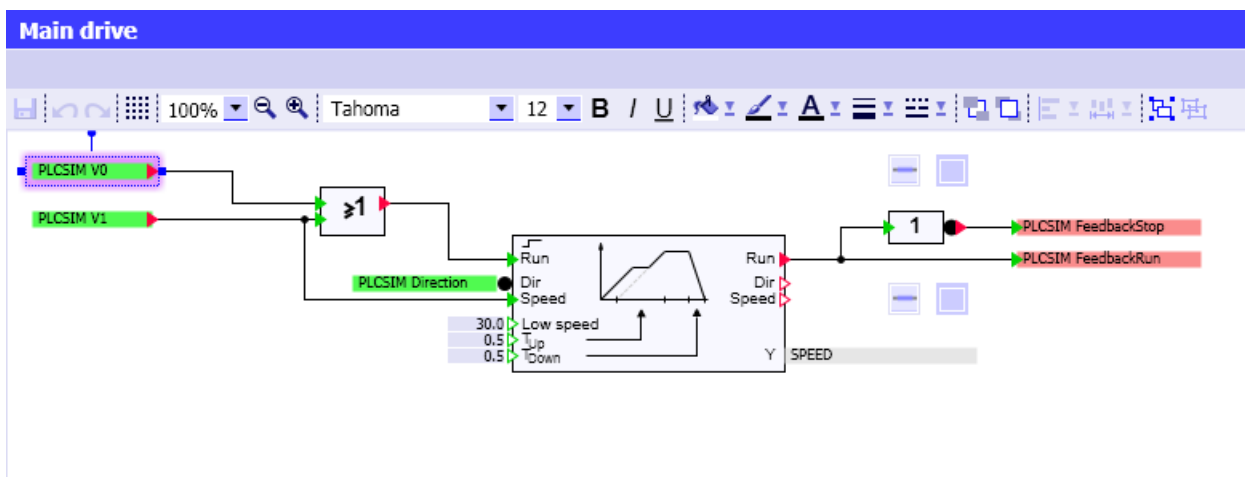


Under *Search results* depending on the selected search option *Part of field content*, all of the signals in your project that contain *V* as part of the signal name are displayed. As can be seen, this only includes the two coupling signals *V0* and *V1*.



For each signal in the search results, the charts and couplings that contain this signal are displayed under Project elements in the Project Manager view. To see this, select a signal in the search results, for example, in this case the *PLCSIM V0* signal. In addition to the PLCSIM coupling, three charts are displayed under Project elements. If you open the Main drive by double-clicking it, for example, you see that the signal for which you searched is indicated with a purple frame. The signal is highlighted in a similar manner in all other charts.

If a search result corresponds to a single chart you can also open the chart by double-clicking on the search result.



To search for a specific signal just drag-and-drop it from the *Signals* task card into either of the two input fields.

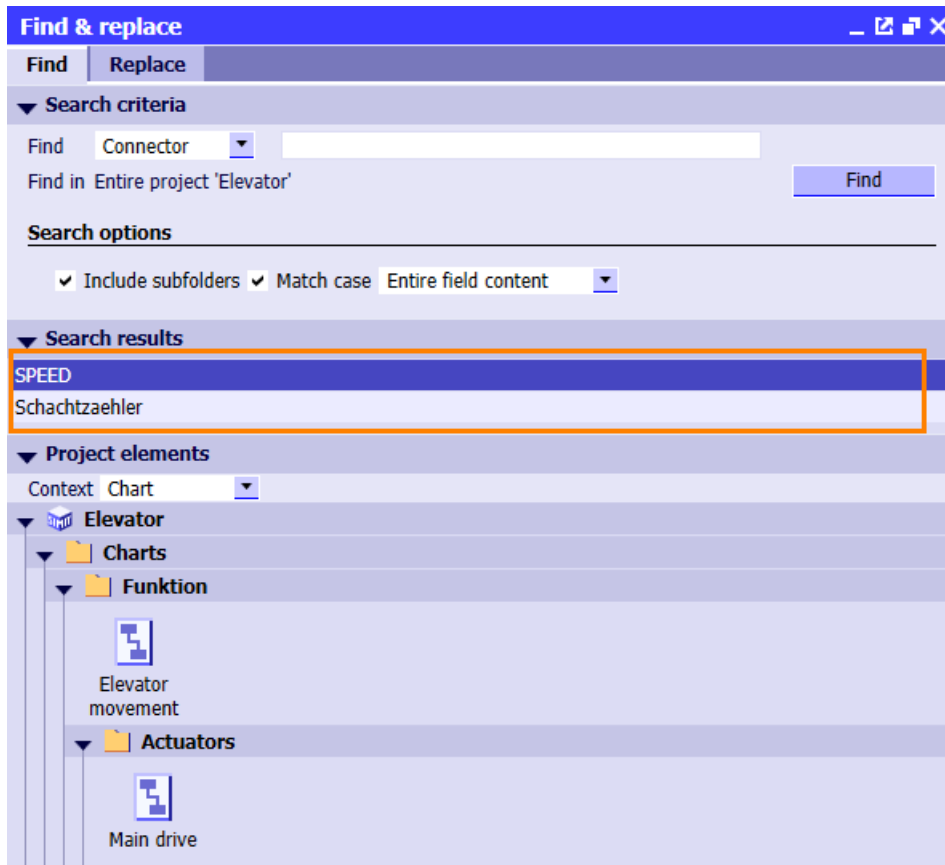
When searching for a component type, enter its name and/or ID. When searching for a specific component type in your project, you may similarly drag-and-drop a component type from the *Components* task card into the input field.

To search for components and connectors the name must be specified.

**Navigation via the search results**

If a search result can be assigned to a specific chart, you can open the corresponding chart by double-clicking the search result. You can also select the search result and then double-click the chart in the *Project elements* area.

If there are several project elements corresponding to a search result, you still need to double-click the desired project element to open it:

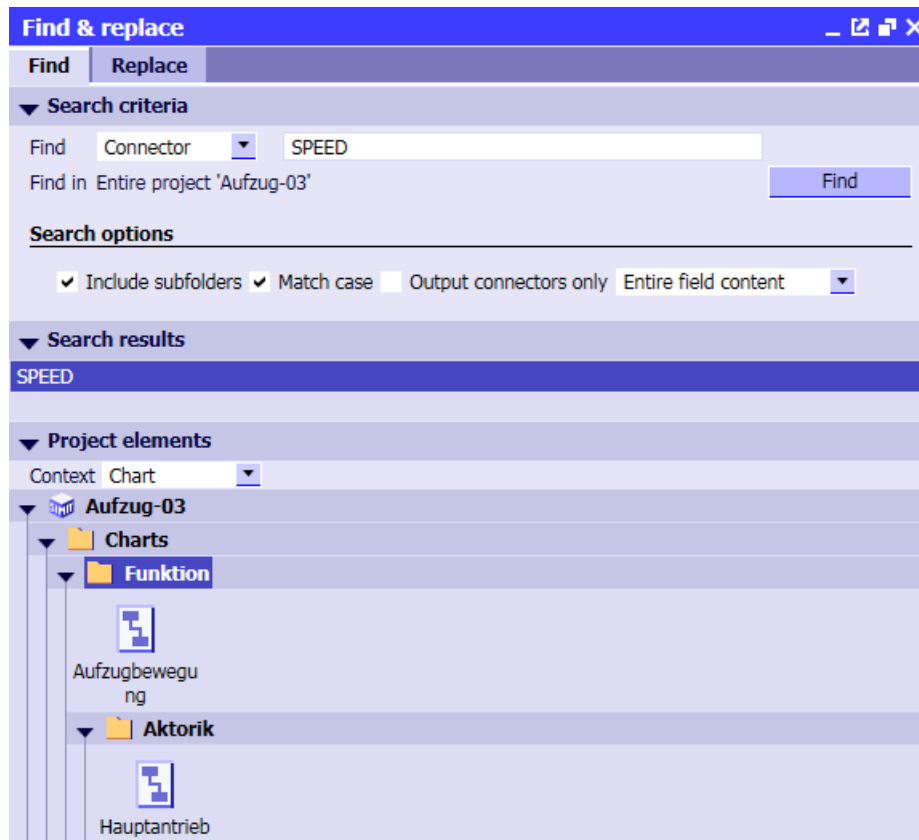


**5.2.1.2 Search using signal properties**

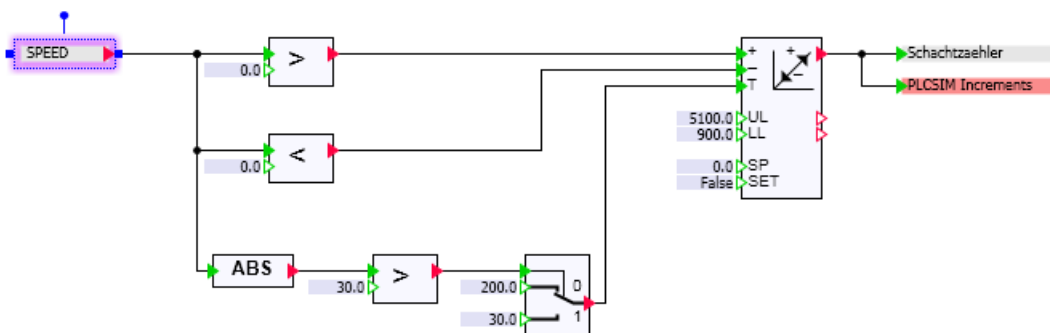
Another way to search for signals and connectors is to use the properties view of a chart. For example, if you want to search a chart for input connectors that are connected to the SPEED output connector, open its properties and click the *Find* button (🔍).

SPEED		
General	Property	Value
	Name	SPEED

The Find & replace editor will then appear with the search results.



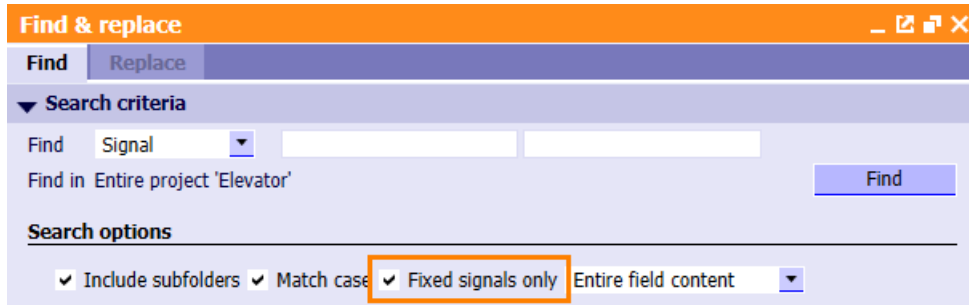
Except for the *Main drive* chart, from which the search was run, the *SPEED* connector is only contained in the *Elevator movement* chart. If you open this chart you see the connector highlighted.



With the search option "Only output connectors", you can hide all search results of input connectors in the search for global connectors and navigate directly to the respective output connector.

### 5.2.1.3 Searching for fixed signals

You can search specifically for fixed signals in the simulation project. To do this for the search for signals, activate the search option *Fixed signals only*. In this way you can get an overview at any time of which signals are currently fixed.



Because signals can only be fixed when the simulation is running, this search option is only available when the simulation is running.

## 5.2.2 Replace

### 5.2.2.1 Replacing with the Find & Replace editor

To replace search results in the editor, click on the "Replace" tab. You can replace

- Signals
- Connectors
- Instances of components and macro components
- Components and macro components by type
- Graphic text

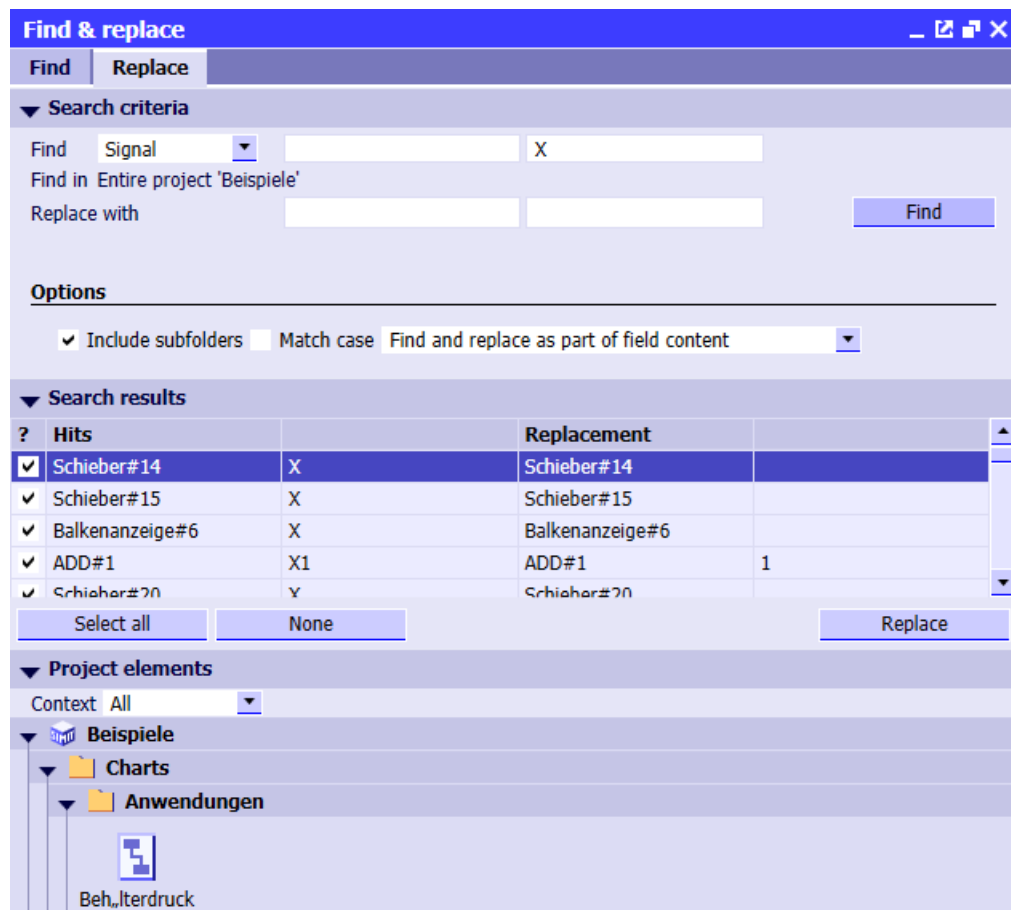
---

#### Note

Replacement works with stored data only, which means that charts and couplings must be saved before replacing.

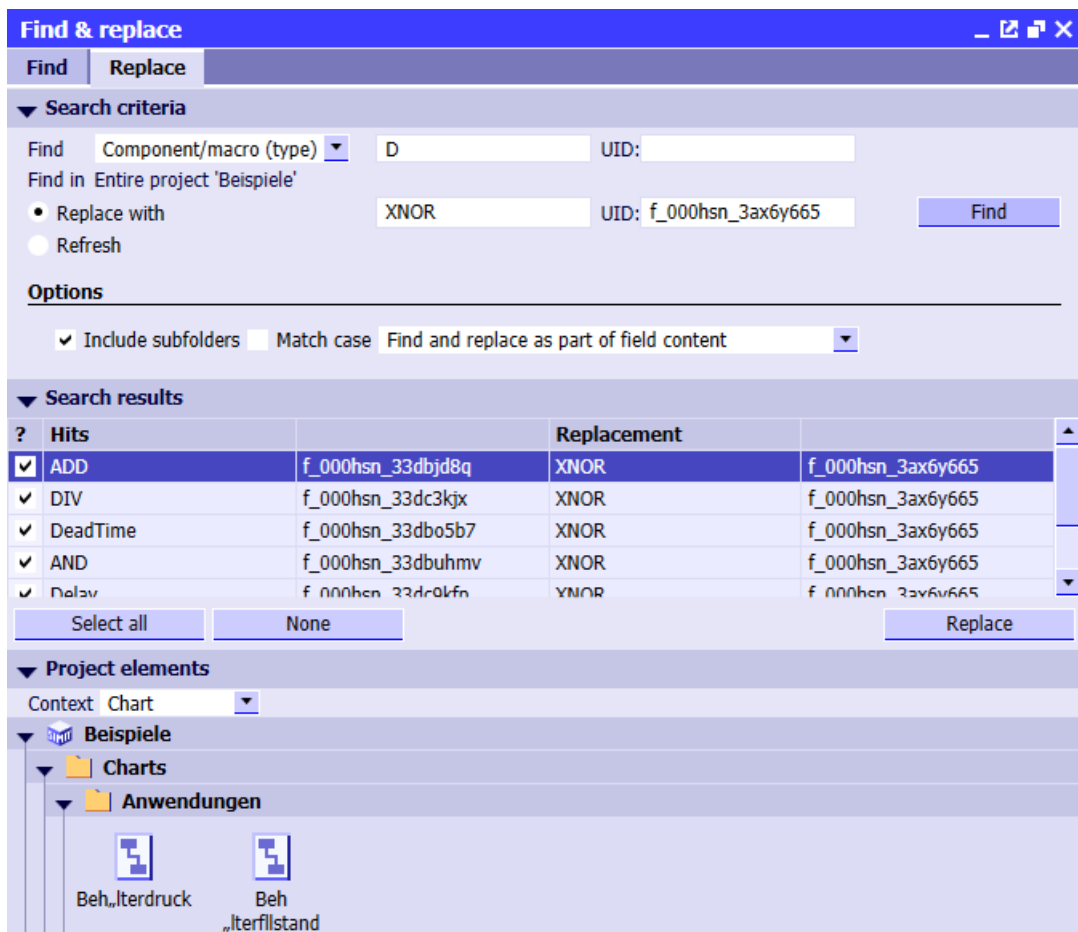
---

Again, please enter the signal name V and an X in the replacement field, for example. After searching you see the search results displayed in the Search results section again.

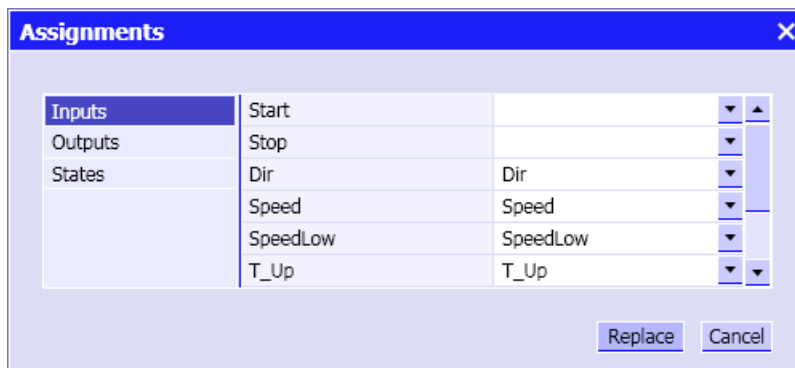


In the Search results section you can now select which results you want to replace. Depending on the patterns that were set for the find and replace, you see the results of the replacement that are performed for each search result. As with the search mode, the Project elements section shows all charts in which the signals for which you searched were found.

In your project, you can also replace components or macro components of a specific type by a component or macro component of another type. You may be searching for component of type *DriveP1*, for example, that simulates the main drive and you want to replace this component with type *DriverP2* because the control logic was modified. To do so just drag-and-drop both component types from the library into the input fields for find and replace. As shown in the figure below, the two types were entered in the fields with their unique identifier, UID.



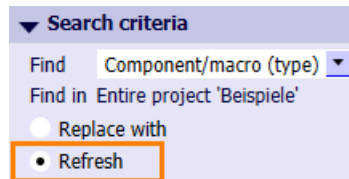
If you now click the *Replace* button, a dialog appears in which the assignment of the inputs and outputs of both component types is displayed. Change the assignment, if necessary.



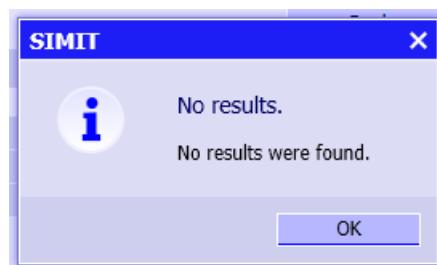


### 5.2.2.2 Refresh

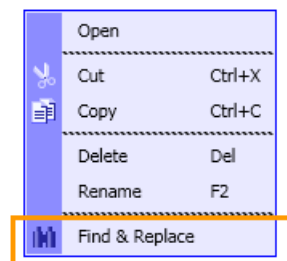
When replacing components, you can update components by type in your simulation project using the *Update* option (see the figure below). If you choose this option, the system searches by component type for all components in the defined search area that have the same name and belong to the same library family but were created (saved) at a later date. The search for current component types takes place in all component libraries of the *Components* task card, in the following order of panes: *Project components*, *User components* and *Basic components*. The most recently created component type is then suggested as the current type for replacement in the search result.



The *Refresh* option works in the same way for all macro components contained in the defined search area, with the search for current macro components taking place in the libraries of the *Macros* task card. If all components and macro components in the selected search area are already up to date, the message dialog shown below is displayed as the result.



The *Refresh* option is also available if you want to update components by type in macro components. Start the update via the shortcut menu for the macro component.

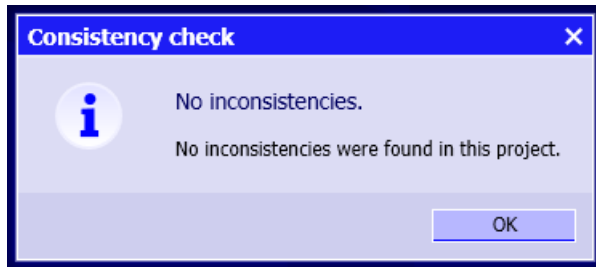


## 5.3 Consistency check

Use the consistency check to check whether your project is consistent. Inconsistent projects cannot be started. The consistency check is always performed automatically when the simulation is started.

5.3 Consistency check

You can also manually start the consistency check by double-clicking on the tree node "Consistency check" in the project tree. When a project is consistent, a dialog appears indicating that there are no inconsistencies:



If you rename the *SPEED* connector as *Increments* in the *Elevator-03* project on the Main drive chart, for example, two output connectors with the same name exist in the project. The project is no longer consistent, and when the project is started the consistency check opens in the work area showing the inconsistencies that were found (see figure below). Because a consistency check is always performed automatically when the simulation is started, the consistency check editor also shows when you try to start this inconsistent project.



Inconsistencies are indicated by being preceded with the red symbol "E". Warnings are indicated with the blue symbol "W". Using a command, you may switch the display on or off for warnings.

In the above example there are two results:

1. There is no output connector that matches the input connector SPEED.
2. The output connector Increment exists in several instances.

The first message is only a warning and indicates that the simulation model may not yet be finished. A warning will not keep you from launching the simulation. The second result indicates an error that needs to be fixed before the simulation can be launched.

To fix an error or a warning select its entry in the list of results. The Project elements section shows the charts in which the error or warning must be resolved. From this view you may then

open the charts to fix the problem. After resolving the inconsistencies, you may recheck the project using the *Recheck* button.

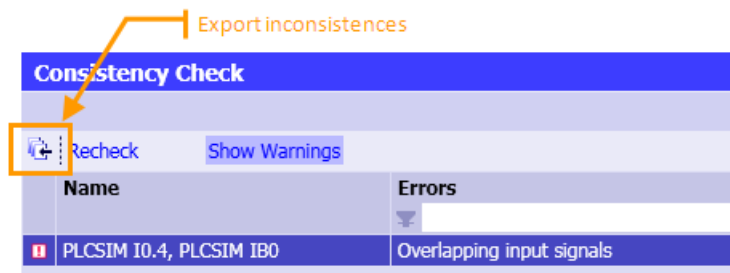
---

**Note**

The consistency check only works with stored data, charts and couplings, which means these have to be saved before the check.

---

The results of the consistency check can also be exported in a tab-separated text file. To do so, activate the button in the consistency check editor as shown in the figure below.





## Automatic control interface

### 6.1 Overview

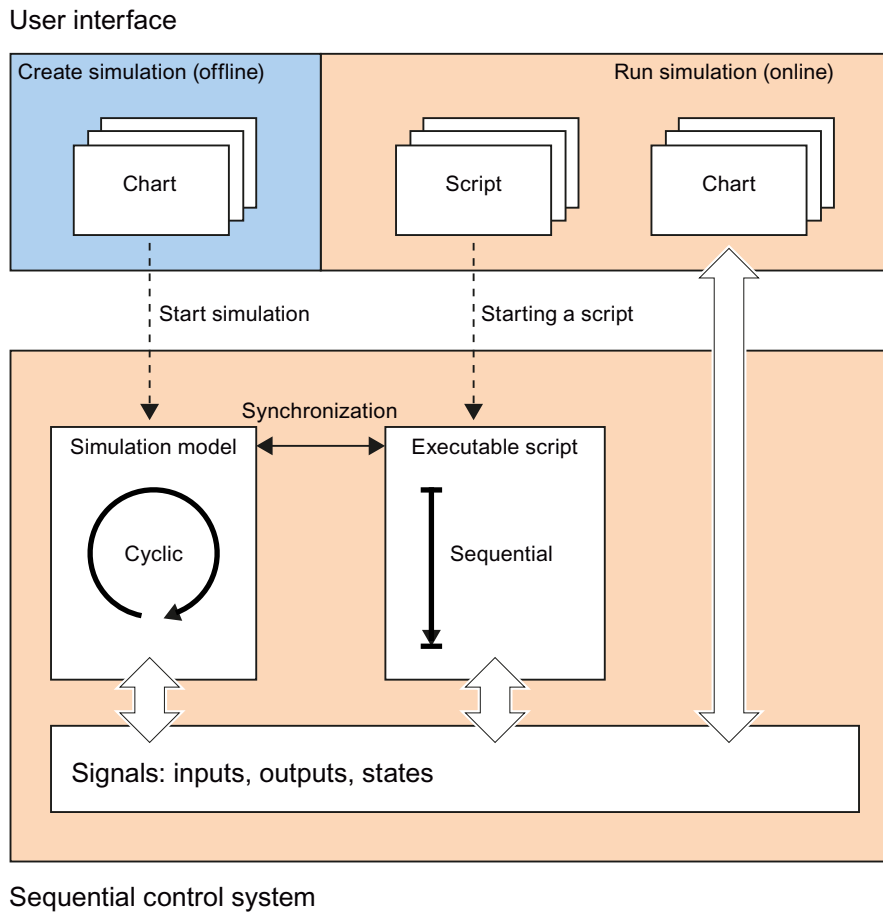
A simulation is made functional in SIMIT by defining the graphical interconnection and parameter assignment of components. When the simulation is started, all the charts of a simulation project are brought together to generate an executable simulation model that is then processed cyclically. The SIMIT user interface gives you access to all input and output signals, statuses and component parameters that can be changed online during ongoing simulation.

The Automatic Control Interface function module from SIMIT provides you with the additional option of using scripts to access simulation variables at specific times in ongoing simulations. You are then able to automate manual interventions through the user interface, monitor processes within the simulation, and create logs from output information.

A script is a sequence of instructions that are processed in consecutive order and without jumps from beginning to end. Time slices are used to synchronize the script and the cyclically executed simulation model. Within each time slice, the script is executed before the simulation model contained in this time slice.

You can stop a script to wait for specific events in the simulation model or for specific points in time. You can also use instructions in the script to influence the control system of the simulation model, such as creating and loading snapshots.

The schematic figure below shows how scripts are incorporated into the architecture of SIMIT:



## 6.2 Handling of scripts

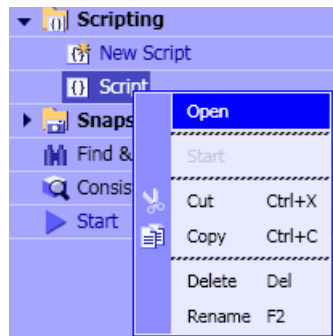
### 6.2.1 Creating a script

You can both create and execute scripts in your simulation project. The script editor can be used to edit scripts.

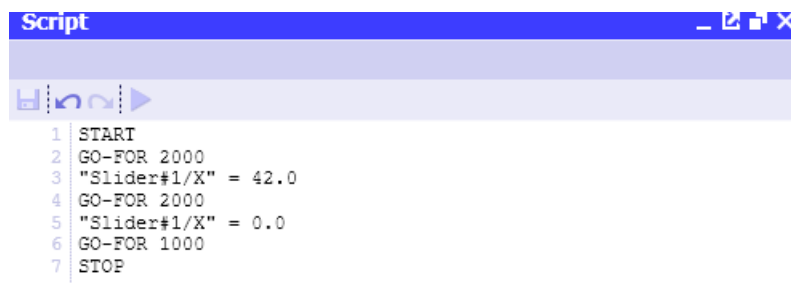
Scripts are elements of a SIMIT project and are managed in the *Scripting* folder of the project tree. You can create and edit scripts whenever SIMIT is open, which means when the simulation has not yet been started (offline) as well as when it has been started (online). Double-click on *New Script* in this folder to create a new script.



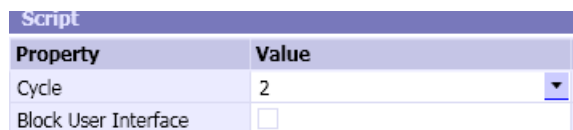
To edit a script, open the script editor from the shortcut menu or double-click on the script in the navigation window.



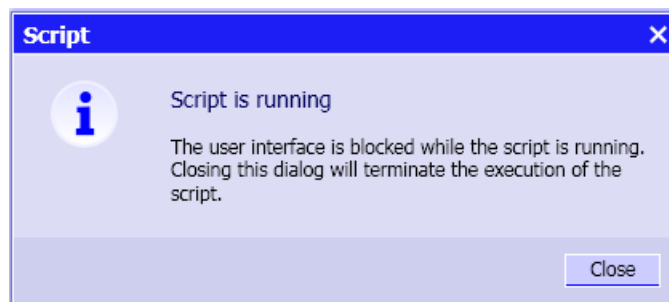
The editor that then opens lists the line number in a column to the left of the pane in order to facilitate the allocation of error messages.



Enter the time slice in which the script is to be executed in the Property view of the script editor. If you include other scripts by means of an instruction, the time slice specified in the included scripts is irrelevant; the applicable time slice is the one specified in the script that was started.





If you enable the *Block interface* option, a dialog like the one shown in the following figure appears when the script is started. This dialog prevents users from accessing the currently running simulation through the interface. When you close this dialog the script is canceled.



### 6.2.2 Executing a script

A script can only be executed when a simulation is running. You can start a script either through its shortcut menu in the project tree or by using the toolbar in the script editor.



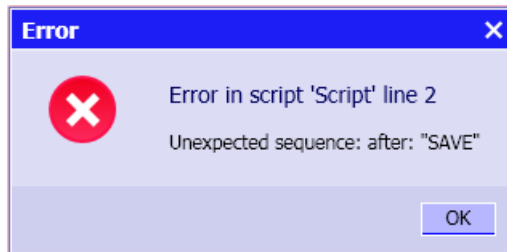
If a script is started, this is indicated by the flashing symbol  in the status bar. The  symbol for the ongoing simulation is also shown.

You can stop a script at any time using the shortcut menu of the *Scripting* folder:



Only one script can be run at a time. Execution of a script is stopped once all instructions in the script have been completed, when it is stopped by a command or when the simulation is closed or reinitialized in the interface.

After a script is started, the syntax is checked and the script is compiled into an executable instruction list. If an error is discovered in the script, this is flagged and the script is not started.



Problems can also arise that only become apparent once the script is being executed. Should such problems occur, a message appears in the status bar and the script is stopped.

`00:01:42:800 Wrong simulation state, script 'Script', line 1.`

### 6.2.3 External scripts

---

#### Note

We recommend that you only create and edit scripts using the SIMIT script editor, as this will ensure that they remain in the simulation project. This is because only scripts located in the SIMIT project tree can be executed and archived with the project.

---

However, if you wish to create scripts externally using a different editor, note the following:



Scripts are text files with the standard Windows character encoding (code page 1252). Script files must always have the extension *.script*. The script properties that are listed in the property view of the script editor must be set in the first two lines of the script file as per the details listed below in the table "Keywords for script properties":

Keyword = Value

Example:

**META\_BLOCKGUI = False**

**META\_CYCLE = 2**

Table 6-1 Keywords for script properties

Keyword	Meaning
META_BLOCKGUI	True: The user interface is blocked while the script is being executed False: The user interface is not blocked while the script is being executed
META_CYCLE	Cycle (1 to 8) in which the script is to run

You can find additional information on this in the section: Composing scripts (Page 283).

#### Note

When you archive a project, only the scripts located in the simulation project are archived.

External scripts are not archived with the project. The archived project is therefore incomplete.

## 6.3 The script syntax

### 6.3.1 Controlling the script

#### General

You can exit a running script using an instruction in the script or by initiating a dialog during the script runtime to display various processing options.

#### Canceling the script

The instruction  
BREAK

is used to cancel the processing of a script. If the *BREAK* instruction is used in an included script, processing resumes in the calling script.

#### Terminating the script

The instruction  
QUIT

is used to terminate the script. If the *QUIT* instruction is in an included script, the calling script will also be terminated.

### The query dialog

The instruction

```
DIALOG "text" MODE
```

is used to pause the simulation and a query dialog with the content text is opened. The *Mode* parameter can be used in conjunction with the "Buttons for dialogs" table to determine which buttons this dialog will have.

Table 6-2 Buttons for dialogs

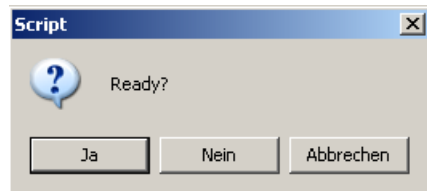
Mode	Button			
	Yes	No	OK	Cancel
YESNOCANCEL	X	X	-	X
OKCANCEL	-	-	X	X
YESNO	X	X	-	-
OK	-	-	X	-

*YESNOCANCEL* is the default mode. Pressing *Cancel* in the dialog will terminate the script. The simulation stays paused. In all other cases, the simulation is restarted and script execution resumes.

In the example

```
DIALOG "Ready?" YESNOCANCEL
```

the dialog shown in the following figure appears.



If a log file was already opened in the script before the *DIALOG* instruction, it will be noted in this file whether the dialog was closed using the *Yes* or *No* button. Based on this, the entry **DIALOG text True**

or

```
DIALOG text False
```

is made in the log file.

In the script, a *YESNOCANCEL* dialog with the binary variables

```
_result
```

and

```
_ok.
```

can be used to query which decision the user has made.

The variable `_result` has the value *True* if the most recently opened dialog was closed with "Yes", and *False* if it was closed with "No". The variable `_ok` is also set to *False* if a dialog was acknowledged with "No". This variable is not automatically reset to *True*, but this can be performed using an instruction if desired.

## 6.3.2 Composing scripts

You can compose a script from several separate scripts. The instruction below means that the current script will continue with another script:

```
INCLUDE "Name"
```

The name of the script is entered relative to the calling script. Backslashes in the name must be doubled.

Example:

```
INCLUDE "..\subscripts\test2"
```

The *INCLUDE* instruction is executed once the included script has ended.

Multiple scripts can also be linked into chains in which the included scripts also contain *INCLUDE* instructions. The maximum depth of such a chain is limited to 10.

External scripts can also be included by specifying an absolute path and the name of the script file:

```
INCLUDE "C:\scripts\test3"
```

The included script file must have the file extension *.script*. The file extension is not specified in the file name of the *INCLUDE* instruction.

---

### Note

Once a script is started, an executable instruction list is created for this script and all the included scripts. The included scripts must therefore exist before the calling script is started; changes made to an included script after the calling script is started have no effect.

---

### Note

When you archive a project, only the scripts located in the simulation project are archived.

External scripts are not archived with the project. The archived project is therefore incomplete.

---

## 6.3.3 Commenting scripts

Comment lines start with two forward slashes. They are ignored when the script is executed.

Example:

```
// This is a single line comment
```

You can also comment out entire areas in the script by starting a comment with */\** and ending it with *\*/*.

### 6.3 The script syntax

Example:

```
/* This is a  
multiline comment */
```

You cannot close one comment area and open a new one in the same line.

#### 6.3.4 Signals in scripts

You can use scripts to access inputs, outputs, states and simulation model parameters that can be changed online. Signals are identified in SIMIT with the *source* of the signal and the *signal name*. The signal designation comprises both items separated by a forward slash and framed by quotation marks as follows:

```
"Source/signal name"
```

Example:

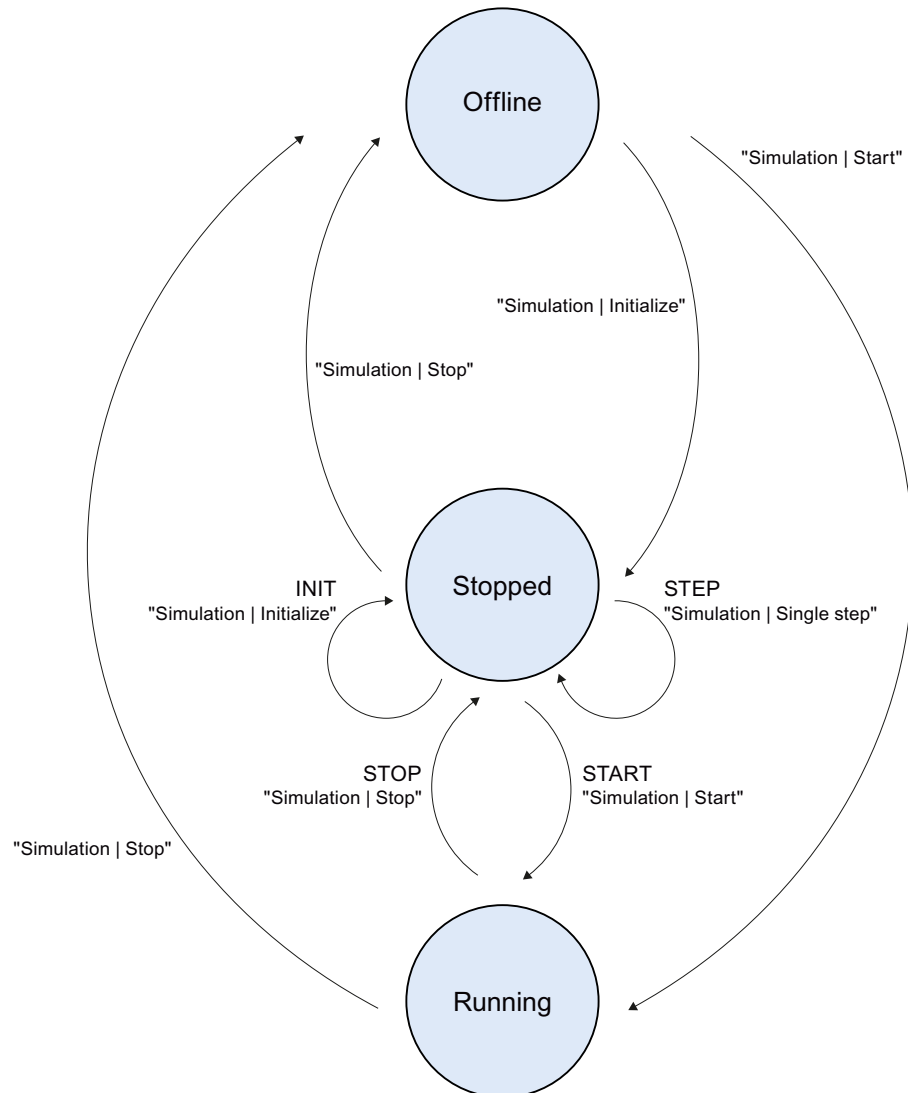
```
"ADD#1/IN1"
```

The following rules apply in the unusual event that the source or signal name contains double quotes or forward slashes:

- All double quotes must be preceded by a '\
- If the signal name itself contains a forward slash, then all forward slashes except for the delimiter between source and signal must be preceded by a '\
- If the source name contains a forward slash but not the signal name, then no marking is required. This is due to the fact that the last forward slash in the signal name is assumed to be the delimiter which would result in a correct result in this case.

### 6.3.5 Controlling the simulation

A simulation is always in one of the following three states: "Offline", "Stopped" or "Running".



The simulation state can be changed by a script. For example, the simulation can be stopped and started again. However, script commands are only effective if the simulation state is "Stopped" or "Running". Scripts cannot start an "Offline" simulation, nor can they end a simulation.

#### 6.3.5.1 Initializing a simulation

The instruction  
INIT

is used to initialize the simulation. This instruction is the same as the *Simulation | Initialize* command in the SIMIT menu.

This command is invalid if the simulation has already been started. In this case, the following error message appears in the status bar and the script is terminated:

**Wrong simulation state, Script '...', Row ... .**

---


**Note**

The */NIT* script command cannot cause the transition from "Offline" to "Online", because a script can only be executed in online mode.

---

### 6.3.5.2 Starting the simulation

The instruction  
`START`

is used to start the simulation. This instruction is the same as the *Simulation | Start* command in the SIMIT menu and the  button on the SIMIT toolbar.

A *START* instruction in the script is ignored if the simulation has already been started.

---

**Note**

The *START* script command cannot cause the transition from "Offline" to "Online", because a script can only be executed in online mode.

Starting with this command automatically sets the simulation time to real-time (100%).

---

### 6.3.5.3 Starting and waiting until an absolute time

The instruction  
`GO-TO time`

is used to pause the script until the simulation time is equal to or exceeds *time* in milliseconds.

The instruction is ignored if the simulation time is already equal to or greater than *time*. If the simulation is initialized but has not yet been started, this command will cause the simulation to begin.

### 6.3.5.4 Starting and waiting until a relative time

The instruction  
`GO-FOR time`

is used to pause the script until *time* in milliseconds has passed.

If the simulation is initialized but has not yet been started, this command will cause the simulation to begin.

### 6.3.5.5 Starting and waiting for a certain number of time slices

The instruction  
`GO n`

is used to pause the execution of the script for  $n$  time slices. The time slices are counted in the script time slice. The parameter  $n$  can be omitted if the script is to be paused for just one cycle, i.e. for  $n=1$ .

If the simulation is initialized but has not yet been started, this command will cause the simulation to begin.

### 6.3.5.6 Starting and waiting for an event

The instruction

```
GO-UNTIL Condition TIMEOUT time
```

is used to pause the script until the *Condition* is met or the *TIMEOUT* time specified in the simulation in milliseconds has expired.

Example:

```
GO-UNTIL "OR#1/Y" TIMEOUT 10000
```

You do not have to enter a time value for *TIMEOUT*. The script would then no longer close of its own accord if the condition was not met.

If the simulation is initialized but has not yet been started, this command will cause the simulation to begin.

### 6.3.5.7 Stopping the simulation

The instruction

```
STOP
```

is used to stop the simulation. This instruction is the same as the *Simulation | Stop* command in the SIMIT menu.

A *STOP* instruction in the script is ignored if the simulation has already been stopped.

### 6.3.5.8 Executing a single step

The instruction

```
STEP
```

is used to carry out a single step in the simulation. This instruction is the same as the *Simulation | Single step* command in the SIMIT menu. The function key F12 is assigned to this function.

This instruction is invalid if the simulation is already running cyclically. Otherwise the following error message appears in the status bar and the script is terminated:

**Wrong simulation state, Script '...', Row ... .**

---

#### Note

A simulation step is always one step of the fastest submodel, which means the submodel with the shortest cycle time.

---

### 6.3 The script syntax

#### 6.3.5.9 Saving a snapshot

The instruction  
`SAVE-IC "name"`

is used to save a snapshot in the simulation project under the name *name*.

The snapshot is always saved at the highest level in the *Snapshot* folder. The specifying of subfolders is not permitted.

<b>NOTICE</b>
---------------

Snapshots with the same name that already exist in your simulation project will be overwritten without any warning prompt and any data saved in this snapshot will be lost.
---

#### 6.3.5.10 Loading a snapshot

The instruction  
`LOAD-IC "name"`

is used to load a snapshot with the name *name* from the *Snapshot* folder of the simulation project .

The specifying of subfolders is permitted. Any backslashes in the name must be doubled.

Example:

```
LOAD-IC "name"  
LOAD-IC "subfolder\\name"
```

If no snapshot with this name exists, the following error message appears in the status bar and the script is terminated:

**Snapshot '...' cannot be loaded, the file may not exist, Script '...', Line ... .**

After a snapshot is loaded, the simulation is stopped and must be restarted using either the *STEP* or *START* instruction.

#### 6.3.5.11 Resetting the simulation time

The instruction  
`SIMTIME-RESET`

is used to reset the simulation time to zero. The simulation state remains unchanged.

---

**Note**

Resetting the simulation time causes inconsistencies in the trends of the Trend and Messaging Editor. This is because all the values in the trend are recorded over the simulation time.

---



## 6.3.6 Logging

You can log the results of your simulation, which means signal values or events. Script instructions can be used to create log files in which outputs are documented.

Script instructions to output data are ignored if no log file is open when the script is executed.

### 6.3.6.1 Opening and closing log files

The instruction  
`OPEN-LOG "name"`

is used to open a log file with the name *name*. Any log file that may already be opened is closed first. The log file must be entered with its absolute path. Any backslashes in the name must be doubled.

Example:  
`OPEN-LOG "c:\\protocol\\log.txt"`

---

#### Note

Any log file that already exists with the same name will be overwritten without warning. Data that is already saved in the log file will be lost.

---

You must close the log file either at the end of the script or at the point where you no longer wish to record any more data.

The instruction  
`CLOSE-LOG`  
closes the log file.

---

#### Note

The outputs are not saved in the log file if the log file was not closed by a instruction before the end of a script. In this case, the log file is empty.

---

### 6.3.6.2 Unformatted output

The instruction  
`PRINT "Signal"`

saves the current value of the signal *Signal* in the open log file. Inputs, outputs, states and component parameters that can be changed online may be used as signals. Binary values are output as *True* or *False*.

### 6.3.6.3 Formatted output

The instruction  
`PRINTF "Formatstring", "Signal1", "Signal2", ...`

## 6.3 The script syntax

saves the current values of the signals listed in the instruction in the open log file. The *Formatstring* contains the format specifier for each signal; it must contain precisely one format specifier for each signal. The format string may also contain text. The allowed format specifiers are listed in the table below.

Table 6-3 Formatting instructions

Signal type	Formatting instruction
analog, integer, binary	%f or %.nf (n: number of decimal places), binary values are output as "0" or "1".
integer	%i
binary	%b (values are output as "True" or "False")

A backslash must be placed in front of the percentage sign in the format string if you would like the percentage sign to be output as a normal character: \%. Use the *PRINTF* instruction without a signal specification if you want to output a fixed text only.

Signals can be any input signals, output signals, states or component parameters that can be changed online. You can find additional information on this in the section: Signals in scripts (Page 284).

In the example

```
PRINTF "Ramp: %.2f [ULR=%b] [LLR=%b]", "Ramp#1/Y", "Ramp#1/ULR",
"Ramp#1/LLR"
```

the following is output to the log file:

```
Ramp: 0.00 [ULR=False] [LLR=True]
```

## 6.3.6.4 Outputting time and date

You can enter the time and date of your computer into the log file using the system variables listed in the following table:

Table 6-4 System variables

Syntax	Meaning
_t_day	Day
_t_mon	Month
_t_year	Year
_t_hour	Hour
_t_min	Minute
_t_sec	Second

All figures appear as two digits.

For example, the following instructions

```
PRINTF "Date: %.0f.%.0f.%.0f", "_t_day", "_t_mon", "_t_year"
PRINTF "Time: %.0f:%.0f:%.0f", "_t_hour", "_t_min", "_t_sec"
```

generate the following entries in the log file:

```
Date: 20.06.11
```

Time: 12:55:13

---

**Note**

The system variables in the "System variables" table can only be used in the context of logging and cannot be used for general calculations within the script.

---

### 6.3.6.5 Outputting version information

You can output the version of your current project and the currently used SIMIT version to the log file using the system variables `_ProjectVersion` and `_SIMITVersion`.

For example, the following instructions

```
PRINTF "Project version: %s", "_ProjectVersion"
```

```
PRINTF "SIMIT version: %s", "_SIMITVersion"
```

generate the following entries in the log file:

```
Project version: AA12345-344332-3.4
```

```
SIMIT version: 7.1.0
```

### 6.3.6.6 The `_printlog` system function

It is also possible to write directly from components to an open log file. However, to do this you will have to modify the behavior description of the relevant component types, which means the system function

```
_printlog("string");
```

must be inserted in order to output text. When called, this function writes the text in the *string* directly to the open log file.

---

**Note**

The SIMIT component type editor (CTE) is required to edit the behavior description of component types. The component type editor is a SIMIT expansion module with which you can create your own component types.

---

## 6.3.7 Signal curves

### 6.3.7.1 Overview

In the script, you can specify signals whose values are to be written cyclically to a plot file. Only one plot file at a time can be open in a script.

6.3 The script syntax

6.3.7.2 Opening and closing a plot file

The instruction

```
OPEN-PLOT "File name"
```

opens the plot file *File name* and starts to record the signals. The name of the file (*File name*) must be specified with its absolute path, in which any backslashes must be doubled.

A plot file that is already open is closed first. If the file *File name* already exists, it will be overwritten without confirmation.

**Note**

The signals to be recorded must be specified before the file is opened.

The first line of the plot file shows the signal names of the signals to be recorded. The remaining rows contain the simulation time in milliseconds, and tab-delimited signal values in the sequence that is specified in the first row.

The instruction

```
CLOSE-PLOT
```

closes the plot file and the list of signals to be recorded is deleted.

The instruction sequence

```
STOP
INIT
START
PLOT "Ramp#2/Y"
PLOT "Ramp#2/ULR"
PLOT-CYCLE 500
OPEN-PLOT "D:\\plot.txt"
GO-FOR 4000
CLOSE-PLOT
```

would add the following to the plot file:

	Ramp#2/Y	Ramp#2/ULR
0	1.666666666666667	Ramp#2/ULR
500	18.33333333333333	False
1000	35.0	False
1500	51.66666666666667	False
2000	68.33333333333333	False
2500	85.0	False
3000	100.0	True
3500	100.0	True
4000	100.0	True

6.3.7.3 Specifying signals

The instruction

```
PLOT "Signalname"
```

adds the signal *Signalname* to the list of signals to be recorded. Please note that the list of signals to be output must be completely created before the plot file is opened.

You can record input signals, output signals, states and component parameters that can be changed online within your simulation project.

#### 6.3.7.4 Specifying a cycle

The instruction  
`PLOT-CYCLE time`

specifies the cycle in which the signals are to be recorded. The cycle time *time* specified in milliseconds is rounded up to the next multiple of the script cycle time. If you do not add this instruction to your script, the script cycle time is used for the recording. Please note that the cycle must be specified before the plot file is opened.

#### 6.3.8 Setting signals

Input signals, state variables and component parameters that can be changed online can be set by instructions in the script. You can also set these variables to pre-defined values or apply a linear change to the values.

---

##### Note

If you set inputs that are linked to the outputs in the simulation project or give values for states of a component that are set in the component itself, the value specified in the script will either be completely ignored or will only be effective during one computing time slice.

---

#### 6.3.8.1 Setting individual values

The instruction  
`"Signal" = Value`

sets the specified *Signal* to the given *Value*.

Example:  
`"Status/1/BI" = True`

Instead of a constant value, you can also specify an expression for the value. An expression consists of constants or signals that are linked to each other by operators. The permitted

6.3 The script syntax

arithmetic operators are listed in the "Arithmetic operators" table, and the permitted Boolean operators are listed in the "Boolean operators" table.

Table 6-5 Arithmetic operators

Operator	Meaning
( )	Parentheses
*	Multiplication
/	Division
+	Addition
-	Subtraction

Table 6-6 Boolean operators

Operator	Meaning
( )	Parentheses
!	not
&&	and
	or

The operators are listed in both tables from highest to lowest priority.

Example:

```
"Display/E" = 2 * ("Slider/A" + 0.5)
```

**Note**

A value that you have assigned to a signal will not be available until the next execution time slice. If you have set a signal using an instruction in the script, for example, and you want to use this signal to assign a value or output its value in the instruction that immediately follows, the signal still has the value that it had at the beginning of the cycle.

**6.3.8.2 Separating connected signals**

You can use the

`FIX "Signal"`  
instruction to disconnect the specified signal ("force") to then be able to set a value.

Example:

```
FIX "Status/1/BI"  
"Status/1/BI"=True
```

You can use the

`UNFIX "Signal"`  
instruction to revoke the disconnection of the specified signal.

Example:

```
UNFIX "Status/1/BI"
```

The two commands, `FIX` and `UNFIX`, correspond to the operation of the signal splitter on the user interface.

### 6.3.8.3 Defining a signal curve

The instructions

```
RAMP "Signal" FROM start TO end IN time
RAMP "variable" TO end IN time
```

allows you to set an *analog* input signal or the state of a component to a linearly rising or falling value. This instruction sets the value *start* and causes the value to increase or decrease so that it reaches the value *end* within *time* milliseconds it, which means the value changes linearly (ramp).

There is no start value (FROM start) in the second instruction. The start value in this case is the current value.

Script execution continues during the specified time *time*, which means other instructions in the script are executed without waiting for the signal to reach its specified end value. A script is not terminated until all signals that were set using ramps have reached their end values.

Examples:

```
RAMP "Display#1/X" FROM 20.0 TO 10.0 IN 15000
```

### 6.3.9 Conditional execution

The instructions

```
IF expression THEN
    block1
ELSE
    block2
ENDIF
```

allow you to set conditions for the execution of instructions. If the condition *expression* is true, the instruction list *block1* is executed; otherwise list *block2* is executed. Both instruction lists comprise one or several rows containing any instructions you wish, and may therefore also contain conditional instructions. The conditional instruction can be nested.

The *ELSE* instruction can also be omitted:

```
IF expression THEN
    block
ENDIF
```

Note that *every* IF instruction must be closed with an *ENDIF* instruction.

Examples:

```
IF "Button/1/Z" == True THEN
    "Display/E" = 2.0 * ("Slider/A" + 0.5)
ELSE
    "Display/E" = "Slider/A"
ENDIF
```

```
IF "Button#1/Z" THEN
    "Display/E" = 2.0 * ("Slider/A" + 0.5)
ELSE
```

## 6.3 The script syntax

```

    IF !"Button#2/Z" THEN
        "Display/E" = 3.0 * ("Slider/A" + 0.5)
    ELSE
        "Display/E" = "Slider/A"
    ENDIF
ENDIF

```

The *expression* condition can be used to access component input signals, output signals as well as and states and parameters that can be changed online. You may choose from the following comparison operators:

Table 6-7 Comparison operators

Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

**Note**

When comparing analog signals for equality/non-equality, please note that a floating-point format is used for these signals. The values calculated for these signals may therefore differ from the expected values due to rounding, and comparisons with them may return unexpected results.

A condition can be made up of several expressions that are linked by Boolean operators. You can find additional information about the Boolean operators in the tables in section: Setting individual values (Page 293).

### 6.3.10 Accessing the simulation time

The system variable

`_Time`

contains the current simulation time in milliseconds.

In the example

```

IF ("_Time" > 20000) THEN
    PRINTF "Simulation time: %.0fms", "_Time"
ENDIF

```

the output sent to the log file might be:

**Simulation time: 26600ms**



## Libraries

### 7.1 The basic library

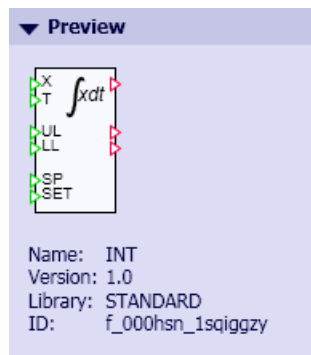
#### 7.1.1 General

##### 7.1.1.1 Introduction

The basic library of SIMIT contains elementary functions for creating simulations, i.e. for modelling plant and machine behavior. These functions are provided in the form of component types and controls. This following sections describe in detail the individual component types and controls contained in the SIMIT basic library.

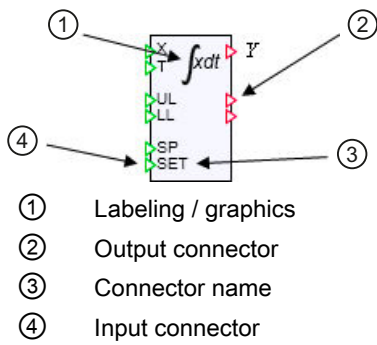
##### 7.1.1.2 Component symbols

Component types are instantiated as components in order to create a simulation. To do this, simply select the required component type with the mouse and drag it into a chart. Each instance of a component is represented by a type-specific symbol in a chart. The symbol for a component type and its version and ID are displayed in the preview in the figure below. Simply click on the component type in the library to select it.

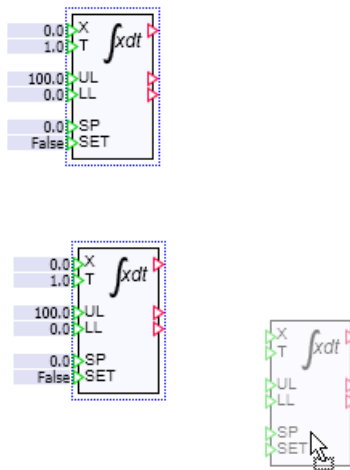


Every symbol has connectors with names and a label or graphic that clearly shows the function of the component on charts (see the figure below). The symbols are designed so that the function of both the component and the connectors can be understood intuitively.

7.1 The basic library

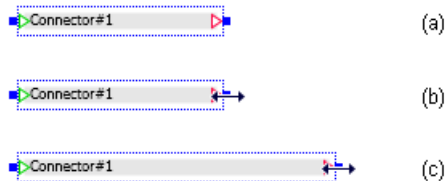


Components are represented by the type-specific symbol in charts. Click on the relevant symbol to select a component in a chart. A blue frame then appears around the symbol for the selected component. Simply hold down the mouse button to drag the symbol and to move it within the chart.

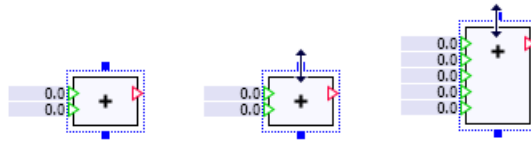


Some components have handles on the selection frame. These handles are used to change the size of the symbol.

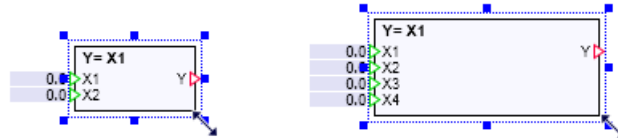
Components such as connectors have handles on the left and right of the selection frame as shown in the figure below under (a). When you roll the cursor over the handles, its appearance changes as shown in the figure below under (b). Hold down the left mouse button to move the handles and thus change the width of the symbol as shown in the figure below under (c).



For components such as *ADD*, the selection frame has handles at the top and bottom. These handles are used to adjust the height of the symbol to suit the number of inputs. Simply click on the top or bottom gripper, hold down the left mouse button and drag it up or down.

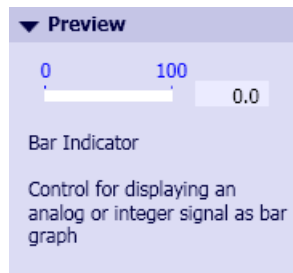


The formula components have handles on all sides and at every corner of the selection frame. These allow you to adjust both the width and the number of inputs. The handles at the corners of the symbol allow you to make these two settings at the same time.



### 7.1.1.3 Symbols for the controls

When you create a simulation, controls are handled in the same way as component types, which means they are positioned in a chart with their symbol. Controls that you selected from the library appear in the preview with their symbol, their designation and a brief description of their function.



When the simulation starts, controls act as active elements and they are represented accordingly as active controls by their symbols as seen in figure below under (b). If there is no active simulation, the symbols represent passive controls and are displayed as such as seen in figure below under (a).



Controls, like components, are represented by the type-specific symbol in charts. Simply click on a symbol to select it. The symbol of the selected control then appears with a blue selection frame; simply hold down the left mouse button to drag the symbol and move it within the chart. Use the handles on the selection frame to change the size of the symbol.

### 7.1.1.4 Component connectors

Component connectors from the Basic Library are inputs or outputs. Inputs (green triangles) are arranged on the left and outputs (red triangles) appear on the right of the symbol. To visually emphasize the direction in which the connectors work, the input triangles point into the symbol while the output triangles point out of the symbol.

Inputs and outputs that belong together from a functional point of view are arranged opposite one another in the symbol for a component if possible. In the above example of the integrator, these are input  $X$  and integrator output  $Y$  for example. Inputs and outputs that belong together from a functional point of view are also grouped together and are separated from other groups by spaces. This means that the functional interactions created by interconnecting components can be more easily identified in charts. In the example from the section: Component symbols (Page 297), the following three groups are formed:

- Inputs  $X$  and  $T$  for calculating the integral value at output  $Y$ ,
- the limits  $UL$  and  $LL$  with their binary feedback and
- set point  $SP$  and set command  $SET$  for setting the integrator output.

The function described by the integral

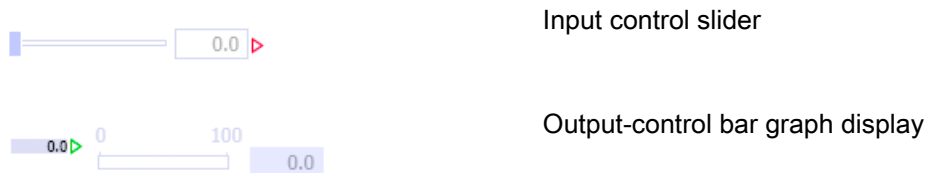
$$Y = \frac{1}{T} \int X dt$$

can thus easily be assigned to connectors  $X$ ,  $T$  and  $Y$ . Connectors are only given names in the symbol if the function of that connector is not obvious.

All inputs and outputs are binary (logical), analog or integer inputs or outputs. Complex connector types are not used in the components of the Basic Library. One exception is the *PROFdrive* type connector that is used to connect the header component and the device-specific component in the PROFIDRIVE library. The values of the binary inputs or outputs are designated by zero and one or, alternatively, by False and True.

### 7.1.1.5 Connectors for controls

Controls for entering signals have only one output as their connector in form of a red triangle on the right side of the symbol (see the table below). A green triangle on the left side of the symbol identifies the input to the display controls. The signal splitter control has only one connector which is always invisible.



As with components, the connectors for controls are either binary (logical), analog or integer inputs or outputs.

### 7.1.1.6 Connecting connectors

The connectors for controls and components can be connected to one another if the following rules are observed:

1. Only inputs may be connected to outputs.
2. An output can only be connected to one input, while an input may be connected to multiple outputs.
3. Connectors to be connected must be of the same type.

Connectors can be connected in different ways:

- Connecting line
- Overlapping connectors
- By implicit connections.

In the first two cases, the connection is made graphically in the work area of the chart editor. The chart editor is designed so that the rules for connecting connectors defined above are automatically followed.

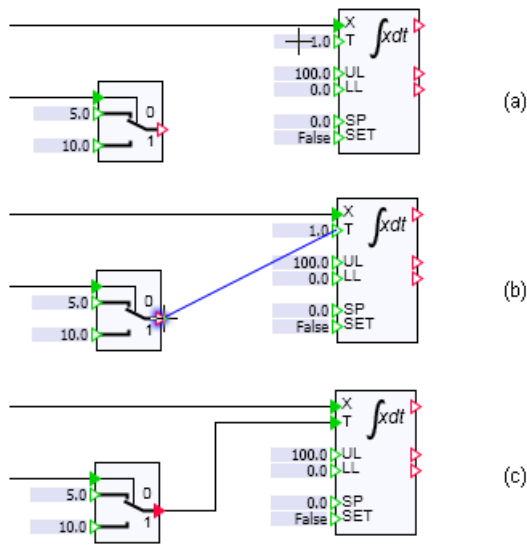
Implicit connections are made by changing the settings in the properties of the inputs or outputs of components to be connected.

### Connecting with connecting lines

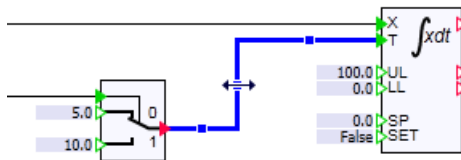
If you want to create a connection between the output of the *Selection* component and the input *T* of the integrator, for example, move the cursor over one of the two connectors. When the cursor changes to a cross, as seen in the figure below under (a), you can click or hold down the mouse button to create the connection. If you then move the cursor, a blue rubber band indicates the connection between the connector and the cursor.

Now move the cursor over the connector to be connected. When the connector to be connected is visually highlighted, as seen in the figure below under (b), the connection can be completed. If you held down the mouse button to open the connection, simply release the button to close the connection. If you opened the connection with a mouse click, simply click on the highlighted connector to close the connection. When the connection is closed, the rubber band is automatically replaced with a connecting line with right angles and the triangles of the connected connectors are filled in with color, as shown in the figure below under (c).

7.1 The basic library



To delete connecting lines, first click on the connecting line to be deleted. The connecting line changes to a thick blue line as shown in the figure below and can be deleted by selecting *Edit | Cut* from the menu bar or by pressing the *Del* delete button.

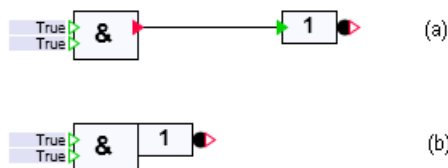


If a connection line selected, this connection can be manually edited.

Connecting by superimposing connectors

Connecting by superimposing connectors is done by positioning two components and/or controls to be connected on the chart so that the input of one component lies directly above the output of the other component to be connected.

The figure below compares this type of connection (b) with the connecting line method (a). The two connectors that are connected by superimposition become invisible.



**Note**

If the connectors do not become invisible when they are overlapping, the connectors are not of the same type and thus cannot be connected.

**Implicit connections**

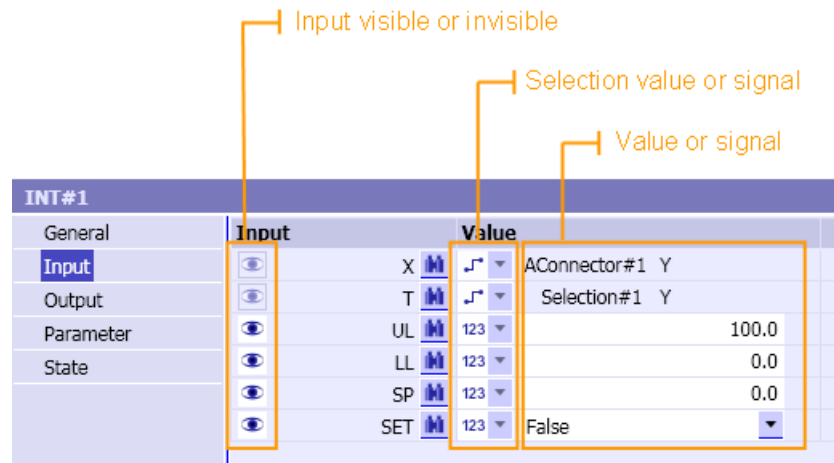
Implicit connections are made by changing the settings in the properties of the inputs and outputs to be connected of components and/or controls. To do this, open the property view of the component (see figure below) and follow these steps:

1. Make the input or output to be connected invisible (👁️).  
Click the 👁️ or 👁️/🚫 symbol to toggle between input visible and input invisible.

**Note**

An invisible input or output is not displayed on the symbol for the component and thus cannot be connected to another output or input using connecting lines.

2. Set the selection field for value/signal specification to signal specification (📡).
3. Enter the signal to be connected with component name (source) and connector name.



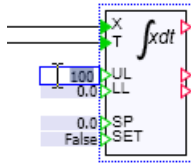
The procedure is the same for a control. Because a control does not allow you to select a value/signal, the second step is omitted and the output to be connected may be entered directly after toggling to invisible.

**7.1.1.7 Setting inputs**

Unconnected inputs may be preassigned with values. You can enter the value in the

- connector box for the input on the chart or in the
- property view for the component

Double-click in the connector box to open the field for entering the value (see figure below). To complete your input, either press Return or click in the chart outside the connector box.



To enter a value in the property view, navigate to the relevant input and click in the input field to open it (see figure below). To complete your input, either press Return or click in the property view outside the input field.

INT#1		Input	Value
General			
Input	<input type="checkbox"/>	X	AConnector#1 Y
Output	<input type="checkbox"/>	T	Selection#1 Y
Parameter	<input type="checkbox"/>	UL	123 100.0
State	<input type="checkbox"/>	LL	123 0.0
	<input type="checkbox"/>	SP	123 0.0
	<input type="checkbox"/>	SET	123 False

Input of True and 1 or False and 0 are equivalent for binary parameters. Binary values are always displayed as True or False.

You can set inputs in the ways described above even when the simulation has started. In this case, however, the input will only take effect for the duration of the started simulation, which means modified input values are reset to their original values when the simulation ends.

### 7.1.1.8 Properties of components



The properties of components can be accessed in the properties window. To display the properties of a component in the property view, right-click or left-click on the component. If a simulation is running, you can only display the properties by right-clicking. The properties of a component are divided into:

- General properties
- Properties of the inputs
- Properties of the outputs
- Parameters
- States



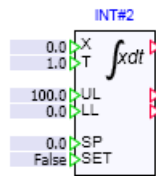
## General properties

General properties of components are the name and the time slice of the component, the unique identifier (UID) of the component type, the position of the component and the width and height of the symbol.

INT#1		
General	Property	Value
Input	Name	INT#1 
Output	Cycle	2 
Parameter	Show Name	<input type="checkbox"/>
State	UID	f_000hsn_33dc54dw
	Position	X: 365.0 Y: 55.0
	Width	50.0
	Height	90.0

The name must be unique for all components and controls used in the project, which means the project must not contain multiple components and/or controls with the same name. When you drag a component from the library onto a chart, it is automatically assigned a name. This name is made up of the designation of the component type and a number for the component type that is unique across the entire project.

Check the *Show Name* check box to display the name of the component on the chart.



### Properties of the inputs

Inputs may be visible or invisible and the signal linked to an input or the input value may be set.

INT#1				
General	Input		Value	
Input	<input type="checkbox"/>	X	<input type="checkbox"/>	AConnector# Y
Output	<input type="checkbox"/>	T	<input type="checkbox"/>	Selection#1 Y
Parameter	<input type="checkbox"/>	UL	123	100.0
State	<input type="checkbox"/>	LL	123	0.0
	<input type="checkbox"/>	SP	123	0.0
	<input type="checkbox"/>	SET	123	False

Each input has the following properties:

- Visibility**  
 In the first column, you can toggle between Input visible (👁️) and Input invisible (👁️/❌). If the input is connected with a connecting line, you cannot toggle between them. In the figure above, for example, the two first inputs *X* and *T* are connected; they can therefore not be set to invisible.
- Name**  
 The name of the input is displayed right-justified in the second column.
- References**  
 The third column is used to search for references, which means for objects that use this input.
- Selection value or signal**  
 The fourth column is used to toggle between value (123) or signal (📡) as the selection for the input. This selection is not active if the input is connected with a connecting line.
- Value or signal**  
 In the fifth column, the input value is set if value was selected. If signal was selected, the output connected with the connecting line is displayed or the output to be connected implicitly may be set.

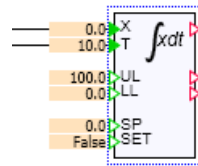
The representation of the properties changes when the simulation is running.

INT#1			
General	Input		Value
Input	123	X	0
Output	123	T	10
Parameter	123	UL	100
State	123	LL	0
	123	SP	0
	123	SET	False

In this case input values are always displayed instead of signals, and it is possible to set each input value.

- **Display On/Off**

In the first column, the value at the input of the component can be shown (123) or hidden (123). When the simulation is started, the display is switched on for non-connected inputs and switched off for connected inputs. The display cannot be switched on for invisible inputs. The figure below shows a component in which the display is switched on for all input values, which means even for values at connected inputs:



- **Forcing input**

In the fourth column you can switch forcing on (T) or off (F) for every connected input.

- **Input value**



The fifth column is used to display or set the input value.

## Properties of the outputs

INT#1			
Allgemein	Name	References	Wert/Signal
Eingang	Y	[Visibility Icon]	
Ausgang	ULR	[Interconnection Icon]	BFormula# X1
Parameter	LLR	[Interconnection Icon]	
Zustand			

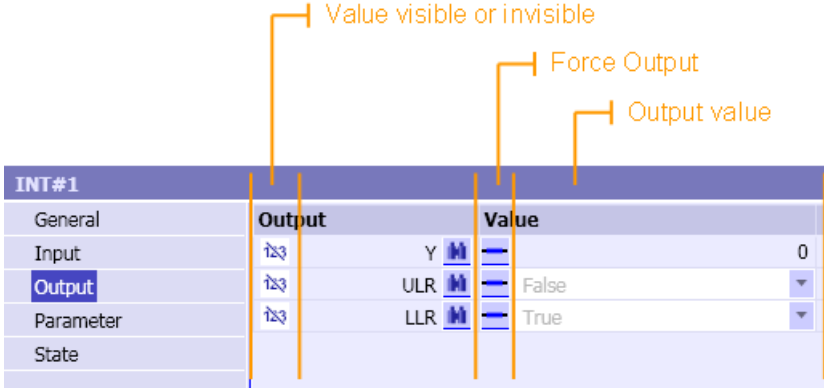
Each output has the following properties:

- Visibility**  
 In the first column, you can toggle between visibility (👁) and invisibility (👁) of the output. If the output is connected with a connecting line or by overlapping connectors, you cannot toggle between them.
- Name**  
 The name of the output is displayed right-justified in the second column.
- References**  
 The third column is used to search for references, which means for objects that use this output.
- Implicit interconnection**  
 In the fourth column, you can set whether this output is to be implicitly connected.

	Output is implicitly connected
	Output is not implicitly connected

If implicit connection is selected, an input field opens. Enter here the signal source to be interconnected and name.

The representation of the properties changes when the simulation is running.

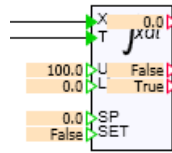


The screenshot shows the 'INT#1' component configuration window. It has a table with columns for 'Output' and 'Value'. The 'Output' column contains visibility icons (👁) and a 'Force Output' icon (👁). The 'Value' column contains a dropdown menu showing 'False' and 'True'. Labels with arrows point to these elements: 'Value visible or invisible' points to the visibility icons, 'Force Output' points to the Force Output icon, and 'Output value' points to the dropdown menu.

INT#1	Output	Value
General		
Input	👁 Y	0
Output	👁 ULR	False
Parameter	👁 LLR	True
State		

- **Display On/Off**

In the first column, the value at the output of the component on the chart can be shown (123) or hidden (123). When the simulation starts, the display is switched off for all outputs. The display cannot be switched on for invisible outputs. The figure below shows a component in which the display is switched on for all output values.



- **Forcing output**

In the fourth column, you can switch forcing on (T) or off (F) for every connected output.

- **Output value**

The fifth column is used to display or set the output value.

## Displaying the input and output values of components

When the simulation is running, the display of individual current input and output values of components can be switched on in the property view of the component by means of the 123 button. To switch on the display of all inputs and outputs of components on a chart with a single mouse click, you can use the same 123 command in the chart toolbar.



If you select one or more components before executing the command, the switch applies only to the selected components. If no components are selected, the switch applies to all components on the chart.

## Parameters

Each parameter is displayed with its name and value in the property view:

INT#1		
General	Parameter	Value
Input	Initial_Value	0.0
Output		
Parameter		
State		

### States

States are displayed with their name and initial values in the property view. When the simulation is started, the current values are displayed for each state.

INT#1		
General	State	Value
Input	z	0.0
Output	zLimitParamFault	False
Parameter	zTimeParamFault	False
State		

### Displaying vectors in the properties window

















Vectors of inputs, outputs, parameters and states are displayed in the property view grouped and in a numerically correct sequence, as shown by way of example in the figure below.

ADD#1		
General	Name	Value/Signal
Input	▼ X [12]	...
Output	👁 X1	123 ▾ 0.0
Parameter	👁 X2	123 ▾ 0.0
State	👁 X3	123 ▾ 0.0
	👁 X4	123 ▾ 0.0
	👁 X5	123 ▾ 0.0
	👁 X6	123 ▾ 0.0
	👁 X7	123 ▾ 0.0
	👁 X8	123 ▾ 0.0
	👁 X9	123 ▾ 0.0
	👁 X10	123 ▾ 0.0
	👁 X11	123 ▾ 0.0
	👁 X12	123 ▾ 0.0

The vector elements can be expanded and collapsed. The figure below shows a vector with collapsed elements.

ADD#1		
General	Name	Value/Signal
Input	▶ X [12]	...
Output		
Parameter		

For connectors with complex connection types an additional expandable and collapsible level has been added.

StorageTankLiquid#1			
General	Name	Value/Signal	
Input	▼ FN [2]	...	
Output	▼ FN1	...	
Parameter	 HSPEC 	123 ▼	83.6
Additional parameter	 MFL 	123 ▼	0.0
State	 PRESSURE 	123 ▼	1.0
	 DENSITY 	123 ▼	997.337
	▼ FN2	...	
	 HSPEC 	123 ▼	83.6
	 MFL 	123 ▼	0.0
	 PRESSURE 	123 ▼	1.0
	 DENSITY 	123 ▼	997.337

### 7.1.1.9 Component error messages

The components are implemented so that critical or nonsensical parameters or input values do not cause unstable component behavior. The component outputs an error message if impermissible values are specified or if input signals are not within the designated range.

In addition, outputs of the component can be set to a defined value in the event of an error to avoid unstable output values. This set value remains in effect until the error condition is eliminated.

All error messages from components in the basic library are assigned to the *ERROR* message category.

Error messages are generated as so-called incoming and outgoing messages. The incoming message is generated when the error occurs; the outgoing message is generated when the error condition has been resolved. Both messages have the same message text; the difference is that the text of outgoing messages is placed in parentheses.

---

#### Note

If you use the *ERROR* message category in your own messages which you generate using the *Message* component from the Trend and Messaging Editor, for example, you will be unable to distinguish between messages from components of the basic library and your own messages based on the message category.

---

### 7.1.1.10 Properties of controls

The properties of controls can be accessed in the property view. To display the properties of a component in the property view, right-click or left-click on the component. If a simulation is running, you can only display the properties by right-clicking. Each control has

- General properties and
- Properties for the connector.

Controls whose representation can be changed also have

- Properties for the view.

You can find additional information on this in the section: Controls (Page 427).

### General properties

General properties of controls are the name, the time slice, the unique identifier (UID), the position as well as the width and height of the control.

Bar Indicator#1		
General	Property	Value
Connector	Name	Bar Indicator#1
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1hwiueeg
	Data Type	Analog
	Position	X: 295.0 Y: 160.0
	Width	160.0
	Height	40.0

The name must be unique for all components and controls used in the project, which means the project must not contain multiple controls and/or components with the same name. When you drag a control from the library onto a chart, it is automatically assigned a name. This name is made up of the designation of the control and a number for the control that is unique across the entire project.

Check the *Show Name* check box to display the name of the control on the chart.



Controls may also have other specific, general properties. You can find additional information on this in the section: Controls (Page 427).

### Properties of connectors

Connectors may be visible or invisible. The figure below shows the property view with the connector of a control by way of example.

Bar Indicator#1		
General	Name	Signal
Connector	 X 	
View		

Each connector has the following properties:

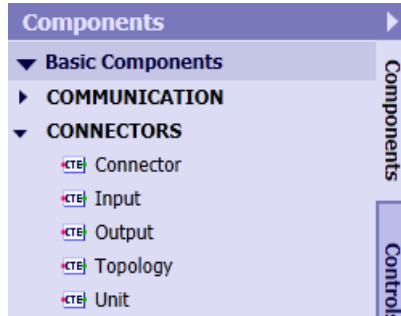
- Visibility**  
 In the first column, you can toggle between visibility () and invisibility () of the connector. If the connector is connected with a connecting line or by superimposing connectors, you cannot toggle between them.
- Name**  
 The name of the connector is displayed right-justified in the second column.
- Signal**  
 In the third column, the connected signal is displayed or can be set for invisible connectors.



## 7.1.2 Connectors

The *CONNECTORS* directory of the Basic Library contains connectors:

- A global *connector*,
- The I/O connectors, *Input* and *Output*
- The special connector, *Unit*
- The *Topology* connector is only used with special SIMIT modules or libraries and is therefore not described here.



The connectors of the basic library have the following shared characteristics:

- The connectors of the global connector and of the I/O connectors have no type. This means the connectors assume the connection type of the connector of the connected component or control.

---

### Note

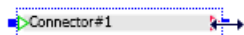
A type check, which means a check to identify whether the connections connected via connectors are of the same type, is carried out automatically before the simulation is started. If connectors are connected with connectors of an incorrect type, a corresponding message is output by the consistency check and the start of the simulation is cancelled.

---

- The width of the connector symbol on a chart can be adjusted to match the length of the connector name. To set the width of a connector, click on the symbol. A blue frame with handles appears on the right and left of the frame (a). If you move the cursor by means of a gripper, the shape of the cursor changes (b). Then press and hold down the left mouse button to change the width by moving the mouse pointer to the left or right (c).
- The connector name is displayed in the connector symbol.



(a)



(b)



(c)

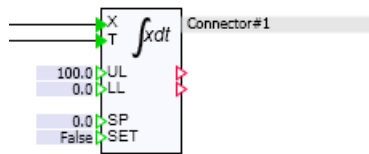
7.1.2.1 Global connector

The global connector *Connector* is used to connect components and/or controls across the boundaries of individual charts.

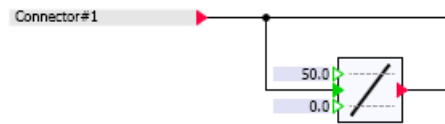
The global connector may be used as either an output connector or an input connector. Its symbol is illustrated in the figure below.



If the global connector is connected to the output of a component or control, the connector on the right side of the connector disappears: the connector is now used as an output connector.



If the global connector is connected to the input of a component or control, the connector on the left side of the connector disappears: the connector is now used as an input connector.

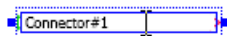


For a connection between an output connector and one or more input connectors, all of these connectors receive the same name: the connector name.

**Note**

Global connector name must be unique in the entire simulation project.

If you drag the global connector from the library onto a chart, the connector is assigned a name that consists of the term *Connector* and a unique sequence number. A connector name can be entered directly in the symbol. Double-click on the connector to open the input box, then press Return or click outside of the input box to confirm your input.



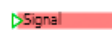
The connector name can also be entered in the property view. The connector name is the only property of the global connector.

Connector#1		
General	Property	Value
	Name	Connector#1

### 7.1.2.2 I/O connectors

I/O connectors establish the link to signals in the SIMIT couplings. An I/O connector can establish a link to signals from any SIMIT coupling.

I/O connectors exist in the form of *Input* and *Output* connectors as shown below.



Input connector *Input*



Output connector *Output*

Connections between coupling signals and I/O connectors can be

- a connection from one or more *Output* connectors to a single output signal of a coupling, or
- a connection from a coupling input signal to an input connector *Input*.

The connection is established by setting the name of the coupling and the name of the coupling signal in the property view of the I/O connector.

PLCSIM Q0.0		
General	Property	Value
	Signal	PLCSIM Q0.0 
	Display Gateway Name	<input checked="" type="checkbox"/>

If you do not want the name of the coupling to be displayed on the chart, you may deselect the option *Display coupling name*.

### 7.1.2.3 The topological connector

Topological connections can be realized beyond the respective chart borders using topological connectors. This connector is present in the CONTEC and in the FLOWNET library. The basic method of operation is the same.

You can find information about the topological connector in the CONTEC library in the section: The topological connector in the CONTEC library (Page 577)

You can find information about the topological connector in the FLOWNET library in the section: The topological connector in the FLOWNET library (Page 471)

### 7.1.2.4 The Unit connector

The *Unit* connector is a special type of connector that can only be used in conjunction with *SIWAREXU* components. Together with these components, it establishes relationships between the components and modules in a PROFIBUS DP coupling.

You can find additional information on this in the section: Linking SIWAREXU components to the coupling (Page 408).

### 7.1.3 Standard components

#### 7.1.3.1 Overview

The standard component types in the Basic Library form the Standard Library. They are contained in the *STANDARD* directory. The component types are divided into component types according to function with

- Analog functions,
- Binary functions,
- Integer functions,
- Conversion functions,
- Mathematical functions and
- Various auxiliary functions.

#### 7.1.3.2 Analog functions

##### General information about the analog functions

All component types with analog functions are contained in the *AnalogBasic* and *AnalogExtended* directories of the Standard Library. *AnalogBasic* contains the basic analog functions, while *AnalogExtended* holds the extended analog functions.

##### Basic analog functions

###### Introduction

The four basic analog functions of Addition, Subtraction, Multiplication and Division, which means the four basic arithmetic operations, are stored as component types in the *AnalogBasic* directory of the Standard Library.

###### ADD – Addition

###### Symbol



## Function

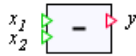
The *ADD* component type maps the sum of the analog values at the  $n$  inputs  $x_1$  to  $x_n$  onto the output  $y$ .

$$y = \sum_{i=1}^n x_i = x_1 + x_2 + \dots + x_n$$

The number of inputs  $n$  is variable and can be set to any value between 2 and 32. All inputs are set to zero by default.

## SUB - Subtraction

### Symbol



### Function

The *SUB* component type maps the difference between the analog values at the inputs  $x_1$  and  $x_2$  onto the output  $y$  according to

$$y = x_1 - x_2$$

All inputs are set to zero by default.

## MUL - Multiplication

### Symbol



### Function

The *MUL* component type maps the product of the analog values at the  $n$  inputs  $x_1$  to  $x_n$  onto the output  $y$ .

$$y = \prod_{i=1}^n x_i = x_1 x_2 \dots x_n$$

The number of inputs  $n$  is variable and can be set to any value between 2 and 32. All inputs are set to one by default.

### DIV – Division

#### Symbol



#### Function

The *DIV* component type maps the quotients of the analog values at the inputs  $x_1$  and  $x_2$  onto the output  $y$  according to

$$y = x_1 / x_2$$

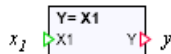
Input  $x_1$  is set to zero by default, while the divisor input  $x_2$  is set to one by default.

The value of the divisor  $x_2$  must not be zero. If the divisor becomes zero during the simulation, the error message "*DIV: division by zero*" (message category *ERROR*) is generated and output  $y$  is set to zero.

### Extended analog functions

#### AFormula – analog formula component

#### Symbol



#### Function

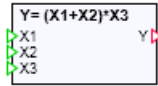
The *AFormula* component type allows the use of explicit algebraic functions. This function  $f$  calculates a value in relation to the  $n$  input values  $x_i$ . The function value is assigned to the output  $y$ .

$$y = f(x_1, \dots, x_n)$$

To define the function, enter the required formula expression in the *Formula* parameter that calculates the output  $y$  in relation to the inputs  $x_i$ . First set the required number of inputs and then enter the formula expression.

AFormula		
General	Parameter	Value
Input	Formula	$(X1 * X2) + (X3 * X4) + 1.2 * \sin(X5)$
Output		
Parameter		
State		

The number  $n$  of inputs can be varied between 1 and 32. Only the inputs that are available according to the number currently set may be used in the formula. The formula is displayed at the top of the component symbol. The figure below shows a component with three inputs by way of example.



The following operators are used in expressions formula:

Table 7-1 Operators in formulas for the AFormula component

Operator	Function
+	Addition
-	Subtraction
/	Division
*	Multiplication
(	Opening parenthesis
)	Closing parenthesis
Numeric constant	Floating point numbers, also in exponential notation
Function calls	Standard mathematical functions

#### Note

When entering floating point numbers, use a decimal point (and not a comma).

#### Note

There is also no check in the formula component to determine whether the arguments of the formula have valid values. If division by zero occurs during the simulation in the formula calculation, output  $y$  has the value **Inf**. If an argument of a formula is undefined during the simulation, as in the case of a division of zero by zero, output  $y$  has the value **NaN** (not a number).

These irregular output values will then be propagated in the model in all values that depend on this output. Your simulation will thus enter an undefined state. To avoid this situation, make sure that the inputs of the formula component can only assume values that ensure the validity of the arguments in the formula.

There is no check in the formula component to determine whether all the set inputs are used in the specified formula.

The mathematical functions listed in the table below are available as standard mathematical functions.

Table 7-2 Mathematical functions in formulas for the AFormula component

Formula	Function
<b>sqrt(x)</b>	$y = \sqrt{x}; x \geq 0$
<b>fabs(x)</b>	$y =  x $

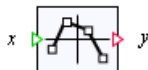
Formula	Function
<b>exp(x)</b>	$y = e^x$
<b>pow(x, z)</b>	$y = X^z$ ;
<b>log(x)</b>	Natural logarithm: $y = \ln(x)$ ; $x > 0$
<b>log10(x)</b>	Common logarithm: $y = \lg(x)$ ; $x > 0$
<b>ceil(x)</b>	Smallest integer greater than or equal to $x$
<b>floor(x)</b>	Largest integer less than or equal to $x$
<b>rand()</b>	Integral random value $y$ , $0 \leq y \leq 32767$
<b>sin(x)</b>	$y = \sin(x)$ ; angle $x$ in radians
<b>cos(x)</b>	$y = \cos(x)$ ; angle $x$ in radians
<b>tan(x)</b>	$y = \tan(x)$ ; angle $x$ in radians; $x \neq \pm(2n+1)\pi/2$
<b>asin(x)</b>	$y = \arcsin(x)$ ; $-\pi/2 \leq y \leq \pi/2$
<b>acos(x)</b>	$y = \arccos(x)$ ; $0 \leq y \leq \pi$
<b>atan(x)</b>	$y = \arctan(x)$ ; $-\pi/2 \leq y \leq \pi/2$
<b>atan2(z, x)</b>	$y = \arctan(x / z)$ ; $-\pi \leq z \leq \pi$
<b>sinh(x)</b>	$y = \sinh(x)$ ; angle $x$ in radians
<b>cosh(x)</b>	$y = \cosh(x)$ ; angle $x$ in radians
<b>tanh(x)</b>	$y = \tanh(x)$ ; angle $x$ in radians

**Note**

You can use the function  $y = \text{rand}()$  to generate random numbers  $y$  within a specified range  $Y_{\text{MIN}} \leq y \leq Y_{\text{MAX}}$  using the formula  $Y_{\text{MIN}} + \text{rand}() * (Y_{\text{MAX}} - Y_{\text{MIN}}) / 32767.0$ .

**Characteristic**

**Symbol**




**Function**


The Characteristic component type is used to define the mapping of input value  $x$  onto output value  $y$  defined by a characteristic:

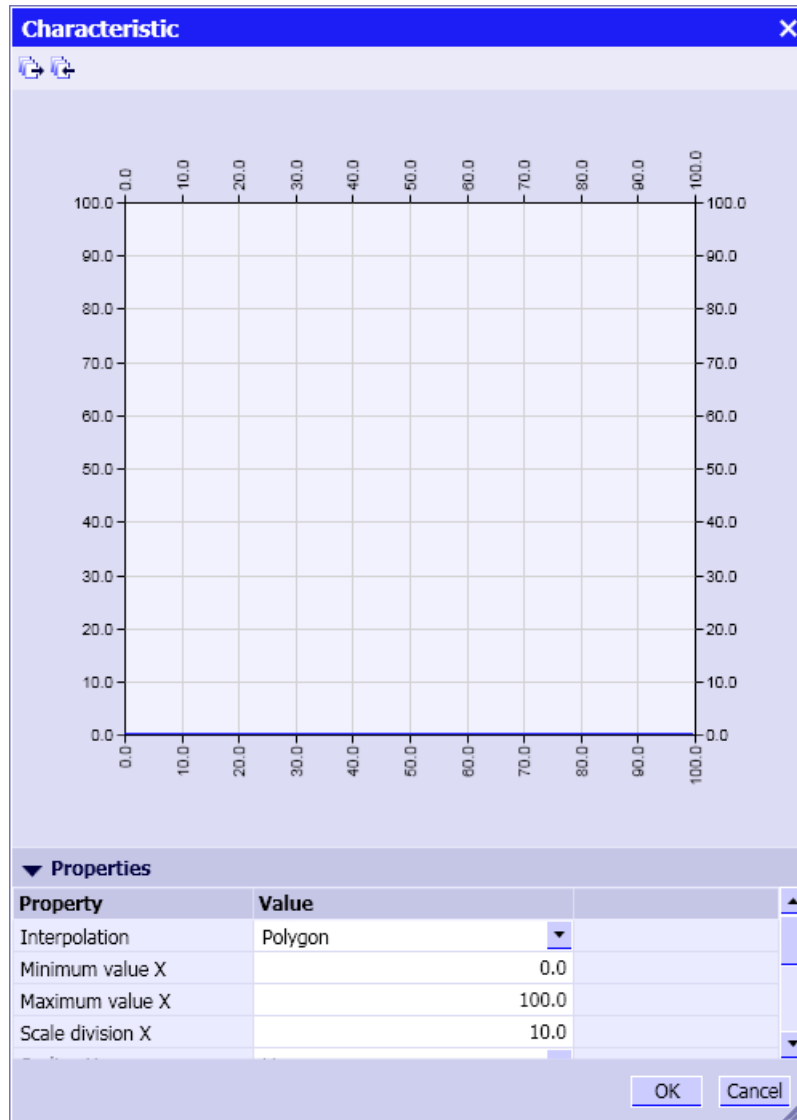
$$y = f_k(x)$$

The characteristic to be used is defined using the *Characteristic* parameter. To do this, open the Parameters section in the component property view.



Characteristic#1	
General	Name Value
Input	Characteristic 
Output	
Parameter	
State	

Use the  button of the *Characteristic* parameter to open a window with the characteristic editor and construct the required characteristic.

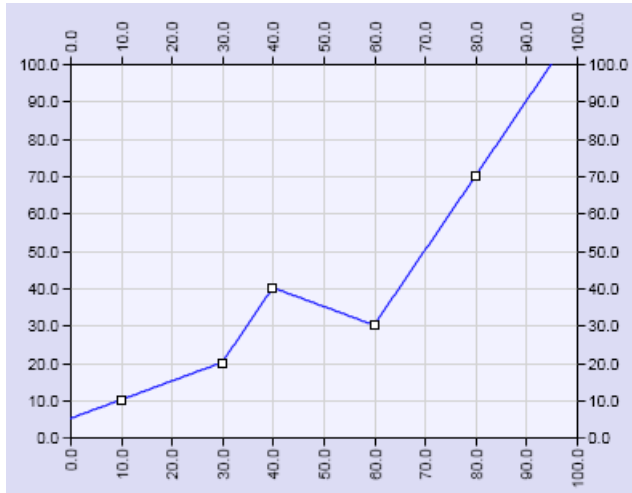
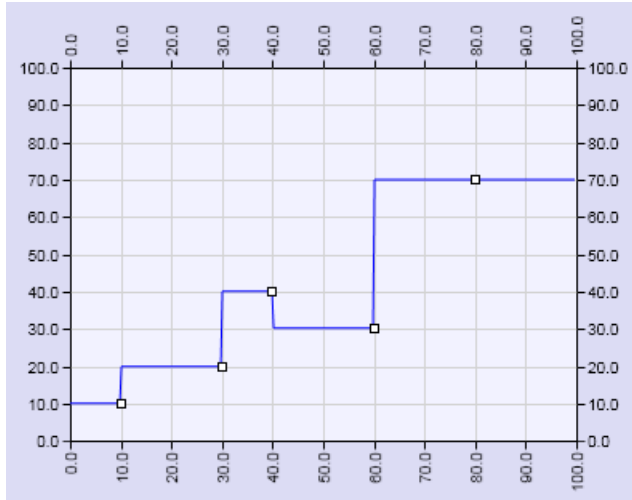


In this editor a characteristic is defined by specifying  $n$  interpolation points  $(x_i, y_i)$ ,  $i = 1, \dots, n$  and by selecting the type of *Interpolation* between these interpolation points. There may be any number  $n$  of interpolation points. The type of interpolation may be either constant or linear, in which case, the characteristic will take the form of a *step curve* or a *polyline*. *Polyline* is the default mode.

7.1 The basic library

Property	Value
Interpolation	Polygon
Minimum value X	Stepcurve
Maximum value X	Polygon

Outside the interpolation point range (the interpolation range), the output values  $y$  are extrapolated. The two figures below show an interpolation with five interpolation points as a step curve and as a polyline by way of example.



The characteristic function for  $n$  interpolation points is thus given as 0.0 for  $n = 0$ ,  $y_1$  for  $n = 1$  and for  $n > 1$  for the step curve using

$$y = \begin{cases} y_1 & \text{for } x \leq x_1, \\ y_i & \text{for } x_{i-1} < x \leq x_i, \quad i = 2, \dots, n \\ y_n & \text{for } x > x_n \end{cases}$$

and for the polyline using

$$y = \begin{cases} y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) & \text{for } x \leq x_1 \\ y_{i-1} + \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}) & \text{for } x_{i-1} < x \leq x_i, i = 2, \dots, n \\ y_n + \frac{y_n - y_{n-1}}{x_n - x_{n-1}}(x - x_n) & \text{for } x > x_n \end{cases}$$

For each of the two axes, which means for the horizontal x-axis and the vertical y-axis of the characteristic chart, the following variables can be set in the properties of the chart:

- the *minimum value* and the *maximum value*,
- the *scale division* and
- the *scale*.

Both axes default to a linear scale with a scale division of ten, a minimum value of zero and a maximum value of one hundred.

Property	Value
Minimum value X	0.0
Maximum value X	100.0
Scale division X	10.0
Scaling X	Linear

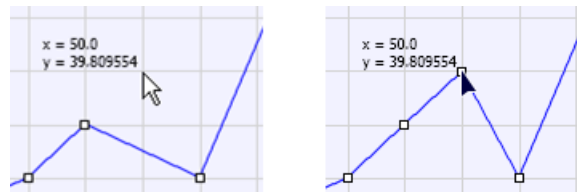
The *scale* may be set to either *linear* or *logarithmic*.

Scaling X	Linear
	Linear
	Logarithmic

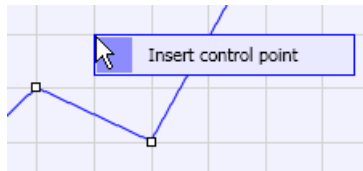
You can insert new interpolation points in the chart as well as remove or move existing interpolation points.

### Inserting interpolation points:

To insert a interpolation point into the chart, move the cursor to the required position. The current coordinates are displayed at the cursor position. A new interpolation point is inserted at the current position with a double-click.

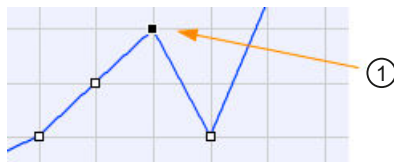


You can also insert a interpolation point at the current position by using the shortcut menu command (right click) *Insert interpolation point*.



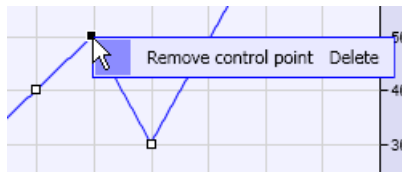
**Removing interpolation points:**

To remove a interpolation point first select it with a left-click on the interpolation point. The selected interpolation point changes its appearance.



① Selected interpolation point

The selected interpolation point can be removed by pressing the Delete key or by using *Remove interpolation point* in the interpolation point shortcut menu.




**Changing the coordinates of interpolation points**


Select a interpolation point with a left-click and by dragging it to the required position while keeping the left mouse button pressed. The interpolation point can only be moved within the boundaries defined by the x-coordinates of its left and right neighbors.

You can also change both coordinates of the selected interpolation point in the properties. Open the input by clicking in the input field, and then enter each desired value. Input is confirmed by pressing *Enter*.

Property	Value
X	50.0
Y	39.809553665896

**Importing and exporting interpolation points**

You can import interpolation points from a file. The command  from the toolbar of the characteristic editor opens the dialog for selecting an Excel file in csv format. In this interpolation point file there is one interpolation point per line, given as pairs of values with their two coordinates: x-coordinate separator y-coordinate. If a dot "." is used as the decimal point, the separator is a comma ",". If a comma "," is used as the decimal point, the separator is a semicolon ";".

With the command  from the toolbar of the characteristic editor, you can also export the current interpolation points into a csv file where you can modify them, for example. A file is written which contains all interpolation points as pairs of values (x-coordinate and y-coordinate). The decimal point is a comma ","; the separator is a semicolon ";".

## Compare – Comparison functions

### Symbol



### Function

The Compare component type compares the analog inputs  $x_1$  and  $x_2$ . Binary output  $b$  is set to one if the comparison expression is true, otherwise  $b$  is set to zero.

The type of comparison is determined with the *Comparison* parameter.

Compare#1		
General	Parameter	Value
Input	Comparison	<
Output		<
Parameter		<=
State		>
		>=

The following comparisons may be set:

- Less than comparison (<), which means

$$b = \begin{cases} 1, & \text{if } x_1 < x_2 \\ 0, & \text{otherwise} \end{cases}$$

- Less than or equal to comparison (<=), which means

$$b = \begin{cases} 1, & \text{if } x_1 \leq x_2 \\ 0, & \text{otherwise} \end{cases}$$

- Greater than comparison (>), which means

$$b = \begin{cases} 1, & \text{if } x_1 > x_2 \\ 0, & \text{otherwise} \end{cases}$$

- Greater than or equal to comparison (>=), which means

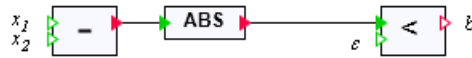
$$b = \begin{cases} 1, & \text{if } x_1 \geq x_2 \\ 0, & \text{otherwise} \end{cases}$$

The selected comparison operator is displayed in the component symbol.

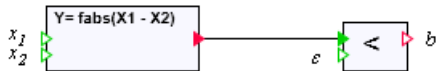
The width of the symbol can be changed to slightly offset the ">=" and "<=" comparison operators from the right edge of the symbol.



In SIMIT, analog variables are mapped onto variables of the type "double" so it is meaningless to compare them directly for equality or inequality. Equality of double variables is only possible within the accuracy defined by the computer (machine accuracy). Equality can be checked using the model illustrated in the figure below, for example.



Alternatively, a functionally identical model may be applied, for example, by using the *Aformula* formula block.



The difference between the two variables  $x_1$  and  $x_2$  is calculated. The amount of this difference is then compared against a definable positive limit  $\epsilon$ :

$$b = \begin{cases} 1, & \text{if } |x_1 - x_2| < \epsilon \\ 0, & \text{otherwise} \end{cases}$$

To check for inequality, only the "greater than" comparison should be used in the comparison component in both of the illustrated models.

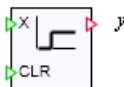
**Note**

You can create your own components to check for equivalence or non-equivalence based on the explanatory notes provided above.

With the Macro editor, for example, you can create corresponding macro components or you can create a comparison component extended by this comparison using the Component Type Editor (CTE).

**DeadTime – Dead time element**

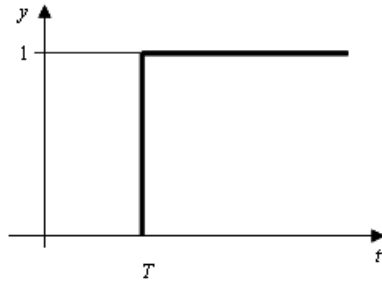
**Symbol**



## Function

The *DeadTime* component type makes a dead time element available. The analog value at input  $x$  is passed to output  $y$  with an adjustable delay.

A step change in the input value  $x$  from zero to one thus gives a curve for the output value  $y$  as shown in the figure below:



The delay time  $T$  is adjustable as a *Delay\_Cycles* parameter of the component type in integral multiples  $n$  of the cycle time  $\Delta t$ .  $T = n\Delta t$  (see figure below). It is set to ten by default and may be set to up to 128. The default cycle time of 100 ms therefore produces a delay of one second.

DeadTime#1		
General	Parameter	Value
Input	Delay_Cycles	10
Output		
Parameter		
State		

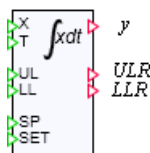
## Note

If you change the cycle time  $\Delta t$  in the project properties, the set dead times change accordingly.

In the component, the input values are saved and sent to the output after a delay according to the selected dead time. Memory is allocated for  $n$  values according to the configured number of delay cycles. All storage locations are initialized to zero. The binary input *CLR* can be used to set the output value  $y$  and all storage locations  $n$  to zero.

## INT – Integration

### Symbol



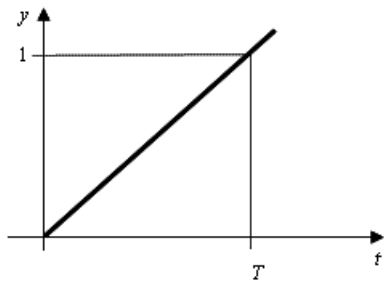
## Function

The *INT* component type forms the integral via the time-specific analog input signal  $x$  using

$$y = \frac{1}{T} \int x \, dt$$

The integral value is assigned to the analog output  $y$ .

For a step-shaped input signal of amplitude one, the result is a linear ascending output signal as shown in the figure below.



The integral value  $y$  is limited to an interval defined by the two limits  $UL$  (high limit) and  $LL$  (low limit):

$$LL \leq y \leq UL.$$

The binary outputs  $ULR$  and  $LLR$  indicate when the integral value has reached the low or high limit:

$$ULR = \begin{cases} 1, & \text{if } y \geq UL \\ 0, & \text{otherwise} \end{cases}$$

and

$$LLR = \begin{cases} 1, & \text{if } y \leq LL \\ 0, & \text{otherwise} \end{cases}$$

The low limit must be less than the high limit. If this condition is violated, the error message "*INT: limits do not match*" (message category) is generated and output  $y$  is set to zero.

The time constant  $T$  for the integration must have a positive value. If  $T$  is not positive, the error message "*INT: zero or negative time constant*" (message category *ERROR*) is generated and output  $y$  is set to zero.

The low limit is set to zero, and the high limit is set to one hundred by default. The time constant  $T$  is set to one second by default. All other inputs are set to zero by default.

The component *INT* only has one parameter - *Initial\_Value*. The integral value  $y$  is set to this value when the simulation is initialized (started). The parameter is set to "0" by default.

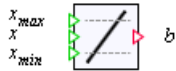


INT#2		
General	Parameter	Value
Input	Initial_Value	0.0
Output		
Parameter		
State		

The binary input *SET* may be used to set the integration value  $y$  to the value at the input *SP*. If *SET* is set to one, the integration value  $y$  is set to the value at the input *SP*. Here, too, the integration value is limited by *LL* and *UL*.

## Interval – Interval query

### Symbol



### Function

The *Interval* component type checks whether an input value  $x$  is within the closed interval  $[x_{min}, x_{max}]$ . If the input value falls within the set interval, the binary output is set to one; otherwise it is zero:

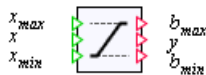
$$b = \begin{cases} 1, & \text{for } x_{min} \leq x \leq x_{max} \\ 0, & \text{otherwise} \end{cases}$$

An interval from zero to one hundred is set, which means  $x_{min}$  has a default of zero and  $x_{max}$  a default of one hundred.

The high interval limit may not be less than the lower interval limit. If the high interval limit becomes less than the low limit during the simulation, the error message "*Interval\_I: limits do not match*" (message category *ERROR*) is generated and output  $b$  is set to zero.

## Limiter

### Symbol



**Function**

The  *Limiter*  component type maps an input value limited to the range from  $x_{min}$  to  $x_{max}$  onto the output  $y$ :

$$y = \begin{cases} x_{max} & \text{for } x \geq x_{max} \\ x & \text{for } x_{min} < x < x_{max} \\ x_{min} & \text{for } x \leq x_{min} \end{cases}$$

The binary outputs  $b_{min}$  and  $b_{max}$  are set to one when the limiter takes effect, which means

$$b_{max} = \begin{cases} 1, & \text{if } x \geq x_{max} \\ 0, & \text{otherwise} \end{cases}$$

and

$$b_{min} = \begin{cases} 1, & \text{if } x \leq x_{min} \\ 0, & \text{otherwise} \end{cases}$$

The limit  $x_{min}$  is set to zero and  $x_{max}$  is set to one hundred by default.

The high limit may not be less than the low limit. If the low limit becomes less than the high limit during the simulation, the error message "*Limiter: limits do not match*" (message category *ERROR*) is generated and output  $y$  is set to zero.

**MinMax – minimum and maximum value selection**

**Symbol**



**Function**

The  *MinMax*  component type maps the minimum or maximum of the  $n$  inputs  $x_1$  to  $x_n$  onto the output  $y$ . The number of inputs  $n$  is variable and can be set to any value between 2 and 32. All inputs are set to zero by default.

The type of mapping is determined with the  *MinMax*  parameter in the property view of the component.

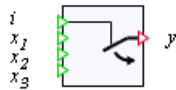
MinMax#1		
General	Parameter	Value
Input	MinMax	MIN
Output		MIN
Parameter		MAX
State		

The *MIN* or *MAX* mapping that is set for a component is displayed in the component symbol as shown in the figure below:



## Multiplexer

### Symbol



### Function

The *Multiplexer* component type interconnects one of the  $n$  different inputs  $x_i$  in relation to the integer value at the selection input  $i$  with the output  $y$ .

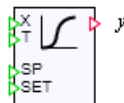
$$y = x_i \text{ for } 1 \leq i \leq n.$$

The number  $n$  of inputs  $x_i$  is variable and can be set to any value between 3 and 32. All inputs are set to zero by default.

The value at the selection input  $i$  is limited internally to 1 through  $n$ , which means for values less than one  $i$  is set to one, while for values greater than  $n$ ,  $i$  is set to the value  $n$ . In the default setting, the first input  $x_1$  is therefore interconnected with the output  $y$ .

## PTn – nth order delay

### Symbol



### Function

The *PTn* component type provides an  $n$ th order delay.

With a first order delay, the function value  $y$  at the output follows the value  $x$  at the input with a delay equal to the differential equation:

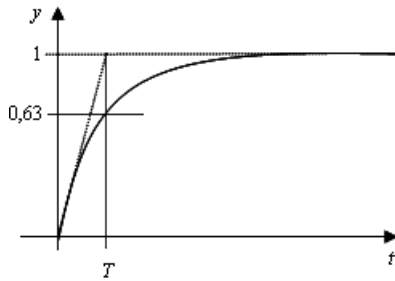
$$\frac{dy}{dt} = \frac{1}{T}(x - y)$$

7.1 The basic library

A step change in the input value  $x$  from zero to one thus gives an exponential curve for the output value  $y$  (step response) as per

$$y = 1 - e^{-t/T}$$

as illustrated in the figure below.



For higher-order delays, the output value  $y$  is obtained by concatenating first-order delays:

$$\frac{dz_i}{dt} = \frac{1}{T}(z_{i-1} - z_i), \quad i = 1, \dots, n$$

$$y = z_n$$

where  $z_i, i = 1, \dots, n$ , are the  $n$  states of the  $n$ th order delay. To illustrate, an  $n$ th order delay corresponds to  $n$  first order delays connected in series.

The order  $n$  of the delay can be set as a parameter of the component. It is set to one by default and may be set to a maximum of 32. The *Initial\_Value* parameter with a default value of zero is used to initialize the states of the delay function.

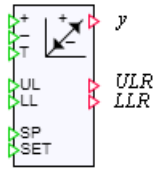
PTn#3		
General	Parameter	Value
Input	Initial_Value	0.0
Output	n	1
Parameter		
State		

To prevent the component behaving in an unstable manner for delay times that are too short, the delay time constant  $T$  is limited to values that are greater than or equal to the cycle time of the component. If the values at input  $T$  are less than the cycle time, the error message "PTn: delay time below cycle time" (message category *ERROR*) is generated. The delay time is set to one second by default. All other inputs of the component are set to zero by default.

The binary input *SET* may be used to set the output value  $y$  and the state values  $z_i, i = 1, \dots, n$ , to the value at the input *SP*. If *SET* is set to one, these values are set equal to the value at the input *SP*.

## Ramp – Ramp function

### Symbol



### Function

The *Ramp* component type increments or decrements its function value  $y$  in every increment of the simulation by the value

$$\Delta y = (UL - LL) \frac{\Delta t}{T}$$

where  $\Delta t$  is the simulation increment width. The ramp value  $y$  is incremented by  $\Delta y$  when the "+" (*UP*) input is set to one. If the "-" (*DOWN*) input is set to one, the ramp value  $y$  is decremented by  $\Delta y$ . If both the "+" and the "-" input are set to one, the ramp value  $y$  does not change.

The ramp value is limited to an interval defined by the two limits  $UL$  (high limit) and  $LL$  (low limit):

$$LL \leq y \leq UL.$$

The binary outputs  $ULR$  and  $LLR$  indicate when the ramp value has reached the low or high limit:

$$ULR = \begin{cases} 1, & \text{if } y \geq UL \\ 0, & \text{otherwise} \end{cases}$$

and

$$LLR = \begin{cases} 1, & \text{if } y \leq LL \\ 0, & \text{otherwise} \end{cases}$$

The low limit must be less than the high limit. If this condition is violated, the error message "*RAMP: limits do not match*" (message category *ERROR*) is generated and output  $y$  is set to zero.

The time constant  $T$  must have a positive value. If  $T$  is not positive, the error message "*Ramp: zero or negative time constant*" (message category *ERROR*) is generated and output  $y$  is set to zero.

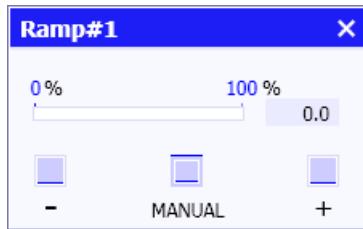
The low limit is set to zero and the high limit is set to one hundred by default. The time constant is set to 10 seconds by default. All other inputs are set to zero by default.

The *Ramp* component has only one parameter - *Initial\_Value*. The ramp function value *y* is set to this value when the simulation is initialized (started). *Initial\_Value* is set to zero by default.

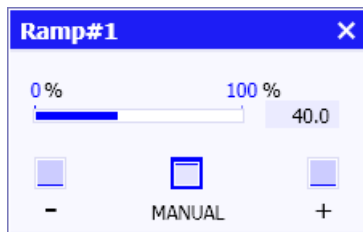
Ramp#1		
General	Parameter	Value
Input	Initial_Value	0.0
Output		
Parameter		
State		

The binary input *SET* may be used to set the ramp value *y* to the value at the input *SP*. If *SET* is set to one, the ramp value *y* is set equal to the value at the input *SP*. Here, too, the ramp value is limited by *LL* and *UL*.

The operating window of the component can be used to manually set the ramp function value during a simulation. To do so, open the operating window by double-clicking on the component symbol.

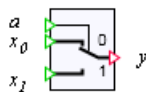


Use the *MANUAL* button to switch the ramp function to manual mode. The color of the border around the button changes to dark blue. You can now press the "-" or "+" button to reduce or increase the ramp function value. The current function value is displayed both as a decimal value and as a percentage.



**Selection – Analog switch**

**Symbol**



## Function

The *Selection* component type interconnects one of the two inputs  $x_0$  or  $x_1$  to output  $y$  in relation to the value of the binary input  $a$ .

If the selection input  $a = 0$ , input  $x_0$  is interconnected; if the selection input  $a = 1$ , input  $x_1$  is interconnected:

$$y = \begin{cases} x_0, & \text{if } a = 0 \\ x_1, & \text{if } a = 1 \end{cases}$$

All inputs are set to zero by default.

### 7.1.3.3 Integer functions

#### Overview

All component types with integer functions are contained in the *IntegerBasic* and *IntegerExtended* directories of the standard library.

The four basic integer functions of Addition, Subtraction, Multiplication and Division, which means the four basic arithmetic operations, are stored as component types in the *IntegerBasic* directory of the standard library.

The symbols of these component types are blue to distinguish them from analog component types.

The *IntegerExtended* directory of the standard library contains further integer functions in the form of component types.

The symbols of these component types are blue to distinguish them from analog component types.

#### Basic integer functions

##### ADD\_I – Addition

#### Symbol



**Function**

The *ADD\_*/component type maps the sum of the analog values at the *n* inputs  $x_1$  to  $x_n$  onto the output  $y$ :

$$y = \sum_{i=1}^n x_i = x_1 + x_2 + \dots + x_n$$

The number of inputs *n* is variable and can be set to any value between 2 and 32. All inputs are set to zero by default.

**SUB\_I – Subtraction**

**Symbol**



**Function**

The *SUB\_*/component type maps the difference between the integer values at the two inputs  $x_1$  and  $x_2$  onto the output  $y$  according to

$$y = x_1 - x_2.$$

All inputs are set to zero by default.

**MUL\_I – Multiplication**

**Symbol**



**Function**

The *MUL\_*/component type maps the product of the integer values at the *n* inputs  $x_1$  to  $x_n$  onto the output  $y$ :

$$y = \prod_{i=1}^n x_i = x_1 x_2 \dots x_n$$

The number of inputs *n* is variable and can be set to any value between 2 and 32. All inputs are set to one by default.



## DIV\_I – Integer division

### Symbol



### Function

The *DIV\_I* component type maps the quotients of the integral values at the inputs  $x_1$  and  $x_2$  according to

$$x_1 = y \cdot x_2 + R$$

as integer division with a remainder. The integral result of the division is available at output  $y$ , and the remainder of the integral division at output  $R$ .

The divider at input  $x_1$  is set to zero by default, while the divisor input  $x_2$  is set to one by default.

The value of the divisor  $x_2$  must not be zero. If the divisor becomes zero during the simulation, the error message "*DIV\_I: division by zero*" (message category *ERROR*) is generated and output  $y$  and the remainder  $R$  are set to zero.

## Extended integer functions

### Compare\_I – Compare functions

### Symbol



### Function

The *Compare\_I* component compares integer inputs  $x_1$  and  $x_2$ . Binary output  $b$  is set to one if the comparison expression is true, otherwise  $b$  is set to zero.

The type of comparison is determined with the *Comparison* parameter.

Compare_I#1		
General	Parameter	Value
Input	Comparison	<
Output		<
Parameter		<=
State		>
		>=
		=
		<>

The following comparisons may be set:

- Less than comparison (<), which means

$$b = \begin{cases} 1, & \text{if } x_1 < x_2 \\ 0, & \text{otherwise} \end{cases}$$

- Less than or equal to comparison (<=), which means

$$b = \begin{cases} 1, & \text{if } x_1 \leq x_2 \\ 0, & \text{otherwise} \end{cases}$$

- Greater than comparison (>), which means

$$b = \begin{cases} 1, & \text{if } x_1 > x_2 \\ 0, & \text{otherwise} \end{cases}$$

- "Greater than or equal to" comparison (>=), which means

$$b = \begin{cases} 1, & \text{if } x_1 \geq x_2 \\ 0, & \text{otherwise} \end{cases}$$

- Equal to comparison (=), which means

$$b = \begin{cases} 1, & \text{if } x_1 = x_2 \\ 0, & \text{otherwise} \end{cases}$$

- Not equal to comparison (<>), which means

$$b = \begin{cases} 0, & \text{if } x_1 \neq x_2 \\ 1, & \text{otherwise} \end{cases}$$

The selected comparison operator is displayed in the component symbol. The width of the symbol can be changed to slightly offset the "<=", ">=" and "<>" comparison operators from the right edge of the symbol.



### Interval\_I – Interval query

#### Symbol



## Function

The *Interval\_I* component type checks whether an integer input value  $x$  is within the closed interval  $[x_{min}, x_{max}]$ . If the input value falls within the set interval, the binary output is set to one; otherwise it is zero:

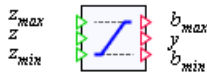
$$b = \begin{cases} 1, & \text{for } x_{min} \leq x \leq x_{max} \\ 0, & \text{otherwise} \end{cases}$$

An interval from zero to one hundred is set, which means  $x_{min}$  has a default of zero and  $x_{max}$  a default of one hundred.

The high interval limit may not be less than the lower interval limit. If the high interval limit becomes less than the low limit during the simulation, then the error message "*Interval\_I: limits do not match*" (message category *ERROR*) is generated and output  $b$  is set to zero.

## Limiter\_I – Limiter

### Symbol



### Function

The *Limiter\_I* component type maps an input value limited to the range from  $x_{min}$  to  $x_{max}$  onto the output  $y$ :

$$y = \begin{cases} x_{max} & \text{for } x \geq x_{max} \\ x & \text{for } x_{min} < x < x_{max} \\ x_{min} & \text{for } x \leq x_{min} \end{cases}$$

The binary outputs  $b_{min}$  and  $b_{max}$  are set to one when the limiter takes effect, which means

$$b_{max} = \begin{cases} 1, & \text{if } x \geq x_{max} \\ 0, & \text{otherwise} \end{cases}$$

and

$$b_{min} = \begin{cases} 1, & \text{if } x \leq x_{min} \\ 0, & \text{otherwise} \end{cases}$$

The limit  $x_{min}$  is set to zero and  $x_{max}$  is set to one hundred by default.

The high limit may not be less than the low limit. If the low limit becomes less than the high limit during the simulation, then the error message "*Limiter\_I: limits do not match*" (message category *ERROR*) is generated and output  $y$  is set to zero.

### MinMax\_I – Minimum and maximum value selection

#### Symbol



#### Function

The *MinMax\_I* component type maps the minimum or maximum of the  $n$  inputs  $z_1$  to  $z_n$  onto the output  $y$ . The number of inputs  $n$  is variable and can be set to any value between 2 and 32. All inputs are set to zero by default.

The type of mapping is determined with the *MinMax* parameter in the property view of the component.

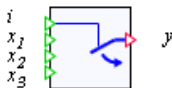
MinMax_I#1		
General	Parameter	Value
Input	MinMax	MIN
Output		MIN
Parameter		MAX
State		

The *MIN* or *MAX* mapping that is set for a component is displayed in the component symbol as shown in the figure below.



### Multiplexer\_I – Integer multiplexer

#### Symbol



#### Function

The *Multiplexer\_I* component type interconnects one of the  $n$  different integer inputs  $x_i$  in relation to the value at the selection input  $i$  with the output  $y$ :

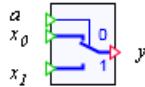
$$y = x_i \text{ for } 1 \leq i \leq n.$$

The number  $n$  of inputs  $x_i$  is variable and can be set to any value between 3 and 32. All inputs are set to zero by default.

The value at the selection input  $i$  is limited internally to 1 through  $n$ , which means for values less than one  $i$  is set to one, while for values greater than  $n$ ,  $i$  is set to the value  $n$ . In the default setting, the first input  $x_1$  is therefore interconnected with the output  $y$ .

## Selection\_I – Integer switch

### Symbol



### Function

The *Selection\_I* component type interconnects one of the two integer inputs  $x_0$  or  $x_1$  with output  $y$  in relation to the value of the binary input  $a$ .

If the selection input  $a = 0$ , input  $x_0$  is interconnected; if the selection input  $a = 1$ , input  $x_1$  is interconnected:

$$y = \begin{cases} x_0, & \text{if } a = 0 \\ x_1, & \text{if } a = 1 \end{cases}$$

All inputs are set to zero by default.

## 7.1.3.4 Mathematical functions

### Overview

The *Math* directory of the Standard Library contains the most commonly used mathematical functions in the form of component types: absolute value generation (*ABS*), square root extraction (*SQRT*), natural logarithm (*LN*) and the exponential function (*EXP*), plus the trigonometrical functions sine (*SIN*), cosine (*COS*) and tangent (*TAN*).

---

#### Note

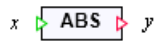
You can also use the *AFormula* component with suitable parameters instead of these components.

---

You can increase the available mathematical functions of this type by creating suitable component types using the SIMIT Component Type Editor (CTE) expansion module.

## ABS – Absolute value

### Symbol



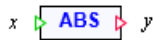
### Function

The *ABS* component type maps the absolute value of input  $x$  onto output  $y$ :

$$y = |x|.$$

## ABS\_I – Absolute integer values

### Symbol



The symbol for *ABS\_I* is blue to distinguish it from the *ABS* component type.

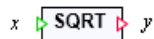
### Function

The *ABS\_I* component type maps the absolute value of integer input  $x$  onto integer output  $y$ :

$$y = |x|.$$

## SQRT – Square root

### Symbol



### Function

The *SQRT* component type maps the square root of input  $x$  onto output  $y$ :

$$y = \sqrt{x}$$

The radicand  $x$  must not be negative. If the radicand becomes negative during the simulation, the error message "*SQRT: invalid argument!*" (message category *ERROR*) is generated and output  $y$  is set to zero.

## EXP – Exponential function

### Symbol



### Function

The *EXP* component type maps the exponential value of input  $x$  onto output  $y$ :

$$y = e^x$$

## LN – Natural logarithm

### Symbol



### Function

The *LN* component type maps the natural logarithm of input  $x$  onto output  $y$ :

$$y = \ln(x).$$

The argument  $x$  must be positive. If the argument becomes less than or equal to zero during the simulation, the error message "*LN: invalid argument!*" (message category *ERROR*) is generated and output  $y$  is set to zero. The argument  $x$  is set to one by default.

## SIN – Sine function

### Symbol



### Function

The *SIN* component type maps the sine value of input  $x$  onto output  $y$ :

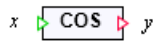
$$y = \sin(x).$$

The unit of measurement for the argument  $x$  (angle) can be set in radians (rad) or degrees (deg) using the *Unit* parameter. The default unit of measurement is degrees (deg).

SIN#1		
General	Parameter	Value
Input	Unit	deg
Output		rad
Parameter		deg
State		

### COS – Cosine function

#### Symbol



#### Function

The *COS* component type maps the cosine value of input  $x$  onto output  $y$ :

$$y = \cos(x).$$

The unit of measurement for the argument  $x$  (angle) can be set in radians (rad) or degrees (deg) using the *Unit* parameter. The default unit of measurement is degrees (deg).

COS#1		
General	Parameter	Value
Input	Unit	deg
Output		rad
Parameter		deg
State		

### TAN – Tangent function

#### Symbol



#### Function

The *TAN* component type maps the tangent value of input  $x$  onto output  $y$ :

$$y = \tan(x).$$

The unit of measurement for the argument  $x$  (angle) can be set in radians (rad) or degrees (deg) using the *Unit* parameter. The default unit of measurement is degrees (deg).



TAN#1		
General	Parameter	Value
Input	Unit	deg
Output		rad
Parameter		deg
State		

## RAND - random numbers

### Symbol



### Function

The RAND component type specifies at its Y output randomly generated numbers, the value ranges and distribution of which can be set using the "Distribution" parameter:

- "Uniform distribution"  
The random numbers have equal probability of being in the open interval (0,1).
- "Normal distribution"  
The random numbers are normally distributed around the mean of 0 with standard deviation of 1. The value range is not limited.

---

### Note

Note that the random numbers are calculated and therefore are not unpredictable.

---

## 7.1.3.5 Binary functions

### Overview

All of the functions for processing binary signals are contained in the *BinaryBasic* and *BinaryExtended* directories.

Component types with the three basic binary operations of conjunction (*AND*), disjunction (*OR*) and negation (*NOT*), plus equivalence (*XNOR*) and non-equivalence (*XOR*) are stored in the *BinaryBasic* directory.

The *BinaryExtended* directory of the standard library contains further binary (logic) functions that go beyond the elementary binary functions in the form of component types.

## Basic binary functions

### AND – Conjunction

#### Symbol



#### Function

The *AND* component type maps the  $n$  binary values at the inputs  $a_i$  as a conjunction, which means using a logical (Boolean) AND function, onto the output  $b$ :

$$b = \bigcap_{i=1}^n a_i$$

The number of inputs  $n$  is variable and can be set to any value between 2 and 32. All inputs are set to one by default.

### OR – Disjunction

#### Symbol



#### Function

The *OR* component type maps the  $n$  binary values at the inputs  $a_i$  as a disjunction, which means using a logical (Boolean) OR function, onto the output  $b$ :

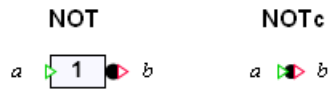
$$b = \bigcup_{i=1}^n a_i$$

The number of inputs  $n$  is variable and can be set to any value between 2 and 32. All inputs are set to zero by default.

### NOT, NOTc – Negation

Negation is available in two component types *NOT* and *NOTc*. These differ only in terms of the symbols used - their functions are completely identical.

## Symbol



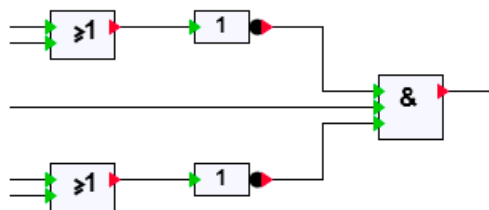
## Function

Output  $b$  is equal to the negated input  $a$ , which means

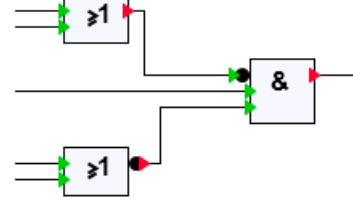
$$b = \bar{a}$$

As can be seen in the figure below, when the *NOTc* component type is used, the negation can be applied clearly and compactly to the inputs or outputs of components.

### Using "NOT"

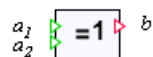


### Using "NOTc"



## XOR – Non-equivalence

### Symbol



### Function

The *XOR* component type maps the  $n$  binary values at the inputs  $a_i$  onto the output  $b$  using the "Exclusive-OR" function:  $b$  is one, if an odd number of inputs  $a_i$  is one. In all other cases  $b$  is zero.

For two inputs

$$b = a_1 \otimes a_2$$

which is the non-equivalence or "unequal" function.

The number of inputs  $n$  is variable and can be set to any value between 2 and 32. All inputs are set to zero by default.

## XNOR – Equivalence

### Symbol



### Function

The *XNOR* component type maps the  $n$  binary values at the inputs  $a_i$  onto the output  $b$  using the "Exclusive NOR" function:  $b$  is one, if an even number of inputs  $a_i$  is one. In all other cases  $b$  is zero.

For two inputs

$$b = \overline{a_1 \otimes a_2}$$

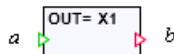
which is the equivalence or "equal" function.

The number of inputs  $n$  is variable and can be set to any value between 2 and 32. All inputs are set to zero by default.

## Extended binary functions

### BFormula – Binary formula component

### Symbol



### Function

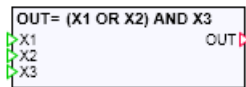
The *BFormula* component type allows explicit logical expressions to be used. This logic function  $f$  calculates a value in relation to the  $n$  values at the inputs  $a_i$ . The function value is assigned to the output  $b$ :

$$b = f(a_1, \dots, a_n)$$

To define the function, enter the required logic formula in the *Formula* parameter that calculates the output value  $b$  in relation to the input values  $a_i$ . Then set the required number of inputs and open the component property view to enter the formula.

BFormula#1		
General	Parameter	Value
Input	Formula	(X1 OR X2) AND X3
Output		
Parameter		
State		

The number of inputs can be varied between 1 and 32. Only the inputs that are available according to the number currently set may be used in the formula. As shown in the figure below, the formula is displayed in the component symbol on the chart.



You can make the component wider to allow longer formulas to be displayed in full.

The operators listed in the following table may be used in formulas.

Table 7-3 Permitted operators in formulas for the BFormula component

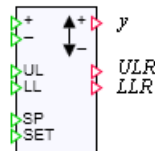
Operator	Function
AND	Conjunction (AND function)
OR	Disjunction (OR function)
NOT	Negation
(	Opening parenthesis
)	Closing parenthesis

**Note**

There is no check in the formula component to determine whether all the set inputs are used in the specified formula.

**Counter – Up and down counters**

**Symbol**



Function

The *Counter* component type is used to count the changes in binary signals. When the binary value at the "+" input changes from zero to one, the counter value is incremented at output *y*. When the binary value at the "-" input changes from zero to one, the counter value is decremented at output *y*.

The values by which the output is decremented or incremented can be set as the parameters (*Decrement* or *Increment*) (see figure below). Both parameters can be modified online, which means while the simulation is running. You can set any analog value for the decrement and increment. Both parameters are set to one by default.

The counter value is limited to an interval defined by the two limits *UL* (high limit) and *LL* (low limit):

$$LL \leq y \leq UL.$$

The binary outputs *ULR* and *LLR* indicate when the counter value has reached the low or high limit:

$$ULR = \begin{cases} 1, & \text{if } y \geq UL \\ 0, & \text{otherwise} \end{cases}$$

and

$$LLR = \begin{cases} 1, & \text{if } y \leq LL \\ 0, & \text{otherwise} \end{cases}$$

The low limit must be less than the high limit. If this condition is violated, the error message "*Counter: limits do not match*" (message category *ERROR*) is generated and the counter value at output *y* is set to zero.

The binary input *SET* may be used to set the counter value *y* to the value at the input *SP*. If *SET* is set to one, the counter value *y* is set equal to the value at the input *SP*. Here, too, the counter value is limited by *LL* and *UL*.

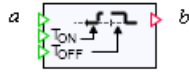
The *Initial\_Value* parameter is used to set the counter value when the simulation is initialized (started). *Initial\_Value* is set to zero by default.

Counter#1		
General	Parameter	Value
Input	Decrement	1.0
Output	Increment	1.0
Parameter	Initial_Value	0.0
State		

The limits are set to zero for the low limit and to one hundred for the high limit by default. All other inputs are set to zero by default.

## Delay – On-Off delay

### Symbol



### Function

With the *Delay* component type, the binary signal *b* at the output is used after a delay to update the binary signal *a* at the input.

If the signal at the input changes from zero to one, the output is set to one once the ON-delay time  $T_{ON}$  has elapsed. If the input signal is reset to zero before the ON-delay time has elapsed, the output signal remains unchanged at zero. If the signal at the input changes from one to zero, the output is set to zero once the OFF-delay time  $T_{OFF}$  has elapsed. If the output signal is reset to one before the OFF-delay time has elapsed, the output signal remains unchanged at one. This interaction is illustrated in the table below.

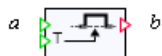
Table 7-4 Signal curves at the input and output of the Delay component

	$T_{ON}$	$T_{OFF}$
Input a		
Output b		

The delay times must not assume a negative value. If a delay time assumes a negative value, the error message "*Delay: on-delay time negative value*" or "*Delay: off-delay time negative value*" (message category *ERROR*) is generated and the corresponding delay time is set to zero.

## Pulse – Pulse

### Symbol



**Function**

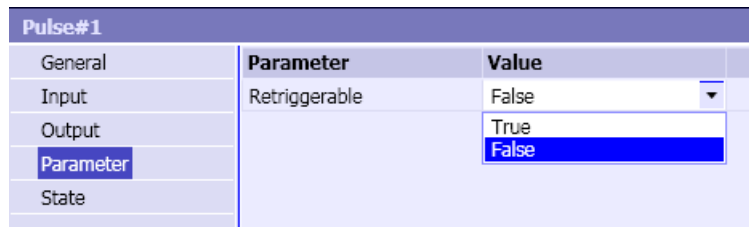
The *Pulse* component type sets the output *b* when an edge change from zero to one occurs at the input *a*. The output *b* is reset once the time *T* has elapsed. A pulse with pulse width *T* is thus generated at output *b*. The pulse width can be set via the signal at analog input *T*.

Table 7-5 Signal curves at the input and output of the Pulse component

Input a	
Output b	

The pulse width *T* must not assume a negative value, otherwise the error message "*Pulse: pulse width negative value*" (message category *ERROR*) is generated and output *b* is set to zero. For pulse widths that are smaller than the simulation cycle time, the output pulse is set to the duration of one simulation cycle.

If the *Retriggerable* parameter, which can be modified online, is set to one (True), the output pulse is restarted whenever the input signal changes from zero to one. The parameter is set to zero (False) by default.



**RS\_FF – Flip-flop with default state of "Reset"**

**Symbol**



**Function**

The *RS\_FF* component type is the simplest type of flip-flop available: an RS flip-flop. If the value at the set input *S* is equal to one, the output value *Q* is set to one. If the input value at the reset input *R1* is set to one, the output is reset to zero. The reset input is dominant, which means if both inputs are set to one, output *Q* is reset to zero. The output *Q* always assumes the inverse value of output *Q*.

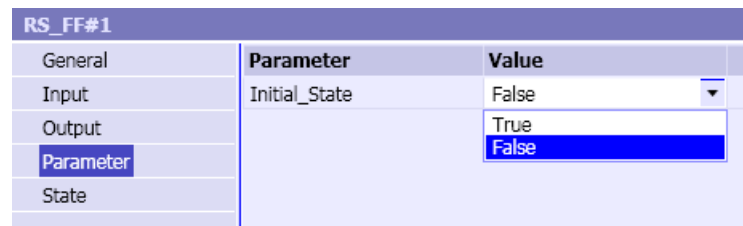


The table below lists the possible states of the *RS\_FF* component type.

Table 7-6 State table for the RS\_FF component

Input S	Input R1	Output Q	Output $\bar{Q}$
0	0	Unchanged	Unchanged
0	1	0	1
1	0	1	0
1	1	0	1

When the simulation starts, the output *Q* is set to the value of the *Initial\_State* parameter. *Initial\_State* is set to zero by default.



## SR\_FF – Flip-flop with default state of "Set"

### Symbol



### Function

The *SR\_FF* component type simulates an SR flip-flop. If the value at the set input *S1* is equal to one, the output value *Q* is set to one. If the input value at the reset input *R* is set to one, the output is reset to zero. The set input is dominant, which means if both inputs are set to one, output *Q* is reset to one. The output  $\bar{Q}$  always has the inverse value of output *Q*.

The table below lists the possible states of the *SR\_FF* component type.

Table 7-7 State table for the SR\_FF component

Input S	Input R1	Output Q	Output $\bar{Q}$
0	0	Unchanged	Unchanged
0	1	0	1
1	0	1	0
1	1	1	0

When the simulation starts, the output *Q* is set to the value of the *Initial\_State* parameter. *Initial\_State* is set to zero by default.

SR_FF#1		
General	Parameter	Value
Input	Initial_State	False
Output		True
Parameter		False
State		

### 7.1.3.6 Converting values

#### Overview

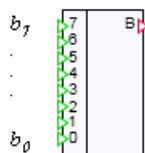
The *Conv* directory of the Standard Library contains component types for converting signals:

- *Bit2Byte* for converting bits to a byte value
- *Byte2Bit* for converting bytes to bits
- *Byte2Word* for converting bytes to a word
- *Word2Byte* for converting a word into bytes
- *Byte2DWord* for converting bytes to a double word
- *DWord2Byte* for converting a double word into bytes
- *Analog2Integer* for converting an analog value into an integer value
- *Integer2Analog* for converting an integer value into an analog value
- *Raw2Phys* for converting a raw value to an analog value
- *Phys2Raw* for converting an analog value to a raw value
- *Unsigned2Signed* for converting an unsigned value into a signed value
- *Signed2Unsigned* for converting a signed value to an unsigned value
- *Real2Byte* for converting a floating point value into its binary representation, and
- *Byte2Real* for converting the binary representation of a floating point number into an analog value.

The connectors of the components for bytes, words and double words are integer connectors. Raw values are integer values too.

#### Bit2Byte – Converting bits into bytes

#### Symbol



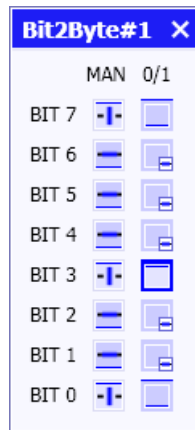
## Function

The *Bit2Byte* component type converts the binary input values  $b_i$ ,  $i = 0, \dots, 7$ , into a byte value  $B$  at the integer output according to

$$B = \sum_{i=0}^7 b_i \cdot 2^i$$

In the conversion,  $b_0$  therefore represents the least significant bit and  $b_7$  the most significant bit.

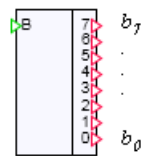
The individual bits  $b_i$  may be set individually while the simulation is running in the component operating window. To set a bit, open the component operating window. Then select the bits that you wish to set manually by pressing the button to the left of those bits. You can then set and reset each bit by pressing the button on the right, which means toggle between zero and one. In the figure below, bits  $b_0$ ,  $b_3$  and  $b_7$  are selected to be set and bit  $b_3$  is set.



Unset bits are identified by a light blue border, while set bits have a dark blue border.

## Byte2Bit – Converting bytes into bits

### Symbol

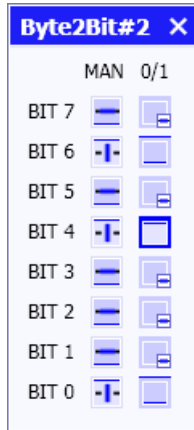


### Function

The *Byte2Bit* component type converts a byte value at the integer input  $B$  into binary values  $b_i$ ,  $i = 0, \dots, 7$  at the outputs. In the conversion,  $b_0$  will be the least significant bit and  $b_7$  the most significant bit of the byte value.

The input is limited to the range of values of one byte, which means to the range from zero to 255. If the input value is not within this range during the simulation, the message "*Byte2Bit: input not a valid byte value*" (message category *ERROR*) is generated.

The individual output bits  $b_i$  may be set individually while the simulation is running in the component operating window. To set a bit, open the component operating window. Then select the bits that you wish to set manually by pressing the button to the left of those bits. You can then set and reset each bit by pressing the button on the right, which means toggle between zero and one. In the figure below, bits  $b_0$ ,  $b_4$  and  $b_6$  are selected to be set and bit  $b_4$  is set:



### Byte2Word – Converting bytes into words

#### Symbol



#### Function

The *Byte2Word* component type combines two byte values  $B_1$ ,  $B_0$  at the integer inputs to form a word at the integer output  $W$ .  $B_1$  is the most significant byte and  $B_0$  is the least significant byte.

The values at the input are limited to the range of values of one byte, which means to the range from zero to 255. If an input value is not within this range during the simulation, the message "*Byte2Word: input not a valid byte value*" (message category *ERROR*) is generated.

### Word2Byte – Converting words into bytes

#### Symbol



## Function

The *Word2Byte* component type breaks down a word at the integer input  $W$  into two byte values  $B_1$ ,  $B_0$  at the integer outputs.  $B_1$  is the most significant byte and  $B_0$  is the least significant byte.

The value at the input is limited to the range of values of one word, which means to the range from zero to  $2^{16}-1$ . If the input value is not within this range during the simulation, the message "*Word2Byte: input not a valid word value*" (message category *ERROR*) is generated.

## Byte2DWord – Converting bytes into double words

### Symbol



### Function

The *Byte2DWord* component type combines four byte values  $B_i$ ,  $i = 0, \dots, 3$  at the integer inputs to form a double word in the order  $B_3 - B_2 - B_1 - B_0$  at the integer output  $DW$ .  $B_3$  is therefore the most significant byte and  $B_0$  is the least significant byte.

The values at the input are limited to the range of values of one byte, which means to the range from zero to 255. If an input value is not within this range during the simulation, the message "*Byte2DWord: input not a valid byte value*" (message category *ERROR*) is generated.

## DWord2Byte – Converting double words into bytes

### Symbol



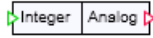
### Function

The *DWord2Byte* component type converts a double word at the integer input  $DW$  into four binary values  $B_i$ ,  $i = 0, \dots, 3$  at the integer outputs.  $B_3$  is the most significant byte and  $B_0$  is the least significant byte.

The value at the input is limited to the range of values of one double word, which means to the range from zero to  $2^{32}-1$ . If the input value is not within this range during the simulation, the message "*DWord2Byte: input not a valid double word value*" (message category *ERROR*) is generated.

### Integer2Analog – Converting from integer to analog

#### Symbol

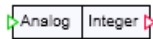


#### Function

The *Integer2Analog* component type converts an integer value at the input into an analog value at the output.

### Analog2Integer – Converting from analog to integer

#### Symbol



#### Function

The *Analog2Integer* component type converts an analog value at the input into an integer value at the output. The integer value is rounded.

### Raw2Phys – Converting from raw to physical

#### Symbol



#### Function

The *Raw2Phys* component type converts an integer value  $x$  at the input to an analog output value  $y$  using the simple linear transformation

$$\frac{y - y_L}{x - x_L} = \frac{y_U - y_L}{x_U - x_L}$$

The input value  $x$  may be a so-called raw value from an automation system, for example. The output value  $y$  is then the value mapped within a defined physical range of values.

The transformation intervals are set using the parameters  $x_L$  (*Raw\_Lower\_Limit*),  $x_U$  (*Raw\_Upper\_Limit*),  $y_L$  (*Phys\_Lower\_Limit*) and  $y_U$  (*Phys\_Upper\_Limit*). They may be modified while a simulation is running and have the following default settings:

- *Raw\_Upper\_Limit*: 27648
- *Raw\_Lower\_Limit*: -27648
- *Phys\_Upper\_Limit*: 100
- *Phys\_Lower\_Limit*: 0

Raw2Phys#1		
General	Parameter	Value
Input	Phys_Lower_Limit	0.0
Output	Phys_Upper_Limit	100.0
Parameter	Raw_Lower_Limit	-27648
State	Raw_Upper_Limit	27648

## Phys2Raw – Converting from physical to raw

### Symbol



### Function

The *Phys2Raw* component type converts an analog value  $x$  at the input to an integer output value  $y$  using the simple linear transformation

$$\frac{x - x_L}{y - y_L} = \frac{x_U - x_L}{y_U - y_L}$$

The input value  $x$  is thus transformed as a measurement of a physical value into the raw value  $y$  for an automation system, for example.

The transformation intervals are set using the parameters  $x_L$  (*Raw\_Lower\_Limit*),  $x_U$  (*Raw\_Upper\_Limit*),  $y_L$  (*Phys\_Lower\_Limit*) and  $y_U$  (*Phys\_Upper\_Limit*). They may be modified while a simulation is running and have the following default settings:

- *Raw\_Upper\_Limit*: 27648
- *Raw\_Lower\_Limit*: -27648

- *Phys\_Upper\_Limit*: 100
- *Phys\_Lower\_Limit*: 0

Phys2Raw#1		
General	Parameter	Value
Input	Phys_Lower_Limit	0.0
Output	Phys_Upper_Limit	100.0
Parameter	Raw_Lower_Limit	-27648
State	Raw_Upper_Limit	27648

### Unsigned2Signed – Converting from unsigned to signed

#### Symbol



#### Function

The *Unsigned2Signed* component type converts an unsigned integer value *x* at the input to a signed value *y* at the output.

The parameter *Width* determines which data width is assigned to the input value: *1 byte*, *2 bytes* or *4 bytes*. The output value is limited in accordance with the set data width. The conversion is done as follows:

Data width: 1 byte

$$y = \begin{cases} -1, & \text{if } x > 2^8 - 1 \\ 0, & \text{if } x < 0 \\ x - 2^8, & \text{if } x > 2^7 - 1 \\ x, & \text{otherwise} \end{cases}$$

Data width: 2 bytes

$$y = \begin{cases} -1, & \text{if } x > 2^{16} - 1 \\ 0, & \text{if } x < 0 \\ x - 2^{16}, & \text{if } x > 2^{15} - 1 \\ x, & \text{otherwise} \end{cases}$$

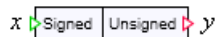
Data width: 4 bytes



$$y = \begin{cases} -1, & \text{if } x > 2^{32} - 1 \\ 0, & \text{if } x < 0 \\ x - 2^{32}, & \text{if } x > 2^{31} - 1 \\ x, & \text{otherwise} \end{cases}$$

## Signed2Unsigned – Converting from signed to unsigned

### Symbol



### Function

The *Signed2Unsigned* component type converts a signed integer value  $x$  at the input to an unsigned value  $y$  at the output.

The parameter *Width* determines which data width is assigned to the output value: *1 byte*, *2 bytes* or *4 bytes*. The output value is limited in accordance with the set data width. The conversion is done as follows:

Data width: 1 byte

$$y = \begin{cases} -2^7, & \text{if } x < -2^7 \\ 2^7 - 1, & \text{if } x > 2^7 - 1 \\ x + 2^8, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

Data width: 2 bytes

$$y = \begin{cases} -2^{15}, & \text{if } x < -2^{15} \\ 2^{15} - 1, & \text{if } x > 2^{15} - 1 \\ x + 2^{16}, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

Data width: 4 bytes

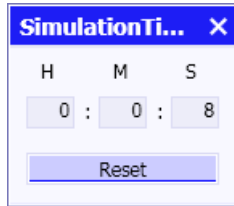
$$y = \begin{cases} -2^{31}, & \text{if } x < -2^{31} \\ 2^{31} - 1, & \text{if } x > 2^{31} - 1 \\ x + 2^{32}, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$





In slow-motion simulation mode, the simulation time is slower than real time; in quick-motion simulation mode, the simulation time is faster than real time. If the simulation is paused, the simulation time is stopped as well. When the simulation is resumed, the simulation time starts running again as well.

The simulation time is also displayed in the component operating window.

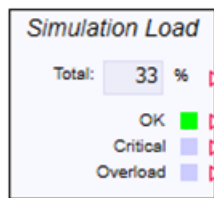


Press the *Reset* button in the component operating window to reset the simulation time to zero.

### SimulationLoad - simulation load

#### "SimulationLoad" components

##### Symbol



##### Function

The *SimulationLoad* component shows the current simulation load. You can find information about the simulation load in the section: Display of the simulation load (Page 31)

The symbol shows the total load in % under "Total". All values in this component are not limited and not smoothed. Therefore, they sometimes may fluctuate more than the value in the symbol for the ongoing simulation.

The three binary displays, *OK*, *Critical* and *Overload*, provide a rough assessment of the load value with the following distribution:

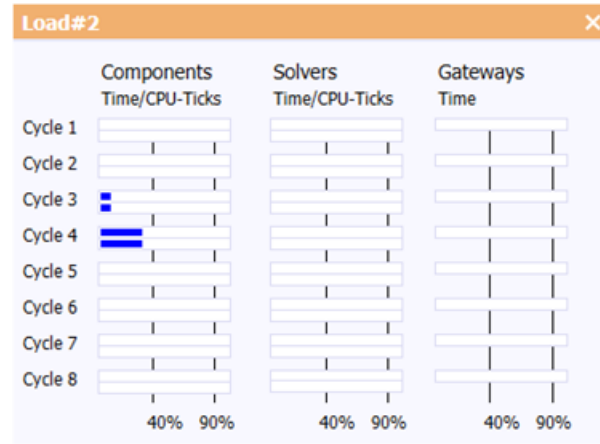
Load < 40%	OK
40% < Load < 90%	Critical
Load > 90%	Overload

The time slice containing the component for the load evaluation only has an impact on how often the values are updated. However, SIMIT always evaluates the values of all eight possible time slices. The worst value since the last update is used.

##### Operating window

This component shows the load values for all eight time slices in the operating window. The following causes are distinguished:

- Components
- Solution procedures (solvers)
- Couplings (gateways)



SIMIT distinguishes between *Time* and *CPU ticks*.

### Time

*Time* the percentage ratio of the actual cycle time to the configured cycle time.

The module with the worst time value is the closest to the overload limit. However, this does not necessarily mean that there is too much calculation in this time slice.

The computational load is distributed across multiple available cores. Usually, a computing core must work through more than one time slice. Time slices with shorter cycle times have a higher priority. This means that the calculation of a slower time slice may be interrupted to permit the calculation of a faster time slice. This is important to calculate all cycles within the configured cycle time, if possible.

The result can be that the calculation in a slow time slice is constantly being interrupted by the calculation of a faster time slice, thereby creating a load problem. The faster time slice, in contrast, is calculated with a higher priority and can therefore have a lower load.

### CPU ticks

To better identify the real cause of a high simulation load, *CPU ticks* are additionally available. They do not indicate the time that was needed for the calculation, but rather a measure for the time spent on the calculation operations. They do not take into account wait times due to interruptions by other time slices. Because the measured absolute value of CPU ticks is not very meaningful, it is normalized:

$$ModelTicks_i' = ModelTicks_i \frac{MAX_{n=1..8}(ModelTime_n + SolverTime_n)}{MAX_{n=1..8}(ModelTicks_n + SolverTicks_n)}$$

$$SolverTicks_i' = SolverTicks_i \frac{MAX_{n=1..8}(ModelTime_n + SolverTime_n)}{MAX_{n=1..8}(ModelTicks_n + SolverTicks_n)}$$

*ModelTicks<sub>n</sub>'*: Displayed value for the computational load of the component code in time slice *n*

*SolverTicks<sub>n</sub>'*: Displayed value for the computational load of the solution procedure in time slice *n*

*ModelTicks<sub>n</sub>*: Measured value for the computational operations of the component code in time slice *n*

*SolverTicks<sub>n</sub>*: Measured value for the computational operations of the solution procedure in time slice *n*

*ModelTime<sub>n</sub>*: Ratio of the time required for the calculation of the component code to the configured cycle time in time slice *n*

*SolverTime<sub>n</sub>*: Ratio of the time required for the calculation of the solution procedure to the configured cycle time in time slice *n*

The ratio of tick values to each other shows where most arithmetic operations are performed in a given time. Relief should be provided in the time slice with the highest values, for example, by relocating parts of the simulation model to a slower time slice or setting the entire critical time slice to a slower speed.

**Simulation load by couplings**

When calculating the tick values, it is assumed that the components and solution procedures only perform calculations and do not perform any file or console tasks. Although these commands require time, they do not fully utilize the calculation cores and therefore lead to misleading information.

Therefore, the couplings are not included in the count of arithmetic instructions. If a high simulation load is caused by a coupling, there are two reasons for this:

- The coupling is being interrupted too often by higher priority calculations
- The coupling itself demands a great deal of time

In the second case, reducing the computational work in the area of components or solution procedure is not very helpful. In general, the coupling must be assigned to a slower time slice, the signal range of the coupling must be reduced, or a part of the signals must be relocated to a second coupling with a slower cycle time.

**Output signals**

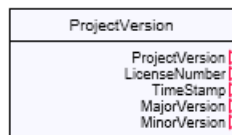
The *SimulationLoad* component has output signals for operating controls in the operating window. These signals are not interconnected to the component symbol, but can be monitored in the property view.

Output signal	Meaning
LoadModel <sub>n</sub> Ticks	Normalized computational work for the component code in time slice <i>n</i>
LoadModel <sub>n</sub> Time	Percentage ratio of the time required for the calculation of the component code to the configured cycle time in time slice <i>n</i>
LoadSolver <sub>n</sub> Ticks	Normalized computational work for the solution procedure in time slice <i>n</i>

LoadSolver <sub>n</sub> Time	Percentage ratio of the time required for the calculation of the solution procedure to the configured cycle time in time slice n
LoadGateway <sub>n</sub> Time	Percentage ratio of the time required for the calculation of the coupling to the configured cycle time in time slice n

## ProjectVersion – Project version

### Symbol

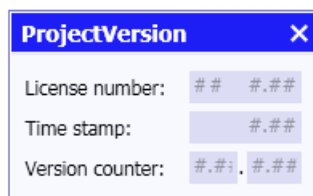


### Function

The following integer values are available at the outputs of a component of this type:

- **ProjectVersion**  
The complete version number coded in numerical form.
- **LicenseNumber**  
The license number coded in numerical form.
- **TimeStamp**  
The time stamp
- **MajorVersion**  
The major version
- **MinorVersion**  
The minor version

The individual version numbers are displayed in readable form in the component operating window.






### Connectors

Three connector component types – referred to simply as connectors – are available:

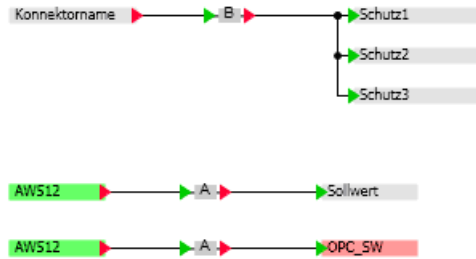
- The analog connector component type *AConnector*,
- the binary connector component type *BConnector*, and
- the integer connector component type *IConnector*.

**Connector components**

-  Analog connector
-  Binary connector
-  Integer connector

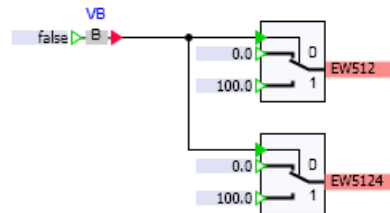
A connector component transfers the input value to its output without change or delay. They are used, for example, in the following cases:

- Connectors (I/O connectors or global connectors) cannot be connected directly to each other. As shown in the example in the figure below, connectors can be connected with the aid of connector components.

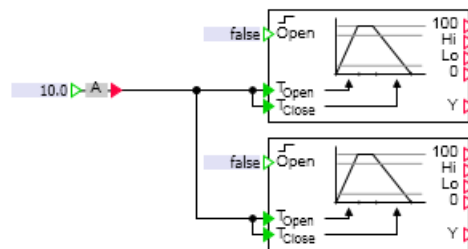


- An input element, for example a converter, is linked directly to the signal to be set. With the aid of a connector, the input element can influence several components directly. As shown in the example in the figure below, the input signal *IN* of the connector *VB* is linked to the input element.

Switch#1	
General	Signal
Connector	Signal VB IN



- Component inputs can be directly assigned a default value. With the aid of a connector, several inputs can be assigned a single default value. As shown in the example in the figure below, a connector is connected to the inputs of the components to be set. The value to be set is then set at the input of the connector.








## Auxiliary components for the creation of macros

Three so-called auxiliary component types are provided for macros:

- The analog constant component type *AConst*,
- The binary constant component type *BConst* and
- The integer constant component type *IConst*

### Constant component types for macros

	Analog constant
	Binary constant
	Integer constant

Parameters of a macro component can only be built from parameters of the components that are used within the macro component. The constant component types thus also provide a parameter for setting inputs of a component as a macro parameter.

You can find additional information on this in the section: Defining parameters of macro components (Page 185)

## 7.1.4 Drive components

### 7.1.4.1 Overview

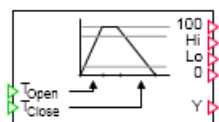
In the directory *DRIVES* of the Basic Library you will find component types for simulation of drives. These component types form the Drive Library. They can be used to simulate general drives for valves and pumps. The types

- *DriveP1* and *DriveP2* are designed for simulating pump drives, while the types
- *DriveV1* to *DriveV4* are intended for simulating valve drives.

The subdirectory *PROFIdrives* contains components for the simulation of variable-speed drives corresponding to the PROFIdrive profile. Components for the simulation of SIMOCODE motor management and control devices can be found in the subdirectory *SIMOCODEpro*.

### 7.1.4.2 Valve drives

Four component types are available (*DriveV1*, *DriveV2*, *DriveV3* and *DriveV4*) to simulate valve drives. All four of these component types share the outputs and some of the inputs as illustrated in the figure below.



The two analog inputs *TOpen* and *TClose* indicate the opening and closing times of the drive, which means the time that the drive takes to fully open or close. If one of the two input values is negative during simulation, the message "*DriveVx: closing or opening time invalid value*" (message category *ERROR*) is generated.

The current position value of the drive is output at analog output *Y* as a percentage, which means in the range from zero to one hundred:

$$0 \leq Y \leq 100.$$

The value zero corresponds to the fully closed valve, while the value one hundred represents the fully open valve.

The four binary outputs *100*, *Hi*, *Lo* and *0* may be interpreted as limit switches for the drive:

- *100* and *0* as limit switches "Valve fully opened" or "Valve fully closed",
- *Hi* and *Lo* as pre-limit switches for "Valve open" or "Valve closed".

The limit switches are permanently set to the position values zero (for the fully closed valve) and one hundred (for the fully open valve). Binary outputs *100* or *0* are therefore set to one when the valve is fully closed or open, which means when the position *Y* of the component has the value zero or one hundred.

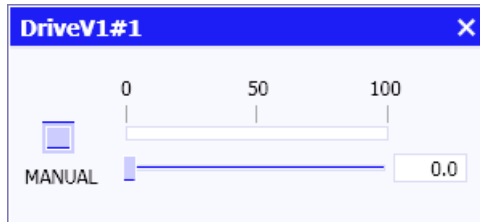
The *Hi* and *Lo* pre-limit switches can be set using the *HI\_Limit* or *LO\_Limit* parameters. Their values are between 0 and 100. Position values of five (*LO\_Limit*) and 95 (*HI\_Limit*) are set by default.

If one of the two parameter values is negative during the simulation, the message "*DriveVx: high and low parameters do not match*" (message category *ERROR*) is generated.

DriveV1#1		
General	Name	Value
Input	HI_Limit	95.0
Output	Initial_Value	Closed
Parameter	LO_Limit	5.0
State		

The *Initial\_Value* parameter is used to set the valve drive to the starting position of *Closed* or *Open*. The default setting for *Initial\_Value* is *Closed*.

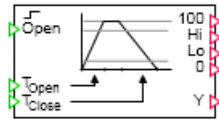
The current position value of the drive is visualized as a bar graph display in the component operating window. The button on the left labeled *MANUAL* is used to change the position value to the desired setting by moving the slider in the operating window.



The position value is then no longer derived from the component inputs, but rather tracks the value set using the slider. The set opening and closing times remain active. The four binary outputs *100*, *Hi*, *Lo* and *0* likewise all remain effective. They are also set as described above if the position value is to be tracked.

## DriveV1 – Type 1 valve drive

### Symbol

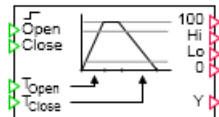


### Function

The *DriveV1* component type simulates a drive unit that sets the position value  $Y$  at the output in relation to a binary value at the input *Open*. If the binary input is equal to zero, the position value  $Y$  is continuously tracked to zero with the closing time  $T_{Close}$ . If the binary input is equal to one, the position value is continuously tracked to one hundred with the opening time  $T_{Open}$ . The position value thus only has the two steady states of zero and one hundred, which means the states "Valve open" and "Valve closed" in relation to the valve function. The drive or the valve is thus opened or closed whenever the binary value at the input changes.

## DriveV2 – Type 2 valve drive

### Symbol



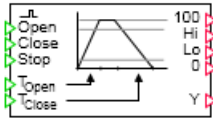
### Function

The *DriveV2* component type simulates a drive unit that sets the position value  $Y$  at the output in relation to two binary values at the *Open* and *Close* inputs. If the binary value at input *Open* is equal to one, the position value  $Y$  is continuously tracked to one hundred with the opening time  $T_{Open}$ . If the binary value at the input *Close* is equal to one, the position value is continuously tracked to zero with the closing time  $T_{Close}$ .

If both input values are set to one or zero at the same time, the position value remains unchanged. The position value thus only changes if the binary value at either the *Open* input or the *Close* input is set to one.

### DriveV3 – Type 3 valve drive

#### Symbol

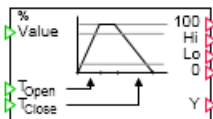


#### Function

The *DriveV3* component type simulates a drive unit that sets the position value *Y* at the output in relation to three binary values at the *Open*, *Close* and *Stop* inputs. If the binary value at the *Open* input changes from zero to one, the position value *Y* is continuously tracked to one hundred with the opening time  $T_{Open}$ . The position value is tracked continuously to zero with the closing time  $T_{Close}$  if the binary value at the *Close* input changes from zero to one. If the binary value at the *Stop* input changes from zero to one, the position value remains unchanged. Opening, closing and stopping of the drive is thus initiated via the edge of the corresponding binary input signal (signal change from zero to one).

### DriveV4 – Type 4 valve drive

#### Symbol

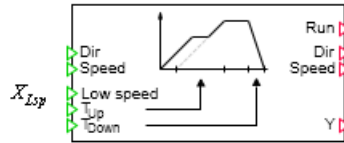


#### Function

The *DriveV4* component type simulates a drive unit that sets the position value *Y* at the output that continuously tracks the analog value at the *Value* input. The tracking occurs in the direction of rising position values with the opening time  $T_{Open}$  and falling position values with the closing time  $T_{Close}$ . The position value is always limited to values from zero to one hundred, even if the input value is outside this interval.

### 7.1.4.3 Pump and fan drives

The two component types *DriveP1* and *DriveP2* may be used in the simulation as drives for pumps, fans or similar units. The outputs and some of the inputs common to these two component types are illustrated in the figure below:



The run-up and run-down speeds of the drive are set at the two analog inputs  $T_{Up}$  and  $T_{Down}$ .  $T_{Up}$  is the time it takes the drive to run up from standstill to the nominal speed in seconds, while  $T_{Down}$  is the time in seconds it takes to run the drive down from nominal speed to standstill. Both times have a default value of one second. If one of the two input values is negative during simulation, the message "*DriveVx: run-up or run-down time invalid value*" (message category *ERROR*) is generated.

The current speed value of the drive is output at analog output  $Y$  as a percentage, which means in the range from zero to one hundred:

$$0 \leq Y \leq 100.$$

The value zero corresponds to the stopped drive, while the value one hundred represents the drive at nominal speed.

The direction of rotation of the drive can be set using binary input *Dir*. When this input is set to zero, the direction of rotation is positive. If the input is set to one, the direction of rotation is negative. This input can therefore be used to specify whether the drive turns clockwise or counterclockwise, for example. If it turns in the negative direction, the speed is output as a negative value at output  $Y$ :

$$-100 \leq Y \leq 0.$$

Positive values for  $Y$  thus indicate speeds in the positive direction, while negative values indicate speeds in the negative direction. The change in direction of rotation only takes effect if the drive is stopped.

The binary input *Speed* is used to toggle the speed between nominal speed (full speed) and partial speed (low speed). If *Speed* is one, the nominal speed was selected; otherwise the partial speed was selected. The default setting for the *Speed* binary input is one. The partial speed is specified as a numerical value (percentage) at the analog input  $x_{LSp}$ :

$$0 \leq x_{LSp} \leq 100.$$

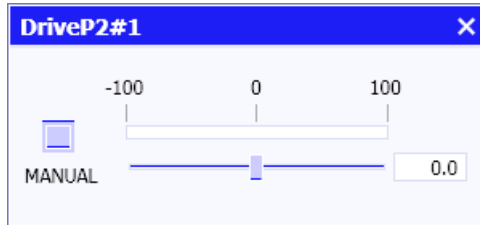
The default setting for the partial speed is 50, which means half the nominal speed. If the partial speed is not within the defined range, the message "*DrivePx: low speed invalid value*" (message category *ERROR*) is generated.

The binary output *Run* is only set to one when the drive has reached the preset speed value in the positive or negative direction of rotation, which means only if the absolute value at the analog output  $Y$  is equal to one hundred (nominal speed) or is equal to the partial speed set at the input  $x_{LSp}$ .

The feedback signal for the selected speed is available at the binary output *Speed*. The binary output is only set to one if the drive has reached its nominal speed in the positive or negative direction of rotation.

The value at the binary output *Dir* takes the form of a feedback signal for the current direction of rotation of the drive. The binary output is zero if the drive is turning in the positive direction; the value is one if the drive is turning in the negative direction.

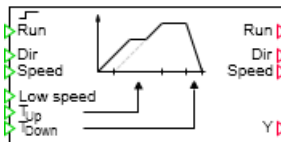
The current speed value of the drive is visualized as a bar graph display in the component operating window. The button on the left labelled *MANUAL* is used to change the speed value to the desired setting by moving the slider in the operating window.



The speed value is then no longer derived from the component inputs, but rather tracks the value set using the slider. The set run-up and run-down times remain active. The binary outputs *Run*, *Speed* and *Dir* likewise remain active. They are also set as described above if the speed value is to be tracked.

### DriveP1 – Type 1 pump drive

#### Symbol

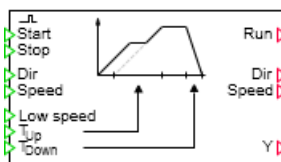


#### Function

The *DriveP1* component type simulates a drive unit that is switched on and off via the binary value at the *Run* input. The drive is switched on while the input value is set to one. If the input value is set to zero, the drive is switched off.

### DriveP2 – Type 2 pump drive

#### Symbol



## Function

The *DriveP2* component type simulates a drive unit that is switched on and off via the two binary inputs *Start* and *Stop*. If the binary value *Start* changes from zero to one, then the drive is switched on. If the binary value *Stop* changes from zero to one, the drive is switched off. Switching the drive on and off is thus initiated via the edge of the corresponding binary input signal (signal change from zero to one).

### 7.1.4.4 PROFIdrive devices

Many variable-speed drives are connected directly to the controller via PROFIBUS DP. To standardize the configuration of such drives, a profile called PROFIdrive was created by the PROFIBUS User Organization (PNO). This profile defines how a drive matching this profile behaves on the PROFIBUS DP with regard to some of its more important functions. The following power converters or frequency converters from the Siemens range of products meets the requirements of a PROFIdrive:

- SIMOREG DC Master
- SIMOVERT Masterdrive
- Micromaster
- Sinamics

A common feature of all PROFIdrive devices is that they are activated by a control word (STW) and provide feedback in a status word (ZSW). In addition, the drives receive a speed setpoint from the controller and provide the actual speed as feedback.

The functionality of a PROFIdrive can be divided roughly into two blocks:

- state machine and
- ramp generator.

The state machine defines the transitions from one status to another. A change in status is triggered by a particular bit in the control word. The current status can be seen in the status word. The purpose of the ramp generator of a PROFIdrive is to convert jumps in the speed setpoint into linear ramp increases with an adjustable gradient.

In addition to this basic function, PROFIdrive devices can of course offer considerably more functions and configuration options. These additional functions are, however, no longer included in the PROFIdrive profile.

### The PROFIdrive library for the simulation of PROFIdrive devices

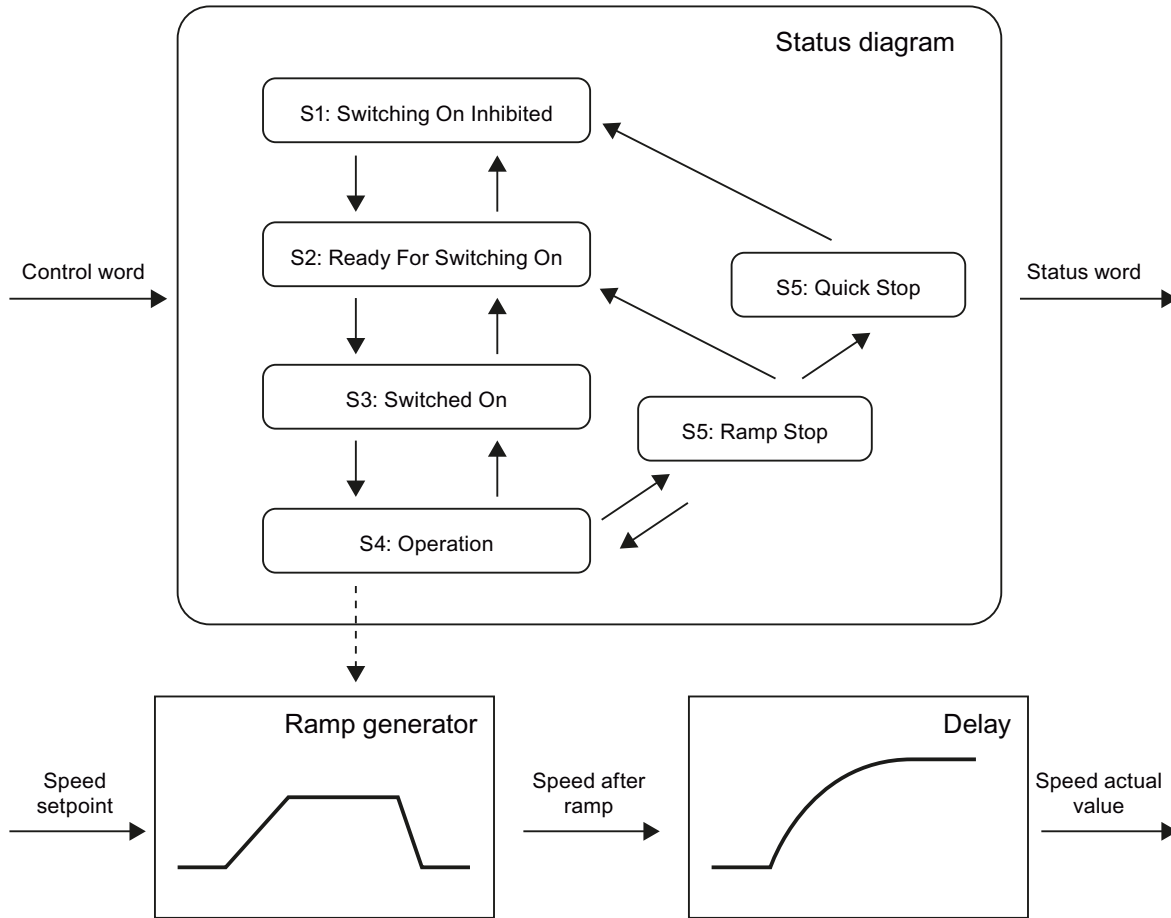
All component types for the simulation of PROFIdrive devices can be found in the *PROFIdrive* directory of the Drives Library:

- The PROFIdrive header component type *PROFIdrive* and
- the PROFIdrive device-specific component types *DCMaster*, *Masterdrive*, *Micromaster3*, *Micromaster4*, *Sinamics* and *Universal*.

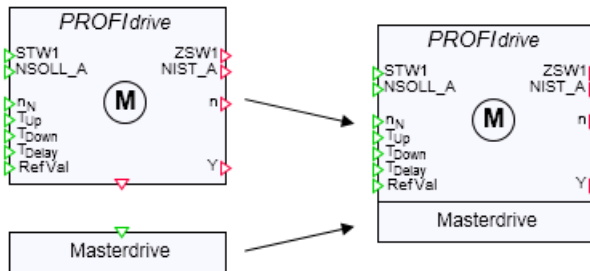
These component types make up the PROFIdrives library.

The state machine and ramp generator functions contained in all PROFIdrive devices are simulated in the *PROFIdrive* component type. In addition, the ramp function value is maintained

by a first order delay element with an adjustable delay time in order to at least partially simulate the inertia of the drive.



Some of the bits in the control and status words are not defined in the PROFIdrive profile. The five bits 11 to 15 in the control and status word can be drive-specific and may therefore have different meanings according to the type of drive. For the SIMOVERT MASTERDRIVE, SINAMICS and MICROMASTER AC converters and the SIMOREC DC-Master DC converters, these drive-specific functions are emulated in the corresponding device-specific components. These drive-specific components are simply attached to the PROFIdrive header component, as illustrated in the figure below.

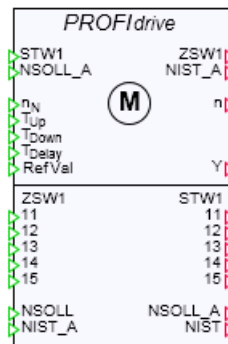




The device-specific component types in the *PROFdrive* library contain standardized emulations of different PROFdrive devices. You can find additional information on this in the section: Specific PROFdrive devices (Page 384).

The meaning of the drive-specific bits can be changed by configuring the converter systems and may therefore deviate from the standard configuration illustrated here. If this is the case, the device-specific component type is set to *Universal*.

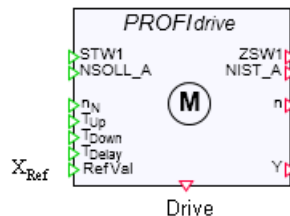
You can find additional information on this in the section: Universal – Extensions to the PROFdrive basic function (Page 382). This component type provides you with bits 11 to 15 of the control and status word as component inputs and outputs, which can be connected as required to other components in the basic library.



In addition to the control and status word and the speed setpoint and actual speed values, PROFdrive devices can exchange other information with the controller. In SIMIT, these additional I/O signals are available via the PROFIBUS DP coupling as regular I/O signals and can therefore be used for specific adaptation of a drive simulation.

## PROFdrive – Basic function of the PROFdrive device

### Symbol

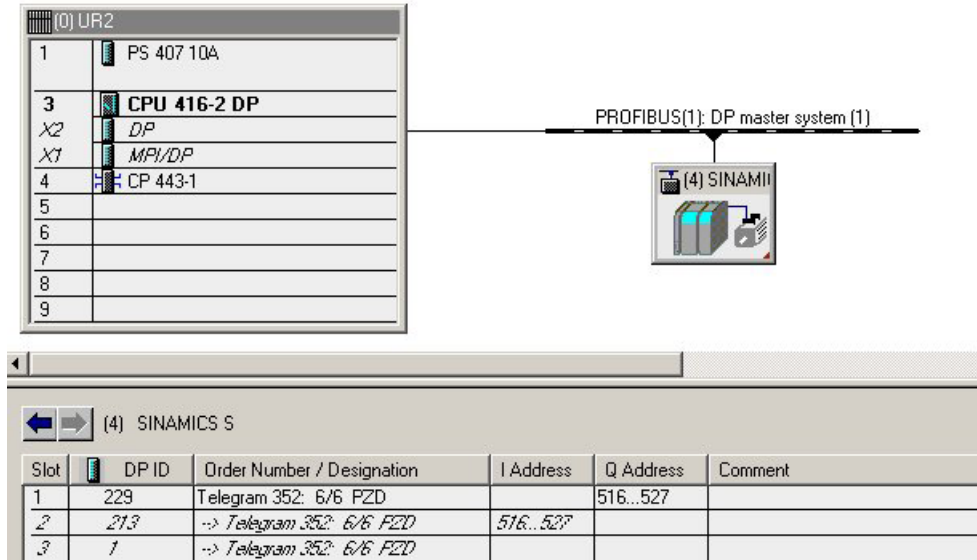


### Function

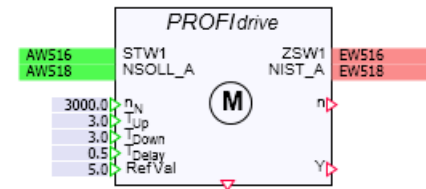
The *PROFdrive* component type emulates the state machine with the ramp function generator according to the PROFdrive profile (Technical Specification, Version 4.1, Edition: May 2006). The inertia of the drive is emulated with a first order delay.

Functionalities for Application Class 1 (AC1) are emulated: "Standard Drive – Speed Control Mode".

The starting point for configuring a *PROFdrive* component is the controller access to the input and output signals of the PROFdrive device. For each drive, a fixed address range is defined within the controller input and output area, and assigned to a PROFdrive component as follows: The control word (first process data word in the output area) is connected to the integer input *STW1* and the status word (first process data word of the controller inputs) is connected to the integer output *ZSW1*. The speed setpoint and actual speed values are transferred in the second process data word. Accordingly, the integer input *NSOLL\_A* is connected to the speed setpoint value. The integer output *NIST\_A* is connected to the speed actual value of the controller. In the figure below, an example of a SIMATIC configuration for SINAMICS is shown.



The figure below shows the *PROFdrive* component and the connected process data:



The process data speed values, which means *NSOLL\_A* and *NIST\_A* are raw values in the range zero to 16384. The value 16384 is the nominal speed. Other process data, for example, torque values or actual current values, are not taken into account in the *PROFdrive* components.

The time in seconds can be specified at inputs  $T_{Up}$  and  $T_{Down}$ . These are the times the ramp generator takes to run up to the nominal speed or to run down to a standstill from the nominal speed. The default values for  $T_{Up}$  and  $T_{Down}$  are three seconds.  $T_{Delay}$  is the delay with which the speed actual value tracks the effective setpoint value (ramp function generator value) via the delay function. The default delay time is half a second.

The reference value at analog input *RefVal* is a percentage speed value. If the speed actual value exceeds the reference value, bit 10 of the status word (*ZSW1.10*) is set to one. The default reference value is five percent. The switchover of this signal is performed using a configurable hysteresis. This is also stated as a percentage speed value in the *Hysteresis* parameter. The default hysteresis value is three percent.

The current speed value of the drive is output at analog output  $Y$  as a percentage, which means in the range from zero to one hundred:  $0 \leq Y \leq 100$ . A value of zero thus corresponds to a stationary drive, while a value of one hundred corresponds to the nominal speed (raw speed value 16384). The nominal speed in rpm can be specified at analog input  $n_N$ . The actual speed in rpm is output at analog output  $n$ . The default is  $n_N$  with 3000 rpm.

The parameter *Tolerance* is a speed value as a raw value. If the speed actual value and the ramp generator value deviate by more than the value specified in *Tolerance*, bit 8 of the status word (ZSW1.8) is set to one. *Tolerance* has a default value of 50.

PROFIdrive#1		
General	Parameter	Value
Input	Hysteresis	3.0
Output	Tolerance	50.0
Parameter		
State		

The contents of the control word and status word are displayed bit-by-bit in the operating window of the component. In addition, the setpoint speed value *NSOLL\_A*, the ramp generator value *NIST\_RFG*, and the speed actual value *NIST\_A*, are displayed as percentage values and in revolutions per minute (*RPM*). The fourth bit and eighth bit in the status word (ZSW1.3 - *Fault* and STW1.7 - *Warning*) can be set to one using a toggle switch.

**PROFIdrive#1** ✕

STW1	ZSW1
On <input type="checkbox"/>	<input type="checkbox"/> Ready To Switch On
No Coast Stop <input type="checkbox"/>	<input type="checkbox"/> Ready To Operate
No Quick Stop <input type="checkbox"/>	<input type="checkbox"/> Operation Enabled
Enable Operation <input type="checkbox"/>	<input type="checkbox"/> Fault
Enable RFG <input type="checkbox"/>	<input type="checkbox"/> No Coast Stop
Unfreeze RFG <input type="checkbox"/>	<input type="checkbox"/> No Quick Stop
Enable Setpoint <input type="checkbox"/>	<input checked="" type="checkbox"/> Switching On Inhibited
Fault Acknowledge <input type="checkbox"/>	<input type="checkbox"/> Warning
8 <input type="checkbox"/>	<input checked="" type="checkbox"/> Speed Error Within Tolerance
9 <input type="checkbox"/>	<input type="checkbox"/> Control Requested
Control by PLC <input type="checkbox"/>	<input type="checkbox"/> Speed Reached
11 <input type="checkbox"/>	<input type="checkbox"/> 11
12 <input type="checkbox"/>	<input type="checkbox"/> 12
13 <input type="checkbox"/>	<input type="checkbox"/> 13
14 <input type="checkbox"/>	<input type="checkbox"/> 14
15 <input type="checkbox"/>	<input type="checkbox"/> 15

---

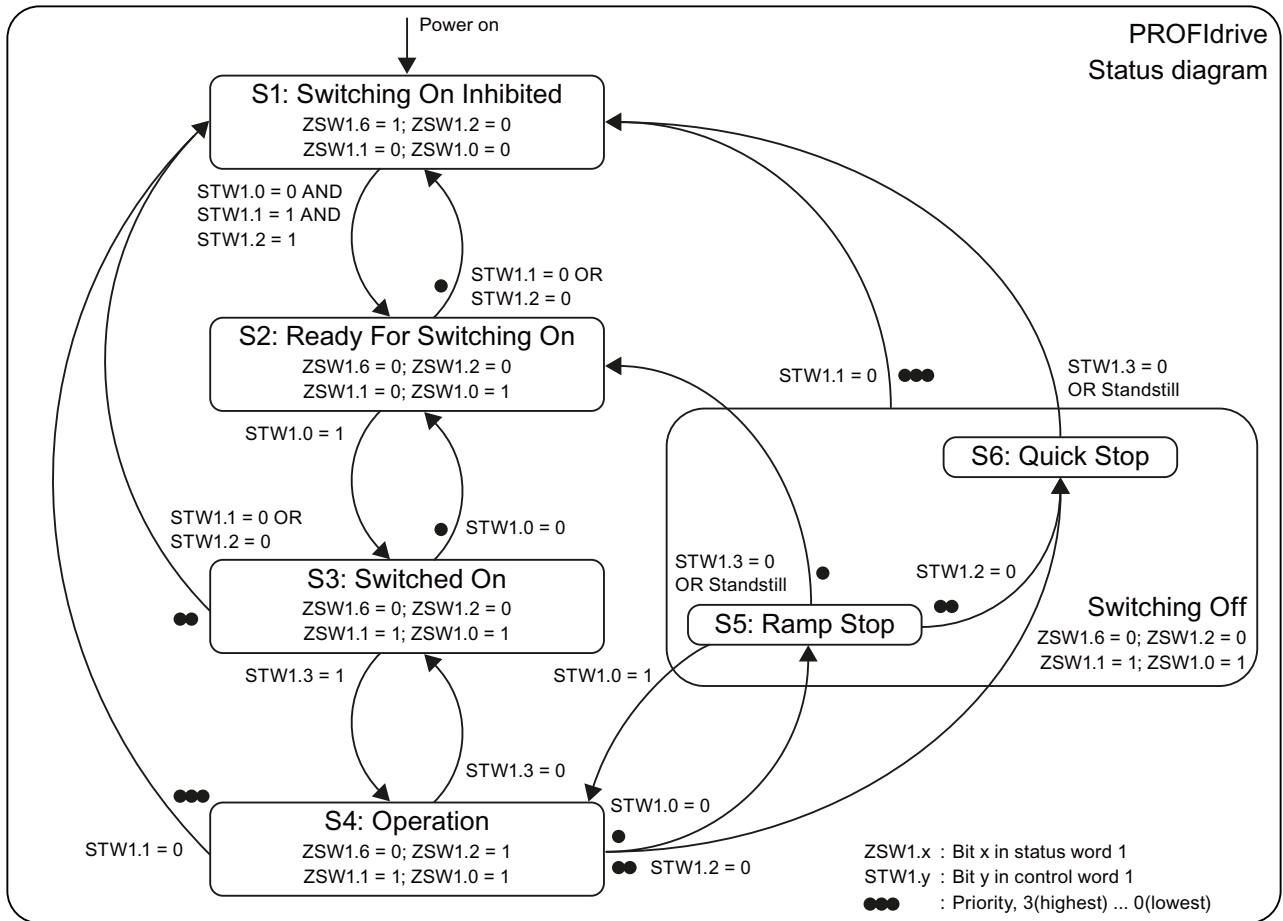
NSOLL_A	<input type="text" value="0.00"/>	%	<input type="text" value="0.00"/>	RPM
NIST_RFG	<input type="text" value="0.00"/>	%	<input type="text" value="0.00"/>	RPM
NIST_A	<input type="text" value="0.00"/>	%	<input type="text" value="0.00"/>	RPM

A specific *Drive* connector on the bottom of the component is used to connect a device-specific component. You can find additional information on this in the section: The PROFIdrive library for the simulation of PROFIdrive devices (Page 375).

This connector is of the *PROFdrive* connection type. It cannot be connected to the analog, binary or integer connectors of components.

**The state machine**

The state machine is described in detail in the PROFdrive specification. The state graph implemented in the *PROFdrive* component type is illustrated in the figure below:



The control bits shown in the table are used to control the state machine (status transitions). Depending on the current status, the corresponding status bits are set as illustrated in the table.

Table 7-8 Status table for the PROFdrive component

State	State	Description	STW1 (bit sequence 15 ... 0)
S1: Switching On Inhibited	1	Off	dddd dmsm a1ss a000
S2: Ready For Switching On	2	Ready	dddd dmsm a0ss a001
S3: Switched On	3	Switched on	dddd dmsm a0ss a011
S4: Operation	4	Normal operation	dddd dmsm a0ss a111
S5: Switching Off (Ramp Stop)	5	Ramp stop of the motor	dddd dmsm a0ss a011
S6: Switching Off (Quick Stop)	6	Quick stop of the motor	dddd dmsm a0ss a011

The individual bits in status word *STW1* shown in the table have the following meaning:

d	device-specific (device-specific component)
s	derived directly from <i>STW1</i>
m	model-specific (here: delay)
a	can be set in the operating window

Table 7-9 Structure of control word

Name	STW1.Bit	Description	Purpose
On/Off (Off1)	STW1.0	Switch off drive	Yes
No Coast Stop (Off2)	STW1.1	No coast stop of drive	Yes
No Quick Stop (Off3)	STW1.2	No quick stop of drive	Yes
Enable Operation	STW1.3	Drive moves to setpoint	Yes
Enable Ramp Generator	STW1.4	RFG is used	Yes
Unfreeze Ramp Generator	STW1.5	RFG frozen	Yes
Enable Setpoint	STW1.6	NSOLL as input for RFG	Yes
Fault Acknowledge	STW1.7	Fault acknowledgement from PLC	Yes
Jog 1 On	STW1.8	not implemented	No
Jog 2 On	STW1.9	not implemented	No
Control by PLC	STW1.10	DO IO data valid	Yes
Device-specific	STW1.11-15		Device-specific component

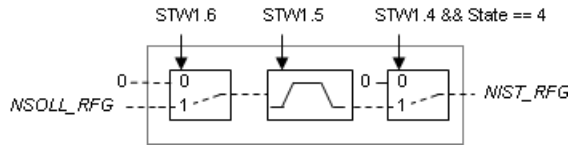
The meaning of all the status word bits is described in the table below.

Table 7-10 Structure of the status word

Name	STW1.Bit	Description	Source
Ready To Switch On	STW1.0	Power supply switched on	State machine
Ready To Operate	STW1.1	No fault present	State machine
Operation Enabled	STW1.2	Operation enabled	State machine
Fault Present	STW1.3	Drive faulty/out of service	Operating window
Coast Stop Not Active	STW1.4	No coast stop of drive	STW1.1
Quick Stop Not Active	STW1.5	No quick stop of drive	STW1.2
Switching On Inhibited	STW1.6	Switching on inhibited	State machine
Warning Present	STW1.7	Warning present/out of service	Operating window
Speed Error Within Tolerance	STW1.8	Setpoint/actual deviation in tolerance range	PT1
Control Requested	STW1.9	Control requested/local operation	STW1.10
Speed Reached	STW1.10	Reference speed reached	PT1
Device-specific	STW1.11-15	Drive-specific	Additional component/0

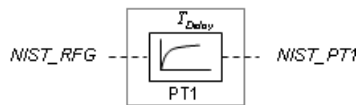
### The ramp generator

The ramp function generator (RFG) is controlled by bits 4, 5 and 6 of the control word (STW1.4, STW1.5, STW1.6). Bit 6 sets the input of the generator to *NSOLL\_RFG*. The generator then ramps up or down to this setpoint value in a linear manner using the time values  $T_{Up}$  and  $T_{Down}$ , assuming the generator is enabled via bit 5. Otherwise, the last output value of the generator is retained. Finally, bit 4 determines whether the ramp generator value (actual value) *NIST\_RFG* is output to the PT1 element. Forwarding of the actual value to the delay element can only be carried out in state S4. Otherwise the actual ramp value 0 is output to the delay element.



### Delay element

The speed actual value *NIST\_PT1* follows the ramp generator value *NIST\_RFG* after a delay by means of the delay element. This at least partly simulates the behavior of the drive under load. The delay time constant  $T_{Delay}$  can be set on the component type input. By setting the delay time constant to zero,  $T_{Delay} = 0$ , the delay element can be deactivated. The speed actual value then corresponds to the ramp generator value.

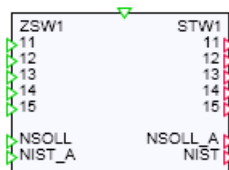


### Use of the PROFIdrive component type without extension

The *PROFIdrive* component type implements the standard functions according to the PROFIdrive profile. The *PROFIdrive* component can therefore be used to simulate PROFIdrive devices with this basic function, without being connected to a device-specific component. Drive-specific bits in the control word are then ignored and drive-specific bits in the status word are set to zero. The speed variables are linked as follows: *NSOLL\_RFG* = *NSOLL\_A* and *NIST\_A* = *NIST\_PT1*.

### Universal – Extensions to the PROFIdrive basic function

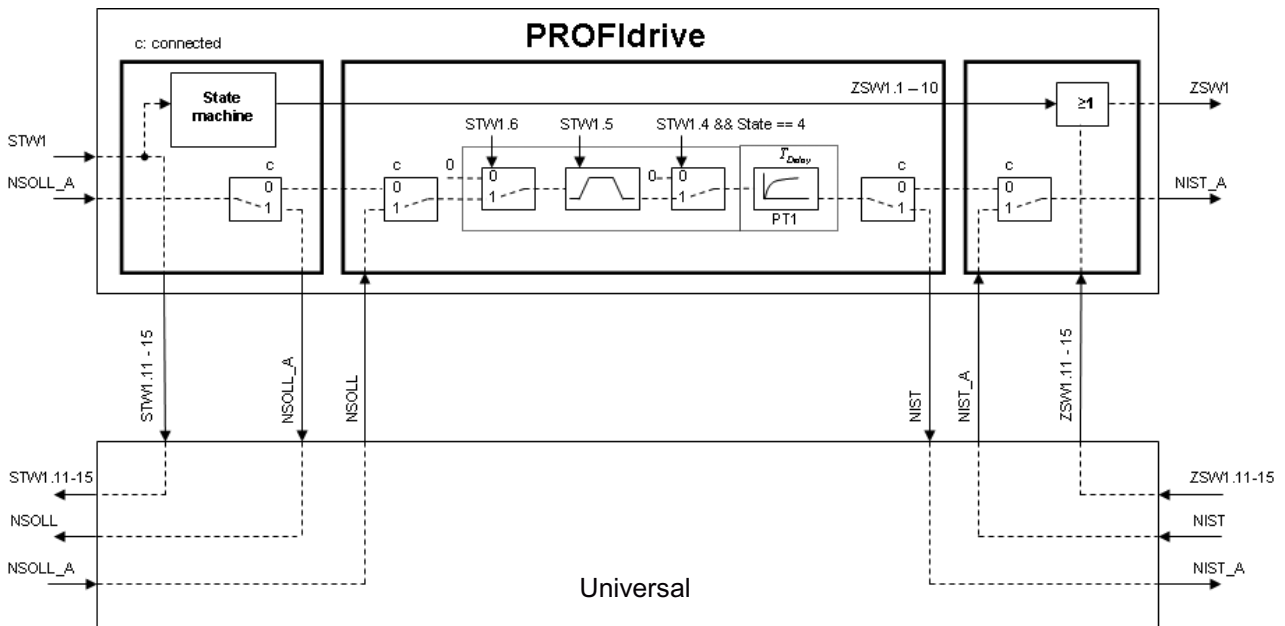
#### Symbol



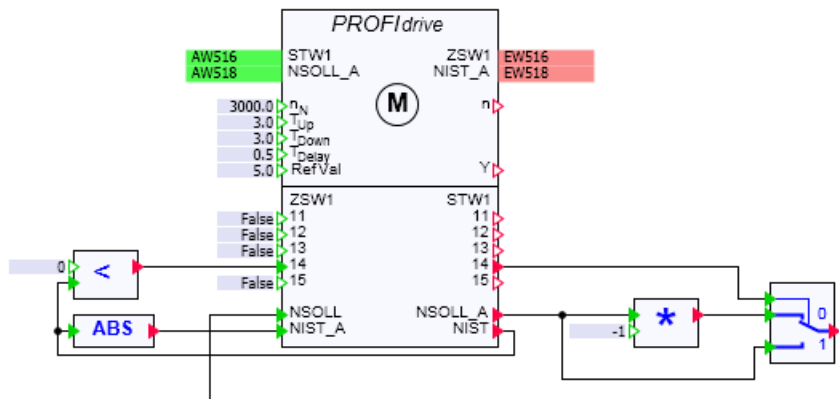
**Function**

The *Universal* component type can only be meaningfully used in combination with the *PROFdrive* component type. It enables drive-specific functions to be realized in addition to the basic PROFdrive functions.

Bits 11 to 15 of the control word (STW1.11-15), the speed setpoint value *NSOLL\_A* and the speed actual value *NIST(NIST\_PT1)* are placed on the outputs of the *Universal* component. By means of appropriate logical and arithmetical operations, bits 11 to 15 of the status word, the speed setpoint value *NSOLL (NSOLL\_RFG)*, and the speed actual value *NIST\_A* are set at the inputs of the component. The signal linkage resulting from the connection of the *Universal* component to the *PROFdrive* component is illustrated in the figure below.



The figure below illustrates an example of how the drive-specific functions for the Type 3 Micromaster can be implemented with the aid of the *Universal* component.



## Specific PROFIdrive devices

### Overview

For some PROFIdrive devices, the additional drive-specific functions are implemented in the form of drive-specific component types. The component types for these can be found in the PROFIdrives library

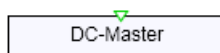
- *DCMaster*,
- *Masterdrive*,
- *Micromaster3*,
- *Micromaster4* and
- *Sinamics*.

These components are always linked to the *PROFIdrive* component. Like the *Universal* component type, these device-specific component types can therefore only be meaningfully used in combination with the *PROFIdrive* component type.

As can be seen in the following detailed descriptions of these component types, the component types *DCMaster*, *Masterdrive*, *Sinamics* and *Micromaster4* are functionally identical. However, to clearly depict in the simulation how the components are assigned to the drives used in the system, all of these component types are contained in the library.

### DCMaster – SIMOREG DC Master power converter

#### Symbol



#### Function

The specific additional functions for the SIMOREG DC Master power converter are implemented in the *DCMaster* component type. The implementation is coordinated with the processing stages contained in the function block *SIMO\_DC* from the function block library *DriveES-PCS7*.

The setpoint value of the ramp function generator *NSOLL\_RFG* is derived from the drive-specific bits 11 and 12 of the control word and the setpoint value *NSOLL\_A* as shown in the table below.

Table 7-11 DCMaster-specific evaluation of the control word

STW1.11 Enable positive direction of rotation	STW1.12 Enable negative direction of rotation	NSOLL_RFG
0	0	0
0	1	- NSOLL_A



STW1.11 Enable positive direction of rotation	STW1.12 Enable negative direction of rotation	NSOLL_RFG
1	0	NSOLL_A
1	1	NSOLL_A

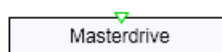
The speed actual value  $NSOLL_A$  and bits 12 and 14 of the status word are set according to the relationships listed in the table below.

Table 7-12 DCMaster-specific states

NIST_PT1	NIST_A	ZSW1.12 Line contactor requirement	ZSW1.14 Positive direction of rotation
NIST_PT1 > 0	NIST_PT1	1	1
NIST_PT1 = 0	0	0	1
NIST_PT1 < 0	NIST_PT1	1	0

## Masterdrive – SIMOVERT Masterdrive frequency converter

### Symbol



### Function

The specific additional functions for the SIMOVERT Masterdrive frequency converter are implemented in the *Masterdrive* component type. The implementation is coordinated with the processing stages contained in the function block SIMO\_MD from the function block library DriveES-PCS7.

The setpoint value of the ramp function generator  $NSOLL\_RFG$  is derived from the drive-specific bits 11 and 12 of the control word and the setpoint value  $NSOLL\_A$  as shown in the table below.

Table 7-13 Masterdrive-specific evaluation of the control word

STW1.11 Enable positive direction of rotation	STW1.12 Enable negative direction of rotation	NSOLL_RFG
0	0	0
0	1	- NSOLL_A
1	0	NSOLL_A
1	1	NSOLL_A

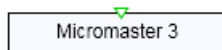
The speed actual value *NIST\_A* and bits 12 and 14 of the status word are set according to the relationships listed in the table below.

Table 7-14 Masterdrive-specific states

NIST_PT1	NIST_A	ZSW1.12 Line contactor requirement	ZSW1.14 Positive direction of rotation
$NIST\_PT1 > 0$	NIST_PT1	1	1
$NIST\_PT1 = 0$	0	0	1
$NIST\_PT1 < 0$	NIST_PT1	1	0

### Micromaster3 – MICROMASTER Type 3 frequency converter

#### Symbol



#### Function

The specific additional functions for the MICROMASTER Type 3 frequency converter are implemented in the *Micromaster3* component type. The implementation is coordinated with the processing stages contained in the function block SIMO\_MM3 from the function block library DriveES-PCS7.

The setpoint value of the ramp function generator *NSOLL\_RFG* is derived from bit 14 of the control word and the setpoint value *NSOLL\_A*.

Table 7-15 Micromaster3-specific evaluation of the control word

STW1.14 Clockwise rotation	NSOLL_RFG
0	- NSOLL_A
1	NSOLL_A

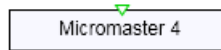
The speed actual value *NIST\_A* and bit 14 of the status word are set according to the relationships listed in the table below.

Table 7-16 Micromaster3-specific states

NIST_PT1	NIST_A	ZSW1.14 Clockwise rotation
$NIST\_PT1 \geq 0$	NIST_PT1	1
$NIST\_PT1 < 0$	NIST_PT1	0

## Micromaster4 – MICROMASTER Type 4 frequency converter

### Symbol



### Function

The specific additional functions for the MICROMASTER Type 4 frequency converter are implemented in the *Micromaster4* component type. The implementation is coordinated with the processing stages contained in the function block SIMO\_MM4 from the function block library DriveES-PCS7.

The setpoint value of the ramp function generator *NSOLL\_RFG* is derived from bit 11 of the control word and the setpoint value *NSOLL\_A*.

Table 7-17 Micromaster4-specific evaluation of the control word

STW1.11 Inversion of setpoint value	NSOLL_RFG
0	NSOLL_A
1	- NSOLL_A

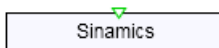
The speed actual value *NIST\_A* and bit 11 of the status word are set according to the relationships listed in the table below.

Table 7-18 Micromaster4-specific states

NIST_PT1	NIST_A	ZSW1.11 Positive direction of rotation
NIST_PT1 >= 0	NIST_PT1	1
NIST_PT1 < 0	NIST_PT1	0

## Sinamics – SINAMICS frequency converter

### Symbol



### Function

The specific additional functions for the SINAMICS frequency converter are implemented in the *Sinamics* component type. The implementation is coordinated with the processing stages contained in the function block SINA\_GS from the function block library DriveES-PCS7.

The setpoint value of the ramp function generator *NSOLL\_RFG* is derived from bit 11 of the control word and the setpoint value *NSOLL\_A*.

Table 7-19 Sinamics-specific evaluation of the control word

STW1.11 Inversion of setpoint value	NSOLL_RFG
0	NSOLL_A
1	- NSOLL_A

The speed actual value *NIST\_A* and bit 11 of the status word are set according to the relationships listed in the table below.

Table 7-20 Sinamics-specific states

NIST_PT1	NIST_A	ZSW1.11 Positive direction of rotation
NIST_PT1 >= 0	NIST_PT1	1
NIST_PT1 < 0	NIST_PT1	0

### Creation of device-specific components

Further device-specific components can be created with the SIMIT add-on module CTE.

A *PROFdrive* connection type must be used to connect the component to the *PROFdrive* header component. A connection of this type must be created on the device-specific component as an *Input*. The *PROFdrive* connection type is defined with five signals in the *forward* direction and four signals in the *backward* direction.

Table 7-21 PROFdrive connection type

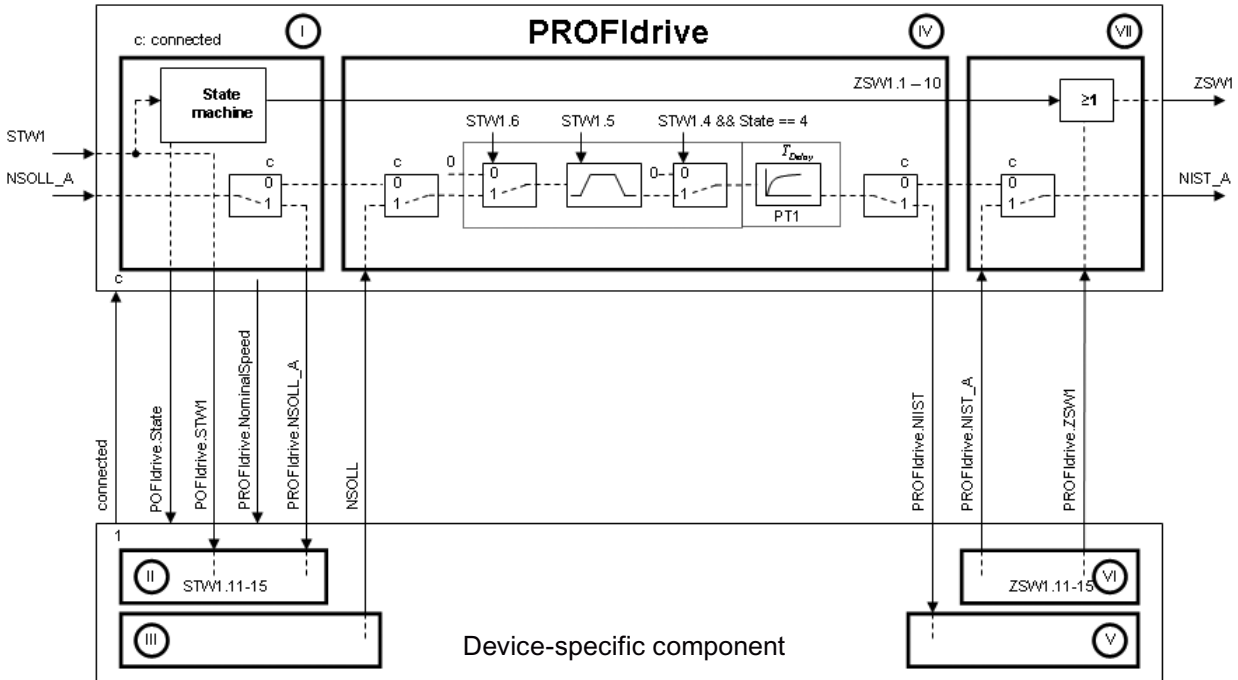
Signal (Forward)	Data type	Signal (Backward)	Data type
NSOLL_A	double	ZSW1	double
NIST	double	connected	logical
State	double	NSOLL	double
NominalSpeed	double	NIST_A	double
STW1	double		

The individual signals have the following meaning:

- *NSOLL\_A*: Low resolution speed setpoint value (one word)
- *NIST*: Speed actual value delayed by the delay element (*NIST\_PT1*)
- *State*: State of the PROFdrive state machine as a numerical value (1 to 6)
- *NominalSpeed*: Nominal speed; input  $n_N$  of the *PROFdrive* component
- *STW1*: Control word
- *ZSW1*: Status word (see table in section: The state machine (Page 380)); the drive-specific bits 11 to 15 of the device-specific component are combined with bits 0 to 10 in the header block

- *connected*: This logical value must be set to one in the device-specific component. In the *PROFdrive* header component this signal indicates that a device-specific component is connected.
- *NSOLL*: Speed setpoint value formed in the device-specific component
- *NIST\_A*: Speed actual value formed in the device-specific component

The use of the signals in the *PROFdrive* header component that is linked with a device-specific component is shown in the figure below:



We recommend that the functionalities shown in the device-specific components as II, III, IV and V are each implemented in separate blocks (BLOCK ... END\_BLOCK). When generating the executable simulation (code generation), the overall functionality of the header component and device-specific component can be introduced into the calculation sequence I to VII without disrupting the calculation cycle.

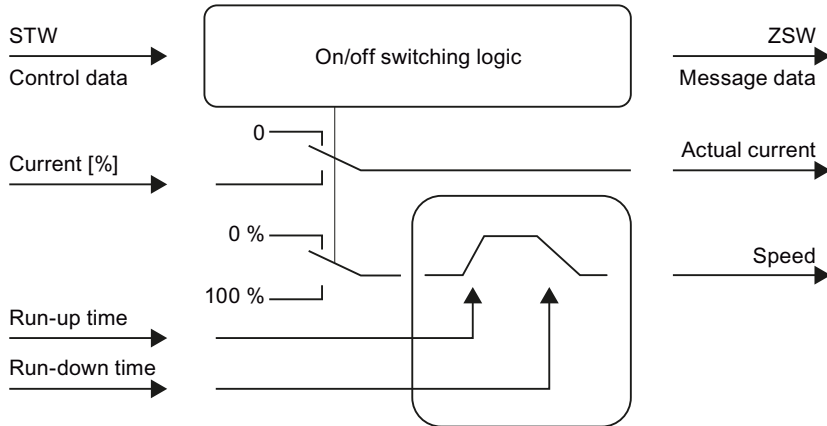
### 7.1.4.5 SIMOCODE pro motor control devices

SIMOCODE pro motor management and control devices are used to switch motors on and off and to monitor the resulting currents. As an option, SIMOCODE pro can also be used to record other measured values and to access comprehensive statistical evaluations. A SIMOCODE pro device is connected to the controller as an individual PROFIBUS DP slave.

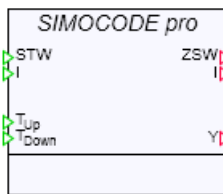
By configuring the device accordingly, a SIMOCODE pro can be employed for very different tasks. For example, it can operate as direct-on-line starter, reversing starter, star-delta starter with or without reversal of rotation, pole switching device with or without reversal of rotation, as well as positioner, solenoid valve drive, overload relay or power circuit breaker. The directory *SIMOCODEpro* in the Drives Library contains component types that emulate the various control functions of SIMOCODE pro. These component types form the SIMOCODEpro Library of SIMIT.

**Basic functions of SIMOCODE pro components**

Each component type of the SIMOCODEpro Library contains, as a basic function, a simple emulation of the motor together with the logic for switching the motor on and off.



The corresponding connectors of the *SIMOCODEpro* component types can be seen in the symbols depicted in the figure below.

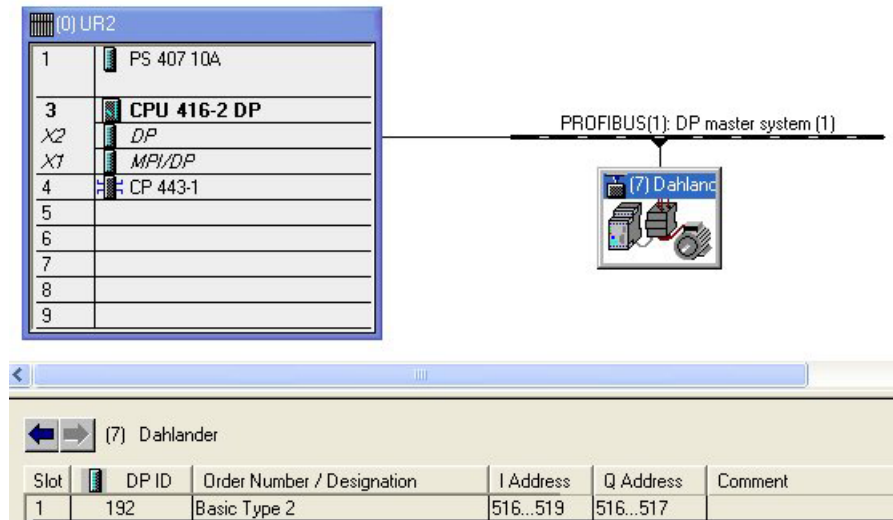


All SIMOCODE pro devices are accessed via control data and return their current status to the controller by means of message data. The content of the control and message data depends on which control function is implemented. Only the cyclical control and message data is processed by the *SIMOCODEpro* component types; acyclical data such as statistical data, for example, are not taken into account. The components can interface with the controller in the following ways:

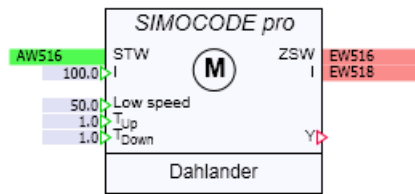
- The control data word (2 bytes) of the controller outputs is connected to the integer input *STW* of the component.
- The first data word of the message data in the controller inputs contains the binary feedback and is connected to the integer output *ZSW*. Analog message data, for example, the actual current value, is transferred to the controller in the second data word.

This configuration corresponds to SIMOCODE pro C or, in the case of SIMOCODE pro V, to basic type 2.

In the figure below, an example of a SIMATIC configuration for a Dahlander motor is shown.



The figure below shows the *SIMOCODEpro* component and the associated process data.



The actual current *I* is communicated to the controller as a percentage value of the current setting (rated motor current). The current value at input *I* is interconnected as the actual current to the output *I*, provided that the motor is switched on. The input has a default value of 100%, which means the actual current equals the current setting. The switching on and off of the motor is controlled by a ramp function.

The relationship between current and motor load is not taken into account. This can however easily be added in the simulation, for example, by not setting the current as a constant value but by using suitable functions to make it dependent on the motor speed or on the process instead. You can find additional information on this in the section: Individual adaptations (Page 395).

The speed of the motor is available as a percentage value at output *Y* of a *SIMOCODEpro* component. The run-up and run-down times of the motor are set on the two analog inputs  $T_{Up}$  and  $T_{Down}$ .  $T_{Up}$  is the time in seconds it takes the motor to run up from standstill to the nominal speed;  $T_{Down}$  is the time in seconds it takes for the motor to run down to a standstill from the nominal speed. Both times have a default value of one second. If one of the two input values is negative during the simulation, the message "x: run-up or run-down time invalid value" (message category *ERROR*) is generated.

**Note**

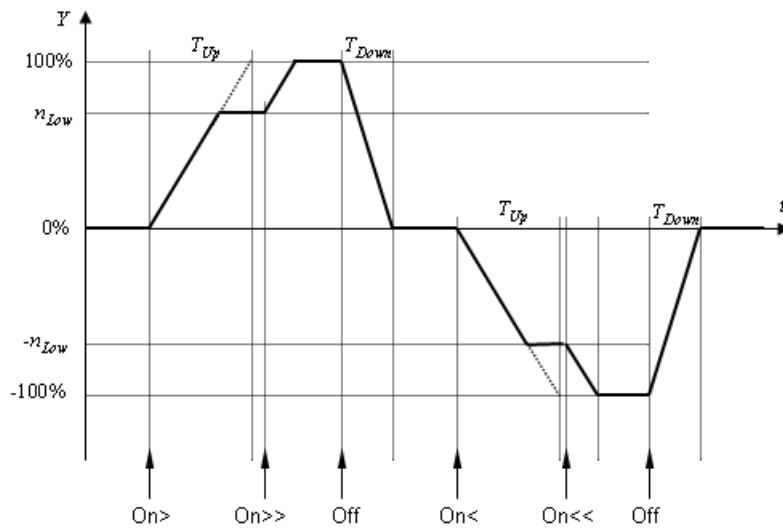
In the case of positioners and solenoid valves, the run-up and run-down times mean the opening and closing time  $T_{Up}$  and  $T_{Down}$  of the valve.

The speed of the motor is not specified by the controller nor detected by the SIMOCODE pro devices. The speed value is therefore not communicated to the controller. Its purpose is to provide another source of motor speed input for the simulation. Consequently, the run-up and run-down times are only of minor importance.

Switchover pauses and interlock times are ignored by the *SIMOCODEpro* component types. Corresponding feedback for the controller is not generated.

### The ramp function

The motor speed is formed using a ramp function. The most general form of this ramp for a drive with two speeds in two directions of rotation is illustrated in the figure below.



The increase and decrease times of the ramp correspond to the run-up and run-down times  $T_{Up}$  and  $T_{Down}$  of the motors. Depending on how SIMOCODE pro is configured, the motor speed  $Y$  is generated as a percentage value in the range -100% to 100%.

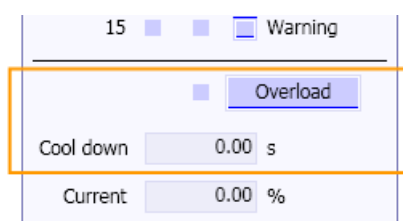
### Overload behavior

Overload can be set using the *Overload* switch in the operating window of a component. The motor is switched off in the event of an overload. A thermal switch-on inhibitor determines when the motor can be switched on again (cooling down time). The cooling down time begins when the overload no longer exists. The cooling down time is set in the *Cool\_Down\_Period* parameter and has a default value of 300 seconds, as shown in the figure below.

ReversingDahlander#1		
General	Parameter	Value
Input	Cool_Down_Period	300.0
Output		
Parameter		
State		

As shown in the figure below, the remaining cooling down time is displayed in the *Cool down* field of the operating window:





During this time the drive is disabled, which means it cannot be restarted.

The thermal switch-on inhibitor is reset by setting the emergency start (*EM-Start*, *STW.12*) bit. The drive can be restarted immediately.

The advance warning for overload ( $I > 115\%$ , *ZSW.11*) is set as soon as the input current exceeds 115%.

### Standard assignments in the control and message data

The functions shown in the table below are used as standard for bits 11 to 15 of the control word (*STW.11* to *STW.14*) in all *SIMOCODEpro* component types. The Emergency Start function does not apply to solenoid valves.

Table 7-22 Standard assignments in the control word

Name	Control data	
Test1	Test function: reset after 5 seconds	<i>STW.11</i>
EM-Start	Emergency start	<i>STW.12</i>
Remote	Operating mode switch S1	<i>STW.13</i>
Reset	Reset device	<i>STW.14</i>

The test function *Test1* resets the *SIMOCODE pro* device five seconds after setting the signal. The motor is switched off.

Setting *EM-Start* resets the thermal switch-on inhibitor if it has been triggered by an overload. The motor can then be switched on again immediately.

The *Remote* command displays the controller default for operating mode switch S1. This command has no functional effect on the simulation.

The *SIMOCODE pro* device is reset using the *Reset* command. This switches the motor off.

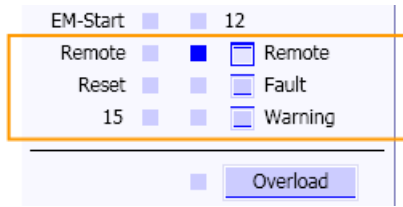
The table below provides an overview of the standard message data for all *SIMOCODEpro* component types. The overload advance warning does not apply to solenoid valves.

Table 7-23 Standard assignments in the status word

Name	Message data	
$I > 115\%$	Overload advance warning	<i>ZSW.11</i>
Remote	Remote operating mode	<i>ZSW.13</i>
Fault	General fault	<i>ZSW.14</i>
Warning	General warning	<i>ZSW.15</i>

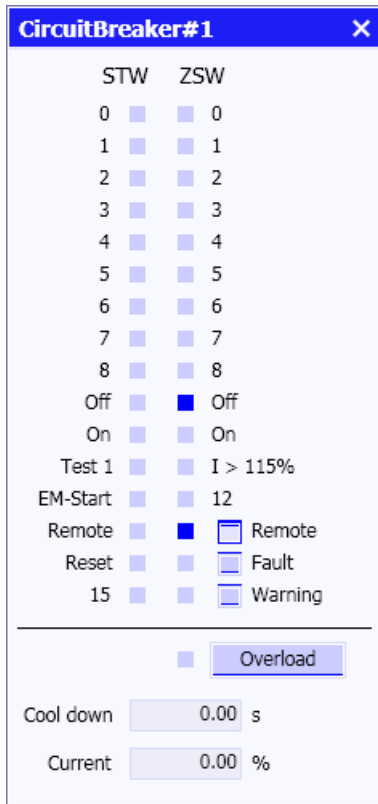
If the current value rises above 115 percent, bit *ZSW.11* is set as advance warning of an overload.

The *Remote*, *Fault* and *Warning* signals can be set in the operating window of the component. The *Remote* signal has a default value of one, which means the switch is closed.



### Operating window of the SIMOCODEpro components

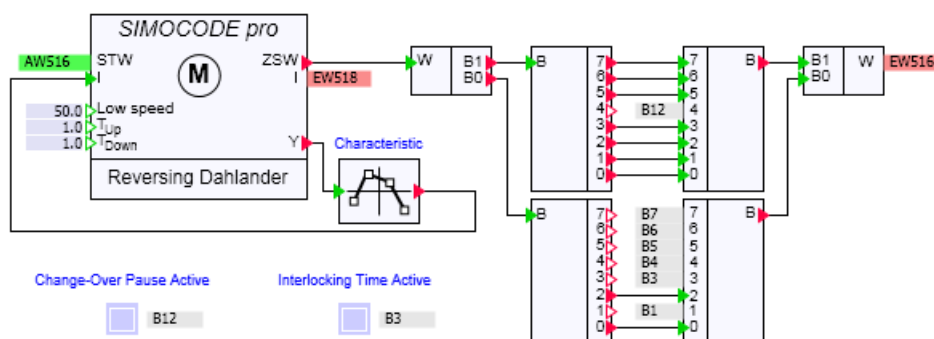
All *SIMOCODEpro* components have an operating window in which the signals of the control word and status word are displayed. The names of the control word signals used by each component are shown in the operating window. The same applies to the message data signals that are manipulated by the components. Signals indicated by numbers have no function in the component and are only displayed in the operating window. The figure below shows the operating window for the component of type *ReversingDahlander*. The operating windows for components of other types are designed accordingly.



## Individual adaptations

Depending on how parameters have been assigned to a SIMOCODE pro device, signals in the control and message data can have meanings that are not present in the SIMOCODEpro Library components. However, these signals can be included in the simulation by a simple connection to other components from the SIMIT Basic Library.

In the example of a *ReversingDahlander* component illustrated in the figure below, the interlocking time and the change-over pause can be set manually. Neither signal is set in the component. Here, the status word on the output of the component has been divided into its individual signals by the conversion components (*Word2Byte*, *Byte2Bit*). The signals set in the component are converted into a word again by the signals linked via the global connectors, and communicated to the *IW516* signal input of the automation system.



An example is also shown of how the current on the input of the *ReversingDahlander* component can be mapped from the speed of the motor on output *Y* of the *ReversingDahlander* component using a suitable trend function (*Characteristic*) and adjusted as required.

## Specific SIMOCODE pro devices

The SIMOCODEpro Library contains ten different *SIMOCODEpro* component types. These component types emulate the various control functions of a SIMOCODE pro. Each control function is emulated by a component type. The simulation function of the component types is coordinated with the corresponding address variants (function blocks) in PCS7.

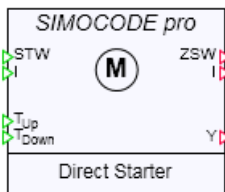
Table 7-24 Supported control functions of the SIMOCODE pro

SIMIT component types	Control function	PCS7- FB
DirectStarter	Direct starter	SMC_DIR
ReversingStarter	Reversing starter	SMC_REV
StarDeltaStarter	Star-delta starter	SMC_STAR
ReversingStarDelta	Reversing star-delta starter	SMC_REVS
Dahlander	Dahlander / pole-changing switch	SMC_D AHL
ReversingDahlander	Dahlander / pole-changing switch with reversal of direction of rotation	SMC_REVD
Valve	Solenoid valve	SMC_VAL
Positioner	Slider	SMC_POS

SIMIT component types	Control function	PCS7- FB
OverloadRelay	Overload	SMC_OVL
CircuitBreaker	Circuit breaker	SMC_CB

**DirectStarter – Direct starter**

**Symbol**



**Function**

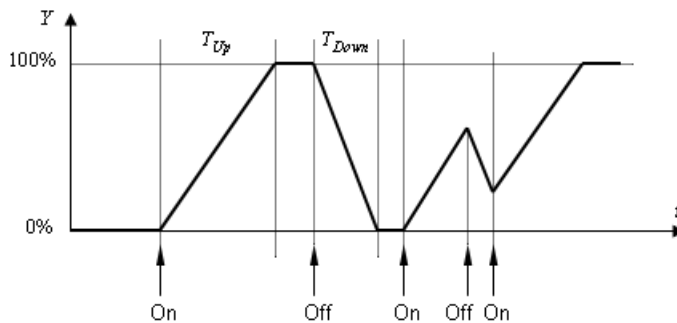
The component type *DirectStarter* simulates drives that have a single direction of rotation and can be switched on and off directly. The table below provides an overview of the relevant control data *STW* and message data *STW* for this application.

Table 7-25 Control and message data of the DirectStarter component

Name	Control data		Message data	
Off	Switch off drive	STW.9	Drive switched off	STW.9
On	Switch on drive	STW.10	Drive switched on	STW.10

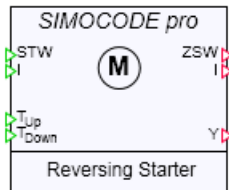
The commands "Switch off drive" (Off, STW.9) and "Reset" (RESET, STW.14) have priority over the command "Switch on drive" (On, STW.10).

The actual current value at output *I* is set to the value at the current input *I* as soon as the drive is switched on. If the drive is switched off, it is set to zero. The speed value at the output *Y* is formed as a percentage value with the help of the ramp function:  $0 \leq Y \leq 100$ .



## ReversingStarter – Reversing starter

### Symbol



### Function

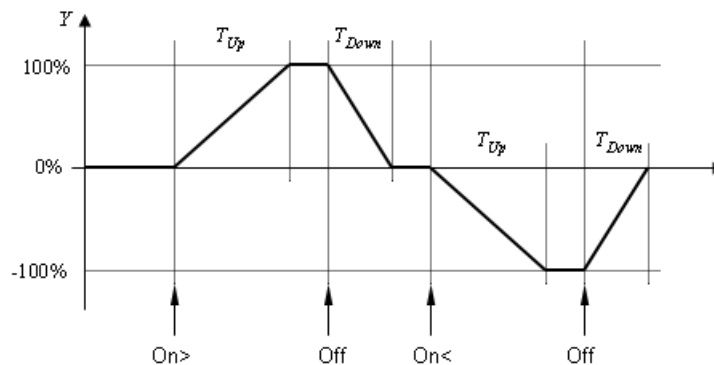
The *ReversingStarter* component type simulates drives with two directions of rotation, which can be switched on and off directly in both directions. The table below provides an overview of the relevant control data *STW* and message data *ZSW* for this application.

Table 7-26 Control and message data of the DirectStarter component

Name	Control data	Message data
On<	Switch on counterclockwise rotation STW.8	Counterclockwise rotation switched on ZSW.8
Off	Switch off drive STW.9	Drive switched off ZSW.9
On>	Switch on clockwise rotation STW.10	Clockwise rotation switched on ZSW.10

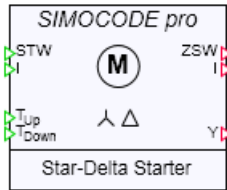
The commands "Switch off drive" (Off, STW.9) and "Reset" (RESET, STW.14) have priority over the command "Switch on counterclockwise rotation" or "Switch on clockwise rotation" (On<, STW.8 or On>, STW.10). Simultaneous switch-on commands for both directions of rotation do not alter the status of the drive; they are ignored.

The actual current value at output *I* is set to the value at the current input *I* as soon as the drive is switched on in one of the two directions of rotation. If the drive is switched off, the actual current value is set to zero. The speed value at the output *y* is formed as a percentage value with the help of the ramp function:  $-100 \leq Y \leq 100$ .



## StarDeltaStarter – Star-delta starter

### Symbol



### Function

The *StarDeltaStarter* component type simulates drives with star-delta switchover. The table below provides an overview of the relevant control data *STW* and message data *STW* for this application.

Table 7-27 Control and message data of the StarDeltaStarter component

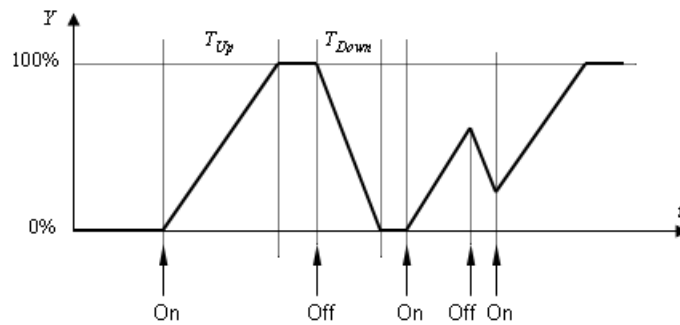
Name	Control data		Message data	
	Off	Switch off drive	STW.9	Drive switched off
On	Switch on drive	STW.10	Delta mode switched on	STW.10

The commands "Switch off drive" (Off, STW.9) and "Reset" (RESET, STW.14) have priority over the command "Switch on drive" (On, STW.10).

The maximum time for star mode is set in the parameter *Max\_Star\_Time*. The default time is 15 seconds. The switchover to delta mode (STW.10) is performed when the maximum star mode time has been reached or if a value of less than 90% is present on the current input.

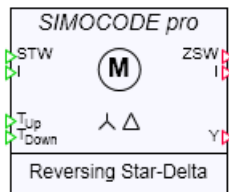
StarDeltaStarter#1		
General	Name	Value
Input	Cool_Down_Period	300.0
Output	Max_Star_Time	15.0
Parameter		
State		

The actual current value at output *I* is set to the value at the current input *I* as soon as the drive is switched on. If the drive is switched off, the actual current value is set to zero. The speed value at the output *Y* is formed as a percentage value with the help of the ramp function:  $0 \leq Y \leq 100$ .



## ReversingStarDelta – Star-delta starter with reversal of direction of rotation

### Symbol



### Function

The *ReversingStarDelta* component type simulates drives with star-delta switchover in both directions of rotation. The table below provides an overview of the relevant control data *STW* and message data *ZSW* for this application.

Table 7-28 Control and message data for the ReversingStarDelta component

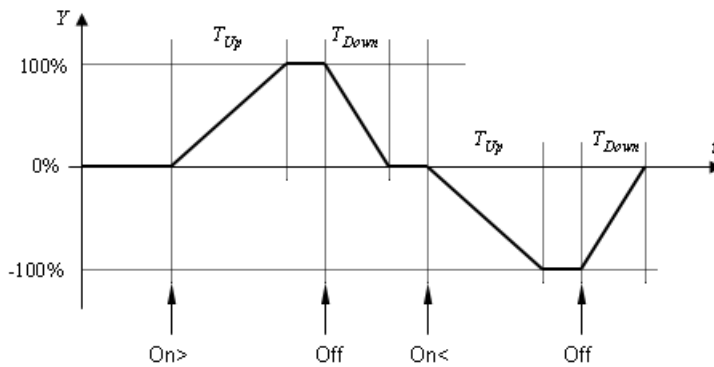
Name	Control data	Message data
On<	Switch on counterclockwise rotation STW.8	Counterclockwise delta mode switched on ZSW.8
Off	Switch off drive STW.9	Drive switched off ZSW.9
On>	Switch on clockwise rotation STW.10	Clockwise delta mode switched on ZSW.10

The commands "Switch off drive" (Off, STW.9) and "Reset" (RESET, STW.14) have priority over the command "Switch on counterclockwise rotation" or "Switch on clockwise rotation" (On<, STW.8 or On>, STW.10). Simultaneous switch-on commands for both directions of rotation do not alter the status of the drive; they are ignored.

The maximum time for star mode is set in the parameter *Max\_Star\_Time*. The default time is 15 seconds. The direction-dependent switchover to delta mode (ZSW.10 or ZSW.8) is performed when the maximum star mode time has been reached or if a value of less than 90% is present at the current input.

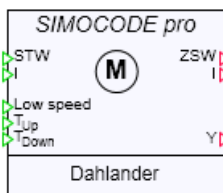
ReversingStarDelta#1		
General	Name	Value
Input	Cool_Down_Period	300.0
Output	Max_Star_Time	15.0
Parameter		
State		

The actual current value at output / is set to the value at the current input / as soon as the drive is switched on in one of the two directions of rotation. If the drive is switched off, the actual current value is set to zero. The speed value at the output y is formed as a percentage value with the help of the ramp function:  $-100 \leq Y \leq 100$ .



Dahlander – Dahlander starter or pole-changing switch

Symbol





## Function

The *Dahlander* component type simulates drives with a single direction of rotation and two speeds: full speed and low speed. The table below provides an overview of the relevant control data *STW* and message data *ZSW* for this application.

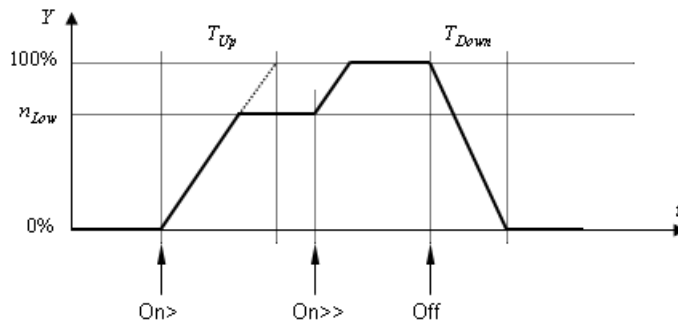
Table 7-29 Control and message data of the Dahlander component

Name	Control data		Message data	
On>>	Switch drive to full speed	STW.8	Drive switched to full speed	ZSW.8
Off	Switch off drive	STW.9	Drive switched off	ZSW.9
On>	Switch drive to low speed	STW.10	Drive switched to low speed	ZSW.10

The commands "Switch off drive" (Off, STW.9) and "Reset" (RESET, STW.14) have priority over the commands "Switch to full speed" (On>>, STW.8) or "Switch to low speed" (On>, STW.10). Simultaneous switch-on commands for both speeds do not alter the status of the drive; they are ignored.

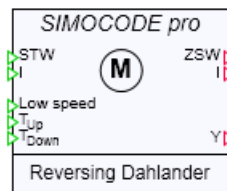
The actual current value at output *I* is set to the value at the current input *I* as soon as the drive is switched to one of the two speeds. If the drive is switched off, the actual current value is set to zero.

The low speed  $n_{Low}$  is set as a percentage value on the input *Low speed*. The default low speed value is 50%. The speed value at the output *Y* is formed as a percentage value with the help of the ramp function:  $0 \leq Y \leq 100$ .



## ReversingDahlander – Dahlander starter or pole-changing switch with reversal of direction of rotation

### Symbol



Function

The *ReversingDahlander* component type simulates drives with two speeds of rotation in two directions. The table below provides an overview of the relevant control data *STW* and message data *ZSW* for this application.

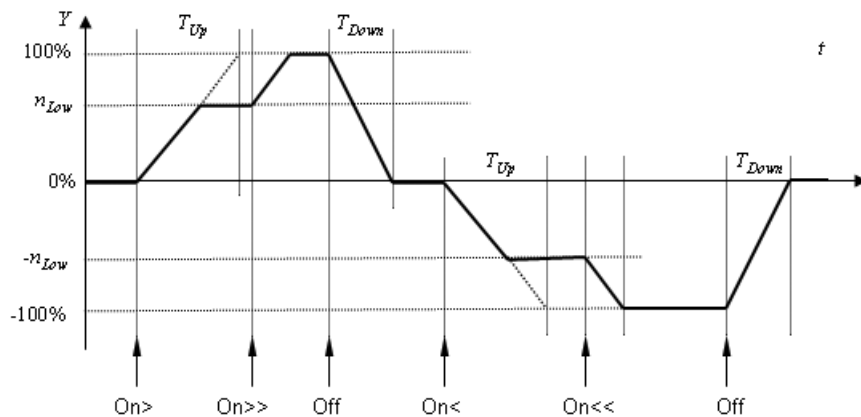
Table 7-30 Control and message data for the ReversingDahlander component

Name	Control data	Message data
On<<	Switch drive to full speed counterclockwise	STW.0
On<	Switch drive to low speed counterclockwise	STW.2
On>>	Switch drive to full speed clockwise	STW.8
On>	Switch drive to low speed clockwise	STW.10
Off	Switch off drive	STW.9

The commands "Switch off drive" (Off, STW.9) and "Reset" (RESET, STW.14) have priority over the switch-on commands "Full speed counterclockwise" (On<<, STW.0), "Low speed counterclockwise" (On<, STW.2), "Full speed clockwise" (On>>, STW. 8) and "Low speed clockwise" (On >, STW.10). Several simultaneous switch-on commands do not alter the status of the drive; they are ignored.

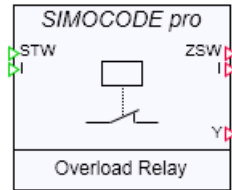
The actual current value at output *I* is set to the value at the current input *I* as soon as the drive is switched on. If the drive is switched off, the actual current value is set to zero.

The low speed  $n_{Low}$  is set as a percentage value on the input *Low speed*. The default low speed value is 50%. The speed value at the output *Y* is formed as a percentage value with the help of the ramp function:  $-100 \leq Y \leq 100$ .



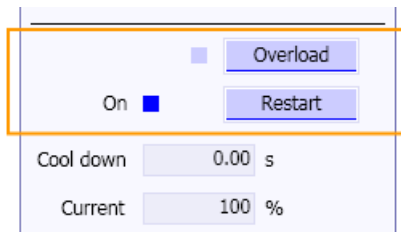
## OverloadRelay – Overload relay

### Symbol



### Function

The *OverloadRelay* component type simulates drives with overload monitoring. The *Overload* command triggers an overload, which means the drive is switched off. The *Restart* command switches the drive on. The commands can only be set in the operating window of the component.



The drive can be initialized as switched on or switched off using the *Initial\_Value* parameter: switched on with the value *Closed*, switched off with the value *Open*. The default setting for *Initial\_Value* is *Closed*.

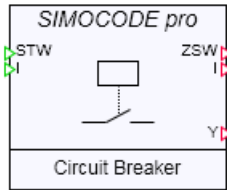
OverloadRelay#1		
General	Parameter	Value
Input	Cool_Down_Period	300.0
Output	Initial_Value	Closed
Parameter		Closed
State		Open

The actual current value at output /is set to the value at the current input /as soon as the drive is switched on. If the drive is switched off, the actual current value is set to zero.

The speed value on the output *Y* is set to one hundred if the drive is on and to zero if the drive is off.

## CircuitBreaker – Circuit breaker

### Symbol



### Function

The *CircuitBreaker* component type simulates drives with switching characteristics. The table below provides an overview of the relevant control data *STW* and message data *ZSW* for this application.

Table 7-31 Control and message data of the CircuitBreaker component

Name	Control data		Message data	
Off	Open switch	STW.9	Switch opening / open	ZSW.9
On	Close switch	STW.10	Switch closing / closed	ZSW.10

The commands "Open switch" (Off, STW.9) and "Reset" (RESET, STW.14) have priority over the closing command (On, STW.10).

The switch can be initialized as closed or open using the *Initial\_Value* parameter: closed with the value *Closed*, opened with the value *Open*. The default setting for *Initial\_Value* is *Closed*.

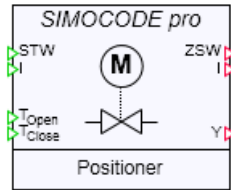
CircuitBreaker#1		
General	Parameter	Value
Input	Cool_Down_Period	300.0
Output	Initial_Value	Closed
Parameter		Closed
State		Open

The actual current value at output / is set to the value at the current input / as soon as the switch closes. If the switch is opened, the actual current value is set to zero.

The speed value on the output Y is set to one hundred if the drive is on and to zero if the drive is off.

## Positioner – Slide valve/Positioner

### Symbol



### Function

The *Positioner* component type simulates positioning drives for slide valves, adjusting valves, etc. The table below provides an overview of the relevant control data *STW* and message data *ZSW* for this application.

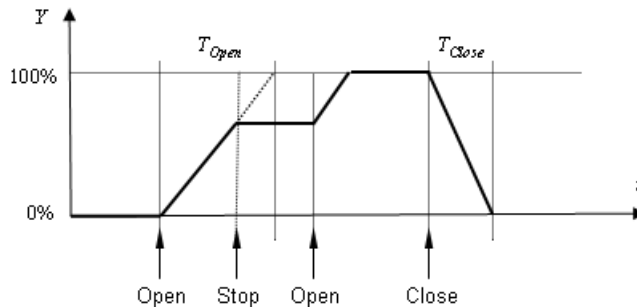
The commands "Switch off drive" (Off, *STW*.9) and "Reset" (RESET, *STW*.14) have priority over the opening and closing commands (Open, *STW*.10 and Close, *STW*.8). Simultaneously set opening and closing commands do not alter the status of the drive; they are ignored.

Table 7-32 Control and message data of the Positioner component

Name	Control data		Message data	
Close	Close slide valve	STW.8	Slide valve closed	ZSW.8
			Slide valve closes	ZSW.2
Stop	Stop slide valve	STW.9	Slide valve stops	ZSW.9
Open	Open slide valve	STW.10	Slide valve open	ZSW.10
			Slide valve opens	ZSW.0

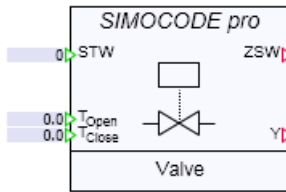
The actual current value at the output *I* is set to the value at the current input *I* as long as the slide valve is opening (*ZSW*.0) or closing (*ZSW*.2). The actual current value is set to zero if the slide valve is stationary and when it is closed or open.

The position value at the output *Y* is formed as a percentage value with the help of a ramp function:  $0 \leq Y \leq 100$ . The value zero represents a closed slide valve, the value one hundred an opened slide valve. The default opening and closing times of the slide valves are both set to one second.



### Valve – Solenoid valve

#### Symbol



#### Function

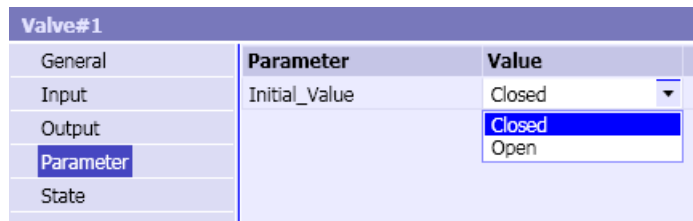
The *Valve* component type simulates drives for solenoid valves. The table below provides an overview of the relevant control data *STW* and message data *STW* for this application.

Table 7-33 Control and message data of the Valve component

Name	Control data		Message data	
Close	Close valve	STW.9	Valve closing / closed	STW.9
Open	Open valve	STW.10	Valve opening / opened	STW.10

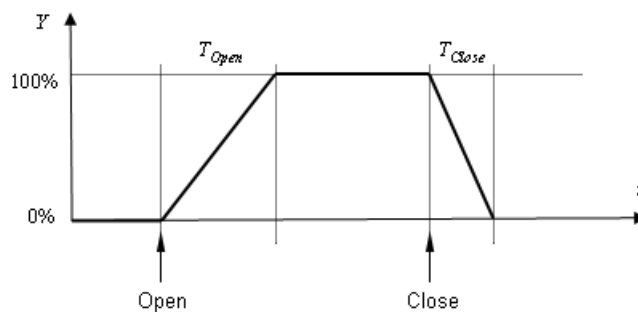
The commands "Close valve" (Close, STW.9) and "Reset" (RESET, STW.14) have priority over the opening command (Open, STW.10 and Close, STW.8).

The valve can be initialized as closed or open using the *Initial\_Value* parameter: closed with the value *Closed*, opened with the value *Open*. The default setting for *Initial\_Value* is *Closed*.



The setting value at the output *Y* is formed as a percentage value with the help of a ramp function. The value zero represents a closed valve, the value one hundred an opened valve.

$$0 \leq Y \leq 100$$



The default opening and closing times of the solenoid valve are zero, which means the opening and closing of the valve is done with a high gradient.

The *Valve* component does not have a current monitoring function and therefore no overload behavior, neither does it have a current input nor a current output.

## 7.1.5 Sensor components

### 7.1.5.1 SIWAREXU components

The SIWAREX U weighing system is used, for example, to measure fill levels in silos and bunkers, to monitor crane loads and for overload protection of industrial lifts. In all these applications, weights are detected with sensors such as load cells or force transducers and transferred to the controller as measured values. Pressure-sensitive sensors deliver a voltage proportional to the weight, which is converted by an analog/digital converter into a numerical value, processed and sent to the controller. The voltage signal is converted and prepared with the help of the SIWAREX U module. SIWAREX U modules can be connected to the PROFIBUS DP via ET 200M or used as a module of the SIMATIC S7-300.

The aim of the simulation with *SIWAREXU* components is to send measured values – which in the real system are transferred to the controller with the SIWAREX U weighing system – to the controller as simulated values. The *SIWAREX* directory of the basic library includes two component types for simulating the SIWAREX U weighing module:

- *SIWAREXU1*
- *SIWAREXU2*

These two component types simulate basic functions of the single-channel and two-channel versions of the SIWAREX U weighing module. The function is the same for both component types *SIWAREXU1* and *SIWAREXU2*. Type *SIWAREXU2* has one additional measuring channel than type *SIWAREXU1*. The detailed descriptions below for one channel of the type *SIWAREXU1* therefore apply to both channels of the type *SIWAREXU2* component type.

### Symbol

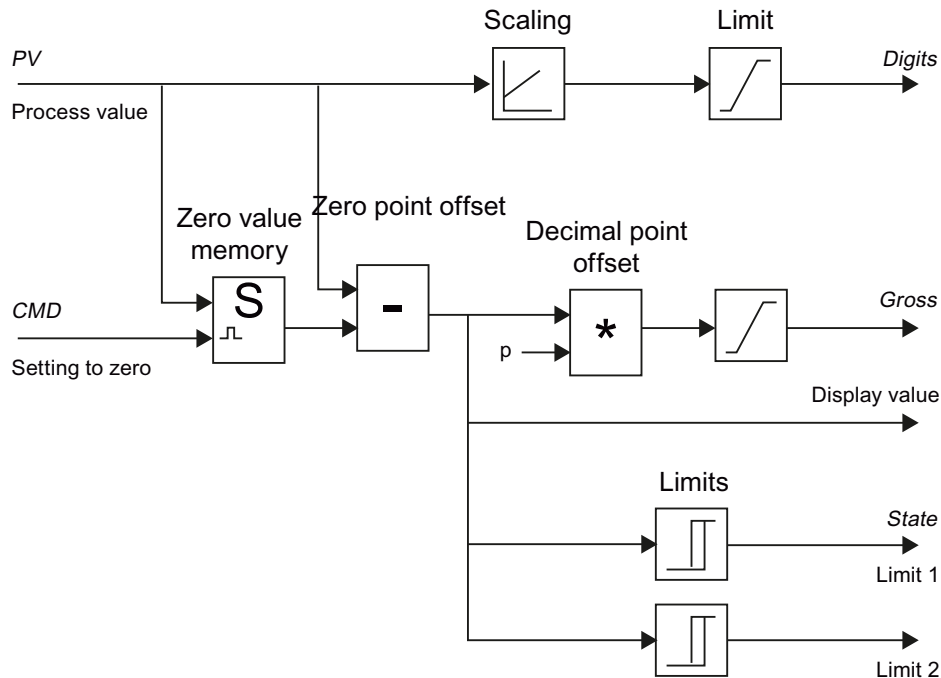


### Function

The SIWAREX U components simulate the following functions of the weighing system:

- Linear characteristic of the weighing system (adjustment diagonal)
- Zero offset,
- Decimal point shift
- Two configurable limits.

A function chart for a measuring channel of the component type is shown in the figure below.



In the SIWAREX U weighing system, the weight to be measured is first converted by a pressure sensor into an electrical voltage and then by an analog/digital converter into a numerical value. In the simulation the numerical value of a weight is already available as the result of a model calculation. The electrical signal transfer between the weighing sensor and the SIWAREX U module is therefore irrelevant for the simulation: the calculated weight value is applied directly to input *PV* of a *SIWAREXU* component as a physical measured variable.

The low pass filtering and sliding averaging, which take place in the SIWAREX U module to suppress disturbances of the electrical voltage signal, are not required in the simulation. Therefore neither function is included in the component types.

The weight value is available in *Digits* as a scaled numerical value limited to the range of zero to 65,535. Scaling is carried out with the help of the linear characteristic of the weighing system (adjustment diagonals). Furthermore, after any decimal point shift and zero offset, the weight value *Gross* is an integer value limited to the range of -32,768 to +32,767.

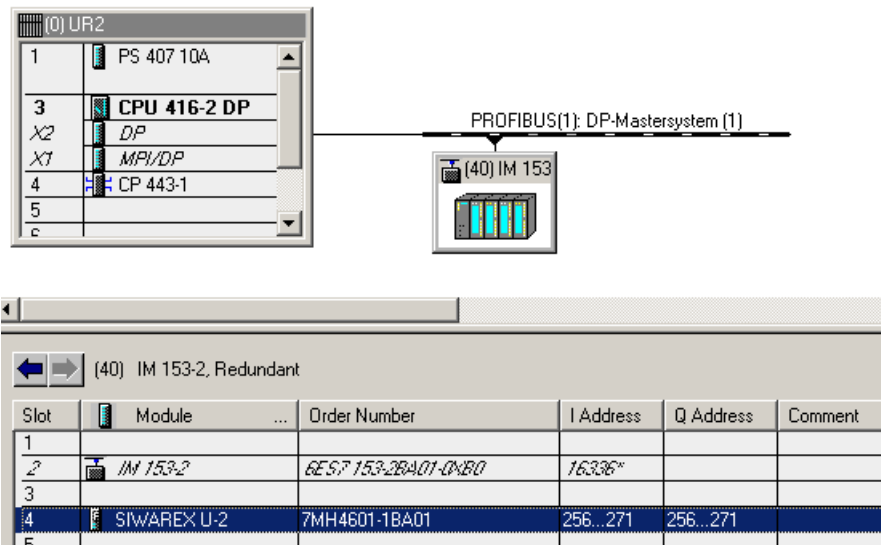
### Linking SIWAREXU components to the coupling

Both component types *SIWAREXU1* and *SIWAREXU2* are designed for communication type SFC/SFB/FB for SIMATIC S7/PCS7. All communication between the automation system and the *SIWAREXU* components takes place exclusively via data records. Therefore, the *SIWAREXU* components have no inputs and outputs that can be connected to signals in the coupling. Instead they have to be linked to data records

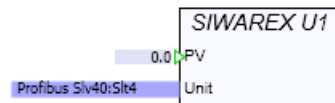
The communication via data records is a property of the SIMIT PROFIBUS DP coupling. If you import a hardware configuration into the PROFIBUS DP coupling that contains SIWAREX U modules, the data records that are relevant to communication with the controller are automatically created in the coupling for each SIWAREX U module. These data records are linked to *SIWAREXU* components using the *Unit* connector.



The figure below shows the SIMATIC configuration for a SIWAREX U module as an example: The slave with Profibus address 40 contains a SIWAREX U module in slot 4.



As shown in the figure below, the assigned *SIWAREXU* component has a *Unit* connector at its *Unit* input.



In the property view of the Unit connector, the component can be easily linked with the data records in the coupling by entering the name of the *Coupling*, and by entering the slave address and the module slot number in *Addressing* in the format *Slv#:Sl4#*. For the example shown, this is *Slv40:Sl4* (see figure below).

Profibus Slv40:Sl4		
General	Property	Value
	Gateway	Profibus
	Addressing	Slv40:Sl4
	Display Gateway Name	<input checked="" type="checkbox"/>

A second way to link a *SIWAREX* component to the data records is provided by the PROFIBUS coupling itself. To do it this way, open the PROFIBUS coupling and its property view. In the navigation area on the left, select the SIWAREX U module that is to be linked to the component by clicking on the module entry with the left mouse button.



SIWAREXU1#1		
General	Name	Value
Input	CH1_Adj_W	10000.0
Output	CH1_Adjust	65.0
Parameter	CH1_Dig_0	2427.0
State	CH1_Dig_1	63107.0
	CH1_OFF_L1	9990.0
	CH1_OFF_L2	1010.0
	CH1_ON_L1	10000.0
	CH1_ON_L2	1000.0
	CH1_Zero	2427.0

Because the scales simulated with the *SIWAREXU* component types are calibrated by setting permanently valid parameters, the simulated scales are always deemed to be calibrated. Therefore a scale adjustment process initiated by the controller is pointless in the simulation. Any such commands of the controller are ignored by the *SIWAREXU* components.

## Zero offset

When a *SIWAREXU* component receives the "Set to zero" command via the *CMD* control word, the weight value applied at that moment is saved as a new zero point. Consequently the zero value is now output as a gross value. All subsequent measurements then relate to this value, which means the difference between the current weight value and the zero value last saved is output as the gross value.

The zero value used on starting the simulation can be set as a digit value in the *CH1\_Zero* parameter of the component. A default digit value of 2427 is set, corresponding to a weight value of the unloaded scale assumed to be zero.

## Decimal point shift

A decimal point shift can be configured to increase the resolution of the gross value to be transferred to the controller. This means that the measured gross value is multiplied by a factor of 10, 100, 1000, 10,000 or 100,000.

The decimal point shift is defined by the parameter *CH1\_Adjust* according to the table below. The default value is 65, which means no decimal point shift.

Table 7-35 Adjustment table for decimal point shift

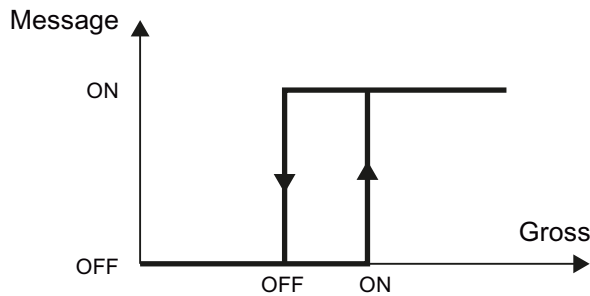
Number of decimal places	Factor	Parameter CH1_Adjust
0	1	65
1	10	69
2	100	73
3	1000	77
4	10,000	81
5	100,000	85

**Note**

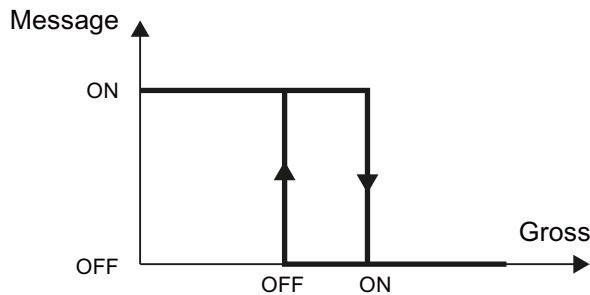
If you change the default setting of the *CH1\_Adjust* parameter in SIMIT, you must also change the C1ADJUST parameter in the instance data block of the controller.

**Limits**

SIWAREX U has two adjustable limits whose switch-on and switch-off points can be freely specified in weight units. The setting "Switch on value > Switch off value" leads to a notification when a weight value is overshoot:



The setting "Switch off value > switch on value" leads to a notification when a weight value is undershot:



In the special case where the limits for *OFF* and *ON* are equal, limit 1 signals an exceedance of this value while limit 2 signals that the set value has been undershot.

The limits are specified with the parameters *CH1\_ON\_L1* and *CH1\_OFF\_L1* for the first limit and with the parameters *CH1\_ON\_L2* and *CH1\_OFF\_L2* for the second limit. The default values are shown in the table below.

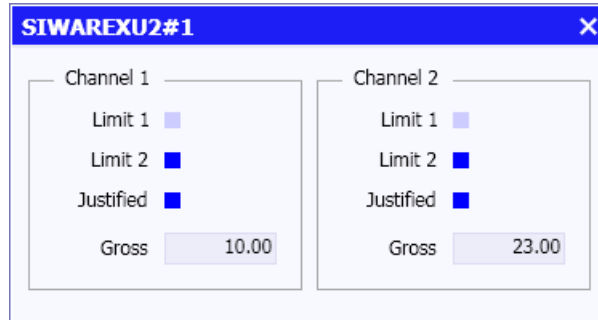
Table 7-36 Default limit settings

Parameter		Default
CH1_ON_L1	CH2_ON_L1	10000
CH1_OFF_L1	CH2_OFF_L1	9900
CH1_ON_L2	CH2_ON_L2	1000
CH1_OFF_L2	CH2_OFF_L2	1010

The status of the limits is available as part of the status information *State* of the component type.

### Operating window of the SIWAREXU components

The operating window of a *SIWAREX U* component shows which limits have been reached and which gross value is transferred to the controller as a weight value. The gross value (*Gross*) is shown without decimal point shift to make it easier to read. In the functional chart this is the *Display value*.

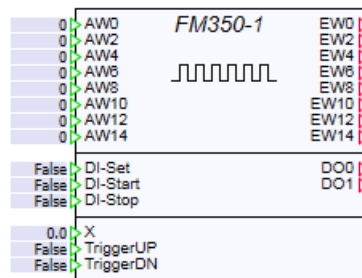


The *Justified* status indicator is always active. It indicates that the simulated scale is always calibrated.

### 7.1.5.2 Counter module FM350-1

#### Overview

#### Symbol



#### Function

The FM 350-1 function module is a fast counter module for use in the S7-300 automation system or as a module in an ET 200M station. Only distributed modules used on PROFIBUS/PROFINET can be simulated with SIMIT.

The module has the following operating modes:

- Counting modes
  - Continuous counting
  - Single counting
  - Periodic counting
- Measurement modes
  - Frequency measurement
  - RPM measurement
  - Continuous periodic measurement

All required functions of these operating modes are simulated in SIMIT.

The data of the real FM350-1 are cyclically transmitted via PROFIBUS DP. The acknowledgment principle was adopted in the SIMIT component.

You can find additional information in the "*S7-300; Counter Module FM 350-1*" documentation.

Some functions will not be discussed here in detail. We will merely point out the differences and describe the basic behavior of the simulated component.

### Behavior of the inputs and outputs in the simulated module

The SIMIT simulation system is based on cyclic calculation of mathematical equations. The fast counting function of the FM350 is simulated in SIMIT, simply stated, by setting the frequency of the counting pulses and adding or subtracting this to or from the current count depending on the current count in each processing cycle. The sign of the frequency makes it possible to also count backwards. The input X should be considered as "count pulses per second" in this case.

Each edge can also be evaluated for slow count pulses when the pulses occur in intervals faster than twice the cycle time. If the input X is zero and a positive edge is detected at one of the trigger inputs, the internal counter is incremented or decremented by one.

The "CNT", "TriggerUP" and "TriggerDN" connectors are used to change the internal count value when the internal gate is open.

The "CNT" input can be directly interpreted as a counting pulse in the counting operating modes and corresponds directly to the measured value in the measurement modes. Additionally, the count can be incremented or decremented by one with every positive edge via the two trigger inputs.

The behavior of the rest of the digital inputs and outputs is identical to the actual module.

---

**Note**

Hardware interrupts are not supported by the simulated module.

Data reading via DS0/1 (diagnostic data record) is not supported in the simulated module.

Isochronous mode is not supported in the simulated module. In other words, synchronization of the function module via PROFIBUS cannot be implemented due to the separation between physical replication of the bus by the simulation unit and simulation of the logic in this simulation component.

---

## The operating modes of the FM350-1

### Counting modes

In the counting modes, the measured value is calculated by adding up the pulses queued at input X. With very slow counts or individual edges, it is possible to increment the counter by one at the "TriggerUP" input with a positive edge or to decrement it at the "TriggerDN" input. This makes it possible to simulate very high frequencies of counting pulses, but also to enable the counting of individual pulses.

#### Count limits

The following applies for the 31-bit counting range:

- The upper count limit is set at +2 147 483 647 (231 - 1).
- The lower count limit is set at -2 147 483 648 (-231).

The following applies for the 32-bit counting range:

- The upper count limit is set at +4 294 967 295 (232 - 1).
- The lower count limit is set at 0 (zero).

#### Continuous counting

In this mode, the simulated FM 350-1 counts continuously starting at the count:

- When the counter reaches the upper count limit while counting forward and an additional count pulse occurs, it jumps to the lower count limit and resumes counting there without skipping a pulse.
- When the counter reaches the lower count limit while counting backwards and an additional count pulse occurs, it jumps to the upper count limit and resumes counting there without skipping a pulse.

#### One-time counting

By reaching the configured count limit in the main counting direction, the gate automatically closes internally and it can only be started again by a new gate start.

### Periodic counting

By reaching the configured count limit in the main counting direction, there is an internal jump to the load value and counting resumes there.

### Measurement modes

The measured value is applied directly at the "CNT" input with the measurement modes. The measured value is equal to the counter value in these modes. Again, the count value can be changed by 1 by positive edges at the "TriggerUP"/"TriggerDN" inputs.

The update time plays no role in the simulated counter module. The pending value is transmitted to the controller via the feedback interface. Limit monitoring and gate control are implemented.

### Frequency measurement

The measured frequency is equal to the value applied at the "CNT" input.

---

#### Note

The value of the real FM350-1 is transferred here in the unit  $10^{-3}$  Hz and must be specified accordingly in the simulation.

---

### Speed measurement

The measured speed is equal to the value applied at the "CNT" input.

---

#### Note

The value of the real FM350-1 is transferred here in the unit  $10^{-3}$  /min and must be specified accordingly in the simulation.

---

### Period duration measurement

The measured period duration is equal to the value applied at the "CNT" input.

---

#### Note

The value of the real FM350-1 is transferred here in the unit  $1\mu\text{s}$  or  $1/16\mu\text{s}$  and must be specified accordingly in the simulation.

---

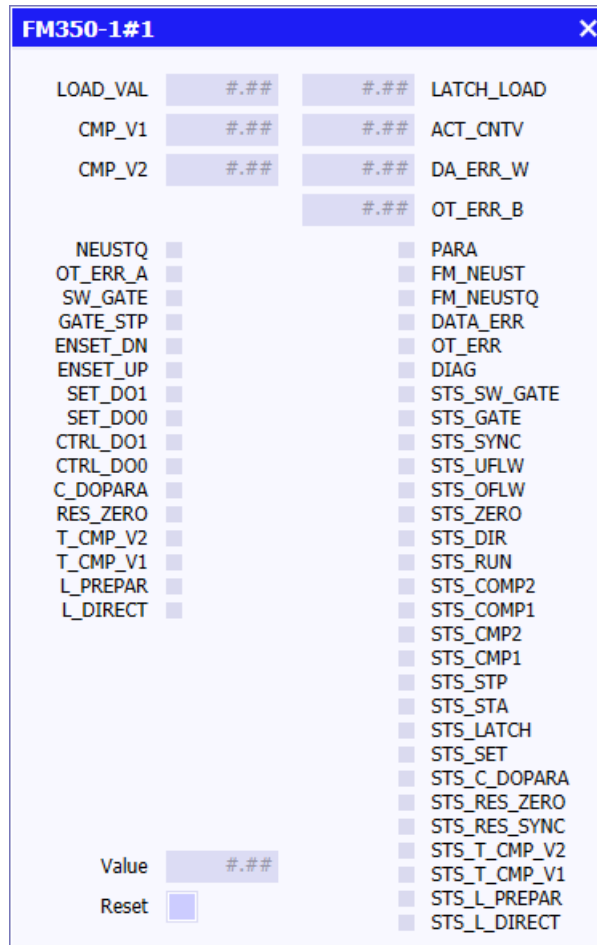
You can find additional information on counting mode and measurement mode in the "*S7-300; Counter Module FM 350-1*" documentation.

### Operating window of the FM 350-1 component

In the operating window, the signals of the control interface are shown on the left and signals from the feedback interface are displayed on the right. All signals are always displayed, regardless of the selected operating mode.



The arrangement of the signals corresponds to the actual module.

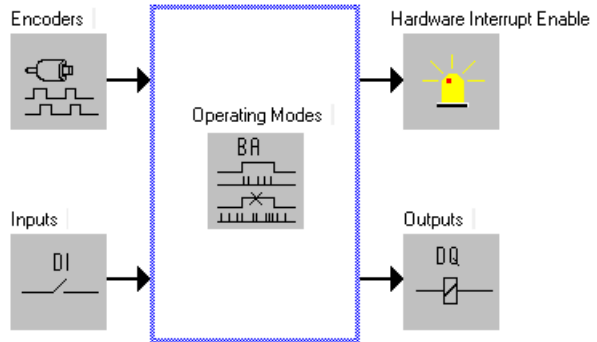


The "Reset" button can be used to manually reset the counter module. Then a restart calibration must be performed.

You can find additional information in the "*S7-300; Counter Module FM 350-1*" documentation.

### Properties of the FM 350-1 component

The parameters of the simulated module correspond to the settings in HW Config in STEP 7. The names of the parameters and selection options correspond to the English notations in the HW Config.



These parameters need to be adjusted for the respective FM350-1 instance. No automatic import from HW Config is performed.

### Properties of the FM 350-1

The figure below shows the property view with selected "Parameters" object in the default setting:

FM350-1#1		
General	Name	Value
Input	OperatingMode	Continuous counting
Output	GateControl	None
Parameter	CountRange	0 to +32 Bits
State	MainCountingDirection	None
	Latch	pos. edge
	SetCounter	Single
	GateFunction	Cancel

The individual parameters and your selection options from the drop-down lists are described in the following sections.

#### "OperatingMode" parameter

This parameter determines the operating mode. The same value must be set as in HW Config in STEP 7.

You can select the following functions from the drop-down list:

- Continuous counting
- Single counting
- Periodic counting
- Frequency measurement
- RPM measurement
- Continuous periodic measurement

**"GateControl" parameter**

This parameter determines the gate control of the counter module. Gate control is used to start and stop the counting operations.

You can select the following functions from the drop-down list:

- "None": Gate is always open
- "SW gate": Gate state depends on the "SW\_GATE" signal of the control interface
- "HW gate (level controlled)": Gate state depends on the "DI\_Start" input (level controlled)
- "HW gate (edge controlled)": A positive edge at the "DI\_Start" input opens the gate; a positive edge at the "DI\_Stop" input closes it
- "Latch": Requirement for saving counts
- "Latch/Retrigger": Requirement for saving counts followed by setting to the load value and counting resumes starting at the load value

**"CountRange" parameter**

This parameter sets the count range. If the counter module is operating in measurement mode, this setting has no effect.

You can select the following count ranges from the drop-down list:

- "0 to +32 bits"
- "-31 to +31 bits"

This setting has an effect on the count limits and signs of the transferred values.

**"MainCountingDirection" parameter**

This parameter determines the main counting direction. The main counting direction is only relevant in the counting modes.

You can select the following functions from the drop-down list:

- "None": No counting direction is specified
- "Forward": Count up
- "Backward": Count down

**"Latch" parameter**

This parameter sets the requirement for saving the counts.

- "pos. edge": The count is stored with a positive edge at the "DI Start" input
- "neg. edge": The count is stored with a negative edge at the "DI Start" input
- "both edge": The count is stored with a positive or negative edge at the "DI Start" input

**"SetCounter" parameter**

This parameter sets the behavior for setting to the load value once or multiple times.

- "Single"
- "Multiple"

**"GateFunction" parameter**

This parameter sets the gate control to "Cancel" or "Interrupt".

- "Cancel"
- "Interrupt"

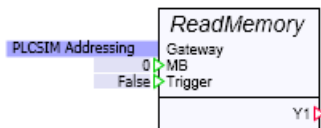
**7.1.6 Communication components**

**7.1.6.1 Components for SIMATIC**

The PLCSIM coupling and the PRODAVE coupling enable access to the bit memory address area and the data block area. This access is not carried out by cyclic communication between the controller and signals that are listed in the coupling editor, but via components which read or write a specified address area of the controller on a trigger signal. The required component types can be found in the basic library in the directory *COMMUNICATION | SIMATIC*.

The components must be provided with a *Unit* connector at their *Gateway* input. You can find additional information on this in the section: Linking SIWAREXU components to the coupling (Page 408).

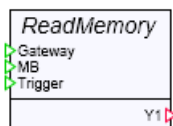
You link the components with the relevant coupling simply by entering the name of the PLCSIM or PRODAVE coupling that you want to access using this component in the property window of the unit connector. Entering the address in the Unit connector is of no significance in this case.



To use this access method for a coupling, you must have already saved the coupling. Open the coupling in the editor, define, for example, an input or output signal and then save the coupling.

**ReadMemory – Reading a bit memory address area**

**Symbol**



## Function

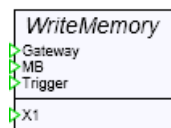
The *ReadMemory* component type enables one or more successive bytes from the bit memory address area of a controller to be read. Enter the address of the first byte to be read at the input *MB*. The read operation is executed when a rising edge occurs at the *Trigger* input, which means a change from False to True.

The number *N* of outputs can be varied by "dragging" the component onto a chart. You can specify a maximum of 32 outputs, which means you can read a maximum of 32 bytes with a component of this type. The bytes that are read are available at the outputs *Y1* to *YN*.

Exactly one read operation is triggered while the simulation is being initialized. This means initial values from the bit memory address area are available after initialization even though no trigger signals have been sent.

## WriteMemory – Writing to a bit memory address area

### Symbol



### Function

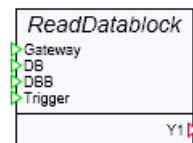
The *WriteMemory* component type allows you to write to one or more successive bytes in the bit memory address area of a controller. Enter the address of the first byte to be written to at the input *MB*. The write operation is executed when a rising edge occurs at the *Trigger* input, which means a change from False to True.

The number *N* of inputs can be varied by "dragging" the component onto a chart. You can specify a maximum of 32 inputs, which means you can write to a maximum of 32 bytes in the bit memory address area with a component of this type. The bytes to be written should be made available at inputs *X1* to *XN*.

Exactly one write operation is triggered while the simulation is being initialized. Initial values can therefore be written to the bit memory address area during the initialization process, even though no trigger signals have been sent.

## ReadDatablock – Reading a data block

### Symbol



## Function

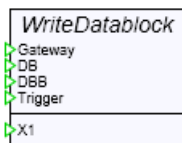
The *ReadDatablock* component type enables one or more successive bytes from a data block of a controller to be read. Enter the data block number at the *DB* input and the address of the first byte to be read at the *DBB* input. The read operation is executed when a rising edge occurs at the *Trigger* input, which means a change from False to True.

The number *N* of outputs can be varied by "dragging" the component onto a chart. You can specify a maximum of 32 outputs, which means you can read a maximum of 32 bytes with a component of this type. The bytes that are read are available at the outputs *Y1* to *YN*.

Exactly one read operation is triggered while the simulation is being initialized. Initial values from the data block are available after initialization, even though no trigger signals have been sent.

## WriteDatablock – Writing to a data block

### Symbol



## Function

The *WriteDatablock* component allows you to write to one or more successive bytes in the data block of a controller. Enter the data block number at the *DB* input and the address of the first byte to be written at the *DBB* input. The write operation is executed when a rising edge occurs at the *Trigger* input, which means a change from False to True.

The number *N* of inputs can be varied by "dragging" the component onto a chart. You can specify a maximum of 32 inputs, which means you can write to a maximum of 32 bytes in the data block with a component of this type. The bytes to be written should be made available at inputs *X1* to *XN*.

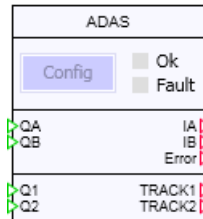
Exactly one write operation is triggered while the simulation is being initialized. Initial values can therefore be written to the data block during the initialization process, even though no trigger signals have been sent.

### 7.1.6.2 Components for SINUMERIK

The *COMMUNICATION / SINUMERIK* directory of the Basic Library contains a component type for communicating axis values from SINUMERIK to SIMIT.

## ADAS – AXIS DATA STREAM PER PROFIBUS

### Symbol



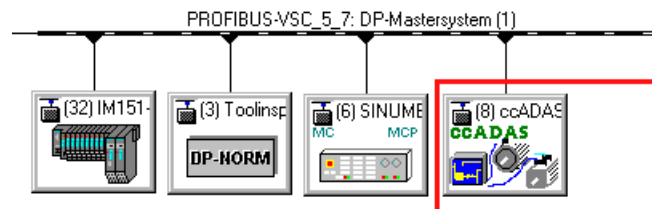
### Function

The *ADAS* component type is used for parameter assignment and preparation of axis values, which are transferred to SIMIT via PROFIBUS DP from a SINUMERIK controller using the *ADAS* software package.

### Note

The *ADAS* software package is not part of the SIMIT product and needs to be purchased separately.

To transfer axis values you need to insert the PROFIBUS slave *ccADAS* into the SINUMERIK hardware configuration. The GSD file of the *ccADAS* slave can be found on the SIMIT Installation CD in the folder *Tools/ADAS*.



This slave has 8 bytes each of input and output data for communication and an additional 4 bytes for each axis value.

**Profibus**

Save and Load

▼ Inputs Reset Filter

Default	Symbol Name	Address	Data Type	System	Slave	Slot	Comment
0		ED17	DWORD	1	8	1	
0		ED21	DWORD	1	8	1	

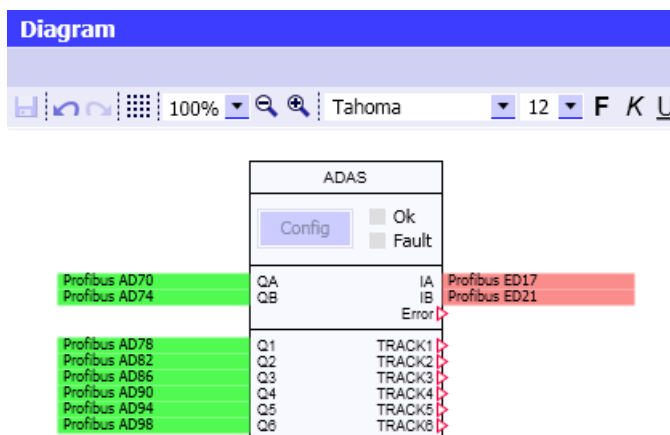
▼ Outputs Reset Filter

Symbol Name	Address	Data Type	System	Slave	Slot	Comment
	AD70	DWORD	1	8	1	
	AD74	DWORD	1	8	1	
	AD78	DWORD	1	8	1	
	AD82	DWORD	1	8	1	
	AD86	DWORD	1	8	1	
	AD90	DWORD	1	8	1	
	AD94	DWORD	1	8	1	
	AD98	DWORD	1	8	1	

**Profibus**

Profibus	Property	Value
▼ System 1	Module	Header + 6 Tracks (4 Byte)
▶ [3] L-SC IM-SC/def.DP	Slot	1
▶ [6] SLAVE_8109	Addresses Inputs	EB17 - EB24
▼ [8] ccADAS	Addresses Outputs	AB70 - AB101
[1] Header + 6 Tracks (4 Byte)	Failsafe	No
▶ [32] ET 200S HighFeature (Cu)	Pull module	<input type="checkbox"/>

All I/O data are configured as double words in the coupling and must be connected to the SIMIT component (*QA, QB, Q1 .. Qn* or *IA* and *IB*):



The transferred axis values are available at the analog outputs *TRACK<sub>i</sub>*,  $i = 2, \dots, N$  of a component of type ADAS. The number *N* of available channels can be specified by scaling the component. A maximum of 28 channels can be specified. You can, for example, connect



the *TRACKi* outputs with the 3D viewer control in SIMIT to animate the 3D view of the machine with current axis values.

### Note

The number of transferred axis values is specified by assigning parameters to the *ccADAS* slave. You should set the number in the ADAS component to the same value as defined in the slave to use all channels within the component.

In case of communication failure the integer output *Error* provides information about the cause of the error.

Table 7-37 Error codes for the ADAS component

Error	Cause
0	No error
1	No reply to configuration of Track 1
2	No reply to configuration of Track 2
..	
28	No reply to configuration of Track 28
100	No reply to Reset command
101	No reply to Set Communication command

## Parameters

The *ADAS* component has the parameter vectors *AxisType* and *AxisNumber*, which have as many elements as the specified number of channels. This allows for each channel to specify which axis of the SINUMERIK-NC is to be transferred and whether the axis moves linearly or is an axis of rotation.

- *AxisType*  
Depending on this parameter assignment, the axis values are provided in mm (translational), angle degrees (rotatory grad) or radians (rotatory rad).
- *AxisNumber*  
This parameter specifies which axis is to be transferred on the corresponding ADAS channel.

The figure below shows the parameters together with their default values:

ADAS#1		
General	Name	Value
Input	▼ AxisType [2]	...
Output	AxisType1	translational ▼
Parameter	AxisType2	translational ▼
Additional parameter	▼ AxisNumber [2]	...
State	AxisNumber1	1
	AxisNumber2	1

**Note**

Note that the parameter assignment will be effective only if the *Config* button in the operating window or on the basic symbol is pressed to transfer the configuration to the SINUMERIK while the simulation is running and the SINUMERIK controller is connected.

As an alternative, you may also specify these settings in the SINUMERIK machine data.

**Additional parameters**

The additional parameter *TimeOut* allows you to specify the period of time after which the component will cancel communication with the SINUMERIK, if the command to transfer the configuration receives no response from the SINUMERIK.

The figure below shows the additional parameter together with its default values:

ADAS#1	
General	Name Value
Input	TimeOut [s] 5.0
Output	
Parameter	
Additional parameter	
State	

**Operating window**

The transfer of the configuration to the SINUMERIK can be triggered in the operating window of the *ADAS* component type by using the *Config* button. While transfer is in progress, the two displays *Ok* and *Fault* are not active. Successful transfer is indicated by the green display *Ok*, and failure is indicated by the red display *Fault*.



The same operating and display elements can be found on the basic symbol and provide the same function.

## 7.1.7 Controls

### 7.1.7.1 Controls for displaying signal values

#### Binary display

#### Symbol



#### Function

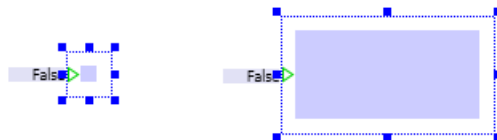
The *Binary display* control is used to display a binary value. The color and shape of the control can be defined in the view properties.

Binary Indicator#1		
General	Property	Value
Connector	Color (off)	<input type="text"/>
View	Color (on)	<input type="text"/>
	Shape	Rectangular
		Rectangular
		Round

You can select any color for the signal Off state (signal value zero) and On state (signal value one). You can also toggle the shape of the control between *Rectangular* and *Round* in a drop-down box.



You can change the size of the control as required using the width and height handles on the selection frame. To change the size using the corner handles, hold down the Shift key and the size of the control will increase or reduce proportionately.



## Analog display

### Symbol



### Function

The Analog display control is used to display an analog or integer value in the form of a pointer instrument. The *Data type* of the signal to be displayed can be toggled between *Analog* and *Integer* in the general properties of the control.

Analog Display#1		
General	Property	Value
Connector	Name	Analog Display#1
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1hwiueeg
	Data Type	Analog
	Position	Analog
	Width	Integer
	Height	90.0

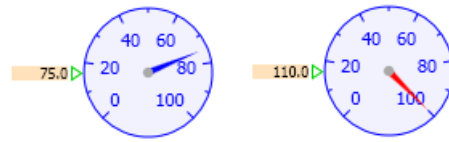
Other settings for the analog display can be adapted individually in the view properties. The figure below shows the property view with the default settings.

Analog Display#1		
General	Property	Value
Connector	Minimal Value	0.0
View	Maximal Value	100.0
	Initial Angle	225
	Final Angle	315
	Scale	20.0
	Decimal Places	0
	Label	
	Font Size	11.0
	Foreground	
	Background	

The range of values and the scale, labels and color can all be set as required:

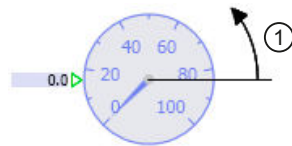
- **Value range**

The value range is determined by the minimum and maximum values. If the value to be displayed is outside the set range of values, the pointer of the analog display appears in red.



- **Scale**

*Initial angle* and *Final angle* identify the start and end of the scale. The initial angle displays the minimum value and the final angle the maximum value. The angle is defined as degrees from the horizontal axis in counterclockwise direction. *Scale marks* indicate the subdivision for values on the scale. Scale values are displayed with the specified number of *decimal places*.



① Angle

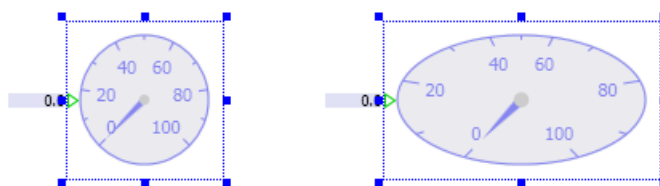
- **Label**

You can enter a text in the *Label* property box that will appear in the analog display with the adjustable *Font size*.

- **Color**

The border, scale and texts are displayed in the adjustable, uniform foreground color. The background color can also be set as the fill color for the control.

You can change the size and thus the round shape of the control as required using the width and height handles on the selection frame. To change the size using the corner handles, hold down the Shift key and the size of the control will increase or reduce proportionately.



## Digital display

## Symbol



Function

The *Digital display* control is used to display the value of an analog or integer signal. The *Data type* of the signal to be displayed can be toggled between *Analog* and *Integer* in the general properties of the control.

Digital Display#2		
General	Property	Value
Connector	Name	Digital Display#2
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1hwiueeg
	Data Type	Analog
	Position	Analog
	Width	Integer
	Height	30.0

For analog signals you can set the *font size* and the number of *decimal places* to be displayed in the *View* properties dialog.

Digital Display#1		
General	Property	Value
Connector	Font Size	12.0
View	Decimal Places	2

For integer signals, you can set the *Font size*, the *Display format* and the *Data width* in the property view.

Digital Display#1		
General	Property	Value
Connector	Font Size	12.0
View	Display format	Decimal
	Data size [bytes]	Decimal
		Hexadecimal
		Character

Decimal numbers can be positive or negative. In hexadecimal notation negative numbers are always represented as a two's complement. When converting to hexadecimal numbers you need to specify how many bytes are to be included (1, 2, 4 or 8 bytes). As hexadecimal numbers are displayed with a fixed number of characters, this setting also determines the number of hexadecimal characters displayed (2, 4, 8 or 16 places).

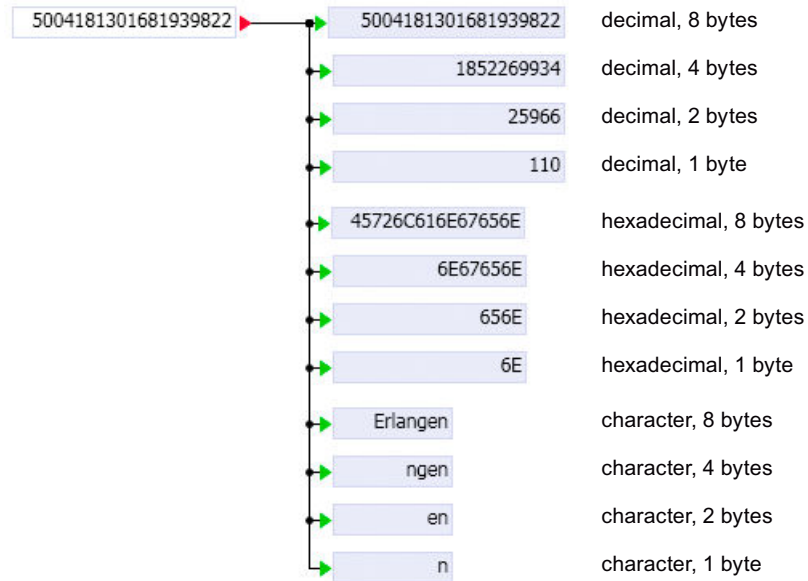
In the *Characters* display format only the specified number of bytes are included, starting with the least significant byte, resulting in a display of 1, 2, 4 or 8 characters. A character corresponding to the coding of the (extended) ASCII code is displayed if it is a displayable character.

**Note**

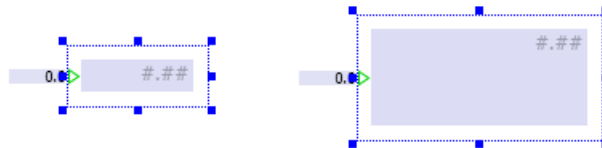
If the data width is set to less than 8, the actual value of an integer input variable might not be displayed in some circumstances.

The data width does not affect the input of a value as the effective value is not limited. However, the way the entered value is displayed will depend on the data width setting.

The figure below shows examples of the effect of different display formats and data widths:



You can change the size of the symbol as required, and thus modify the set font size, using the width and height handles on the selection frame. To change the size using the corner handles, hold down the Shift key and the size of the control will increase or reduce proportionately.



### Bar graph display

### Symbol



Function

The *Bar graph display* is used to display an analog or integer signal in the form of a bar graph. The *Data type* of the signal to be displayed can be toggled between *Analog* and *Integer* in the general properties of the control.

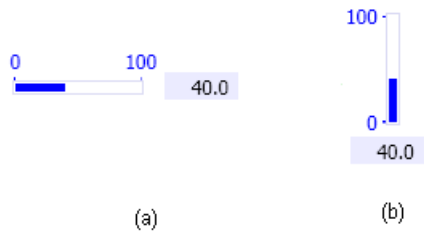
Bar Indicator#1		
General	Property	Value
Connector	Name	Bar Indicator#1
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1hwiueeg
	Data Type	Analog
	Position	Analog
	Width	Integer
	Height	40.0

Other settings for the bar graph display can be adapted individually in the property view.

Bar Indicator#1		
General	Property	Value
Connector	Start Value	0.0
View	End Value	100.0
	Orientation	Horizontal
	Show Scale	<input checked="" type="checkbox"/>
	Show Value	<input checked="" type="checkbox"/>

The following properties can be set:

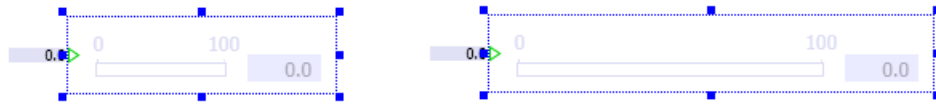
- Scale**  
 The displayed range of values is defined by the *Start value* and the *End value*. You can toggle the scale display on and off with the *Show scale* check box.
- Orientation**  
 The possible orientations for the control are *Horizontal* (a) and *Vertical* (b).



- Bar value display**  
 Use the *Show value* check box to toggle the additional numerical bar value display on or off.

You can change the size of the symbol using the handles on the selection frame. This allows you to make the horizontally-aligned bar graph display wider, for example.





### 7.1.7.2 Controls for entering signal values

#### Pushbutton

#### Symbol



#### Function

The *Pushbutton* control is used to enter a binary signal.

The pushbutton can be defined as *Normally Open* or *Normally Closed* in the general properties. The default setting is *Normally Open*.

Pushbutton#1		
General	Property	Value
Connector	Name	Pushbutton#1
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1i0ipgaq
	Type	Normally Open
	Position	Normally Closed
	Width	Normally Open
	Height	30.0

To activate the pushbutton while the simulation is running, simply click the button. The figure below shows the unpressed button (a) and the pressed button (b):



(a)

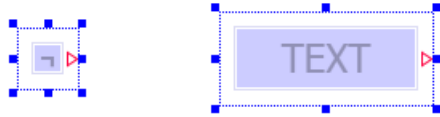


(b)

As can be seen in the properties window of the figure below, you can specify a *text* for the *pushbutton* that will appear on the button in the configured *font size*.

Pushbutton#1		
General	Property	Value
Connector	Text	
View	Font Size	12.0

The handles on the button frame can be used to change the size and thus adjust the size of the specified text, for example. The text is aligned centered in the button (both horizontally and vertically).



Pushbutton with image

Symbol



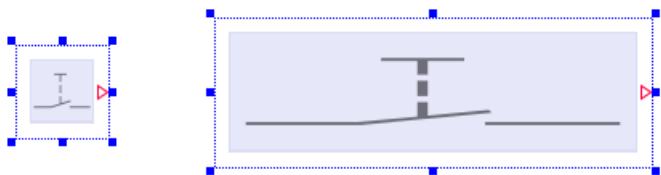
Function



The *Pushbutton with image* is used to enter a binary signal; images are used to represent the button position.





The pushbutton can be defined as *Normally Open* or *Normally Closed* in the general properties. The default setting is *Normally Open*.

Pushbutton with Image#1		
General	Property	Value
Connector	Name	Pushbutton with Image#1
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1i0ipgaq
	Type	Normally Open
	Position	Normally Closed
	Width	Normally Open
	Height	50.0

To activate the pushbutton while the simulation is running, simply click the symbol. The size of the symbol and thus the size of the sensitive area can be changed using handles on the selection frame.



In the property view you can specify images that represent the *unpressed* (Off) and *pressed* (On) pushbutton while the simulation is running. Click  to open the dialog for selecting the corresponding graphic files. Click  to delete the selection.

Pushbutton with Image#1		
General	Property	Value
Connector	Adapt to Image Size	<input type="checkbox"/>
<b>View</b>	Image (off)	 
	Image (on)	 

The *Adapt to image size* check box is not selected by default. The selected images will then be adapted to the size of the button, which means the width and height are scaled accordingly. If the check box is selected, the size of the button is matched to the size of the graphic for *Image (not pressed)*.

## Switch

### Symbol



### Function





The *Switch* is used to switch a binary signal.

The switch can be defined as *Normally Open* or *Normally Closed* in the *Off* or *On* position in the general properties. It is set to *Normally Open* in the *Off* position by default. The selected position takes effect as the *Default* when the simulation starts.

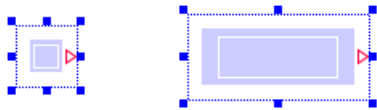
Switch#1		
General	Property	Value
Connector	Name	Switch#1
	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1i0ipgaq
	Type	Normally Open
	Default	Off
	Position	X: 525.0 Y: 180.0
	Width	30.0
	Height	30.0

7.1 The basic library

To activate the switch while the simulation is running, simply click the button. The closed switch is represented by a dark-blue border. The table shows the switch as Normally Closed and Normally Open with the default settings *Off* and *On*.

Default setting	NC contact	NO contact
On		
Off		

Handles on the button frame are used to change the size.



Switch with image

Symbol



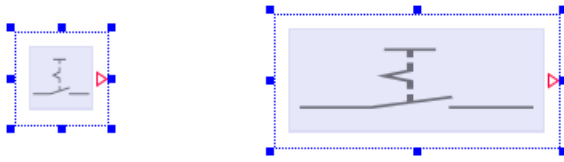
Function

The *Switch with image* is used to switch a binary signal; images are used to represent the switch position.

The switch can be defined as *Normally Open* or *Normally Closed* in the *Off* or *On* position in the general properties. It is set to *Normally Open* in the *Off* position by default. The selected position takes effect as the *Default* when the simulation starts.

Switch with Image#1		
General	Property	Value
Connector	Name	Switch with Image#1
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1i0ipgaq
	Type	Normally Open
	Default	Off
	Position	X: 110.0 Y: 35.0
	Width	50.0
	Height	50.0

To activate the switch while the simulation is running, simply click the symbol. The size of the symbol and thus the size of the sensitive area can be changed using handles on the selection frame.



In the property view you can specify images that represent the two possible switch states - *Off* and *On* - while the simulation is running. Click **...** to open the dialog for selecting the corresponding graphic files. Click **X** to delete the selection.

Switch with Image#1		
General	Property	Value
Connector	Adapt to Image Size	<input type="checkbox"/>
<b>View</b>	Image (off)	<b>...</b> <b>X</b>
	Image (on)	<b>...</b> <b>X</b>

The *Adapt to image size* check box is not selected by default. The selected images will then be adapted to the size of the button, which means the width and height are scaled accordingly. If the check box is selected, the size of the button is matched to the size of the graphic for *Image (off)*.

### Step switch

### Symbol



### Function

The *Step switch* is used to step through the switching of an integer value. The numerical value is increased or reduced by one every time it is switched.

The minimum and maximum values for the switched signal can be defined in the general properties. The default setting is a ten-step switch with a range of values from zero to ten, starting with the switch value zero.

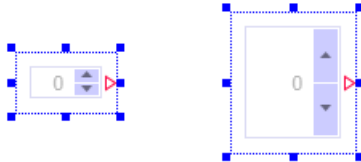
Stepping Switch#1		
General	Property	Value
Connector	Name	Stepping Switch#1
	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1hwiuig
	Minimal Value	0
	Maximal Value	10
	Default	0
	Position	X: 330.0 Y: 230.0
	Width	55.0
	Height	30.0

7.1 The basic library

To activate the step switch while the simulation is running, simply click the top (▲) or the bottom (▼) button. The switch value that is currently set is displayed in the symbol.



The size of the symbol and thus the size of the two buttons can be changed using handles on the selection frame.



Step switch with image

Symbol







Function

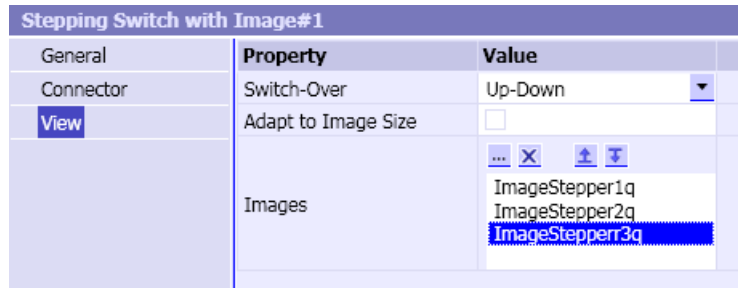
The *Step switch with image* is used to step through the switching of an integer value; images are used to represent the switch position.

The switch value from which the step switch starts is set in the general properties. The *default setting* is zero. The numerical value of the switched signal is increased or reduced by one every time it is switched.

Stepping Switch with Image#1		
General	Property	Value
Connector	Name	Stepping Switch with Image#1
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1hwiuig
	Default	0
	Position	X: 495.0 Y: 135.0
	Width	50.0
	Height	50.0

The number of switching steps is defined by the number of images. The images are set in the view properties. They change for every step while the simulation is running in the order set in the *Images* list.

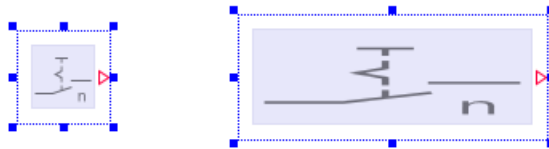
Click  to open the dialog for selecting the corresponding graphic files. The selected image is added to the list. The order of the images can be changed by swapping an image with the image above it () or the image below it () in pairs. Click  to delete an image.



To activate the step switch while the simulation is running, simply click the symbol. The *Switchover* is selected via the drop-down list.

In the "Up-Down" parameter setting, the value of the step switch is incremented when you click in the top half of the control and decremented when you click in the bottom half.

Handles on the selection frame are used to change the size of the symbol.



The *Adapt to image size* check box is not selected by default. The images are then adapted to the size of the button, which means the width and height are scaled accordingly. If the check box is selected, the size of the button is matched to the size of the graphic for the first image.

## Digital input

### Symbol



### Function

The *Digital input* is used to enter an analog or integer signal value in digital form.

The *Data type* of the signal can be set to *Analog* or *Integer* in the general properties. The *Default* signal value can also be set. It is set to zero by default.

Digital Input#2		
General	Property	Value
Connector	Name	Digital Input#2
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1hwiueeg
	Data Type	Analog
	Default	Analog
	Position	Integer
	Width	80.0
	Height	30.0

For analog signals you can set the *Font size* and the number of *Decimal places* to be displayed in the View properties.

Digital Input#1		
General	Property	Value
Connector	Font Size	12.0
View	Decimal Places (Display Only)	2

For integer signals, you can set the *Font size*, the *Display format* and the *Data width* in the property view.

Digital Input#1		
General	Property	Value
Connector	Font Size	12.0
View	Display format	Decimal
	Data size [bytes]	Decimal
		Hexadecimal
		Character

Entries must be made in the specified display format. If the syntax used does not correspond to the specified display format, the input is interpreted as zero.

Decimal numbers can be positive or negative. In hexadecimal notation negative numbers are always represented as a two's complement. When converting to hexadecimal numbers you need to specify how many bytes are to be included (1, 2, 4 or 8 bytes). As hexadecimal numbers are displayed with a fixed number of characters, this setting also determines the number of hexadecimal characters displayed (2, 4, 8 or 16 places).

In the *Characters* display format only the specified number of bytes are included, starting with the least significant byte, resulting in a display of 1, 2, 4 or 8 characters. A character corresponding to the coding of the (extended) ASCII code is displayed if it is a displayable character.

The data width does not affect the input of a value because the effective value is not limited. The specified value is only displayed in accordance with the configured data width.

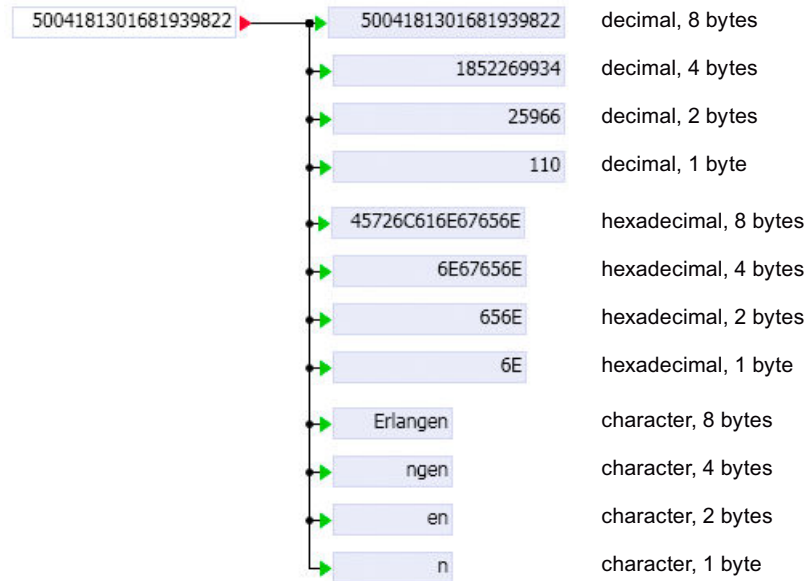
**Note**

If the data width is set to less than 8, the actual value of an integer input variable might not be displayed in some circumstances.

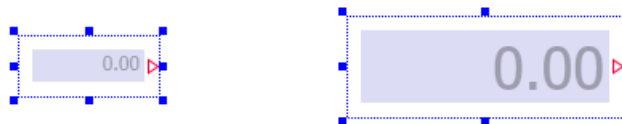


When entering values via the Digital Input control, you must use the specified display format. If the entry is syntactically incorrect, it is interpreted as "0".

The figure below shows examples of the effect of different display formats and data widths:



You can change the size of the symbol as required, and thus modify the set font size, using the width and height handles on the selection frame. To change the size using the corner handles, hold down the Shift key and the size of the control will increase or reduce proportionately.



The signal values appear right-justified in the symbol while the simulation is running.

## Slider

## Symbol



## Function

The *Slider* is used to set an analog or integer signal.

The default setting for the signal value is set in the general properties. The default is zero.

Slider#1		
General	Property	Value
Connector	Name	Slider#1
View	Cycle	2
	Show Name	<input type="checkbox"/>
	UID	a_02hipv_1hwiueeg
	Default	0.0
	Position	X: 565.0 Y: 195.0
	Width	160.0
	Height	30.0

The slider is moved with the mouse while the simulation is running. Position the mouse pointer over the button of the slider, hold down the left mouse button and move the button to the desired position. You can also click to the right or left of the button to increase or reduce the set value by one step.

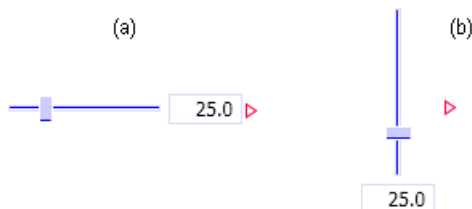


Other settings for the slider view can be changed individually in the property view.

Slider#1		
General	Property	Value
Connector	Start Value	0.0
View	End Value	100.0
	Step	1.0
	Orientation	Horizontal
	Show Value	<input checked="" type="checkbox"/>

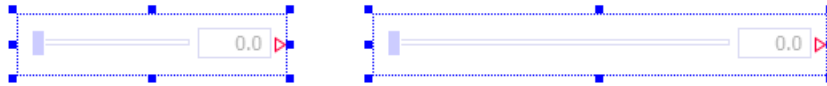
The following properties can be set:

- Scale**  
 The scale is determined by the *Start value*, the *End value* and the *Increment*.
- Orientation**  
 The possible orientations for the control are *Horizontal* and *Vertical*.



- Bar value display**  
 Use the *Show value* check box to toggle the additional numerical display of the signal value on or off.

You can change the size of the symbol using the handles on the selection frame.



### 7.1.7.3 Miscellaneous controls

#### Signal splitter

#### Symbol



#### Function

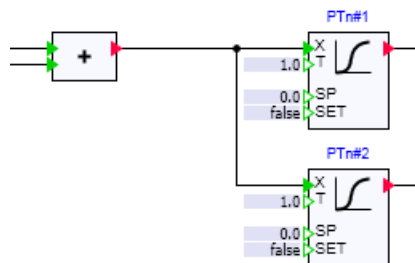
The *Signal splitter* is used to force, which means to set, values at the inputs and outputs of components or coupling signals while the simulation is running using any of the Input controls.

Forcing is already available in the property view for all inputs and outputs of components.

The *Signal splitter* control also allows you to force inputs and outputs using any Input control, as described below by way of example.

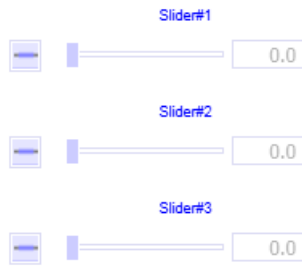
**Tip:** You can use the signal splitter without an assigned Input control and connect it, for example, to the component input or component output that is to be forced. In this case, the signal splitter will perform the same function as the signal splitter in the properties of the input or output.

The figure below shows a section from a chart containing an adder whose output is connected to the inputs of two PTn elements.



If you want to implement the forcing of the adder output and the two inputs of the PTn elements using sliders, add a slider and a signal splitter to a chart for each input and output to be forced.

7.1 The basic library



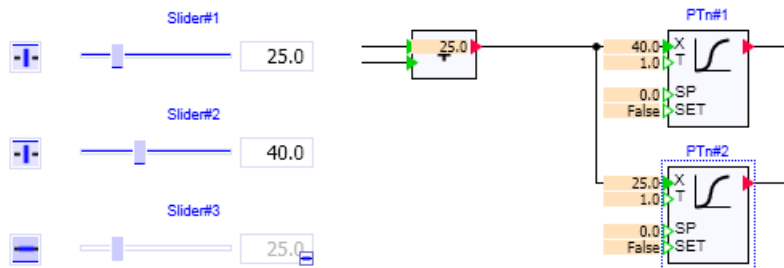
Then open the property view for the first slider, set its connector to invisible and enter the output of the adder as the connected signal. Then open the connector properties for the associated signal splitter. Its connector is always invisible, as can be seen in the figure below.

Signal Disconnecter#1	
General	Name
Connector	Signal

Here, too, you enter the adder output as the signal.

Signal Disconnecter#1	
General	Name
Connector	Signal ADD#1 Y


Connect the other two sliders and signal splitters to the inputs of the two PTn elements in the same way. You can now force the output and the inputs while the simulation is running, as shown by way of example in the figure below.



The signal splitters for the two controls *Slider#1* and *Slider#2* are switched on (☑). Forcing of the adder output and of the input to PTn#1 via the two sliders is thus activated. The signal splitter for *Slider#3* is not switched on (☐), which means the input PTn#3 cannot be forced this way. *Slider#3* is shown as inactive and is also identified with the overlay ☐.

Values for the output and input can now be set using the first two sliders. In the example, the adder output is set to the value 25 and the input of PTn#1 is set to the value 40. The input of PTn#2 is connected to the adder output, and thus takes on its value (25). The (inactive) Slider#3 displays the value of the input connected to it.

The following points should thus be noted when using signal splitters:

- The signal splitter and the associated Input control should be linked to the input or output to be forced.
- Forcing is not activated for signal splitters that are not switched on. The associated input control is shown as inactive and identified by the overlay . It indicates the value of the connected input or output.
- Switching on the signal splitter activates forcing.

## See also

Properties of the inputs (Page 306)

Properties of the outputs (Page 307)

## Action


### Symbol



### Function

The action is used to open a chart or trend.

You can determine if a chart or trend is to be opened under "Action" in the general properties. As the "Target", specify the chart or trend to open starting with a slash followed by the complete folder hierarchy separated by slashes. Alternatively, you can drag the appropriate chart or trend and drop it into the "Target" property field in the project tree.

Action#1		
General	Property	Value
View	Name	Action#1
	Show names	<input type="checkbox"/>
	UID	f_000hsn_3x9lhfnj
	Action	Open chart 
	Target	/New folder/Chart
	Position	X: 235.0 Y: 120.0
	Width	70.0
	Height	35.0

Click the button to trigger the action while simulation is running.

Text can be specified for the action. This text is shown with the adjustable font size on the button:

Action#1		
General	Property	Value
View	Text	Overview
	Font size	14.0

The size of the button can be changed and adjusted, for example, to fit the size of the specified text. The text is aligned centered in the button (both horizontally and vertically).

#### 7.1.7.4 The 3D Viewer control

The graphical editor in SIMIT allows you to clearly visualize the behavior of a machine using simple graphical basic elements. Two-dimensional graphics can be used to draw a machine and to show movements of the machine by animating relevant parts of this drawing. The 3D Viewer control gives you the added option of incorporating three-dimensional animated views of a machine in your simulation. The representation of the machine is then clearer, and the movements of the machine are displayed more realistically.

In order to use the 3D Viewer control you must have a three-dimensional geometry model of the machine. This 3D model must be made kinematic, which means it must be designed in such a way that it can be linked to signals from the functional simulation and then execute the animations controlled by these signals in the simulation. The kinematic 3D model includes elements that are evaluated by the 3D Viewer control and converted into animations of the 3D model.

Like the other controls in the basic library, the 3D Viewer control is inserted in a chart and its parameters are assigned accordingly. It also has connectors with which it can be connected to signals of the functional model. A special feature of the 3D Viewer control is a menu that can be used to adapt the view of 3D model by means of commands.

#### Data format requirements

In order to use the 3D Viewer control you need a three-dimensional geometry model in VRML V2.0 format. This VRML format can be exported from most CAD systems. In some cases, however, you will also need to restructure the VRML model after export to identify and capture the shapes or shape groups that are to be animated as kinematic simulation points. In terms of the size of a model exported from CAD systems, it is also worth reducing it by eliminating details of the geometry model that are not necessary for visualization.

To restructure the 3D model and make it kinematic, you can use a suitable VRML editor that shows the geometric structure of the 3D model and allows you to modify the VRML code. Information on editing environments for VRML can be found on the web pages of the Web3D Consortium, for example: [www.web3d.org](http://www.web3d.org).

#### Animating the 3D model

The 3D Viewer control allows you to animate individual shapes or shape groups of the 3D model in various ways. You can:

- Move and rotate shapes or shape groups in space (translational and rotational movement)
- Reshape shapes or shape groups (size scaling)
- Change the color and transparency of individual shapes

In order to perform these operations, you need to assign appropriate elements or identifiers to the individual shapes or shape groups in the 3D model:

- For movement animations specific motion sensors are added to the 3D model
- For reshaping and changes to the surface of a shape appropriate identifiers are added to the shape definition.

The functioning of the various motion sensors is explained and the other options for modifying models are described in the sections below. Examples are used to show how the various sensors and identifiers can be added to a VRML file.

## Animation sensors

A 3D model usually uses sensors to respond to user actions. Sensors in the 3D Viewer control, on the other hand, are routed to connectors. This means you can connect the connectors of the 3D Viewer control to signals from your functional model. This means movements are calculated by the functional model during the simulation and visualized by the 3D Viewer control.

The 3D Viewer control supports the following sensors for animating the 3D model:

- Plane sensors for the translational movement of objects
- Cylinder sensors for rotating objects about the local coordinate axes
- Sphere sensors for rotating objects around a vector

For each motion sensor of the 3D model the following applies:

- At least one higher-level Transform node must exist for a sensor. The sensor can be placed anywhere in the Transform node.
- The position of the sensor alone determines which shapes or shape groups are moved: The Transform node to which the sensor is assigned and all its child nodes are moved by the sensor.
- Routes to sensors are not necessary and are not evaluated.

For each sensor a specific number of connectors is made available in the connector properties of the control. Connect the connectors that perform the desired movement in the 3D model to the corresponding signals in your functional model.

Faulty sensors, which means sensors that are not assigned to a Transform node, are interpreted as not being present. For this reason no animation connectors are available for faulty sensors in the 3D Viewer control properties.

## PlaneSensor – the plane sensor

A plane sensor is used to translate an object, which means a shape or a shape group. In the VRML standard plane sensors can be used to move objects in two spatial directions – in the X and Y direction of the local coordinate system. The 3D Viewer control allows a translation in all three spatial directions at each plane sensor. A plane sensor is placed in the VRML model with the keyword **PlaneSensor**.

Once the VRML file has been loaded into the 3D Viewer control, there are three analog connectors available for a plane sensor in the control properties:

Sensorname#TX     for translating the object in the X direction  
 Sensorname#TY     for translating the object in the Y direction  
 Sensorname#TZ     for translating the object in the Z direction

If no *Sensorname* is defined for a plane sensor, *TranslationN* is set as the sensor name, whereby *N* is a sequential number for the plane sensor, for example, *N*= 1, 2, ...

The example below defines a cone to which a plane sensor called *ConeSensor* is assigned:

```
#VRML V2.0 utf8
DEF ConeTransform Transform
{
  children
  [
    DEF Cone Shape
    {
      appearance Appearance
      {
        material Material {}
      }
      geometry Cone {}
    }
    DEF ConeSensor PlaneSensor {}
  ]
}
```

Once this VRML file has been loaded into the 3D Viewer control, the following three analog connectors are available in the 3D Viewer control properties for translating the cone:

ConeSensor#TX     (translation in X direction)  
 ConeSensor#TY     (translation in Y direction)  
 ConeSensor#TZ     (translation in Z direction)

You can now connect each of these connectors to an analog signal of your functional model to animate the desired movement of the cone. For connectors that have not been connected to signals, no movement of the cone occurs in the corresponding direction.

### CylinderSensor – the cylinder sensor

Cylinder sensors are used to rotate an object, which means a shape or shape group by one of the three local coordinate axes. Cylinder sensors should be used as follows to rotate objects about the local coordinate axes X, Y or Z. A cylinder sensor is placed in the VRML model with the keyword **CylinderSensor**. The angle of rotation for an axis is given in degrees.



Once the VRML file has been loaded into the 3D Viewer control, there are three analog connectors available for a cylinder sensor in the control properties:

Sensorname#RX for rotating the object about the local X axis  
 Sensorname#RY for rotating the object about the local Y axis  
 Sensorname#RZ for rotating the object about the local Z axis

If no *Sensorname* is defined for a cylinder sensor, *RotationN* is set as the sensor name, whereby *N* is a sequential number for the cylinder sensor, for example, *N*= 1, 2, ...

The example below defines a cone to which a cylinder sensor called *ConeSensor* is assigned:

```
#VRML V2.0 utf8
DEF ConeTransform Transform
{
  children
  [
    DEF Cone Shape
    {
      appearance Appearance
      {
        material Material {}
      }
      geometry Cone {}
    }
    DEF ConeSensor CylinderSensor {}
  ]
}
```

Once this VRML file has been loaded into the 3D Viewer control, the following three analog connectors for rotating the cone are available in the 3D Viewer control properties:

ConeSensor#RX (rotation about the X axis)  
 ConeSensor#RY (rotation about the Y axis)  
 ConeSensor#RZ (rotation about the Z axis)

You can now connect each of these connectors to an analog signal of your functional model to animate the desired rotation of the cone. For connectors that have not been connected to signals, no rotation of the cone occurs about the corresponding axis.

## SphereSensor – the sphere sensor

Sphere sensors resolve the restriction associated with cylinder sensors of only being able to rotate about coordinate axes. A direction vector is specified for sphere sensors about which the shape is to be rotated. A sphere sensor is placed in the VRML model with the keyword **SphereSensor**. The angle of rotation is given in degrees.

Once the VRML file has been loaded into the 3D Viewer control, there are four analog connectors available for a sphere sensor in the control properties:

Sensorname#R Angle of rotation  
 Sensorname#X X-coordinate of the direction vector

Sensorname#Y      Y-coordinate of the direction vector  
 Sensorname#Z      Z-coordinate of the direction vector

If no *Sensorname* is defined for a sphere sensor, *SphereSensorN* is set as the sensor name, whereby *N* is a sequential number for the sphere sensor, for example, *N* = 1, 2, ...

The example below defines a cone to which a sphere sensor called *ConeSensor* is assigned:

```
#VRML V2.0 utf8
DEF ConeTransform Transform
{
  children
  [
    DEF Cone Shape
    {
      appearance Appearance
      {
        material Material {}
      }
      geometry Cone {}
    }
    DEF ConeSensor SphereSensor {}
  ]
}
```

Once this VRML file has been loaded into the 3D Viewer control, the following three analog connectors for rotating the cone are available in the 3D Viewer control properties:

ConeSensor#R      Angle of rotation  
 ConeSensor#X      X-coordinate of the direction vector  
 ConeSensor#Y      Y-coordinate of the direction vector  
 ConeSensor#Z      Z-coordinate of the direction vector

You can now connect each of these connectors to an analog signal of your functional model to set the direction vector and animate the desired rotation of the cone.

## Scaling objects

If you wish to scale the size of an object, which means a shape or shape group, you need to modify the name of the Transform node to which the object is assigned or the Transform node containing the object to be scaled: Prefix the name of the Transform node with the identifier **SCALE**. Negative scale values flip the object in the scaling axis.

---

### Note

A scale value of zero in two degrees of freedom, which means in two directions, will cause the object to disappear from the animation.

---

Once the VRML file has been loaded into the 3D Viewer control, there are three analog connectors available for a scaled Transform node in the control properties:

SCALETransformname#SX    for scaling in the X direction  
 SCALETransformname#SY    for scaling in the Y direction  
 SCALETransformname#SZ    for scaling in the Z direction

*Transformname* is the name of the Transform node.

The example below shows a VRML model with a box for which a scale has been defined:

```
#VRML V2.0 utf8
DEF SCALEBoxTransform Transform
{
  children
  [
    Shape
    {
      appearance Appearance
      {
        material Material {}
      }
      geometry Box {}
    }
  ]
}
```

Once this VRML file has been loaded into the 3D Viewer control, the following three analog connectors for scaling the box are available in the 3D Viewer control properties:

SCALEBoxTransform #SX    (scaling in X direction)  
 SCALEBoxTransform #SY    (scaling in Y direction)  
 SCALEBoxTransform #SZ    (scaling in Z direction)

You can now connect each of these connectors to an analog signal of your functional model to animate the desired reshaping of the box. For connectors that have not been connected to signals, no scaling of the box occurs in the corresponding axis.

## Changing the color and transparency of a shape

In VRML a shape is defined by a Shape node. In order to change the color or transparency properties of a shape, the name of the shape must be modified. Prefix the name of the Shape node with the identifier **RGBT**.

Once the VRML file has been loaded into the 3D Viewer control, there are four analog connectors available for a Shape node prefixed with the identifier RGBT in the control properties:

RGBTShapename#CT        for the shape transparency value  
 RGBTShapename#CR        for the red component of the shape color  
 RGBTShapename#CG        for the green component of the shape color  
 RGBTShapename#CB        for the blue component of the shape color

*Shapename* is the name of the Shape node.

The shape color is determined by corresponding values for the red, green and blue components. Valid values for a color component are in the range 0, ... ,1. The values for the transparency are also in the range 0, ... ,1. The transparency value 1 means that the shape is transparent and therefore invisible; the transparency value 0 means that the shape is not transparent.

The example below constructs a cylinder for which an animation of the color and transparency is defined:

```
#VRML V2.0 utf8
Transform
{
  children
  [
    DEF RGBTCylinder Shape
    {
      appearance Appearance
      {
        material Material {}
      }
      geometry Cylinder {}
    }
  ]
}
```

Once this VRML file has been loaded into the 3D Viewer control, the following four analog connectors for the transparency and color of the cylinder are available in the 3D Viewer control properties:

RGBTCylinder #CT	(transparency of the shape)
RGBTCylinder #CR	(red component of the shape)
RGBTCylinder #CG	(green component of the shape)
RGBTCylinder #CB	(blue component of the shape)

You can now connect each of these connectors to an analog signal of your functional model to animate the desired color and transparency of the cylinder.

If you only want to animate the visibility of a shape, simply connect connector *#CT* to a signal. The shape is then displayed in its original color (as defined in the VRML file) and you can switch its visibility off and on by means of the signal values zero and one.

---

**Note**

An invisible shape remains invisible even if you change the color values. Only changing the transparency value makes it visible again.

---

To make an entire group of shapes invisible, move the group to a new transformation node and scale the node in two axis directions to zero or one. You can find additional information on this in the section: Scaling objects (Page 450).

If you want to assign the same color to several shapes and animate it, you can utilize the inheritance of properties. The material property of a primary shape can be passed onto any

number of other shapes. The primary shape includes the *RGBT* identifier in its name. The resulting four connectors of the primary shape can be used to switch the color (and also the visibility/transparency) of all other shapes that inherit this material property.

This is illustrated by the example below: Two cylinders change color at the same time, but only one has the *RGBT* identifier.

```
#VRML V2.0 utf8
Transform
{
  children
  [
    DEF RGBTCylinder Shape
    {
      appearance Appearance
      {
        # Definition of the primary material property
        material DEF CylinderColor Material
        {
          diffuseColor 0.2 0 0.8
        }
      }
      geometry Cylinder {}
    }
  ]
  Translation 2 0 0
}
Transform
{
  children
  [
    Shape
    {
      appearance Appearance
      {
        # Inheritance of the material property
        material USE CylinderColor
      }
      geometry Cylinder {}
    }
  ]
  Translation -2 0 0
}
```

## Switching viewpoints

If viewpoints are included in a VRML file, they can be switched both dynamically in the simulation and also by manual operation of the 3D Viewer control. If a VRML file is loaded into the 3D Viewer control, the integer **VIEWPOINT** connector appears in the connector properties of the 3D Viewer control, provided that viewpoints have been defined in the file. The *VIEWPOINT* connector can be set in the value range 1, ..., *N*, whereby *N* is the number of defined viewpoints.

The example below shows the syntax for two viewpoints called Main View and Front View:

```

DEF MainView Viewpoint
{
  position      -2.076697e3 574.432739 4.039152e3
  orientation    0.192129 -0.980301 4.578408e-2 0.482376
  description    "Main view"
}
DEF FrontView Viewpoint
{
  Position      -78.220909 1.075813e3 4.449046e3
  orientation    -0.631093 -0.427644 0.647179 2.717778e-2
  description    "Front view"
}
    
```

To switch viewpoints, connect the *VIEWPOINT* connector to an integer signal whose values you can use to switch to the corresponding viewpoint.

### Configuring the 3D Viewer control

In order to display a 3D model in the 3D Viewer control and modify it dynamically, you need to add a 3D Viewer control to the simulation project. Select the VRML file that describes the 3D model in the control view properties. Once you have selected the VRML file, all degrees of freedom of the 3D model are available to you in the control properties as connectors for animating the model. You can then link the connectors to simulation signals from the "Signals" task card, using drag-and-drop for example.

### Importing the 3D model

The 3D Viewer control is located in the *Controls* task card in the *Miscellaneous* pane. To add a 3D model to the simulation, you need to place an instance of the 3D Viewer control on a chart. As with all other controls, this is done by dragging and dropping it from the "Controls" task card. The size of the control on the chart – and hence the size of the 3D model – can be changed using the width and height sizing handles on the selection frame.

You can then load the VRML file containing the description of the 3D model in the control view properties.

3D-Viewer#1		
General	Property	Value
Connector	3D Model	box <span>...</span>
<b>View</b>	Operable	<input checked="" type="checkbox"/>

You can use the *Operable* property to operate the 3D viewer control (even when the simulation is stopped) and set the view of the 3D model. You can find additional information on this in the section: *Simulating with the 3D Viewer control* (Page 455)

The scene settings (camera settings) are saved with the chart and restored when you open the chart.

## Linking the connectors to signals

When the VRML file is loaded, all sensors and modifications defined in the file are recorded and made available as connectors in the connector settings.

3D-Viewer#1		
General	Name	Signal
Connector	BoxSensor#TX ▶	▶
View	BoxSensor#TY ▶	▶
	BoxSensor#TZ ▶	▶

The connectors that can be used for translating, rotating, scaling, hiding or changing the color of shapes are all provided with a ▶ button. Clicking this button starts an explanatory animation for this connector, making it easy to identify the animated object or its movement axis in the 3D scene. Then link the connectors that execute the animations you require to the corresponding signals of the "Signals" task card using drag-and-drop.

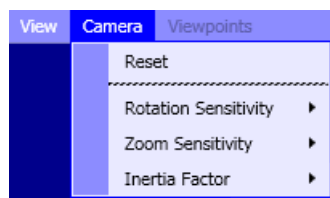
### Note

Of the connectors of a sphere sensor, only connector #R, which is used to animate the angle of rotation, has an animation button ▶. When this animation button is activated, a rotation of the object about the X-axis is animated.

## Simulating with the 3D Viewer control

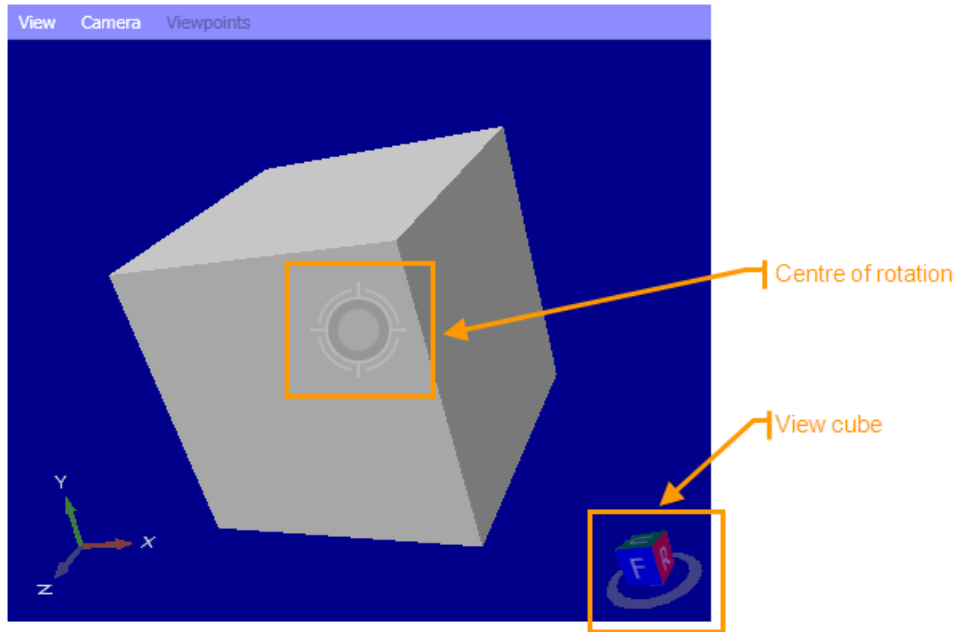
Once the simulation has started you can adjust the view of the 3D model in the 3D Viewer control by means of commands. You can rotate, move or zoom the scene (camera settings) and switch to defined viewpoints. You can also make the same adjustments to the scene before starting the simulation by setting the *Operable* view property for the 3D Viewer control.

The standard view can be restored at any time by selecting the *Camera > Reset* menu command or by pressing the *Home* key.



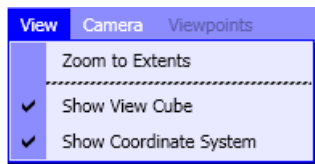
### Rotating the scene

You can rotate the scene manually using the mouse or keyboard. Move the mouse pointer over a point on the scene and press the left mouse button. The center of rotation, which is also the viewpoint as seen by the observer (camera), appears in the center of the scene. Hold down the mouse button and rotate the scene in the direction you want. Alternatively you can also rotate the scene vertically or horizontally using the arrow keys.



You can reset the center of rotation/viewpoint by double-clicking an element of the scene.

The view cube in the bottom right corner of the 3D Viewer control shows you the current viewing direction of the 3D scene. Click one side of the cube to reset the scene to the corresponding viewing plane. Double-click to set the opposite viewing plane. You can switch the view cube display on or off by selecting the *View > Show view cube* menu command.



You can also use the shortcut keys listed in the table below to switch the viewing plane.

Table 7-38 Shortcut keys for switching the viewing plane

Viewing plane	Abbreviation/acronym	Shortcut keys
Front view	F (Front)	Ctrl-F
Back view	B (Back)	Ctrl-B
Left view	L (Left)	Ctrl-L
Right view	R (Right)	Ctrl-R



Viewing plane	Abbreviation/acronym	Shortcut keys
Top view	T (Top)	Ctrl-T
Bottom view	B (Bottom)	Ctrl-B

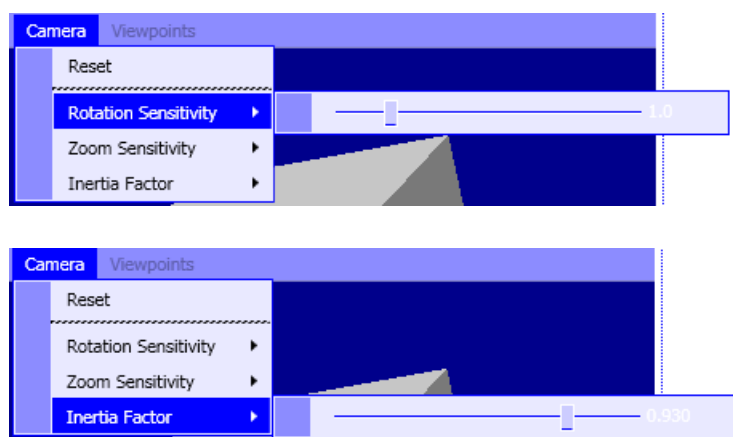
The bottom left corner of the 3D Viewer control shows the coordinate system with the three axes *X*, *Y* and *Z*. You can switch the coordinate system display on or off using the *View / Show coordinate system* menu command (see figure below).

### Note

The viewing planes are defined in the VRML specification: a Cartesian, left-handed, three-dimensional coordinate system is used in which the positive *Y*-axis points upwards and the viewer looks from the positive *Z*-axis towards the negative *Z*-axis.

When you create the 3D model or export it to a VRML file, make sure that the *Y* axis is pointing upwards in accordance with the VRML specification.

You can influence how the scene responds to being rotated with the mouse or arrow keys by selecting the *Camera > Rotation sensitivity* and *Camera > Inertia* menu commands. The sensitivity and inertia factor can be reduced or increased by means of a slider. The default setting for both sliders is one.



### Swiveling the camera

By panning the camera you can move the scene in the window of the 3D Viewer control, which means move the scene away from or into the center of the window. In contrast to a rotation, if you pan the camera the position of the viewer (camera) remains constant, only the viewpoint (target) changes.

The operating instructions for panning the scene are the same as for rotating, except that you also need to hold down the shift key. You can find information about rotating the scene in the section: Rotating the scene (Page 456).

### Zooming the scene

In order to view parts of a scene in more detail you can zoom in or out as required using the Page Up and Page Down keys. The same effect is achieved if you turn the mouse wheel or drag with the left mouse button while holding down the Ctrl key. If you press the Ctrl and shift key at the same time and hold down the left mouse button, you can draw an area to be zoomed.

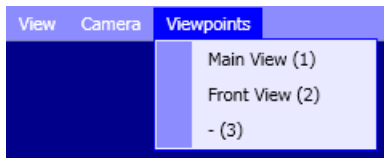
Using the *View > Adjust zoom* menu command, you can adjust the zoom at any time so that the complete scene fills the 3D viewer control. All other camera settings remain unchanged.

You can influence how the scene responds to zooming by selecting the *Camera > Zoom sensitivity* and *Camera > Inertia* menu commands. The sensitivity and inertia factor can be reduced or increased by means of a slider. The default setting for both sliders is one.



### Switching the viewpoint

A viewpoint describes a particular viewing position. All viewpoints defined in the VRML file are listed in the 3D Viewer control in the *Viewpoints* menu with their name and a sequential number. If a viewpoint does not have a name, it is listed with the identifier "-". You can switch to a different viewpoint by selecting it in this menu.



## 7.2 The FLOWNET library

### 7.2.1 Introduction

The FLOWNET library is an extension of SIMIT, which provides component types for creating simulations of pipeline networks. By connecting components in this library, a model of a pipeline network, a flownet, which simulates the thermodynamic processes in pipeline networks is created in SIMIT. In conjunction with the FLOWNET library, a special solution method can be applied in SIMIT that calculates flows, pressures and specific enthalpies in the pipeline network simulation.

Although the SIMIT flownets are based on a modeling approach that employs physical balance equations, the aim is not to use the dynamic process simulations to facilitate the design of system components or systems, but instead to provide a physically plausible simulation of the thermodynamic variables in pipeline networks for virtual commissioning. This simulation should

be easy to create from components in a graphical interface and should be stable even in extreme situations. During implementation of the component types in the FLOWNET library, the focus was on simple parameter assignment of the components and stable behavior in the flownet, rather than a detailed simulation of the physics.

The component types in the FLOWNET library can be used to create flownets for a variety of media:

- water/steam,
- liquids or
- ideal gases.

You can use the Component Type Editor (CTE) module of SIMIT to create your own flownet components and thus extend your flownet library. The flownet solution method can also be used by the components via FLOWNET-specific connection types.

---

**Note**

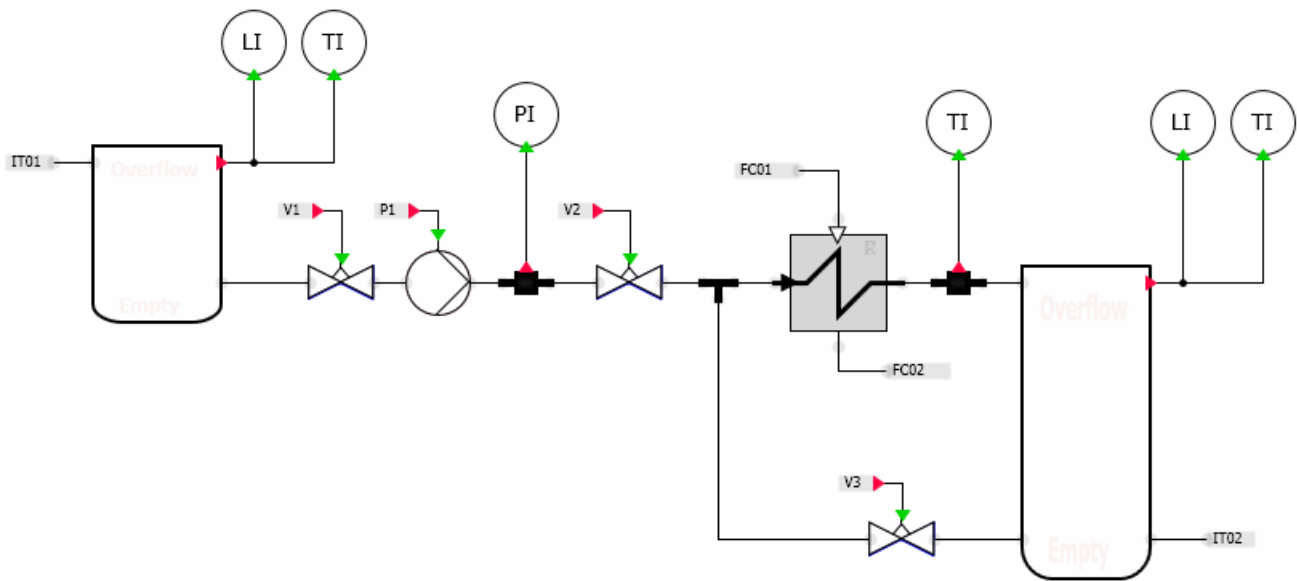
When the simulation starts, it will check whether your SIMIT installation has a license for the FLOWNET library. If the simulation project includes flownet components, which means components that use the solution method for flownets, the simulation can only be started if you have a FLOWNET license.

---

## 7.2.2 Flownets

A SIMIT flownet is an interconnection of flownet components used for simulation of thermodynamic processes in pipeline networks. The simulation of flownets is based on a special solution procedure, which is assigned parameters and configured using flownet components. The modeling approach described below limits flownets to homogenous media, but it can be used for liquids, ideal gases or water in either liquid or steam aggregate state.

The FLOWNET library provides component types that can be used to configure or model flownets. As usual in SIMIT, the chart editor is used to model flownets. The symbol used for the flownet component types, such as valves or pumps, is the same as the one normally used in pipeline diagrams. This means you can easily construct a flownet model, as seen in the figure below, in the form of a pipe diagram using the symbols of the component types:



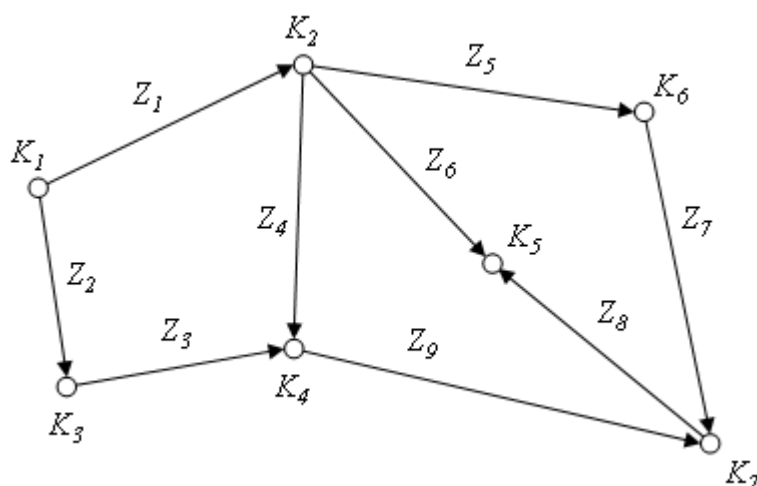
The flownet topology for configuring the flownet solution procedure is derived from the interconnection of flownet components. While the simulation is running, the flownet solution procedure and the flownet components exchange data: calculated values or flownet parameters.

### 7.2.2.1 Flownet basics

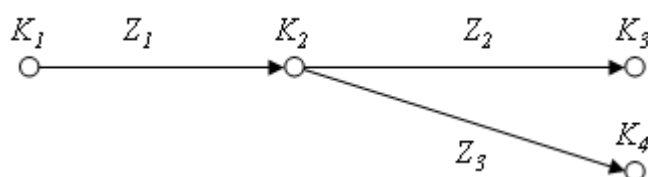
The process for simulating pipeline networks is based on mapping the connection of flownet components to flownets as a graph of nodes and branches. The branches model the flow paths and the nodes model the connections, which means the intersections or joints of the flow paths. The determining variables for the nodes are the pressures and specific enthalpies and the flow for the branches (mass flows).

All physical variables with a vector nature, such as the flow of fluids, are represented in the flownet as one-dimensional variables with direction information, which means as vector quantities (lines of flow). The direction is indicated by the sign. The variables can be numbered arbitrarily for reference.

Flownets can be depicted by graphs, whose branches (edges) form the pipelines with their fittings (valves, pumps, etc.), while their nodes form the pipe joints. The graph is directional, which means the direction of a branch indicates the direction of flow. The graph is also interconnected just like the flownet. The graph represents the topology of the flownet on the level of pipelines and joints. As an example, the following figure shows a graph with 7 nodes  $K_i$  and 9 branches  $Z_i$ .



A graph, as shown in the following figure, with three branches and four nodes results from the example in the section: Flownets (Page 459)



Nodes  $K_1$ ,  $K_3$  and  $K_4$  are used to set the boundary conditions for the flownet. For example, the pressure on the connections of both tanks is specified with these external nodes. In contrast, node  $K_2$  is an internal node, for which the relevant variables, such as pressure, are calculated using the flownet solution procedure.

The solution procedure for flownets is based on determining the flow in the branches depending on the pressure, using the momentum balance, and on balancing the flow of matter and enthalpy in the nodes. The state variables for such a system therefore are the mass flows in the branches and the pressures and specific enthalpies in the nodes. Other variables, such as density and temperature in the flownet, can be derived according to the medium in question.

If the flownet components in a branch change neither the rate nor the enthalpy of flow in that branch, the branch is removed from the flownet, which means both nodes are merged to form one node.

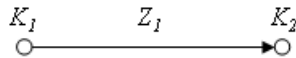
The solution procedure for flownets is a cyclical solution procedure with equidistant time slices that extends the standard solution procedure. Flownet component types can also be used in addition to other component types, such as those in the basic library. Specific connections are used for data exchange between flownet components and the flownet solution procedure. Through these connections, the components receive values calculated by the flownet solution

procedure; these are then used to calculate variables that are sent back to the flownet solution procedure.

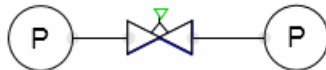
**Note**

Depending on the structure of the flownet and on the parameter assignment of the flownet components, the calculation of the flownet can become unstable during simulation and the values of the flownet state variables can grow beyond all limits as a result. The stability of flownets is not checked in SIMIT. In this case, you can establish stability by setting a smaller value for the cycle time of the flownet, and/or by modifying the parameters of the flownet balances.

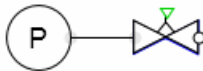
Obviously a flownet must consist of at least one branch with two nodes. The following figure shows the minimal graph.



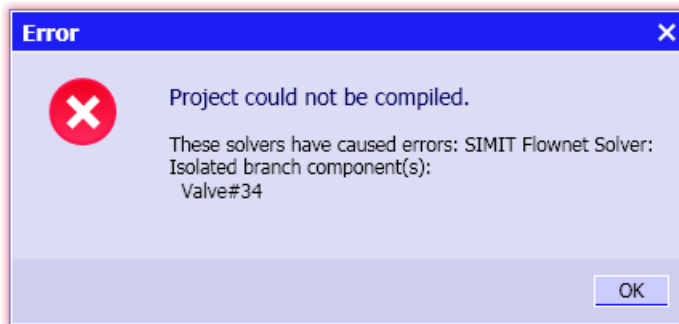
The figure below shows a corresponding minimal flownet. The nodes can be external (as in the figure) or internal.



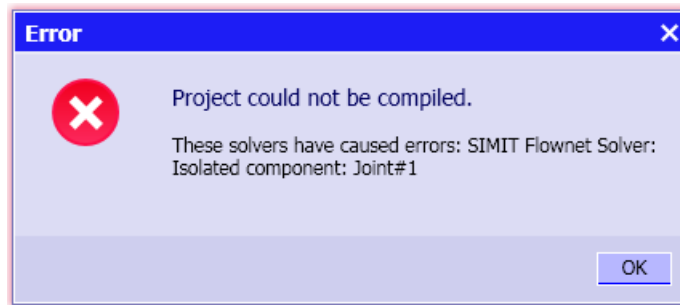
If a branch is not completed with two nodes, as shown in the figure below, an error message is output when you start the simulation and the simulation is not started.



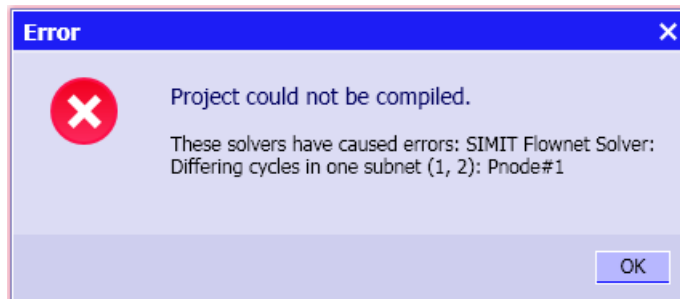
The figure below shows this error message, "*Isolated branch component(s)*", and lists the components in this branch.



A similar error message appears if the flownet only contains a single isolated component. The corresponding graph then consists of only one branch or node and does not meet the minimum requirements for a flownet.



The flownet components and the flownet solution procedure are also processed cyclically in SIMIT. The flownet components must be assigned to a time slice for this purpose. All components of a flownet must be assigned parameters with the same time slice. Otherwise the simulation start is canceled with an error message as the one in the figure below.



### 7.2.2.2 Variables used in flownets

The mass flows, pressures and specific enthalpies as well as the derived densities and temperatures are considered basic physical variables in flownets. These variables are listed in the following table along with the symbols and units used in this manual.

Table 7-39 Flownet variables

Variable	Symbol	Unit
Mass flow rate	$\dot{m}$	kg/s
Pressure	$p$	bar (absolute)
Specific enthalpy	$h$	kJ/kg
Density	$\rho$	kg/m <sup>3</sup>
Temperature	$T$	°C

### 7.2.2.3 Modeling of flownet branches

It is assumed that branches do not store any mass. The mass flow and density are therefore consistent throughout the entire branch.

These assumptions are always met for incompressible media. For compressible media no mass is stored in the branch if the branches have "no" volume. The density decreases for compressible media in the direction of falling pressure, which means in the direction of flow;

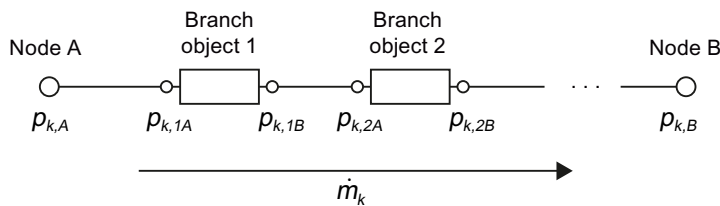
the density change is negligible for slight throttling, so it is acceptable to regard the density as constant.

Thus only the pressures at the connection points need to be considered for branch objects. The relationships between the pressures on either side of a branch object and the mass flow in the branch, such as  $\rho \Delta p \sim \dot{m}^2$ , are purely of an algebraic nature.

Momentum balance is applied to each branch  $k$  using pressure forces. Friction forces, acceleration forces and gravity are ignored. With a uniform cross-section  $A_k$  in a branch of length  $L_k$ , the following applies

$$L_k \frac{d\dot{m}_k}{dt} = A_k \Delta p_k - A_k \sum_K \Delta p_{k,K}$$

where  $\Delta p_k = p_{k,A} - p_{k,B}$  is the pressure difference over the entire branch  $k$ , and  $\Delta p_{K,k} = p_{K,kA} - p_{K,kB}$  are the pressure differences of the individual branch elements (see the figure below).



If pressure is applied in bars, the rate of change of the mass flow  $\dot{m}_k$  is given by

$$\frac{d\dot{m}_k}{dt} = 10^5 \frac{A_k}{L_k} \left( \frac{\Delta p_k}{\text{bar}} - \sum_K \frac{\Delta p_{k,K}}{\text{bar}} \right) \frac{\text{kg}}{\text{ms}^2}$$

The length and cross-section of a branch are generally unknown so a reasonable estimate must be made. Assuming for example a cross-section of 0.05 m<sup>2</sup> and a length of 10 m results in a factor that is subsequently described as the momentum factor

$$A_k = 10^5 \frac{A_k}{L_k}$$

of 500m.

### 7.2.2.4 Modeling of flownet nodes

The inflow and outflow of the medium are dynamically balanced at each node. Each node is assigned a material balance envelope, which means a volume. The mass inflow and outflow are balanced as well as the enthalpy inflow and outflow as a measure of the energy conversion.



### Mass balance for the node

The pressure in a node  $i$  is determined through the mass balance, which means through the balancing of the inflow and outflow from the branches that are connected with the node:

$$V_i \frac{d\rho_i}{dt} = \sum_{\kappa} \dot{m}_{\kappa}$$

$V_i$  is the volume of the material balance envelope assigned to the node,  $\rho_i$  is the density of the medium within the material balance envelope and the  $\dot{m}_{\kappa}$  are the inflows and outflows. Inflows are positive ( $\dot{m}_{\kappa} > 0$ ), outflows are negative ( $\dot{m}_{\kappa} < 0$ ).

Using the equation of state  $\rho = \rho(p, h)$  and assuming an isenthalpic change of state for the pressures in the node, the following applies:

$$\frac{d\rho_i}{dt} = \left( V_i \frac{d\rho_i}{d\rho_i} \Big|_{h=\text{const}} \right)^{-1} \sum_{\kappa} \dot{m}_{\kappa}$$

The term

$$V_i \frac{d\rho_i}{d\rho_i}$$

is a measure of the compressibility of the medium. Using the compressive modulus  $K_i$

$$K_i = M_i \left( V_i \frac{d\rho_i}{d\rho_i} \right)^{-1}$$

the following applies for the mass balance:

$$\frac{d\rho_i}{dt} = \frac{K_i}{M_i} \sum_{\kappa} \dot{m}_{\kappa} = c_i \sum_{\kappa} \dot{m}_{\kappa}$$

The default setting for the specific compression modulus  $c_i = K_i / M_i$  in the flownet solution procedure is the same for all nodes in the flownet. However, various factors result in increased compressibility  $c_i$  of gases and steam in contrast to liquids, and this is taken into consideration for media with higher and lower densities. You can find additional information on this in the section: Parameter assignment of flownets (Page 467).

### Enthalpy balance for the node

The specific enthalpy is treated as another state variable of a node. In principle, the convective inflow of enthalpy and the existing enthalpy in the node form a mixed enthalpy, which applies to the outflow. The balancing for a node  $i$  is given by

$$\frac{d(h_i M_i)}{dt} = \sum_{\kappa} h_{\kappa} \dot{m}_{\kappa}$$

where  $M_i$  is the mass and  $h_i$  is the specific enthalpy of the medium within the material balance envelope. It follows from

$$\frac{dh_i}{dt} = \frac{1}{M_i} \sum_{k \in Z_i} \dot{m}_k (h_k - h_i)$$

that by using the difference in enthalpy only the inflows to the node  $i$  ( $k \in Z_i$ ) need to be summed.

Just like the mass balance factors  $c_i$ , the default thermal factors  $m_i = 1/M_i$  in the flownet solution method are equal for all nodes, but have different values for media with higher and lower densities.

### Determining the density of the medium in the nodes

The values for the density in the nodes are calculated from the pressure and specific enthalpy values. The relations used depend on the medium in the flownet.

#### Water/steam medium

In the case of water/steam, the density in the nodes is calculated using the equation of state for water/steam with pressure  $p$  and specific enthalpy  $h$ :

$$\rho = \rho(p, h).$$

#### Liquid medium

In the case of liquids, calculation is based on constant density throughout the entire flownet. The default is a density of 997.337 kg/m<sup>3</sup>.

#### Calculation

The gas equation  $pV = mR_s T$  is used for ideal gas. Using the specific heat capacity  $c_p$  the density is given by

$$\rho = \frac{m}{V} = \frac{p \cdot 10^2 \frac{\text{Pa}}{\text{bar}}}{R_s \left( \frac{h - h_0}{c_p} + T_0 \right)}$$

with pressure  $p$  in bar, the specific heat capacity  $c_p$  in kJ/kgK and the specific gas constant  $R_s$  in kJ/kgK. If the triple point of water is used for the zero point ( $T_0, h_0$ ), we can set  $T_0 = 273.15$  K and  $h_0 = 0$ . The density is then calculated using the following relationship

$$\rho = \frac{p \cdot 10^2 \frac{\text{Pa}}{\text{bar}}}{R_s \left( \frac{h}{c_p} + 273,15 \text{ K} \right)}$$

In the flownet solution procedure, the specific gas constant has a value of 0.287 kJ/kgK (specific gas constant for dry air).

### Determining the temperature of the medium in the nodes

The temperatures used in the FLOWNET library components are also calculated from the values for pressure  $p$  and specific enthalpy  $h$ . The equation of state for water/steam medium

$$T = T(p, h)$$

is used. For ideal gases and liquids the temperature is determined using the specific heat capacity  $c_p$  from the specific enthalpy in accordance with

$$T = h / c_p$$

.

#### 7.2.2.5 Heat exchange with the environment

Heat exchange with the environment is accounted for with a corresponding term in the enthalpy balance equation for the node:

$$\frac{dh_i}{dt} = \frac{1}{M_i} \left( \sum_{k \in Z_i} \dot{m}_k (h_k - h_i) - c_i (T_i - T_{Env}) \right)$$

. Where  $T_{Env}$  is the ambient temperature,  $T_i$  is the temperature of the node and

$$c_i = \alpha_i A_i$$

is the determining heat transfer factor, which is the product of the heat transfer coefficient  $\alpha_i$  and the heat transfer surface  $A_i$ .

#### 7.2.2.6 Parameter assignment of flownets

The variables for the branches and nodes of a flownet are calculated using the various parameters described above. Default values for these parameters are set in the flownet solution procedure but can be changed using special component types from the FLOWNET library.

You can use the following component types for parameter assignment of networks:

- *NetParam* for general parameter assignment of flownets (NetParam – network parameter assignment (Page 482)),
- *NetWS* for parameter assignment of networks with water/steam medium (NetWS – water/steam network parameter assignment (Page 493)),
- *NetLiquid* for parameter assignment of networks with liquid medium (NetLiquid – liquid network parameter assignment (Page 514)) and
- *NetGas* for parameter assignment of networks with gas medium (NetGas – gas network parameter assignment (Page 529)).

You can use the following component types for specific parameter assignment of individual nodes:

- *JointParam* for general parameter assignment of nodes (*JointParam* – parameterizable joint (Page 485)),
- *JointParamWS* for parameter assignment of a node in a network with water/steam medium (*JointParamWS* – water/steam parameterizable joint (Page 497)),
- *JointParamLiquid* for parameter assignment of a node in a network with liquid medium (*JointParamLiquid* – liquid parameterizable joint (Page 517)),
- *JointParamGas* for parameter assignment of a node in a network with ideal gas medium (*JointParamGas* – parameterizable joint (Page 532)).

The component type

- *BranchParam* (*BranchParam* – branch parameter assignment (Page 484))

can be used to assign parameters to individual branches.

The following sections describe the connection types with which you can use parameter assignments in your user-created flownet components:

- Connector type FLN5 for parameters of a flownet (Page 551)
- Connector type FLN6 for parameter assignment of a branch (Page 552)
- Connector type FLN7 for parameter assignment of an internal node (Page 553)

## Flownet media

The following media can be specified for a flownet:

- water/steam,
- ideal gas or
- liquid

Water/steam is the default medium.

## Parameters for branches

The momentum factor  $A$  can be set for flownet branches. The same factor  $A = 450\text{m}$  is used as default for all branches in the flownet solution method. If there are several momentum factors in a branch, the effective factor  $A$  in the branch is given by

$$\frac{1}{A} = \sum_k \frac{1}{A_k}$$

using the  $k$  factors  $A_k$ .

## Node parameters

The following variables can be parameterized for nodes in a flownet:

- Specific compression modulus  $c = K / M$
- Thermal factor  $m = 1 / M$

Both variables can be individually specified for water and steam as well as liquid and gas.

Transitions between the parameters  $c_L$  and  $m_L$  and the parameters  $c_G$  and  $m_G$  for water/steam medium are calculated by the flownet solution method based on the density according to the following scheme:

$$c = \begin{cases} c_L & \text{for } \rho \geq 500 \text{ m}^3/\text{kg} \\ c_G & \text{for } \rho < 500 \text{ m}^3/\text{kg} \end{cases}$$

and

$$m = \begin{cases} m_L & \text{for } \rho \geq 500 \text{ m}^3/\text{kg} \\ m_G & \text{for } \rho < 500 \text{ m}^3/\text{kg} \end{cases}$$

The transition between both parameter values can also be set with a linear transition function:

$$c = \begin{cases} c_L & \text{for } \rho > \rho_L \\ c_G & \text{for } \rho < \rho_G \\ c_L + (c_G - c_L) \frac{\rho_G(\rho_L - \rho)}{\rho(\rho_L - \rho_G)} & \text{otherwise} \end{cases}$$

and

$$m = \begin{cases} m_L & \text{for } \rho > \rho_L \\ m_G & \text{for } \rho < \rho_G \\ m_L + \frac{m_G - m_L}{v_G - v_L} (v - v_L) & \text{otherwise} \end{cases}$$

with the two reference values  $\rho_G = 5 \text{ kg/m}^3$  and  $\rho_L = 1000 \text{ kg/m}^3$ .

For liquid medium the corresponding variables  $c_L$  and  $m_L$  are always applicable, and for ideal gas the variables  $c_G$  and  $m_G$  are applicable.

For heat exchange with the environment, the ambient temperature  $T_{\text{env}}$  and the heat transfer factor  $c = \alpha A / M$  can be parameterized. The default temperature is set at  $T_{\text{env}} = 20^\circ\text{C}$  with factor  $c = 0$ , which means there is no heat exchange with the default settings.

## Parameters for liquid medium

If liquid is set as the medium for the flownet, constant density is assumed. The density value to be used can be specified as a parameter.

The specific heat capacity  $c_p$  of the medium can be specified as an additional variable for the flownet as well as specifically for each individual node. The default value is 4.18 kJ/kgK.

**Parameters for ideal gas medium**

For ideal gas as the flownet medium, the gas constant  $R_s$  and the specific heat capacity  $c_p$  can be specified. The default values are  $R_s = 0.287$  kJ/kgK and  $c_p = 1$ kJ/kgK.

**Initialization of variables**

Each time the simulation is started, the flownet state variables are initialized as follows:

All branch mass flow rates are initialized with a value of zero ( $\dot{m} = 0$ ). This initialization cannot be changed. The default values for the pressures in the nodes are  $p = 1$ bar, but they can be changed to other values by means of a parameter. The initial value  $h$  for the specific enthalpy depends on the medium in the flownet as follows:

- $h = 100$  kJ/kg for water/steam,
- $h = 83.6$  kJ/kg for liquid and
- $h = 20$  kJ/kg for ideal gas.

These initial values can be changed by means of a parameter.

**Parameter overview**

The table below provides an overview of the available flownet parameters. The signal column lists the names of the input or output signals for the component, the designation column gives the names under which the parameters can be found in the component properties dialog. The default values and units are shown in the last column. If explicit parameters are not set, the defaults for the flownet as described above apply.

Table 7-40 List of flownet parameters

Signal	Term	Description	Default	
			Value	Unit
MEDIUM	Medium	Characteristic number for the medium in the flownet: <ul style="list-style-type: none"> <li>• water/steam: 0</li> <li>• liquid: 2</li> <li>• ideal gas: 1</li> </ul>	0	-
CG	sCompressionGas	Specific compression modulus $c = K / M$ for ideal gas or steam	10	bar/kg
CL	sCompressionLiquid	Specific compression modulus $c = K / M$ for liquids	100	bar/kg
MG	FactorThermalGas	Thermal factor $m = 1 / M$ for ideal gas or steam	100	kg <sup>-1</sup>
ML	FactorThermalLiquid	Thermal factor $m = 1 / M$ for liquids	0.1	kg <sup>-1</sup>
P_INIT	PressureInit	Initial pressure value	1	bar
H_INIT	sEnthalpyInit	Initial value for specific enthalpy	20 / 83.6 / 100	kJ/kg

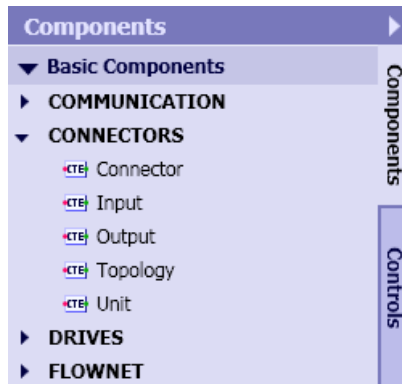
Signal	Term	Description	Default	
DENSITY	Density(Liquid)	Density (only applies to liquid)	997.337	kg/m <sup>3</sup>
T_ENV	TemperatureEnvironment	Ambient temperature $T_{env}$	20	°C
C_ENV	FactorHeatExchangeEnv	Heat transfer factor $c = \alpha A$	0	kW/K
L_CR	sHeatCapLiquid	Specific heat capacity $c_p$ for liquids	4.18	kJ/kgK
IG_R	GasConstant	Gas constant $R_g$	0.287	kJ/kgK
IG_CR	sHeatCapGas	Specific heat capacity $c_p$ for gas medium	1.0	kJ/kgK
ST	SmoothTransition	Switch to linear transition of the compression modulus and thermal factor for water/steam	False	
AL	FactorMomentum	Momentum factor A	450	m

## 7.2.3 Components in FLOWNET library

### 7.2.3.1 The topological connector in the FLOWNET library

A connector is available in the *CONNECTORS* directory of the SIMIT basic library that can be used to create topological connections for flownet components across chart limits:

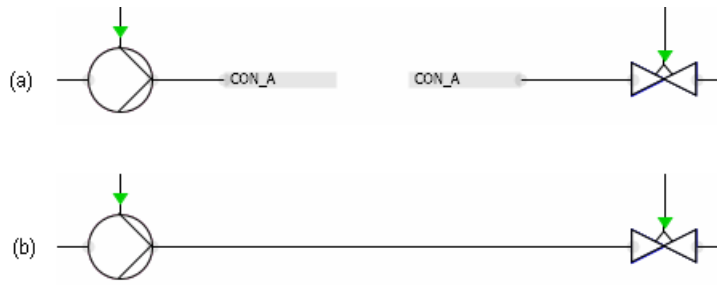
- the *Topology* connector



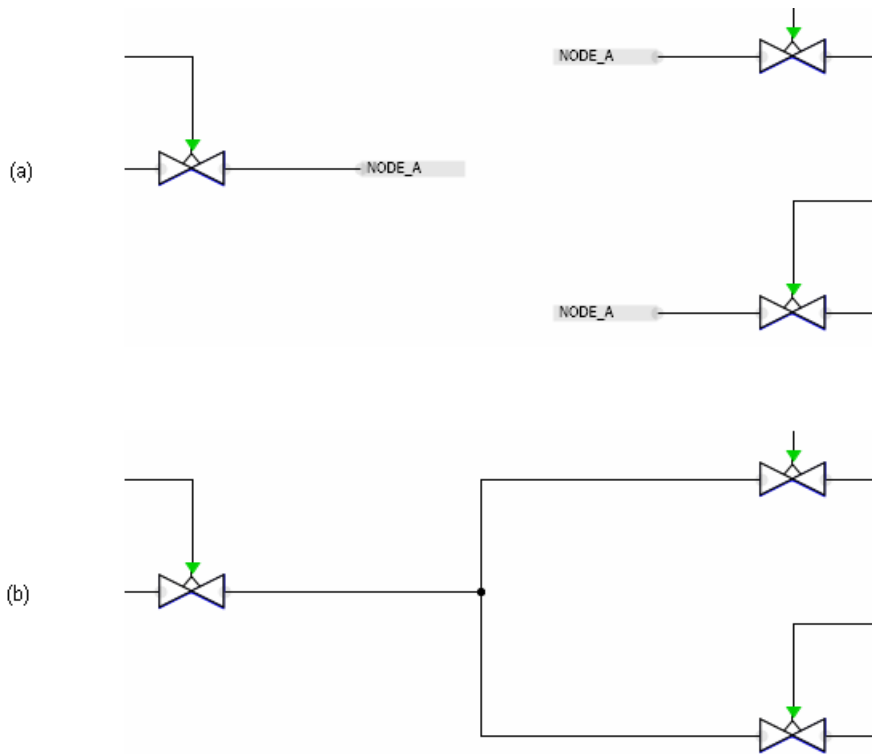
The *Topology* symbol is shown in the figure below:



Use the *Topology* connector to create a topological flownet connection between two or more branch components. Two components are connected by the *CON\_A* connector under (a) in the figure below. The connection is functionally identical to the direct connection of both components via a connecting line, as shown under (b) in the figure below:



Three components are connected by the *NODE\_A* connector under (a) in the figure below. This configuration is functionally identical to the connection using a connection node that is shown under (b) in the figure below.

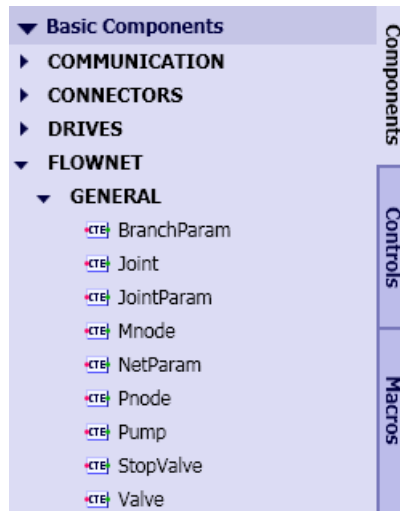




### 7.2.3.2 General components

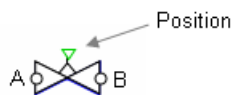
#### Selection of the general components

The *GENERAL* directory of the FLOWNET library includes component types that can be used in flownets with any medium.



#### Valve

#### Symbol



#### Function

The *Valve* component type is used for simulating a control valve. Depending on the control valve position the pressure drop over the control valve is calculated as

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_V^2 \rho \frac{\text{kg}}{\text{m}^3}} 12960 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

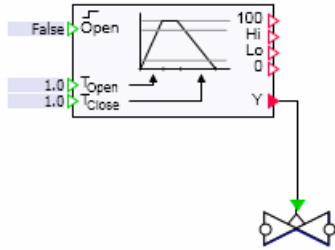
. The following applies:

- $\Delta = p_B - p_A$  is the pressure drop over the valve in bar
- $\dot{m}$  is the mass flow in kg/s

- $\rho$  is the density of the medium in  $\text{kg/m}^3$
- $c_v$  is the flow coefficient in  $\text{m}^3/\text{h}$

The reference direction is defined for the mass flow  $\dot{m}$  from connector *A* to connector *B*, which means it is  $\dot{m} > 0$  for mass flow in the reference direction. The pressure drop is therefore a negative value:  $\Delta p < 0$ .

At the *Position* connector, the *H* position of the valve drive is given as a percentage value for which the drive component types of the SIMIT basic library can be used.



The position value is limited to the range  $0 \leq H \leq 100\%$ . The control valve position is mapped using the control valve characteristic to the flow coefficient  $c_v$ . The following applies

$$\frac{k_v}{k_{v100}} = \frac{1}{S_v} + \left(1 - \frac{1}{S_v}\right) \frac{H}{100\%}$$

for a linear characteristic,

$$\frac{k_v}{k_{v100}} = \frac{1}{S_v} + \left(1 - \frac{1}{S_v}\right) \left(\frac{H}{100\%}\right)^2$$

for a quadratic characteristic,

$$\frac{k_v}{k_{v100}} = S_v \left(\frac{H}{100\%}\right)^{-1}$$

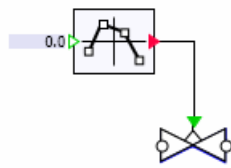
for an equal percentage characteristic.

The position ratio

$$S_v = \frac{k_{v100}}{k_{v0}}$$

is applied here as a quotient consisting of the flow coefficient  $c_{V100}$  for a completely open value ( $H = 100\%$ ) and the flow coefficient  $c_{v0}$  for a completely closed valve ( $H = 0$ ).

Any valve characteristic curves can be created with the *Characteristic* component type from the SIMIT standard library. The characteristic component is simply placed upstream of the valve *Position* input.



The linear characteristic is then configured at the control valve so that the characteristic specified with the characteristic component is simply scaled with the factor  $(c_{V100} - c_{V0}) / c_{V0}$  and offset by  $c_{V0}$ .

### Parameters

The valve characteristic curve and flow coefficient can be adjusted by means of parameters:

- *Characteristic*  
Configured valve characteristic curve: linear, quadratic, equal percentage; can be adjusted online
- *Cvs*  
Flow coefficient  $c_{V100}$  with  $c_{V100} \geq c_{V0} + 10^{-6} \text{m}^3/\text{h}$ ; can be adjusted online
- *Cv0*  
Flow coefficient  $c_{V0}$  with  $c_{V0} \geq 10^{-6} \text{m}^3/\text{h}$

The parameters with their units and default values are shown in the figure below.

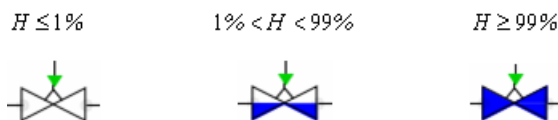
Name	Value
Characteristic	Linear
Cvs [m <sup>3</sup> /h]	360.0
Cv0 [m <sup>3</sup> /h]	0.000001

### Additional parameters

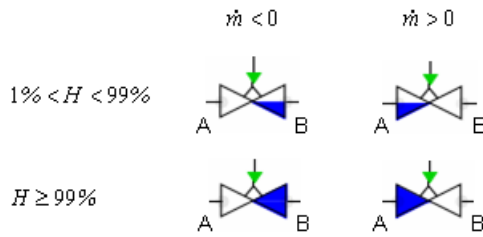
Additional parameters can be used to visualize the operating states of the components in the symbol.

Name	Value
ShowFlow	False
ShowFlowDirection	False

When *ShowFlow* is set to *True* the valve position is displayed as in the figure below.



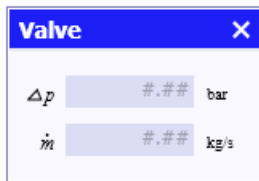
If *ShowFlowDirection* is also set to true, the valve position and the direction of flow is displayed as in the figure below.



Both additional parameters can be adjusted online.

**Operating window**

The pressure drop  $\Delta p$  and the flow rate  $\dot{m}$  are shown in the operating window.



**StopValve – non-return valve**

**Symbol**



**Function**

The *StopValve* component type is used for simulating a non-return valve. Depending on the flow direction the pressure drop over the control valve is calculated as

$$\frac{\Delta p}{\text{bar}} = \begin{cases} \frac{-\dot{m}^2}{k_{V100}^2 \rho \frac{\text{kg}}{\text{m}^3}} 12960 \left(\frac{\text{sec}}{\text{h}}\right)^2 & \text{for } \dot{m} > 0 \\ \frac{-\dot{m}^2}{k_{V0}^2 \rho \frac{\text{kg}}{\text{m}^3}} 12960 \left(\frac{\text{sec}}{\text{h}}\right)^2 & \text{for } \dot{m} < 0 \end{cases}$$

. The following applies:

- $\Delta p = p_B - p_A$  is the pressure drop over the non-return valve in bar
- $\dot{m}$  is the mass flow in kg/s,
- $\rho$  is the density of the medium in kg/m<sup>3</sup> and
- $c_v$  is the flow coefficient in m<sup>3</sup>/h.

The reference direction is defined for the mass flow  $\dot{m}$  from connector *A* to connector *B*, which means it is  $\dot{m} > 0$  for mass flow in the reference direction.

## Parameters

Both flow coefficients for the valve can be set via parameters:

- *Cvs*  
Flow coefficient  $c_{v100}$  with  $c_{v100} \geq c_{v0} + 10^6 \text{ m}^3/\text{h}$
- *Cv0*  
Flow coefficient  $c_{v0}$  with  $c_{v0} \geq 10^6 \text{ m}^3/\text{h}$

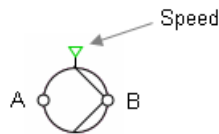
The parameters with their units and default values are shown in the figure below.

Name	Value
<i>Cvs</i>	[m <sup>3</sup> /h] 360.0
<i>Cv0</i>	[m <sup>3</sup> /h] 0.000001

In order to implement the non-return function, the flow coefficient  $c_{v0}$  must be sufficiently small.

## Pump – pump

### Symbol



### Function

The component type *Pump* calculates the pressure boost, which depends on the flow and speed according to

$$\Delta p = n^2 \Delta p_0 + (\Delta p^* - \Delta p_0) \frac{\dot{m}^2}{(\dot{m}^*)^2} \quad \text{for } \dot{m} > 0$$

Where

$\Delta p = p_B - p_A$  is the pressure boost in bar,

$\dot{m}$  is the mass flow in kg/s,

$\Delta p_0$  is the zero flow head in bar,

$\Delta p^*$  is the nominal pressure boost in bar,

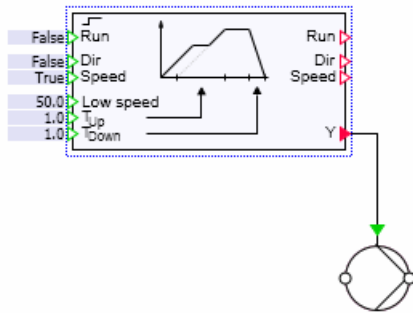
$\dot{m}^*$  is the nominal flow in kg/s and

$n$  is the dimensionless speed value.

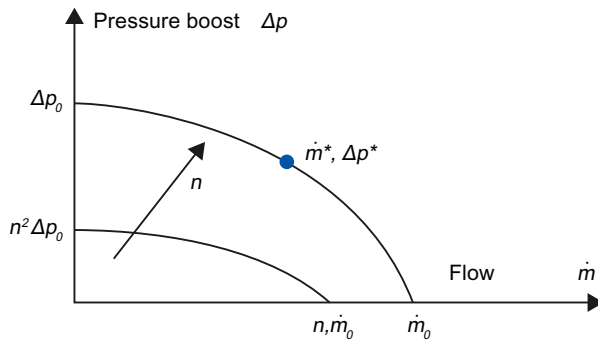
7.2 The FLOWNET library

The reference direction is defined for the flow  $\dot{m}$  from connector *A* to connector *B*, which means the following applies for a flow in reference direction  $\dot{m} > 0$ .

The speed *N* is specified as a percentage value at the *Speed* input. The speed value is limited to the range  $0 \leq N \leq 100\%$ . This means  $n = N / 100\%$  and thus  $0 \leq n \leq 1$ . Drives from the basic SIMIT library can for example be used here as shown in the figure below.



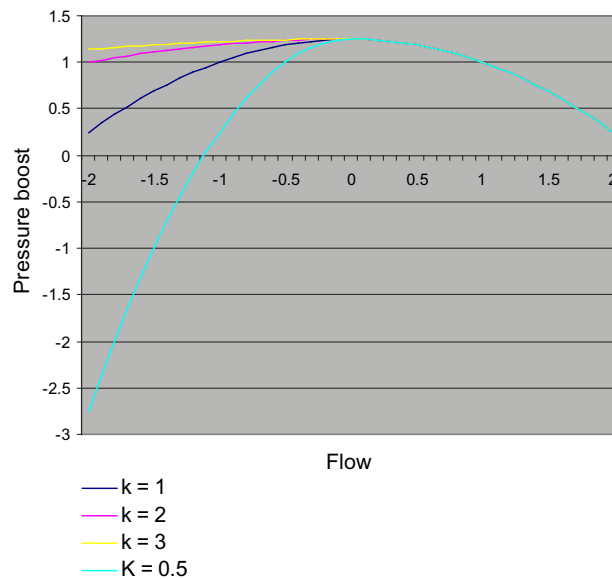
The quadratic relation between pressure boost and flow defined above is illustrated in the figure below for operation of the pump in the normal range, which means for  $\dot{m} > 0, \Delta p > 0$ .



The pump characteristic is increased steadily, according to

$$\Delta p = n^2 \Delta p_0 + (\Delta p^* - \Delta p_0) \frac{\dot{m}^2}{(k \dot{m}^*)^2} \quad \text{for } \dot{m} < 0$$

when the flow is reversed ( $\dot{m} < 0$ ). The throttling effect can be influenced by means of the flow coefficient *c*. The figure below shows the extended characteristic qualitatively for various characteristic values *c*.



It is clear that the throttling effect is greater for smaller values of the flow coefficient  $c$ .

To stabilize a simulation that contains a component of this type, pressure reversal ( $\Delta p < 0$ ) is also calculated using the above equations. For switched off pumps, which means for zero speed, this results in pure throttling for the flow in pump direction as well as for the flow against pump direction:

$$\Delta p = \begin{cases} (\Delta p^* - \Delta p_0) \frac{\dot{m}^2}{(k\dot{m}^*)^2} & \text{for } \dot{m} < 0 \\ (\Delta p^* - \Delta p_0) \frac{\dot{m}^2}{(\dot{m}^*)^2} & \text{for } \dot{m} > 0 \end{cases}$$

## Parameters

The determining variables for the pump characteristic are set with parameters:

- ZeroFlowHead  
Zero flow head  $\Delta p_0$ ,  $\Delta p_0 > \Delta p^*$ ; can be adjusted online
- NominalPressure  
Nominal pressure  $\Delta p^*$ ,  $\Delta p^* > 0$ ; can be adjusted online
- NominalMassflow  
Nominal mass flow  $\dot{m}^*$ ,  $\dot{m}^* > 0$ ; can be adjusted online

The parameters with their units and default values are shown in the figure below.

Name	Value	
ZeroFlowHead	[bar]	10.0
NominalPressure	[bar]	8.0
NominalMassflow	[kg/s]	10.0

### Additional parameters

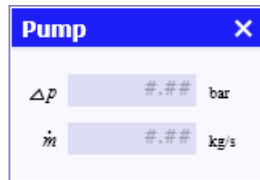
The additional parameter *Throttling* sets a positive flow coefficient  $c(c > 0)$ .

Name	Value
Throttling	1.0
ShowFlow	False

If the additional parameter *Showflow* is set to True, the operating status is displayed in the pump symbol.



The pressure boost  $\Delta p$  and the flow rate  $\dot{m}$  are displayed in the operating window.



### Pnode – pressure setting

#### Symbol



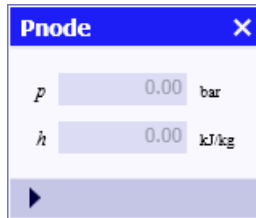
#### Function

The *Pnode* component type defines values for the pressure  $p$  and specific enthalpy  $h$  at its connector A. This type of component forms a boundary for the flownet. When the flownet is represented as a graph, it corresponds to an (external) node for which the pressure and specific enthalpy are predefined.



## Operating window

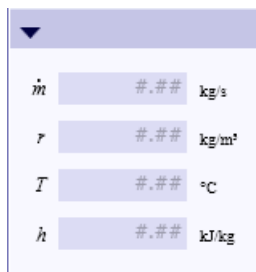
The values for pressure  $p$  and specific enthalpy  $h$  can be specified digitally in the operating window.



The inputs for these variables have the following default values:

- Pressure  $p = 1$  bar (*Pressure* input)
- Specific enthalpy  $h = 100$  kJ/kg (*sEnthalpy* input)

The corresponding values for the medium flowing into or out of the external node are shown in the extended operating window: mass flow  $\dot{m}$ , density  $r$ , temperature  $T$  and specific enthalpy  $h$ .



For outflow  $\dot{m} > 0$ , for inflow  $\dot{m} < 0$ .

## Mnode – mass flow setting

### Symbol



### Function

The *Mnode* component type defines values for the mass flow  $\dot{m}$  and specific enthalpy  $h$  at its connector A. This type of component forms a boundary for the flownet. When the flownet is represented as a graph, it corresponds to an inflow or outflow through an (internal) node or branch. An internal node with defined inflow or outflow is added to the flownet for each component of this type.

**Operating window**

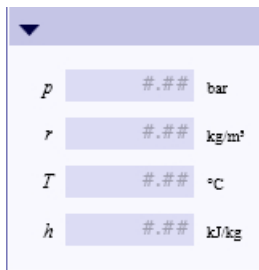
The values for mass flow  $\dot{m}$  and specific enthalpy  $h$  can be specified digitally in the operating window.



The inputs for these variables have the following default values:

- Mass flow  $\dot{m} = 0$  (*Massflow* input)
- Specific enthalpy  $h = 100$  kJ/kg (*sEnthalpy* input)

The variables of the assigned internal node are displayed in the extended operating window: pressure  $p$ , density  $r$ , temperature  $T$  and specific enthalpy  $h$ .



**NetParam – network parameter assignment**

**Symbol**



**Function**


The *NetParam* component type is used for parameter assignment of a flownet. The components can be added at any point on any branch of the flownet.

## Parameters

The variables for the flownet can be specified as parameters for the components:

- *Medium*  
The following can be selected as medium for the flownet: "Water/Steam", "Liquid", or "Ideal Gas".
- *FactorMomentum*  
Momentum factor for the flow through the branches of the flownet
- *sCompressionGas*  
Specific compression modulus for the "Water/Steam" medium with density  $\rho < 500 \text{ kg/m}^3$  or the "Ideal Gas" medium
- *sCompressionLiquid*  
Specific compression modulus for the "Water/Steam" medium with density  $\rho < 500 \text{ kg/m}^3$  or the "Liquid" medium
- *FactorThermalGas*  
Enthalpy balancing factor for the "Water/Steam" medium with density  $\rho < 500 \text{ kg/m}^3$  or the "Ideal Gas" medium
- *FactorThermalLiquid*  
Enthalpy balancing factor for the "Water/Steam" medium with density  $\rho < 500 \text{ kg/m}^3$  or the "Liquid" medium
- *DensityLiquid*  
Medium density; only applies to the "Liquid" medium
- *sHeatCapGas*  
Specific heat capacity for gas, only applies to the "Ideal Gas" medium
- *sHeatCapLiquid*  
Specific heat capacity for liquids, only applies to the "Liquid" medium
- *GasConstant*  
Specific gas constant for the medium, only applies to the "Ideal Gas" medium

The parameters with their units and default values are shown in the figure below:

Name	Value
Medium	Water/Steam 
FactorMomentum [m]	450.0
sCompressionGas [bar/kg]	10.0
sCompressionLiquid [bar/kg]	100.0
FactorThermalGas [1/kg]	100.0
FactorThermalLiquid [1/kg]	0.1
DensityLiquid [kg/m³]	997.337
sHeatCapGas [kJ/kgK]	1.0
sHeatCapLiquid [kJ/kgK]	4.18
GasConstant [kJ/kgK]	0.287

### Additional parameters

Initial values and specific characteristics for internal nodes of the flownet can be set by means of additional parameters:

- *PressureInit*  
Initialization value for the pressure in the internal nodes of the flownet
- *sEnthalpyInit*  
Initialization value for the specific enthalpy in the internal nodes of the flownet
- *SmoothTransition*  
When this additional parameter is set to True, the variables *sCompression* and *FactorThermal* are set with a density dependent linear transfer function
- *TemperatureEnvironment*  
Ambient temperature
- *FactorHeatExchangeEnv*  
Proportionality factor for the heat exchange of the medium in the internal nodes with the environment; for a value of zero, no heat exchange occurs

The additional parameters, their units and default values are shown in the figure below:

Name	Value
PressureInit	[bar] 1.0
sEnthalpyInit	[kJ/kg] 100.0
SmoothTransition	False
TemperatureEnvironment	[°C] 20.0
FactorHeatExchangeEnv	[kW/K] 0.0

### BranchParam – branch parameter assignment

#### Symbol



#### Function

The *BranchParam* component type is used for parameter assignment of the branches in a flownet. The components are placed anywhere along the branch to be assigned parameters.

#### Parameters

The momentum balance factor can be defined in the branch that is to be assigned parameters.

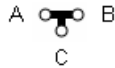
- *FactorMomentum*  
Momentum factor for the flow through the branches of the flownet

The default value for the parameter is shown in the figure below.

Name	Value
FactorMomentum	[m] 450.0

## Joint – joint

### Symbol

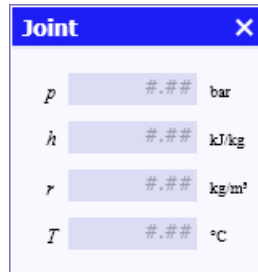


### Function

The component type *Joint* can be used to join three branches connected at *A*, *B* and *C* in one node. The *Joint* component adds an internal node to the flownet.

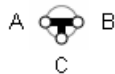
### Operating window

The pressure  $p$ , specific enthalpy  $h$ , density  $r$  and the temperature  $T$  of the node are displayed in the operating window.



## JointParam – parameterizable joint

### Symbol



### Function

The component type *JointParam* can be used to join three branches connected at *A*, *B* and *C* in one node. The *JointParam* component adds an internal node to the flownet.

## Parameters

The following variables of the node associated with *JointParam* can be assigned parameters:

- *sCompressionGas*  
Specific compression modulus for low density media (gas/steam,  $\rho < 500 \text{ kg/m}^3$ )
- *sCompressionLiquid*  
Specific compression modulus for high density media (liquid,  $\rho < 500 \text{ kg/m}^3$ )
- *FactorThermalGas*  
Factor for enthalpy balancing for low density media (gas/steam,  $\rho < 500 \text{ kg/m}^3$ )
- *FactorThermalLiquid*  
Factor for enthalpy balancing for high density media (liquid,  $\rho < 500 \text{ kg/m}^3$ )

The parameters with their units and default values are shown in the figure below.

Name	Value	Value
sCompressionGas	[bar/kg]	10.0
sCompressionLiquid	[bar/kg]	100.0
FactorThermalGas	[1/kg]	100.0
FactorThermalLiquid	[1/kg]	0.1

## Additional parameters

Initial values and specific characteristics for the node can be set by means of additional parameters:

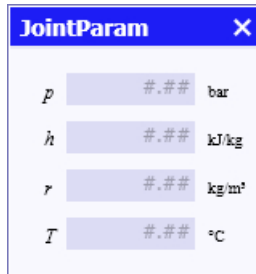
- *PressureInit*  
Initialization value for the pressure in the node
- *sEnthalpyInit*  
Initialization value for the specific enthalpy in the node
- *TemperatureEnvironment*  
Ambient temperature
- *FactorHeatExchangeEnv*  
Proportionality factor for the heat exchange of the medium in the node with the environment; for a value of zero, no heat exchange occurs

The additional parameters, their units and default values are shown in the figure below.

Name	Value	Value
PressureInit	[bar]	1.0
sEnthalpyInit	[kJ/kg]	100.0
TemperatureEnvironment	[°C]	20.0
FactorHeatExchangeEnv	[kW/K]	0.0

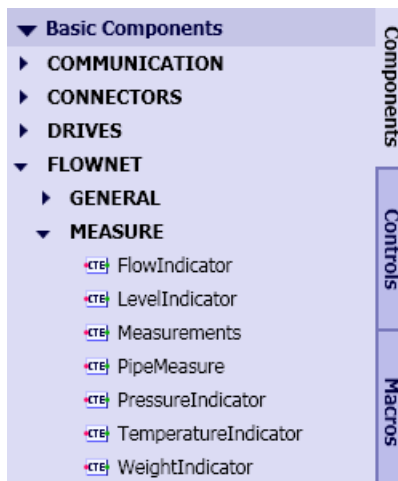
## Operating window

The pressure  $p$ , specific enthalpy  $h$ , density  $\rho$  and the temperature  $T$  of the node are displayed in the operating window.



### 7.2.3.3 Measuring components

Components for simulating measurements in pipeline networks can be found in the *MEASURE* directory.



Connectors of the *Measure* connection type are used in the measuring components. The meaning of the individual signals for this type are given in the following table.

Table 7-41 Signals of the Measure connection type

Signal	Meaning	Unit
Temperature	Temperature of measurement	°C
Pressure	Pressure of measurement	bar
Level	Measured level	m
Weight	Measured weight	kg
Flow	Measured flow	kg/s

### PipeMeasure – pipe measuring point

#### Symbol



#### Function

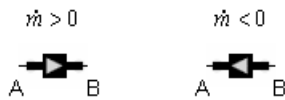
The *PipeMeasure* component type creates a measuring point in the pipe. It is inserted with its connectors *A* and *B* at the desired measuring point in the flownet. The measuring process for the various variables is not simulated with suitable models; the variables calculated by the flownet solution procedure are output.

The measurement variables are output at the *Measure* connection:

- absolute value  $|\dot{m}|$  of the flow rate value,
- Pressure  $p_A$  at connector *A* and
- Temperature *T*.

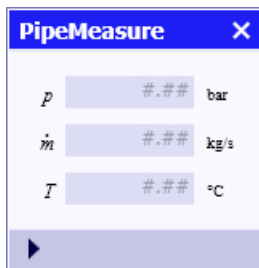
No other signals are set for the *Measure* connector.

When the simulation is running, the direction of the medium flow is indicated by an arrow on the symbol.



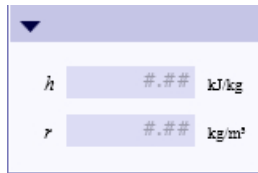
#### Operating window

The variables output via the *Measure* connector are also displayed in the operating window.



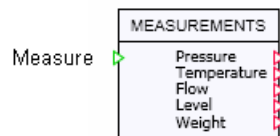
Values for the specific enthalpy and the density of the medium at the measuring point are displayed in the extended operating window.





## Measurements – measurement indicators

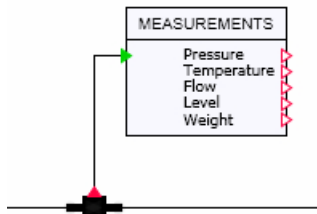
### Symbol



### Function

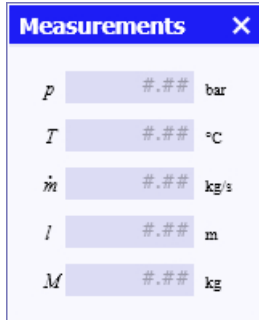
The *MeasureAll* component type provides the measured values bundled by the *Measure* input as individual signals at its outputs: pressure, temperature, flow, level and weight.

This type of component can be connected to the pipe measuring point, for example, and thereby output its measurement variables as individual signals.



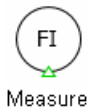
### Operating window

The output measurement variables are also displayed in the operating window for the component type.



### FlowIndicator – flow measurement indicator

#### Symbol



#### Function

The *FlowIndicator* component type displays the flow specified by its *Measure* input.

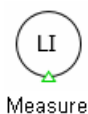
### Operating window

The flow rate value obtained via the *Measure* input is displayed in the operating window.



### LevelIndicator – level indicator

#### Symbol



## Function

The *LevelIndicator* component type displays the level that is specified by its *Measure* input.

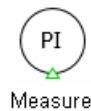
## Operating window

The level value obtained via the *Measure* input is displayed in the operating window.



## PressureIndicator – pressure indicator

### Symbol

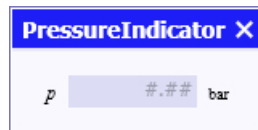


## Function

The *PressureIndicator* component type displays the pressure that is specified by its *Measure* input.

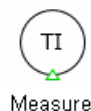
## Operating window

The pressure value obtained via the *Measure* input is displayed in the operating window.



## TemperatureIndicator – temperature indicator

### Symbol



**Function**

The *TemperatureIndicator* component type displays the temperature that is specified by its *Measure* input.

**Operating window**

The temperature value obtained via the *Measure* input is displayed in the operating window.



**WeightIndicator – weight indicator**

**Symbol**



**Function**

The *WeightIndicator* component type displays the weight that is specified by its *Measure* input.

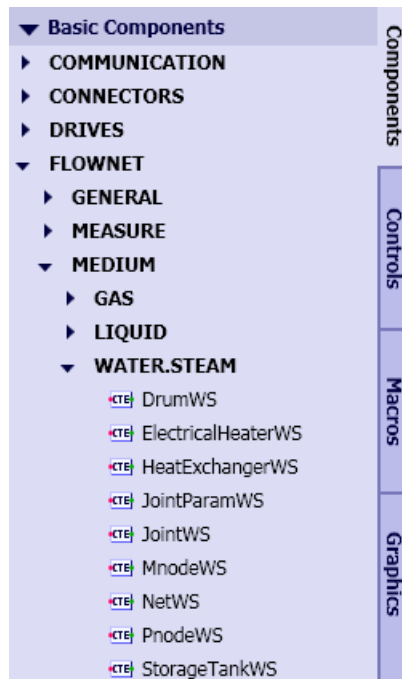
**Operating window**

The weight value obtained via the *Measure* input is displayed in the operating window.



### 7.2.3.4 Component types for "water/steam" medium

Component types that can be used in the flownet with the water/steam medium are located in the *MEDIUM.WATER.STEAM* folder of the FLOWNET library. The medium parameter for a flownet that contains these components must be set to the value "*Water/Steam*".



### NetWS – water/steam network parameter assignment

#### Symbol



#### Function

The *NetWS* component type is used to assign parameters to a network for the water/steam medium. The components can be added at any point on any branch of the flownet.

#### Parameters

The variables for the flownet can be specified as parameters for the components:

- *FactorMomentum*  
Momentum factor for the flow through the branches of the flownet
- *sCompressionSteam*  
Specific compression modulus for density  $\rho < 500 \text{ kg/m}^3$  (steam)

7.2 The FLOWNET library

- *sCompressionWater*  
Specific compression modulus for density  $\rho < 500 \text{ kg/m}^3$  (water)
- *FactorThermalSteam*  
Factor for enthalpy balancing for density  $\rho < 500 \text{ kg/m}^3$  (steam)
- *FactorThermalWater*  
Factor for enthalpy balancing for density  $\rho < 500 \text{ kg/m}^3$  (water)

The parameters with their units and default values are shown in the figure below:

Name	Value
FactorMomentum	[m] 450.0
sCompressionSteam	[bar/kg] 10.0
sCompressionWater	[bar/kg] 100.0
FactorThermalSteam	[1/kg] 100.0
FactorThermalWater	[1/kg] 0.1

**Additional parameters**

Initial values and specific characteristics for internal nodes of the flownet can be set by means of additional parameters:

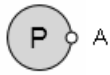
- *PressureInit*  
Initialization value for the pressure in the internal nodes of the flownet
- *sEnthalpyInit*  
Initialization value for the specific enthalpy in the internal nodes of the flownet
- *SmoothTransition*  
When this additional parameter is set to True, the variables *sCompression* and *FactorThermal* are set with a density dependent linear transfer function
- *TemperatureEnvironment*  
Ambient temperature
- *FactorHeatExchangeEnv*  
Proportionality factor for the heat exchange of the medium in the internal nodes with the environment; for a value of zero, no heat exchange occurs.

The additional parameters, their units and default values are shown in the figure below:

Name	Value
PressureInit	[bar] 1.0
sEnthalpyInit	[kJ/kg] 100.0
SmoothTransition	False <input type="checkbox"/>
TemperatureEnvironment	[°C] 20.0
FactorHeatExchangeEnv	[kW/K] 0.0

## PnodeWS – water/steam pressure setting

### Symbol

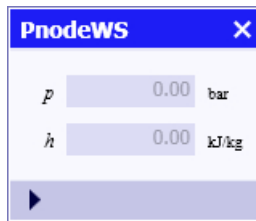


### Function

The *PnodeWS* component type defines values for the pressure  $p$  and specific enthalpy  $h$  at its connector A. This type of component forms a boundary for the flownet. When the flownet is represented as a graph, it corresponds to an (external) node for which the pressure and specific enthalpy are predefined.

### Operating window

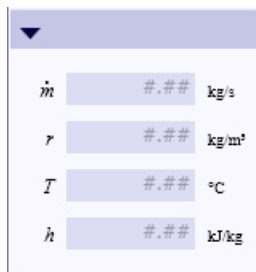
The values for pressure  $p$  and specific enthalpy  $h$  can be specified digitally in the operating window.



The inputs for these variables have the following default values:

- Pressure  $p = 1$  bar (*Pressure* input)
- Specific enthalpy  $h = 100$  kJ/kg (*sEnthalpy* input)

The corresponding values for the medium flowing into or out of the external node are shown in the extended operating window: mass flow  $\dot{m}$ , density  $r$ , temperature  $T$  and specific enthalpy  $h$ .



For outflow  $\dot{m} > 0$ , for inflow  $\dot{m} < 0$ .

### MnodeWS – water/steam mass flow setting

#### Symbol



#### Function

The *MnodeWS* component type defines values for the mass flow  $\dot{m}$  and specific enthalpy  $h$  at its connector A. This type of component forms a boundary for the flownet. When the flownet is represented as a graph, it corresponds to an inflow or outflow through an (internal) node or branch. An internal node with defined inflow or outflow is added to the flownet for each component of this type.

#### Operating window

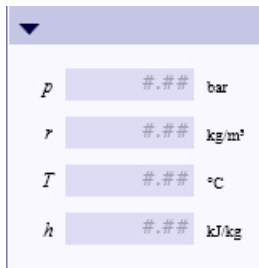
The values for mass flow  $\dot{m}$  and specific enthalpy  $h$  can be specified digitally in the operating window.



The inputs for these variables have the following default values:

- Mass flow  $\dot{m} = 0$  (*Massflow* input)
- Specific enthalpy  $h = 100$  kJ/kg (*sEnthalpy* input)

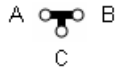
The variables of the assigned internal node are displayed in the extended operating window: pressure  $p$ , density  $r$ , temperature  $T$  and specific enthalpy  $h$ .





## JointWS – water/steam joint

### Symbol

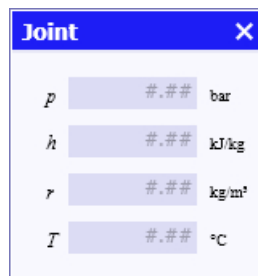


### Function

The component type *JointWS* can be used to join three branches connected at its connectors *A*, *B* and *C* in one node. The *JointWS* component adds an internal node to the flownet.

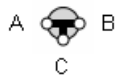
### Operating window

The pressure  $p$ , specific enthalpy  $h$ , density  $r$  and the temperature  $T$  of the node are displayed in the operating window.



## JointParamWS – water/steam parameterizable joint

### Symbol



### Function

The component type *JointParamWS* can be used to join three branches connected at its connectors *A*, *B* and *C* in one node. The *JointParamWS* component adds an internal node to the flownet.

## Parameters

The node associated with *JointParamWS* can be assigned parameters:

- *sCompressionSteam*  
Specific compression modulus for density  $\rho < 500 \text{ kg/m}^3$  (steam)
- *sCompressionWater*  
Specific compression modulus for density  $\rho < 500 \text{ kg/m}^3$  (water)
- *FactorThermalSteam*  
Factor for enthalpy balancing for density  $\rho < 500 \text{ kg/m}^3$  (steam)
- *FactorThermalWater*  
Factor for enthalpy balancing for density  $\rho < 500 \text{ kg/m}^3$  (water)

The parameters with their units and default values are shown in the figure below:

Name	Value	
sCompressionSteam	[bar/kg]	10.0
sCompressionWater	[bar/kg]	100.0
FactorThermalSteam	[1/kg]	100.0
FactorThermalWater	[1/kg]	0.1

## Additional parameters

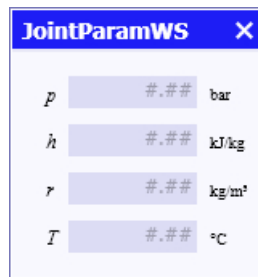
Initial values and specific characteristics for the node can be set by means of additional parameters:

- *PressureInit*  
Initialization value for the pressure in the node
- *sEnthalpyInit*  
Initialization value for the specific enthalpy in the node
- *TemperatureEnvironment*  
Ambient temperature
- *FactorHeatExchangeEnv*  
Proportionality factor for the heat exchange of the medium in the node with the environment; for a value of zero, no heat exchange occurs

The additional parameters, their units and default values are shown in the figure below:

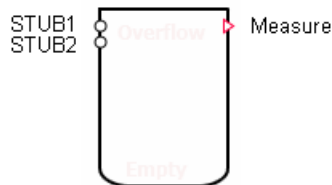
Name	Value	
PressureInit	[bar]	1.0
sEnthalpyInit	[kJ/kg]	100.0
TemperatureEnvironment	[°C]	20.0
FactorHeatExchangeEnv	[kW/K]	0.0

The pressure  $p$ , specific enthalpy  $h$ , density  $r$  and the temperature  $T$  of the node are displayed in the operating window.



## StorageTankWS – water/steam storage tank

### Symbol



### Function

The *StorageTankWS* component type provides the simulation with an open water tank, which means a tank that is not sealed from the environment.

Inflows and outflows occur via the *STUBx* connectors on the tank. For each of the *N* connectors, a throttling effect defined by

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_V^2 \rho \frac{\text{kg}}{\text{m}^3}} 12960 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

is assumed. The tank can have a minimum of one and a maximum of 16 connectors. The connectors can be moved to any position on the outline of the component symbol by using the mouse while simultaneously pressing ALT.

It is assumed that the water in the tank is immediately completely mixed, which means a homogenous medium with an overall consistent density and enthalpy in the tank.

The inflows and outflows of water are balanced across the *N* connectors of the tank. The balanced mass *M* of the *N* flow rates  $\dot{m}_i$  of the medium in the tank is thus defined by

$$\frac{dM}{dt} = \sum_{i=1}^N \dot{m}_i$$

The water volume balancing via the volume flows results in the rate of change of density  $\rho$  given by

$$\frac{d\rho}{dt} = \frac{\rho}{M} \left( \sum_{i=1, i \in Z}^N \dot{m}_i - \rho \sum_{i=1, i \in Z}^N \frac{1}{\rho_i} \dot{m}_i \right)$$

where only the inflows ( $i \in Z$ ) need to be summed.

The specific enthalpy of the water is determined from the enthalpy balance according to

$$\frac{dh}{dt} = \frac{1}{M} \left( \sum_{i=1, i \in Z}^N h_i \dot{m}_i - h \sum_{i=1, i \in Z}^N \frac{1}{\rho_i} \dot{m}_i \right)$$

. Here, too, only the inflows ( $i \in Z$ ) need to be summed.

The dynamic behavior of the water in the tank is described by these balances for mass  $M$ , density  $\rho$  and specific enthalpy  $h$ .

The calculated variables for the level  $l$  of water in the tank, its mass  $M$  and temperature  $T$  as well as pressure  $\rho_0$  on the tank base are output at the *Measure* connector. The temperature  $T$  is calculated using the equation of state

$$T = T(\rho, h)$$

from the density and specific enthalpy of the water. The pressure  $\rho$  on the tank base is calculated as the sum of the weight pressure  $\rho g l$  of the water and the atmospheric pressure  $\rho_U$  according to

$$\rho = \rho_U + \rho g l.$$

### Borderline case "empty tank"

When the tank is empty, the outflow is strongly throttled. The flow coefficient is thereby set to the value  $c_{v0}$  for maximum throttling of all connectors through which water flows out.

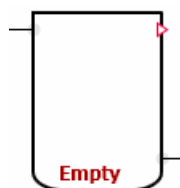
The tank is considered empty when the tank fill level  $V$  is lower than a specified minimum tank filling  $V_{min}$ :

$$V < V_{min}$$

When the tank is empty, balancing of the states is stopped, which means changes to the three state variables mass, density and specific enthalpy are rejected. The empty state continues until there is a sufficient increase in the fill level. In each increment the validity of

$$M \geq V_{min} \rho \left( 1 + \frac{H_{min}}{100\%} \right)$$

is checked. The required increase can be set via the hysteresis  $H_{min}$ . A corresponding indicator for an empty tank is shown in the component symbol.

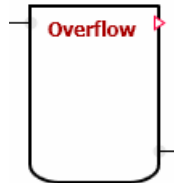


### Borderline case "full tank"

For a full tank, the balanced contents are limited in the calculation of the state variables according to:

$$M = V\rho$$

A corresponding indicator is shown in the symbol.




---

### Note

It is not checked whether the state variables (pressure, enthalpy) for the inflows and tank contents always have values for the water state.

In the simulation, inflows specified accordingly may result in a situation where the state values for the tank contents describe a water-steam mixture or pure steam, which has no physical meaning.

---

### Parameters

All relevant geometric variables of the tank are set via parameters:

- *Volume*  
Volume  $V$  of the tank; can be adjusted online
- *Height*  
Height of the tank; can be adjusted online
- *NbrOfStubs*  
Number  $N$  of connectors

The parameters with their units and default values are shown in the figure below:

Name		Value
Volume	[m <sup>3</sup> ]	10.0
Height	[m]	5.0
NbrOfStubs		1

### Additional parameters

The atmospheric pressure *PressureOutside* and initialization values for the tank fill level (*LevelInit*), the specific enthalpy (*EnthalpyInit*) and density (*DensityInit*) of the water are set via additional parameters:

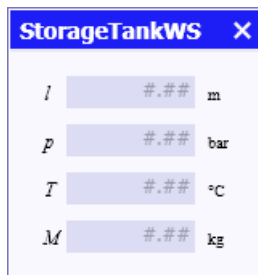
- *PressureOutside*  
Atmospheric pressure  $p_0$ ; can be adjusted online
- *LevelInit*  
Initialization value for the level
- *EnthalpyInit*  
Initialization value for the specific enthalpy in the contents
- *DensityInit*  
Initialization value for the density of the contents
- *Cvs*  
Uniform flow coefficient  $c_v$  for all connectors
- *Cv0*  
Flow coefficient  $c_{v0}$  for severe throttling in the tank connector
- *MinVolume*  
Minimum tank volume  $V_{min}$ ; can be adjusted online
- *MinVolumeHys*  
Hysteresis  $H_{min}$ ; can be adjusted online

The additional parameters, their units and default values are shown in the figure below:

Name	Value	
PressureOutside	[bar]	1.0
LevelInit	[%]	50.0
EnthalpyInit	[kJ/kg]	100.0
DensityInit	[kg/m <sup>3</sup> ]	997.337
Cvs	[m <sup>3</sup> /h]	360.0
Cv0	[m <sup>3</sup> /h]	0.000001
MinVolume	[m <sup>3</sup> ]	0.001
MinVolumeHys	[%]	50.0

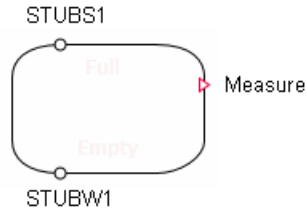
### Operating window

The level *l*, the pressure *p*, the temperature *T* and the mass *M* of the water in the tank is displayed in the operating window.



## DrumWS – water/steam drum

### Symbol



### Function

The *DrumWS* component type provides the simulation with a drum, which means a closed container for water/steam that is sealed off from the environment.

The drum is assumed to be a cylindrical, horizontal or vertically standing container, whose inflows and outflows occur via the *STUBWx* and/or *STUBSx* connectors. For each of the connectors, a throttling effect defined by

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_V^2 \rho \frac{\text{kg}}{\text{m}^3}} 12960 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

is assumed. At the  $N_W$  connectors *STUBWx*, medium with the state variables of the saturated water can be drawn out and medium with the state variables of saturated steam can be drawn out at the  $N_S$  connector *STUBSx*. The drum can have a minimum of one and a maximum of eight connectors of each type. The connectors can be moved to any position on the outline of the component symbol by using the mouse while simultaneously pressing ALT.

It is assumed that the medium in the drum immediately separates into two homogenous saturated phases: the saturated liquid phase and the saturated steam phase.

The inflows and outflows of water/steam are balanced via the  $N = N_W + N_S$  drum connectors. This results in a mass of water and steam in the drum, balanced via the mass flows, given by

$$\frac{dM}{dt} = \sum_{i=1}^N \dot{m}_i$$

or given

$$M = \rho V$$

the rate of change in the mean density results in

$$V \frac{d\rho}{dt} = \sum_{i=1}^N \dot{m}_i$$

In addition, the energy is balanced as per

$$\frac{d(hM)}{dt} = \sum_{i=1}^N h_i \dot{m}_i$$

Whereby  $h$  is the specific enthalpy (mean enthalpy) of water/steam in the drum and  $h_i$  is the specific enthalpy of the inflow and outflow at the  $i$ -th connector. The specific enthalpy of the medium in the drum must be applied for outflows, which means  $h'$  for outflows of saturated water and  $h''$  for outflows of saturated steam. The balancing is calculated as follows

$$M \frac{dh}{dt} = \sum_{i=1, i \in Z}^N (h_i - h) \dot{m}_i + (h' - h) \sum_{i=1, i \in A'}^N \dot{m}_i + (h'' - h) \sum_{i=1, i \in A''}^N \dot{m}_i$$

Whereby the inflows  $Z$ , outflows of saturated water  $A'$  and/or outflows of saturated steam  $A''$  are summed.

In order to model a heat exchange with an ideally insulated drum, the enthalpy balance equation is extended as follows

$$M \frac{dh}{dt} = \sum_{i=1, i \in Z}^N (h_i - h) \dot{m}_i + (h' - h) \sum_{i=1, i \in A'}^N \dot{m}_i + (h'' - h) \sum_{i=1, i \in A''}^N \dot{m}_i + A\alpha(T_T - T_S)$$

$T_T$  is the temperature of the drum wall,  $T_S$  is the saturation temperature of water/steam.

The heat storage in the drum wall is described by the heat balance

$$\frac{dT_T}{dt} = \frac{1}{M_T c_T} A\alpha(T_S - T_T)$$

The level of water  $l$ , the saturation temperature  $T_S$ , saturation pressure  $p_s$  and the mass  $M$  of water/steam are output at the *Measure* connector.

The STUBWx connectors are assumed to be located at the bottom of the drum. The pressure at these connectors therefore is the sum

$$p = \rho'gl + p_s$$

of the saturated steam and the hydrostatic pressure of the water.

### Borderline case "empty drum"

When the drum is empty, the outflow is strongly throttled. The flow coefficient is thereby set to the value  $c_{v0}$  for maximum throttling of all connectors through which water or steam flows out.

The drum is considered empty when the fill level  $M_{W/\rho}$  is lower than a specified minimum fill level  $V_{\min}$ :

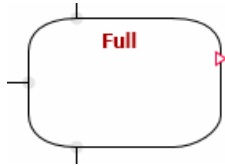
$$M_W < V_{\min} \rho.$$

When the drum is empty, balancing of the states is stopped, which means changes to the density and specific enthalpy are rejected. The empty state continues until there is a sufficient increase in the amount of water. In each increment the validity of



$$M_W \geq V_{min} \rho \left( 1 + \frac{H_{min}}{100\%} \right)$$

is checked. The required increase can be set via the hysteresis  $H_{min}$ . A corresponding indicator for an empty drum is shown in the component symbol.



### Borderline case "full drum"

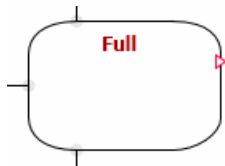
The drum is regarded as full when the amount of water reaches the maximum possible value:

$$M_W < (V - V_{min}) \rho.$$

When the drum is full, balancing of the states is stopped, which means changes to the density and specific enthalpy are rejected. The full state continues until there is a sufficient decrease in the amount of water. In each increment the validity of

$$M_W \leq \rho V - \rho V_{min} \left( 1 + \frac{H_{min}}{100\%} \right)$$

is checked. The required decrease can be set via the hysteresis  $H_{min}$ . A corresponding indicator for a full drum is shown in the component symbol.



### Parameters

All relevant geometric variables of the drum are set via parameters:

- *NbrOfStubsW*  
Number  $N_W$  the *STUBW*x connectors
- *NbrOfStubsS*  
Number  $N_S$  of *STUBS*x connectors
- *Position*  
Position of the drum: *Vertically* or *Horizontally*
- *VolumeDrum*  
Volume  $V$  of the drum; can be adjusted online
- *HeightOrLength*  
Height or length of the drum; can be adjusted online
- *MassDrum*  
Mass  $M_T$  of the drum; can be adjusted online

- *SurfaceDrum*  
Inner surface  $A$  of the drum; can be adjusted online
- *sHeatCapDrum*  
Specific heat capacity  $c_T$  of the drum; can be adjusted online
- *HeatTransCoe*  
Heat transfer coefficient  $\alpha$  for water/steam of the drum; can be adjusted online

The parameters with their units and default values are shown in the figure below:

Name	Value
NbrOfStubsW	1
NbrOfStubsS	1
Position	Vertically
VolumeDrum [m <sup>3</sup> ]	10.0
HeightOrLength [m]	5.0
MassDrum [kg]	5000.0
SurfaceDrum [m <sup>2</sup> ]	12.5
sHeatCapDrum [kJ/kgK]	0.5
HeatTransCoe [kW/m <sup>2</sup> K]	4.0

**Additional parameters**

Initialization values and other variables can be set via additional parameters:

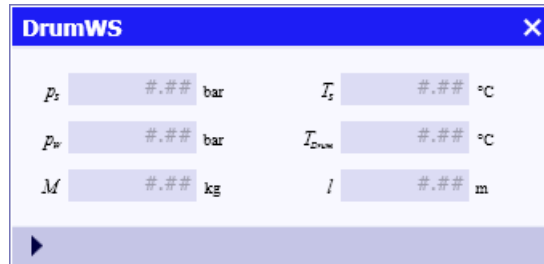
- *VolumeInit*  
Initialization value for the volume of water
- *TemperatureInit*  
Initialization value for the temperature of water/steam (saturation temperature)
- *Cvs*  
Uniform flow coefficient  $c_v$  for all connectors
- *Cv0*  
Flow coefficient  $c_{v0}$  for severe throttling in the drum connector
- *MinVolume*  
*Minimal drum* volume  $V_{min}$ ; can be adjusted online
- *MinVolumeHys*  
Hysteresis  $H_{min}$ ; can be adjusted online

The additional parameters, their units and default values are shown in the figure below.

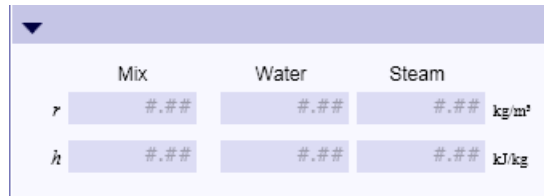
Name	Value
VolumeInit [%]	50.0
TemperatureInit [°C]	20.0
Cvs [m <sup>3</sup> /h]	360.0
Cv0 [m <sup>3</sup> /h]	0.000001
MinVolume [m <sup>3</sup> ]	0.01
MinVolumeHys [%]	50.0

### Operating window

The saturation pressure  $p_s$  and the pressure  $p_w$  at the STUBWx connectors are displayed in the operating window. The saturation temperature  $T_s$ , the drum temperature  $T_T$ , the mass  $M$  of water/steam and the water level  $l$  are also displayed.

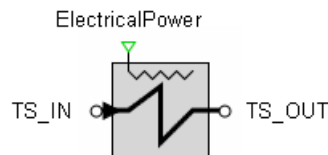


The values for density  $\rho$  and specific enthalpy  $h$  of water/steam are shown in the extended operating window. *Mix* indicates the mean variables, *Water* the variables for saturated water and *Steam* the variables for saturated steam.



## ElectricalHeaterWS – electrically heated heat exchanger for water/steam

### Symbol



### Function

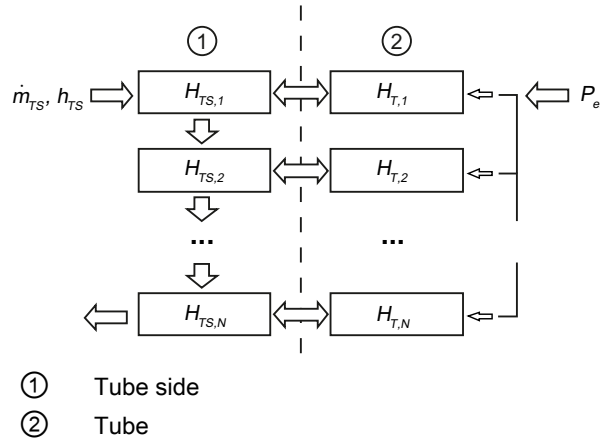
The *ElectricalHeaterWS* component type is used to simulate an electrically heated heat exchanger. The electric heating power  $P_{el}$  in kW is specified via the connector *ElectricalPower*.

Water/steam is directed via the *TS\_IN* and *TS\_OUT* connectors as a heated medium. The flow  $\dot{m}$  is throttled according to the relationship

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_V^2 \rho_{TS} \frac{\text{kg}}{\text{m}^3}} 12900 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

with throttle coefficient  $\alpha_V$ . The reference direction for the flow is chosen as from *TS\_IN* to *TS\_OUT*.

For the heat transfer, a simple one tube model divided into segments is applied. The number  $N$  of segments can be set to a value between 4 and 16.



It is assumed that the supplied electrical energy is completely converted into heat. The heat balance for a segment  $i$  of the spare tube with extremely simplified heat transfer is then given by

$$\frac{dT_{T,i}}{dt} = a_T P_{el} + b_T (T_{TS,i} - T_{T,i})$$

with

$$a_T = \frac{1}{M_T c_T}, \quad b_T = \frac{A_{TS} \alpha_{TS}}{M_T c_T}$$

In addition, the water/steam-side enthalpy balancing for a segment  $i$  is described by

$$\frac{dh_{TS,i}}{dt} = a_{TS} (h_{TS,i-1} - h_{TS,i}) + b_{TS} (T_{T,i} - T_{TS,i})$$

with

$$a_{TS} = \frac{N \dot{m}}{\rho_{TS} V_{TS}}, \quad b_{TS} = \frac{A_{TS} \alpha_{TS}}{\rho_{TS} V_{TS}}$$

$T_{T,i}$  and  $T_{TS,i}$  are the temperatures of the tube segments and of the medium in the segments, and  $h_{T,i}$  and  $h_{TS,i}$  are the corresponding specific enthalpies.

### Parameters

The segmentation and coefficients for the balance equations are defined by parameters:

- *NbrOfSegments*  
Number  $N$  of segments:  $4 \leq N \leq 16$
- *Cvs*  
Throttle coefficient  $c_v$

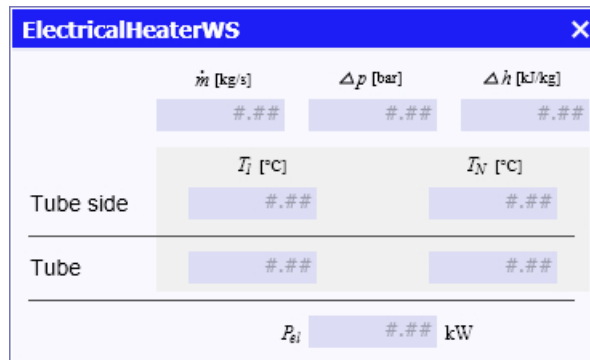
- *Volume*  
Volume  $V_{TS}$  of the medium (inner tube volume)
- *Surface*  
Surface  $A_{TS}$  of the tube on the water/steam side
- *HeatTransCoef*  
Heat transfer coefficient  $\alpha_{TS}$  of the tube to water/steam
- *sHeatCapTube*  
Specific heat capacity  $c_T$  of the tube
- *MassTube*  
Mass  $M_T$  of the tube

The parameters with their units and default values are shown in the figure below:

Name	Value
NbrOfSegments	4
Cvs [m³/h]	360.0
Volume [m³]	2.25
Surface [m²]	550.0
HeatTransCoef [kW/m²K]	4.0
sHeatCapTube [kJ/kgK]	1.3
MassTube [kg]	10000.0

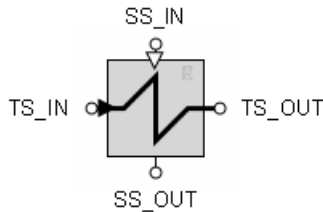
### Operating window

The flow rate  $\dot{m}$ , pressure drop  $\Delta p$  and enthalpy difference  $\Delta h$  of the heated medium as well as the electrical heating power  $P_{ei}$  are displayed in the operating window. The temperatures of the tube and of the heated medium (tube side) for the first and last segment ( $T_1$  and  $T_N$ ) are also displayed.



HeatExchangerWS – heat exchanger water/steam to water/steam

Symbol



Function

The *HeatExchangerWS* component type is used to simulate a heat exchanger for the media water/steam on the tube side and shell side. The simulation is implemented for the three types:

- Parallel-flow heat exchanger
- Counter-flow heat exchanger
- Cross-flow heat exchanger

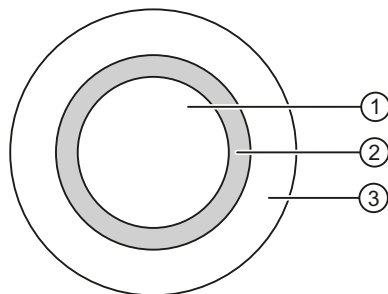
Both media are routed on the tube side via the connectors *TS\_IN* and *TS\_OUT*, and on the shell side via the connectors *SS\_IN* and *SS\_OUT*.

The flow  $\dot{m}$  is throttled on the tube side and shell side according to the relationship

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_v^2 \rho \frac{\text{kg}}{\text{m}^3}} 12900 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

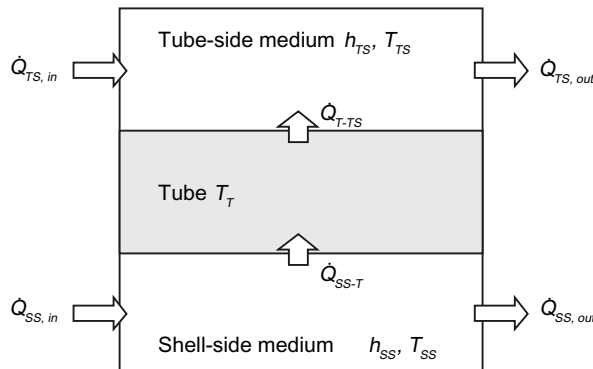
with the relevant throttle coefficient  $c_v$ . The chosen reference direction for the flow is from connector *\_IN* to *\_OUT* on the tube side and shell side.

For the heat transfer, a simple one tube model divided into segments is applied. The number *N* of segments can be set to a value between 4 and 16.



- ① Tube side space
- ② Tube
- ③ Shell side space

The water/steam heat on the tube and shell side and the heat in the tube itself are balanced.



The two heat transfers, from the shell side medium to the tube and from the tube to the tube side medium, are set according to

$$\dot{Q}_{SS-T} = A_{SS} \dot{q}_{SS-T} = A_{SS} \alpha_{SS} (T_{SS} - T_T)$$

$$\dot{Q}_{T-TS} = A_{TS} \dot{q}_{T-TS} = A_{TS} \alpha_{TS} (T_T - T_{TS})$$

in an extremely simplified manner. For each segment  $i$  the heat balances

$$\frac{dh_{TS,i}}{dt} = a_{TS} (h_{TS,i-1} - h_{TS,i}) + b_{TS} (T_{T,i} - T_{TS,i})$$

$$\frac{dT_{T,i}}{dt} = a_T (T_{SS,i} - T_{T,i}) + b_T (T_{TS,i} - T_{T,i})$$

$$\frac{dh_{SS,i}}{dt} = a_{SS} (h_{SS,i-1} - h_{SS,i}) + b_{SS} (T_{T,i} - T_{SS,i})$$

apply with the following coefficients, which are the same for all segments:

$$a_{TS} = \frac{N \dot{m}_{TS}}{\rho_{TS} V_{TS}}, \quad b_{TS} = \frac{A_{TS} \alpha_{TS}}{\rho_{TS} V_{TS}}, \quad a_{SS} = \frac{N \dot{m}_{SS}}{\rho_{SS} V_{SS}}, \quad b_{SS} = \frac{A_{SS} \alpha_{SS}}{\rho_{SS} V_{SS}}, \quad a_T = \frac{A_{SS} \alpha_{SS}}{M_T c_T}, \quad b_T = \frac{A_{TS} \alpha_{TS}}{M_T c_T}$$

For the densities  $\rho_{TS}$ ,  $\rho_{SS}$  and the specific heat capacities  $c_{TS}$ ,  $c_{SS}$ , the values apply that are set for the media in the shell side flownet and tube side flownet.

For initialization, the temperatures of the tube segments are set to the temperature of the tube side medium, calculated from the pressure (input *FNTS.PRESSURE*) and the specific enthalpy (input *FNTS.HSPEC*).

**Parameters**

The segmentation and coefficients for the balance equations are defined by parameters:

- *Type*  
Type of heat exchanger: *ParallelFlow* (parallel flow), *CounterFlow* (counter flow), *CrossFlow* (cross flow)
- *NbrOfSegments*  
Number *N* of segments:  $4 \leq N \leq 16$
- *CvsSS*  
Shell side throttle coefficient  $\alpha_V$
- *CvsTS*  
Tube side throttle coefficient  $\alpha_V$
- *VolumeSS*  
Shell side volume  $V_{SS}$
- *VolumeTS*  
Tube side volume  $V_{TS}$
- *SurfaceSS*  
Shell side surface  $A_{SS}$  of the tube (exterior surface of the tube)
- *SurfaceTS*  
Tube side surface  $A_{TS}$  of the tube (interior surface of the tube)
- *HeatTransCoefSS*  
Heat transfer coefficient  $\alpha_{SS}$  on the tube exterior
- *HeatTransCoefTS*  
Heat transfer coefficient  $\alpha_{TS}$  on the tube interior
- *sHeatCapTube*  
Specific heat capacity  $c_T$  of the tube
- *MassTube*  
Mass  $M_T$  of the tube

The parameters with their units and default values are shown in the figure below:

Name	Value
Type	ParallelFlow ▾
NbrOfSegments	4
CvsSS [m³/h]	360.0
CvsTS [m³/h]	360.0
VolumeSS [m³]	1.65
VolumeTS [m³]	2.25
SurfaceSS [m²]	720.0
SurfaceTS [m²]	550.0
HeatTransCoefSS [kW/m²K]	4.0
HeatTransCoefTS [kW/m²K]	4.0
sHeatCapTube [kJ/kgK]	1.3
MassTube [kg]	10000.0



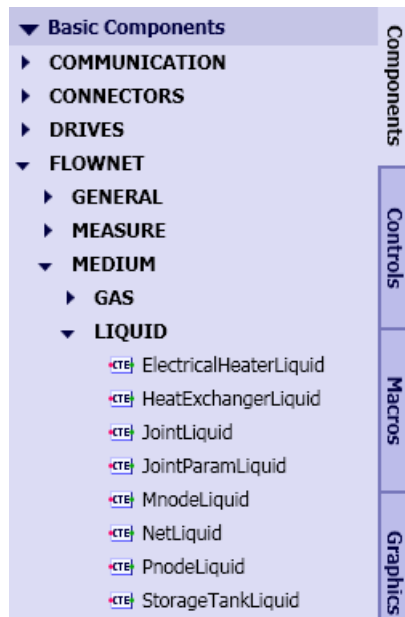
## Operating window

The flow rate  $\dot{m}$ , pressure drop  $\Delta p$  and enthalpy difference  $\Delta h$  of the shell side and tube side media are displayed in the operating window. The temperatures of the tube and the medium (shell side and tube side) for the first and last segment ( $T_1$  and  $T_N$ ) are also displayed.

	$\dot{m}$ [kg/s]	$\Delta p$ [bar]	$\Delta h$ [kJ/kg]
	###	###	###
	$T_1$ [°C]	$T_N$ [°C]	
Shell side	###	###	
Tube	###	###	
Tube side	###	###	
	###	###	###

### 7.2.3.5 Component types for liquid medium

Component types that can be used in the flownet with the liquid medium are located in the directory *MEDIUMLIQUID* of the FLOWNET library. The medium parameter for a flownet that contains these components should be set to the value "Liquid", if applicable.



## NetLiquid – liquid network parameter assignment

### Symbol



### Function

The *NetLiquid* component type is used for parameter assignment of a flownet for liquids. The components can be added at any point on any branch of the flownet.

### Parameters

The variables for the flownet can be specified as parameters for the components:

- *Density*  
Density of the liquid
- *sHeatCapacity*  
Specific heat capacity of the liquid
- *FactorMomentum*  
Momentum factor for the flow through the branches of the flownet
- *sCompression*  
Specific compression modulus of the liquid
- *FactorTherma*  
Factor for enthalpy balancing

The parameters with their units and default values are shown in the figure below:

Name	Value
Density	[kg/m <sup>3</sup> ] 997.337
sHeatCapacity	[kJ/kgK] 4.18
FactorMomentum	[m] 450.0
sCompression	[bar/kg] 100.0
FactorThermal	[1/kg] 0.1

### Additional parameters

Initial values and specific characteristics for internal nodes of the flownet can be set by means of additional parameters:

- *PressureInit*  
Initialization value for the pressure in the internal nodes of the flownet
- *TemperatureInit*  
Initialization value for the temperature in the internal nodes of the flownet

- *TemperatureEnvironment*  
Ambient temperature
- *FactorHeatExchangeEnv*  
Proportionality factor for the heat exchange of the medium in the internal nodes with the environment; for a value of zero, no heat exchange occurs.

The additional parameters, their units and default values are shown in the figure below:

Name		Value
PressureInit	[bar]	1.0
TemperatureInit	[°C]	20.0
TemperatureEnvironment	[°C]	20.0
FactorHeatExchangeEnv	[kW/K]	0.0

## PnodeLiquid – liquid pressure setting

### Symbol

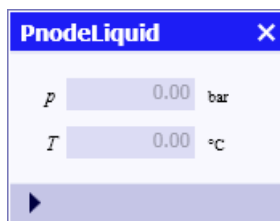


### Function

The *PnodeLiquid* component type specifies values for pressure and temperature at its connector. It forms a boundary for the flownet to which it is connected. When the flownet is represented as a graph, *PnodeLiquid* corresponds to an (external) node, for which the pressure and temperature are predefined.

### Operating window

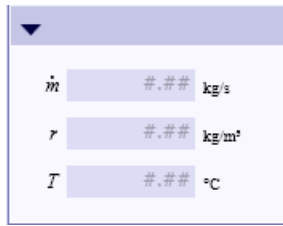
The values for pressure and temperature can be input digitally in the operating window.



The inputs for these variables have the following default values:

- Pressure  $p = 1 \text{ bar}$  (*Pressure* input)
- Temperature  $T = 20 \text{ °C}$  (*Temperature* input)

The corresponding values for the medium flowing into or out of the node are shown in the extended operating window: mass flow  $\dot{m}$ , density  $r$  and temperature  $T$ .



For outflow  $\dot{m} > 0$ , for inflow  $\dot{m} < 0$ .

### MnodeLiquid – liquid mass flow setting

#### Symbol

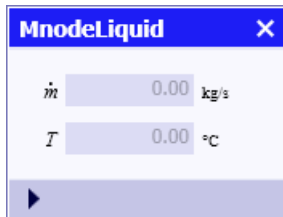


#### Function

The *MnodeLiquid* component type outputs values for mass flow and temperature at its connector. It forms a boundary for the flownet to which it is connected. When the flownet is represented as a graph, *MnodeLiquid* forms an inflow or outflow to an (internal) node or branch. Thus an internal node with a defined inflow or outflow is added to the flownet.

#### Operating window

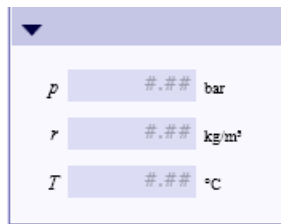
The values for mass flow and temperature can be input digitally in the operating window.



The inputs for these variables have the following default values:

- Mass flow  $\dot{m} = 0$  (*Massflow* input)
- Temperature  $T = 20$  °C (*Temperature* input)

The variables of the assigned internal node are shown in the extended operating window: pressure  $p$ , density  $r$  and temperature  $T$ .



For inflow  $\dot{m} > 0$ , for outflow  $\dot{m} < 0$ .

## JointLiquid – liquid joint

### Symbol

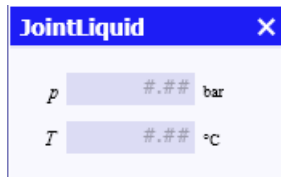


### Function

The *JointLiquid* component type can be used to join three branches connected at its connectors in one node. All connectors are equal. The *JointLiquid* component adds an internal node to the flownet.

### Operating window

The pressure  $p$  and temperature  $T$  of the node are displayed in the operating window.



## JointParamLiquid – liquid parameterizable joint

### Symbol



### Function

The *JointParamLiquid* component type can be used to join three branches connected at its connectors in one node. All connectors are equal. The *JointParamLiquid* component adds an internal node to the flownet. This node can be assigned parameters.

### Parameters

The node associated with *JointParamLiquid* can be assigned parameters.

- *sHeatCapacity*  
Specific heat capacity of the liquid
- *sCompression*  
Specific compression modulus of the liquid
- *FactorThermal*  
Factor for enthalpy balancing

The parameters with their units and default values are shown in the figure below:

Name	Value
sHeatCapacity [kJ/kgK]	4.18
sCompression [bar/kg]	100.0
FactorThermal [1/kg]	0.1

### Additional parameters

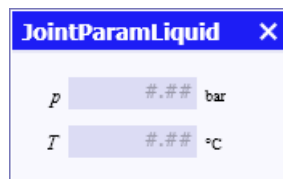
Initial values and specific characteristics for the node can be set by means of additional parameters:

- *PressureInit*  
Initialization value for the pressure in the node
- *TemperatureInit*  
Initialization value for the temperature in the node
- *TemperatureEnvironment*  
Ambient temperature
- *FactorHeatExchangeEnv*  
Proportionality factor for the heat exchange of the liquid with the environment in the node; for a value of zero no heat exchange occurs.

The additional parameters, their units and default values are shown in the figure below:

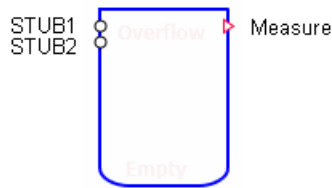
Name	Value
PressureInit [bar]	1.0
TemperatureInit [°C]	20.0
TemperatureEnvironment [°C]	20.0
FactorHeatExchangeEnv [kW/K]	0.0

The pressure  $p$  and temperature  $T$  of the node are displayed in the operating window.



## StorageTankLiquid – liquid storage tank

### Symbol



### Function

The *StorageTankLiquid* component type provides the simulation with an open tank for liquids, which means a tank that is not sealed from the environment.

Inflows and outflows occur via the *STUBx* connectors on the tank. For each of the *N* connectors, a throttling effect defined by

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_V^2 \rho \frac{\text{kg}}{\text{m}^3}} 12900 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

is assumed. The tank can have a minimum of one and a maximum of 16 connectors. The connectors can be moved to any position on the outline of the component symbol by using the mouse while simultaneously pressing ALT.

It is assumed that the liquid in the tank is immediately completely mixed, which means it is a homogenous medium with an overall consistent density and temperature.

The inflows and outflows are balanced across the *N* connectors of the tank. The balanced mass *M* of the *N* mass flows  $\dot{m}_i$  of the liquid in the tank, is defined by

$$\frac{dM}{dt} = \sum_{i=1}^N \dot{m}_i$$

The temperature of the liquid is balanced as a mixing temperature from the inflows ( $i \in Z$ ) according to

$$\frac{dT}{dt} = \frac{1}{M} \left( \frac{1}{c_L} \sum_{i=1, i \in Z}^N h_i \dot{m}_i - T \sum_{i=1, i \in Z}^N \frac{1}{\rho_i} \dot{m}_i \right)$$

The dynamic behavior of the liquid in the tank is described by these balances for mass *M* and temperature *T*.

The calculated variables for the level *l* of the liquid in the tank, its mass *M* and temperature *T* as well as pressure  $p_0$  on the tank base are output at the *Measure* connector. The pressure *p*

on the tank base is calculated from the weight pressure  $\rho g l$  of the liquid and the atmospheric pressure  $\rho_U$  according to

$$\rho = \rho_U + \rho g l$$

**Borderline case "empty tank"**

When the tank is empty, the outflow is strongly throttled. The flow coefficient is thereby set to the value  $c_{v0}$  for maximum throttling of all connectors through which the medium flows out.

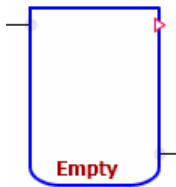
The tank is considered empty when the tank fill level  $V$  is lower than a specified minimum tank filling  $V_{min}$ :

$$V < V_{min}$$

When the tank is empty, balancing of the states is stopped, which means changes to the mass and temperature of the liquid are rejected. The empty state continues until there is a sufficient increase in the fill level. In each increment the validity of

$$M \geq V_{min} \rho \left( 1 + \frac{H_{min}}{100\%} \right)$$

is checked. The required increase can be set via the hysteresis  $H_{min}$ . A corresponding indicator for an empty tank is shown in the component symbol.

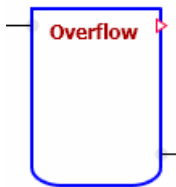


**Borderline case "full tank"**

For a full tank, only the balanced contents are limited in the calculation:

$$M = V\rho$$

A corresponding indicator is shown in the symbol.





## Parameters

Set the following as geometric variables of the tank using parameters:

- *Volume*  
Volume  $V$  of the tank; can be adjusted online
- *Height*  
Height of the tank; can be adjusted online
- *NbrOfStubs*  
Number  $N$  of connectors

The parameters with their units and default values are shown in the figure below:

Name		Value
Volume	[m <sup>3</sup> ]	10.0
Height	[m]	5.0
NbrOfStubs		1

## Additional parameters

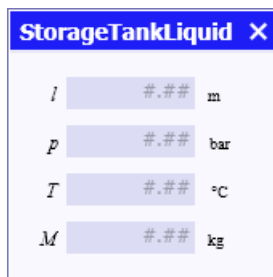
The initial values for the atmospheric pressure *PressureOutside*, the fill level (*LevelInit*) of the tank and the temperature (*TemperatureInit*) of the liquid are set via additional parameters:

- *PressureOutside*  
Atmospheric pressure  $p_j$ ; can be adjusted online
- *LevelInit*  
Initialization value for the level
- *TemperatureInit*  
Initialization value for the temperature of the contents
- *Cvs*  
Uniform flow coefficient  $c_v$  for all connectors
- *Cv0*  
Flow coefficient  $c_{v0}$  for maximum throttling in the tank connector
- *Density*  
Density  $\rho$  of the liquid in the tank
- *sHeatCapacity*  
Specific heat capacity  $c_l$  of the liquid in the tank
- *MinVolume*  
*Minimum tank* volume  $V_{\min}$ ; can be adjusted online
- *MinVolumeHys*  
Hysteresis  $H_{\min}$ ; can be adjusted online

The additional parameters, their units and default values are shown in the figure below:

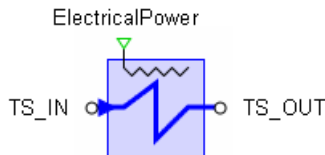
Name	Value
PressureOutside [bar]	1.0
LevelInit [%]	50.0
TemperatureInit [°C]	20.0
Cvs [m³/h]	360.0
Cv0 [m³/h]	0.000001
Density [kg/m³]	997.337
sHeatCapacity [kJ/kgK]	4.18
MinVolume [m³]	0.001
MinVolumeHys [%]	50.0

The level  $l$ , the pressure  $p$ , the temperature  $T$  and the mass  $M$  of the liquid in the tank are displayed in the operating window:



### ElectricalHeaterLiquid – electrically heated heat exchanger for liquid

#### Symbol



#### Function

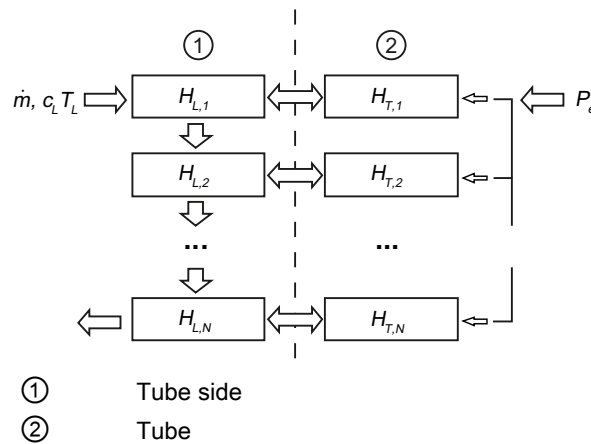
The *ElectricalHeaterLiquid* component type is used to simulate an electrically heated heat exchanger. The electric heating power  $P_{el}$  in kW is specified via the connector *ElectricalPower*.

The liquid is directed via the connectors *TS\_IN* and *TS\_OUT* as a heated medium. The flow  $\dot{m}$  is throttled according to the relationship

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_v^2 \rho \frac{\text{kg}}{\text{m}^3}} 12900 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

with throttle coefficient  $c_v$ . The reference direction for the flow is chosen as from *TS\_IN* to *TS\_OUT*.

For the heat transfer, a simple one tube model divided into segments is applied. The number  $N$  of segments can be set to a value between 4 and 16.



It is assumed that the supplied electrical energy is completely converted into heat. The heat balance for a segment  $i$  of the spare tube with extremely simplified heat transfer is then given by

$$\frac{dT_{T,i}}{dt} = a_T P_{el} + b_T (T_{L,i} - T_{T,i})$$

with

$$a_T = \frac{1}{M_T c_T}, \quad b_T = \frac{A \alpha}{M_T c_T}$$

In addition, the heat balance of the liquid in a segment  $i$  is given by

$$\frac{dT_{L,i}}{dt} = a (T_{L,i-1} - T_{L,i}) + b (T_{T,i} - T_{L,i})$$

with

$$a = \frac{N \dot{m}}{\rho V}, \quad b = \frac{A \alpha}{c_L \rho V}$$

$T_{T,i}$  and  $T_{L,i}$  are the temperatures of the tube segment and the liquid in the segment  $i$ . The flownet values apply for the density  $\rho$  and the specific heat capacity  $c_L$  of the liquid.

### Parameters

The segmentation and coefficients for the balance equations are defined by parameters:

- *NbrOfSegments*  
Number  $N$  of segments:  $4 \leq N \leq 16$
- *Cvs*  
Throttle coefficient  $c_v$
- *Volume*  
Volume  $V$  of the liquid (inner tube volume)

7.2 The FLOWNET library

- *Surface*  
Surface  $A$  of the tube interior
- *HeatTransCoef*  
Heat transfer coefficient  $\alpha$  from tube to liquid
- *sHeatCapTube*  
Specific heat capacity  $c_T$  of the tube
- *MassTube*  
Mass  $M_T$  of the tube

The parameters with their units and default values are shown in the figure below:

Name	Value
NbrOfSegments	4
Cvs [m³/h]	360.0
Volume [m³]	2.25
Surface [m²]	550.0
HeatTransCoef [kW/m²K]	4.0
sHeatCapTube [kJ/kgK]	1.3
MassTube [kg]	10000.0

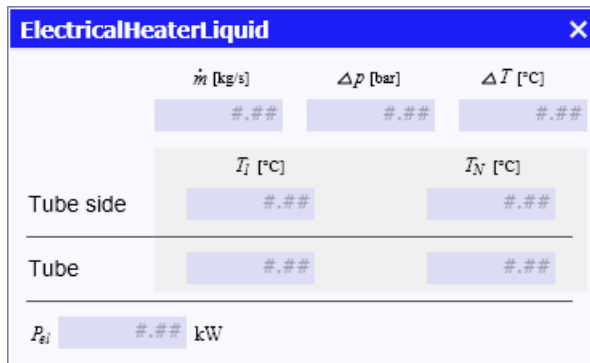
**Additional parameters**

For initialization of the heat exchanger, the temperatures of the liquid and the tube can be set to the same value via the additional parameter *TemperatureInit*.

Name	Value
TemperatureInit [°C]	20.0

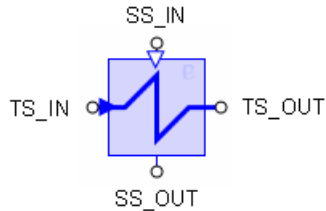
**Operating window**

The flow rate  $\dot{m}$ , pressure drop  $\Delta p$  and temperature difference  $\Delta T$  of the liquid as well as the electrical heating power  $P_{el}$  are displayed in the operating window. The temperatures of the tube and of the liquid (tube side) for the first and last segment ( $T_1$  and  $T_N$ ) are also displayed.



## HeatExchangerLiquid – heat exchanger liquid to liquid

### Symbol



### Function

The *HeatExchangerLiquid* component type is used to simulate a heat exchanger for liquids on the tube side and shell side. The simulation is implemented for the three types:

- Parallel-flow heat exchanger
- Counter-flow heat exchanger
- Cross-flow heat exchanger

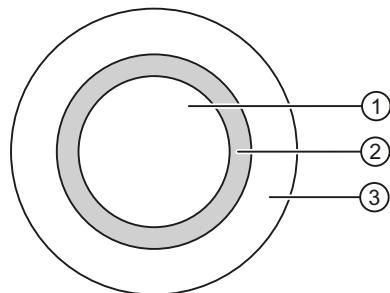
Both media are routed on the tube side via the connectors *TS\_IN* and *TS\_OUT*, and on the shell side via the connectors *SS\_IN* and *SS\_OUT*.

The flow  $\dot{m}$  is throttled on the tube side and shell side according to the relationship

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_V^2 \rho_{TS} \frac{\text{kg}}{\text{m}^3}} 12900 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

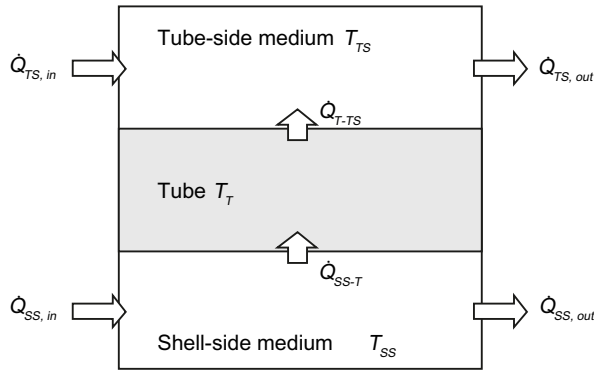
with the relevant throttle coefficient  $c_v$ . The chosen reference direction for the flow is from connector *\_IN* to *\_OUT* on the tube side and shell side.

For the heat transfer, a simple one tube model divided into segments is applied. The number  $N$  of segments can be set to a value between 4 and 16.



- ① Tube side space
- ② Tube
- ③ Shell side space

The liquid heat on the tube and shell side and the heat in the tube itself are balanced.



The two heat transfers, from the shell side liquid to the tube and from the tube to the tube side liquid, are set according to

$$\dot{Q}_{SS-T} = A_{SS} \dot{q}_{SS-T} = A_{SS} \alpha_{SS} (T_{SS} - T_T)$$

$$\dot{Q}_{T-TS} = A_{TS} \dot{q}_{T-TS} = A_{TS} \alpha_{TS} (T_T - T_{TS})$$

in an extremely simplified manner. For each segment  $i$  the heat balances

$$\frac{dT_{TS,i}}{dt} = a_{TS} (T_{TS,i-1} - T_{TS,i}) + b_{TS} (T_{T,i} - T_{TS,i})$$

$$\frac{dT_{T,i}}{dt} = a_T (T_{SS,i} - T_{T,i}) + b_T (T_{TS,i} - T_{T,i})$$

$$\frac{dT_{SS,i}}{dt} = a_{SS} (T_{SS,i-1} - T_{SS,i}) + b_{SS} (T_{T,i} - T_{SS,i})$$

apply with the following coefficients, which are the same for all segments:

$$a_{TS} = \frac{N \dot{m}_{TS}}{\rho_{TS} V_{TS}}, \quad b_{TS} = \frac{A_{TS} \alpha_{TS}}{\rho_{TS} V_{TS}}, \quad a_{SS} = \frac{N \dot{m}_{SS}}{\rho_{SS} V_{SS}}, \quad b_{SS} = \frac{A_{SS} \alpha_{SS}}{\rho_{SS} V_{SS}}, \quad a_T = \frac{A_{SS} \alpha_{SS}}{M_T c_T}, \quad b_T = \frac{A_{TS} \alpha_{TS}}{M_T c_T}$$

The values set for the liquid in the shell side and tube side of the flownet apply to the densities  $\rho_{TS}$ ,  $\rho_{SS}$  and the specific heat capacities  $c_{TS}$ ,  $c_{SS}$ .

For initialization, the temperatures of the tube segments are set to the temperature of the tube side medium, calculated from the specific enthalpy (*FNTS.HSPEC* input).

## Parameters

The segmentation and coefficients for the balance equations are defined by parameters:

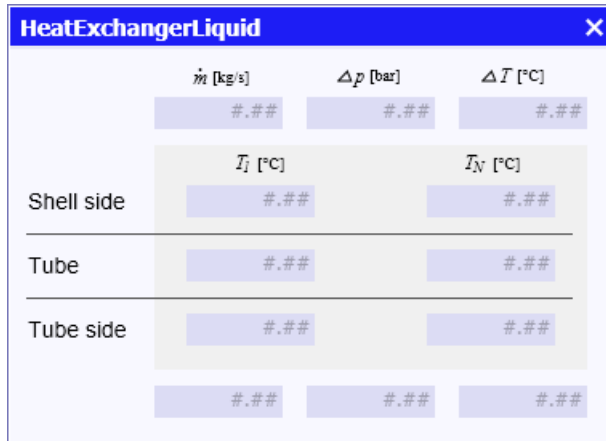
- *Type*  
Type of heat exchanger: *ParallelFlow* (parallel flow), *CounterFlow* (counter flow), *CrossFlow* (cross flow)
- *NbrOfSegments*  
Number  $N$  of segments:  $4 \leq N \leq 16$
- *CvsSS*  
Shell side throttle coefficient  $c_v$
- *CvsTS*  
Tube side throttle coefficient  $c_v$
- *VolumeSS*  
Shell side volume  $V_{SS}$
- *VolumeTS*  
Tube side volume  $V_{TS}$
- *SurfaceSS*  
Shell side surface  $A_{SS}$  of the tube (exterior surface of the tube)
- *SurfaceTS*  
Tube side surface  $A_{TS}$  of the tube (interior surface of the tube)
- *HeatTransCoefSS*  
Heat transfer coefficient  $\alpha_{SS}$  on the tube exterior
- *HeatTransCoefTS*  
Heat transfer coefficient  $\alpha_{TS}$  on the tube interior
- *sHeatCapTube*  
Specific heat capacity  $c_T$  of the tube
- *MassTube*  
Mass  $M_T$  of the tube

The parameters with their units and default values are shown in the figure below:

Name	Value
Type	ParallelFlow <span style="float: right;">▼</span>
NbrOfSegments	4
CvsSS	[m <sup>3</sup> /h] 360.0
CvsTS	[m <sup>3</sup> /h] 360.0
VolumeSS	[m <sup>3</sup> ] 1.65
VolumeTS	[m <sup>3</sup> ] 2.25
SurfaceSS	[m <sup>2</sup> ] 720.0
SurfaceTS	[m <sup>2</sup> ] 550.0
HeatTransCoefSS	[kW/m <sup>2</sup> K] 4.0
HeatTransCoefTS	[kW/m <sup>2</sup> K] 4.0
sHeatCapTube	[kJ/kgK] 1.3
MassTube	[kg] 10000.0

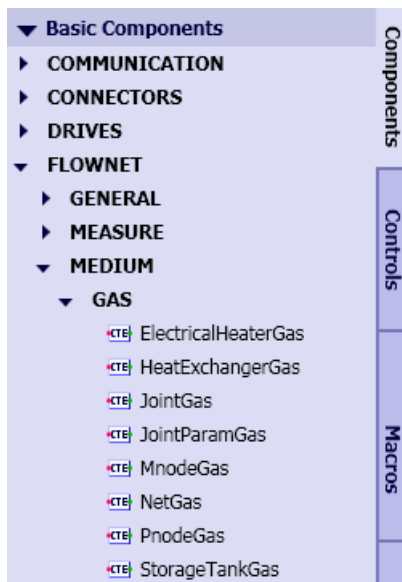
**Operating window**

The flow rate  $\dot{m}$ , pressure drop  $\Delta p$  and enthalpy difference  $\Delta h$  of the shell side and tube side liquids are displayed in the operating window. The temperatures of the tube (Tube), and the liquid (shell side and tube side) for the first and last segment ( $T_1$  and  $T_N$ ) are also displayed.



**7.2.3.6 Component types for gas medium**

Component types that can be used in the flownet with the gas medium are located in the directory *MEDIUMGAS* of the FLOWNET library. The medium parameter for a flownet that contains these components should be set to the value "Ideal Gas", if applicable.





## NetGas – gas network parameter assignment

### Symbol



### Function

The *NetGas* component type is used for parameter assignment of a network for gases. The components can be added at any point on any branch of the flownet.

### Parameters

The variables for the flownet can be specified as parameters for the components:

- **GasConstant**  
Specific gas constant
- **sHeatCapacity**  
Specific heat capacity of the gas
- **FactorMomentum**  
Momentum factor for the flow through the branches of the flownet
- **sCompression**  
Specific compression modulus of the gas
- **FactorTherma**  
Factor for enthalpy balancing

The parameters with their units and default values are shown in the figure below:

Name	Value	
GasConstant	[kJ/kgK]	0.287
sHeatCapacity	[kJ/kgK]	1.0
FactorMomentum	[m]	450.0
sCompression	[bar/kg]	10.0
FactorThermal	[1/kg]	100.0

### Additional parameters

Initial values and specific characteristics for internal nodes of the flownet can be set by means of additional parameters:

- **PressureInit**  
Initialization value for the pressure in the internal nodes of the flownet
- **TemperatureInit**  
Initialization value for the temperature in the internal nodes of the flownet

- *TemperatureEnvironment*  
Ambient temperature
- *FactorHeatExchangeEnv*  
Proportionality factor for the heat exchange of the gas with the environment in the internal nodes; for a value of zero no heat exchange occurs.

The additional parameters, their units and default values are shown in the figure below.

Name	Value
PressureInit	[bar] 1.0
TemperatureInit	[°C] 20.0
TemperatureEnvironment	[°C] 20.0
FactorHeatExchangeEnv	[kW/K] 0.0

### PnodeGas – gas pressure setting

#### Symbol

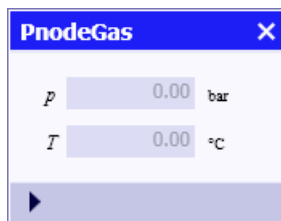


#### Function

The *PnodeGas* component type specifies values for pressure and temperature at its connector. It forms a boundary for the flownet to which it is connected. When the flownet is represented as a graph, *PnodeGas* corresponds to an (external) node for which the pressure and temperature are predefined.

#### Operating window

The values for pressure and temperature can be input digitally in the operating window.



The inputs for these variables have the following default values:

- Pressure  $p = 1 \text{ bar}$  (*Pressure* input)
- Temperature  $T = 20 \text{ °C}$  (*Temperature* input)

The corresponding values for mass flow  $\dot{m}$ , density  $r$  and temperature  $T$  for medium flowing into or out of the node are displayed in the extended operating window.

For outflow  $\dot{m} > 0$ , for inflow  $\dot{m} < 0$ .

## MnodeGas – gas mass flow setting

### Symbol



### Function

The *MnodeGas* component type specifies values for mass flow and temperature at its connector. It forms a boundary for the flownet to which it is connected. When the flownet is represented as a graph, *Mnode* corresponds to an inflow or outflow through an (internal) node or branch. Thus an internal node with a defined inflow or outflow is added to the flownet.

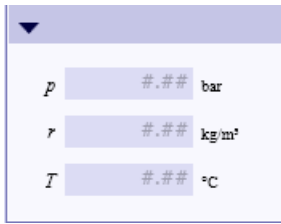
### Operating window

The values for mass flow and temperature can be input digitally in the operating window.

The inputs for these variables have the following default values:

- Mass flow  $\dot{m} = 0$  (*Massflow* input)
- Temperature  $T = 20$  °C (*Temperature* input)

The variables of the assigned internal node are shown in the extended operating window: pressure  $p$ , density  $r$  and temperature  $T$ .



For inflow  $\dot{m} > 0$ , for outflow  $\dot{m} < 0$ .

### JointGas – gas joint

#### Symbol

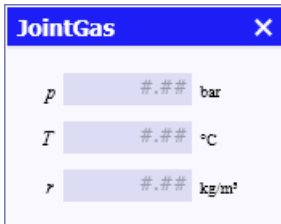


#### Function

The *JointGas* component type can be used to join three branches connected at its connectors in one node. All connectors are equal. The *JointGas* component adds an internal node to the flownet.

#### Operating window

The pressure  $p$ , temperature  $T$  and density  $r$  of the node are displayed in the operating window.



### JointParamGas – parameterizable joint

#### Symbol



#### Function

The *JointParamGas* component type can be used to join three branches connected at its connectors in one node. All connectors are equal. The *JointParamGas* component adds an internal node to the flownet. This node can be assigned parameters.

## Parameters

The node associated with *JointParamGas* can be assigned parameters:

- *GasConstant*  
Specific gas constant
- *sHeatCapacity*  
Specific heat capacity of the gas
- *sCompression*  
Specific compression modulus of the gas
- *FactorThermal*  
Factor for enthalpy balancing

The parameters with their units and default values are shown in the figure below:

Name	Value
GasConstant	[kJ/kgK] 0.287
sHeatCapacity	[kJ/kgK] 1.0
sCompression	[bar/kg] 10.0
FactorThermal	[1/kg] 100.0

## Additional parameters

Initial values and specific characteristics for the node can be set by means of additional parameters:

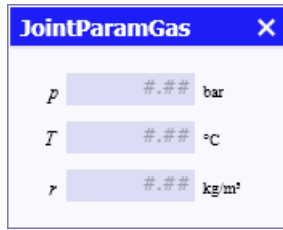
- *PressureInit*  
Initialization value for the pressure in the node
- *TemperatureInit*  
Initialization value for the temperature in the node
- *TemperatureEnvironment*  
Ambient temperature
- *FactorHeatExchangeEnv*  
Proportionality factor for the heat exchange of the gas with the environment in the node; for a value of zero no heat exchange occurs.

The additional parameters, their units and default values are shown in the figure below:

Name	Value
PressureInit	[bar] 1.0
TemperatureInit	[°C] 20.0
TemperatureEnvironment	[°C] 20.0
FactorHeatExchangeEnv	[kW/K] 0.0

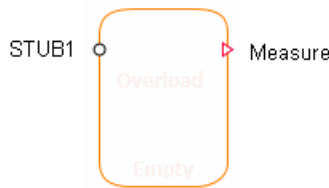
**Operating window**

The pressure  $p$ , temperature  $T$  and density  $\rho$  of the node are displayed in the operating window.



**StorageTankGas – gas storage tank**

**Symbol**



**Function**

The *StorageTankGas* component type provides the simulation of a gas tank.

Inflows and outflows occur via the *STUBx* connectors on the tank. For each of the  $N$  connectors, a throttling effect defined by

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_v^2 \rho \frac{\text{kg}}{\text{m}^3}} 12900 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

is assumed. The tank can have a minimum of one and a maximum of 16 connectors. The connectors can be moved to any position on the outline of the component symbol by using the mouse while simultaneously pressing ALT.

It is assumed that the gas in the tank is immediately completely mixed, which means a homogenous medium in the tank with an overall consistent density and temperature in the tank.

The inflows and outflows are balanced across the  $N$  connectors of the tank. The balanced mass  $M$  of the  $N$  flow rates  $\dot{m}_i$  of the gas in the tank, is defined by

$$\frac{dM}{dt} = \sum_{i=1}^N \dot{m}_i$$

The specific enthalpy  $h$  of the gas is balanced from the inflows ( $i \in \mathbb{Z}$ ) according to

$$\frac{dh}{dt} = \frac{1}{M} \left( \sum_{i=1, i \in Z}^N h_i \dot{m}_i - h \sum_{i=1, i \in Z}^N \dot{m}_i \right)$$

The dynamic behavior of the gas in the tank is described by these balances for mass  $M$  and specific enthalpy  $h$ .

At the connector *Measure*, the variables for pressure  $p$ , temperature

$$T = \frac{h}{c_G}$$

and mass  $M$  of the gas are output. The pressure is calculated from the equation for ideal gas:

$$p = \frac{R_s (T + 273,15\text{K}) M}{V \cdot 10^5 \frac{\text{Pa}}{\text{bar}}}$$

### Borderline case "empty tank"

When the tank is empty, the outflow is strongly throttled. The flow coefficient is thereby set to the value  $c_{v0}$  for maximum throttling of all connectors through which gas flows.

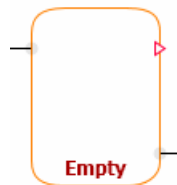
The tank is considered empty when the fill level  $M$  is lower than a specified minimum tank fill level  $M_{\min}$ :

$$M < M_{\min}$$

When the tank is empty, balancing of the states is stopped, which means changes to the mass and specific enthalpy of the gas are rejected. The empty state continues until there is a sufficient increase in the fill level. In each increment the validity of

$$M \geq M_{\min} \left( 1 + \frac{H_{\min}}{100\%} \right)$$

is checked. The required increase can be set via the hysteresis  $H_{\min}$ . A corresponding indicator for an empty tank is shown in the component symbol.



### Borderline case "full tank"

In order to limit the pressure in the tank to meaningful values while the simulation is running, the tank is considered full when its pressure  $p$  reaches a specified maximum pressure  $p_{\max}$ :

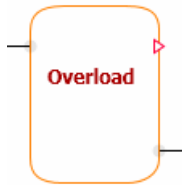
$$\rho < \rho_{\max}$$

When the tank is full, the inflow is strongly throttled. The flow coefficient is thereby set to the value  $c_{v0}$  for maximum throttling of all connections through which gas flows in.

When the tank is full, balancing of the states is stopped, which means changes to the mass and specific enthalpy of the gas are rejected. The full state continues until there is a sufficient decrease  $\Delta M$  in the fill level. In each increment the validity of

$$\Delta M \geq M_{\min} \left( 1 + \frac{H_{\min}}{100\%} \right)$$

is checked. The required decrease can be set via the hysteresis  $H_{\min}$ . A corresponding indicator for a full tank is shown in the component symbol.



### Parameters

Set the following as geometric variables of the tank using parameters:

- *Volume*  
Volume  $V$  of the tank; can be adjusted online
- *NbrOfStubs*  
Number  $N$  of connectors

The parameters with their units and default values are shown in the figure below:

Name	Value
Volume	[m³] 10.0
NbrOfStubs	1

### Additional parameters

Initial values and other variables can be specified via additional parameters:

- *PressureInit*  
Initialization value for pressure; can be adjusted online
- *TemperatureInit*  
Initialization value for the temperature of the gas
- *PressureMax*  
Maximum tank pressure  $\rho_{\max}$ ; can be adjusted online
- *Cvs*  
Uniform flow coefficient  $c_v$  for all connectors



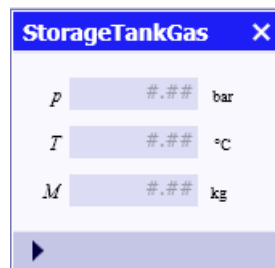
- *Cv0*  
Flow coefficient  $c_{v0}$  for maximum throttling in the tank connector
- *GasConstant*  
Specific gas constant  $R_g$  for the gas in the tank
- *sHeatCapacity*  
Specific heat capacity  $c_g$  of the gas in the tank
- *MinMass*  
Minimum fill quantity  $M_{\min}$  of the tank; can be adjusted online
- *MinVolumeHys*  
Hysteresis  $H_{\min}$ ; can be adjusted online

The additional parameters, their units and default values are shown in the figure below:

Name	Value
PressureInit	[bar] 1.0
TemperatureInit	[°C] 20.0
PressureMax	[bar] 100.0
Cvs	[m³/h] 36.0
Cv0	[m³/h] 0.000001
GasConstant	[kJ/kgK] 0.287
sHeatCapacity	[kJ/kgK] 1.0
MinMass	[kg] 0.015
MinMassHys	[%] 50.0

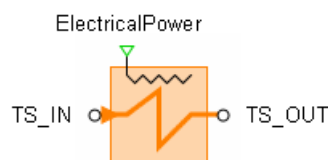
## Operating window

The pressure  $p$ , temperature  $T$  and mass  $M$  of the gas in the tank are displayed in the operating window.



## ElectricalHeaterGas – electrically heated heat exchanger for gas

### Symbol



**Function**

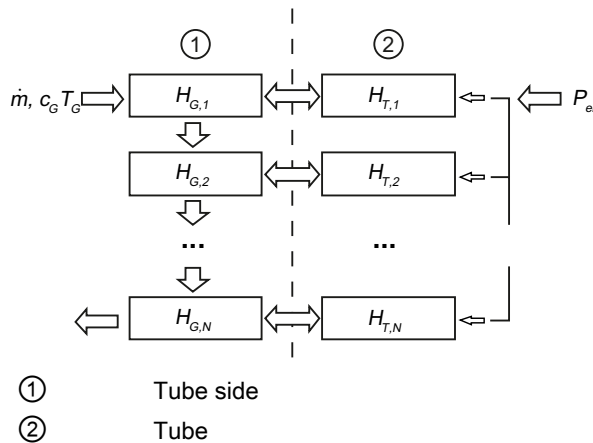
The *ElectricalHeaterGas* component type is used to simulate an electrically heated heat exchanger for gas. The electric heating power  $P_{el}$  in kW is specified via the connector *ElectricalPower*.

The heated gas is directed via the connectors *TS\_IN* and *TS\_OUT*. The flow  $\dot{m}$  is throttled according to the relationship

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_V^2 \rho_{TS} \frac{\text{kg}}{\text{m}^3}} 12900 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

with throttle coefficient  $c_V$ . The reference direction for the flow is chosen as from *TS\_IN* to *TS\_OUT*.

For the heat transfer, a simple one tube model divided into segments is applied. The number  $N$  of segments can be set to a value between 4 and 16.



It is assumed that the supplied electrical energy is completely converted into heat. The heat balance for a segment  $i$  of the spare tube with extremely simplified heat transfer is then given by

$$\frac{dT_{G,i}}{dt} = a_T P_{el} + b_T (T_{G,i} - T_{T,i})$$

with

$$a_T = \frac{1}{M_T c_T}, \quad b_T = \frac{A \alpha}{M_T c_T}$$

In addition, the heat balance of the gas in a segment  $i$  is given by

$$\frac{dT_{G,i}}{dt} = a (T_{G,i-1} - T_{G,i}) + b (T_{T,i} - T_{G,i})$$

with

$$a = \frac{N \dot{m}}{\rho V}, \quad b = \frac{A \alpha}{c_G \rho V}$$

$T_{T,i}$  and  $T_{G,i}$  are the temperatures of the tube segment and the gas in the segment  $i$ . The flownet values apply for the density  $\rho$  and the specific heat capacity  $c_G$  of the gas.

### Parameters

The segmentation and coefficients for the balance equations are defined by parameters:

- *NbrOfSegments*  
Number  $N$  of segments:  $4 \leq N \leq 16$
- *Cvs*  
Throttle coefficient  $c_V$
- *Volume*  
Volume  $V$  of the gas (inner tube volume)
- *Surface*  
Surface  $A$  of the tube interior
- *HeatTransCoef*  
Heat transfer coefficient  $\alpha$  from tube to gas
- *sHeatCapTube*  
Specific heat capacity  $c_T$  of the tube
- *MassTube*  
Mass  $M_T$  of the tube

The parameters with their units and default values are shown in the figure below:

Name	Value
NbrOfSegments	4
Cvs [m <sup>3</sup> /h]	360.0
Volume [m <sup>3</sup> ]	2.25
Surface [m <sup>2</sup> ]	550.0
HeatTransCoefTS [kW/m <sup>2</sup> K]	0.1
sHeatCapTube [kJ/kgK]	1.3
MassTube [kg]	1000.0

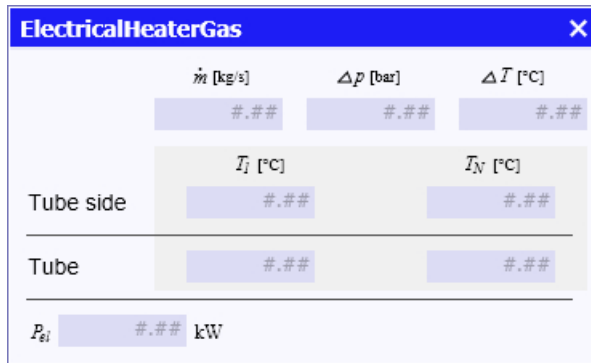
### Additional parameters

For initialization of the heat exchanger, the temperatures of the gas and the tube can be set to the same value via the additional parameter *TemperatureInit*.

Name	Value
TemperatureInit [°C]	20.0

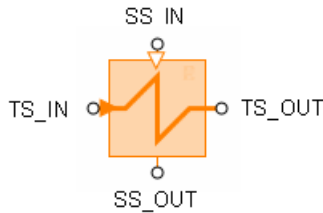
### Operating window

The flow rate  $\dot{m}$ , pressure drop  $\Delta p$  and temperature difference  $\Delta T$  of the gas as well as the electrical heating power  $P_{el}$  are displayed in the operating window. The temperatures of the tube, and of the gas (tube side) for the first and last segment ( $T_1$  and  $T_N$ ) are also displayed.



### HeatExchangerGas – heat exchanger gas to gas

#### Symbol



#### Function

The *HeatExchangerGas* component type is used to simulate a heat exchanger for the gases on the tube side / shell side. The simulation is implemented for these three types:

- Parallel-flow heat exchanger
- Counter-flow heat exchanger
- Cross-flow heat exchanger

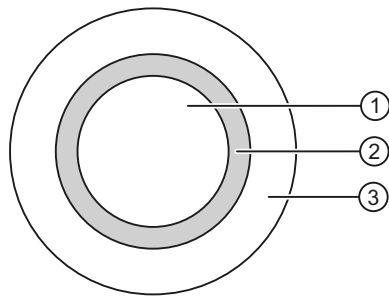
Both media are routed on the tube side via the connectors *TS\_IN* and *TS\_OUT*, and on the shell side via the connectors *SS\_IN* and *SS\_OUT*.

The flow  $\dot{m}$  is throttled on the tube side and shell side according to the relationship

$$\frac{\Delta p}{\text{bar}} = \frac{-\dot{m}^2}{k_V^2 \rho_{TS} \frac{\text{kg}}{\text{m}^3}} 12900 \left( \frac{\text{sec}}{\text{h}} \right)^2$$

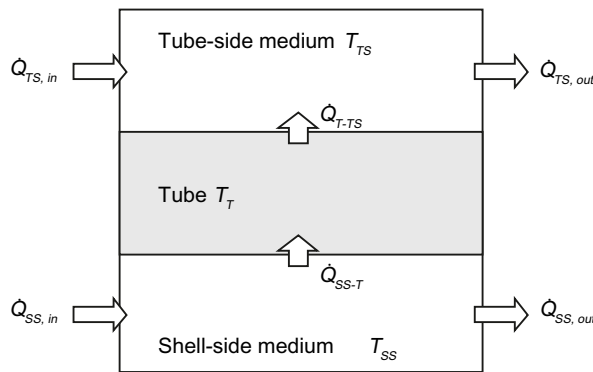
with the relevant throttle coefficient  $c_v$ . The chosen reference direction for the flow is from connector *\_IN* to *\_OUT* on the tube side and shell side.

For the heat transfer, a simple one tube model divided into segments is applied. The number *N* of segments can be set to a value between 4 and 16.



- ① Tube side space
- ② Tube
- ③ Shell side space

The gas heat on the tube and shell side and the heat in the tube itself are balanced.



The two heat transfers, from the shell side gas to the tube, and from the tube to the tube side gas are set according to

$$\dot{Q}_{SS-T} = A_{SS} \dot{q}_{SS-T} = A_{SS} \alpha_{SS} (T_{SS} - T_T)$$

$$\dot{Q}_{T-TS} = A_{TS} \dot{q}_{T-TS} = A_{TS} \alpha_{TS} (T_T - T_{TS})$$

in an extremely simplified manner. For each segment  $i$  the heat balances

$$\frac{dT_{TS,i}}{dt} = a_{TS} (T_{TS,i-1} - T_{TS,i}) + b_{TS} (T_{T,i} - T_{TS,i})$$

$$\frac{dT_{T,i}}{dt} = a_T (T_{SS,i} - T_{T,i}) + b_T (T_{TS,i} - T_{T,i})$$

$$\frac{dh_{SS,i}}{dt} = a_{SS} (h_{SS,i-1} - h_{SS,i}) + b_{SS} (T_{T,i} - T_{SS,i})$$

apply with the following coefficients, which are the same for all segments:

$$a_{TS} = \frac{N\dot{m}_{TS}}{\rho_{TS}V_{TS}}, \quad b_{TS} = \frac{A_{TS}\alpha_{TS}}{c_{TS}\rho_{TS}V_{TS}}, \quad a_{SS} = \frac{N\dot{m}_{SS}}{\rho_{SS}V_{SS}}, \quad b_{SS} = \frac{A_{SS}\alpha_{SS}}{c_{SS}\rho_{SS}V_{SS}}, \quad a_T = \frac{A_{SS}\alpha_{SS}}{M_T c_T}, \quad b_T = \frac{A_{TS}\alpha_{TS}}{M_T c_T}$$

The values set for the gases in the shell side and tube side of the flownet apply to the densities  $\rho_{TS}$ ,  $\rho_{SS}$  and the specific heat capacities  $c_{TS}$ ,  $c_{SS}$ .

For initialization, the temperatures of the tube segments are set to the temperature of the tube side gas, calculated from the specific enthalpy ( *FNTS.HSPEC* input).

## Parameters

The segmentation and coefficients for the balance equations are defined by parameters:

- *Type*  
Type of heat exchanger: *ParallelFlow* (parallel flow), *CounterFlow* (counter flow), *CrossFlow* (cross flow)
- *NbrOfSegments*  
Number  $N$  of segments:  $4 \leq N \leq 16$
- *CvsSS*  
Shell side throttle coefficient  $c_v$
- *CvsTS*  
Tube side throttle coefficient  $c_v$
- *VolumeSS*  
Shell side volume  $V_{SS}$
- *VolumeTS*  
Tube side volume  $V_{TS}$
- *SurfaceSS*  
Shell side surface  $A_{SS}$  of the tube (exterior surface of the tube)
- *SurfaceTS*  
Tube side surface  $A_{TS}$  of the tube (interior surface of the tube)
- *HeatTransCoefSS*  
Heat transfer coefficient  $\alpha_{SS}$  on the tube exterior
- *HeatTransCoefTS*  
Heat transfer coefficient  $\alpha_{TS}$  on the tube interior
- *sHeatCapTube*  
Specific heat capacity  $c_T$  of the tube
- *MassTube*  
Mass  $M_T$  of the tube

The parameters with their units and default values are shown in the figure below:

Name	Value
Type	ParallelFlow
NbrOfSegments	4
CvsSS [m <sup>3</sup> /h]	360.0
CvsTS [m <sup>3</sup> /h]	360.0
VolumeSS [m <sup>3</sup> ]	1.65
VolumeTS [m <sup>3</sup> ]	2.25
SurfaceSS [m <sup>2</sup> ]	720.0
SurfaceTS [m <sup>2</sup> ]	550.0
HeatTransCoefSS [kW/m <sup>2</sup> K]	0.1
HeatTransCoefTS [kW/m <sup>2</sup> K]	0.1
sHeatCapTube [kJ/kgK]	1.3
MassTube [kg]	1000.0

### Operating window

The flow rate  $\dot{m}$ , pressure drop  $\Delta p$  and enthalpy difference  $\Delta h$  of the shell side and tube side gases are displayed in the operating window. The temperatures of the tube and the gases (shell side and tube side) for the first and last segment ( $T_1$  and  $T_N$ ) are also displayed.

	$\dot{m}$ [kg/s]	$\Delta p$ [bar]	$\Delta T$ [°C]
	###	###	###
	$T_1$ [°C]		$T_N$ [°C]
Shell side	###		###
Tube	###		###
Tube side	###		###
	###	###	###

## 7.2.4 Creating your own component types for flownets

The CTE component type editor enables you to create your own component types, which utilize the mechanisms of the FLOWNET process. You can expand the functions of the components supplied with the FLOWNET library, for example, add physical effects that are not covered by the supplied library components and thus enhance your flownet simulations. You can also extend the FLOWNET library by creating your own component types from scratch.

Two aspects must be considered when creating components:

- The topological aspects
- The connection to the solution procedure

The topological aspect is covered by appropriate extensions to the component type definition. Special connection types are provided for the connection to the solution procedure.

In addition, the general descriptions of component properties in the "SIMIT - Component Type Editor" manual form the basis for creating flownet components. The general properties also apply in full to flownet components.

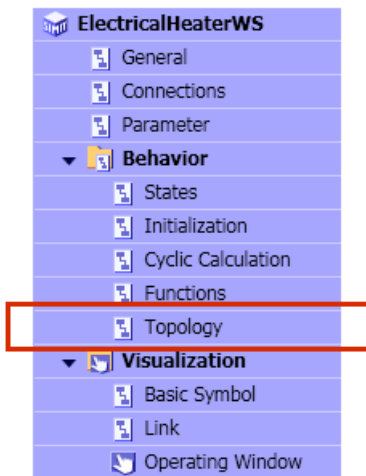
### 7.2.4.1 Topological properties

The topology of a flownet is automatically determined when compiling a simulation project from interconnected flownet components. Each flownet component must thus provide relevant topological information about itself, which means information about how it is to be treated topologically in the flownet. From a topological point of view it is necessary to know how the reference directions for variables in a flownet are defined for a component and how the data exchange between the component and the solution procedure for the flownet is set up.

The flownet elements with topological information are:

- Internal nodes
- External nodes
- Branch objects

Relationships with the topological connectors of the flownet component must be defined for each of these elements. To do this, open the topology editor by double-clicking on the *Topology* node in the navigation of the component type.





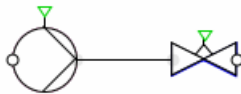
## Connectors of the FLN1 connector type

The *FLN1* connection type indicates the connectors of flownet components that are used to connect with one another. The topology of a flownet is derived from interconnected connectors of this type and the topological information on the individual flownet components. *FLN1* is therefore a purely topological connection type, which means it does not carry any signals. Connectors of this type are simply referred to below as **topological connectors**. Topological connectors are represented by circles.

Connectors of type  
FLN1



If topological connectors are connected with each other by a connecting line in the chart editor, then the connected connectors are hidden. They are still visible only as pale shadows.



They can also be joined by superimposing the connectors. Once linked, both connectors are hidden.



## Topology of the internal node

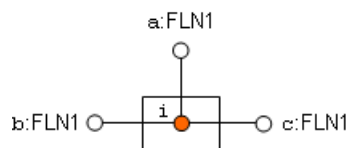
An internal flownet node is defined as follows in a component type in the topology description:

```
INTERNAL_NODE i;
```

Topological connectors of the component must also be assigned to this internal node. Three connectors can be assigned in the topological description as per the example below:

```
FROM i TO a;  
FROM i TO b;  
FROM i TO c;
```

The three connectors *a*, *b* and *c* must be defined as type *FLN1* connectors in the component type. The figure below shows a diagram of the resulting topological structure of the component type.



### Topology of the external node

External nodes represent the boundaries of the flownet. External nodes can be used to predefine pressures and enthalpies in nodes and inflows and outflows for the flownet.

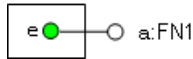
In the topological description of a component type, an external node *e* is defined as follows:

```
EXTERNAL_NODE e;
```

This external node must be also be linked to at least one topological connector of the component. The topology description is, for example, to be completed as

```
FROM e TO a;
```

. The topological connector *a* must be defined in the component type as a connector of type *FLN1*. The figure below shows a diagram of the resulting topological structure of the component type.



### Topology of a branch object

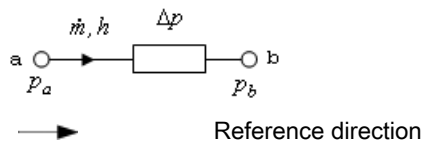
Branch objects are part of a flownet branch, such as throttling of pressure.

The reference direction for the variables of a branch object is defined in the topological description of the flownet component. For example, the definition

```
FROM a TO b;
```

defines the branch object with a reference direction from connector *a* to connector *b* of the component. Both of the topological connectors *a* and *b* must be defined in the component type as connectors of type *FLN1*.

Flow variables are positive in the reference direction. For a branch object, these are the mass flow  $\dot{m}$  and the enthalpy flow  $H = \dot{m}h$ .



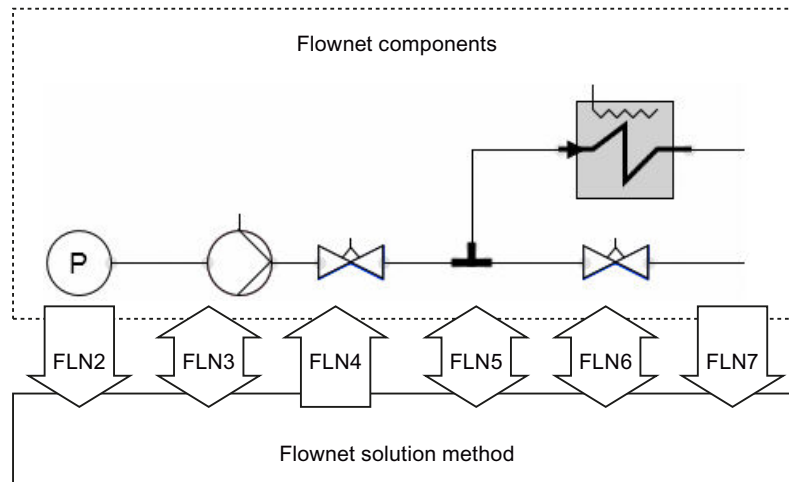
Pressure boosts are applied in the reference direction, which means the pressure change is defined here as

$$\Delta p = p_b - p_a$$

. For pressure drops in the reference direction (throttling),  $\Delta p$  is therefore negative and positive  $\Delta p$  for pressure boosts in the reference direction.

### 7.2.4.2 Connection to the solution procedure

The flownet objects of a component and the flownet solution procedure exchange data via special flownet connectors. There are six different connector types *FLN2* to *FLN7* available for this purpose. Their use with flownet components is explained in the sections 5.2.1 (Page 547) through 5.2.6 (Page 553).



Connectors of the types *FLN2* to *FLN7* provide a connection to the flownet solution procedure and therefore must be set as **invisible** in the component symbol. In the connector properties, the usage "Only in property view" or "Only in CTE" must be set.

Property	Value
Usage	Property view only
Visibility Default	

### Connector type FLN2 for branch objects

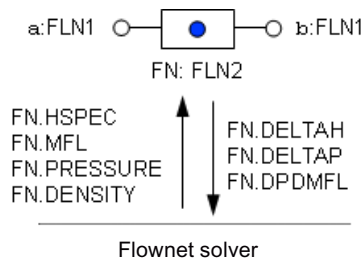
A branch object can exchange variables with the flownet solution procedure via a connector of type *FLN2*. The topological description of a branch object is completed as follows for a connector with the name *FN*:

```
FROM a TO b : FN;
```

The connector must always be defined in the *OUT* direction. It connects the component to the flownet solution procedure to exchange various variables that are relevant to the branch object between the flownet solution procedure and the component. It maps the branch-influencing effect (which means the effect that the branch object has on the branch) to the flownet.

Name	Connection type	Direction	Dimension
FNTS	FLN2	OUT	1

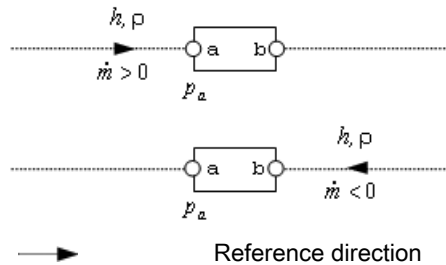
The figure below shows the input and output signals with the direction of data flow between the component and the flownet solution procedure.



The component receives four variables for calculating the effect of the branch object via the input signals:

1. Specific enthalpy  $h$  ( $FN.HSPEC$ ),
2. Mass flow  $\dot{m}$  ( $FN.MFL$ )
3. Pressure  $p_a$  ( $FN.PRESSURE$ ),
4. Density  $\rho$  ( $FN.DENSITY$ ).

The pressure  $p_a$  relates to the "from" connector (*FROM*) of the topological description, which is the connector *a* in our example. Density  $\rho$  and enthalpy  $h$  are variables of the supplied medium.



The output signals for the branch object are calculated in the component and sent to the flownet solution procedure:

1. Change in specific enthalpy  $\Delta h$  ( $FN.DELTAH$ ),
2. Pressure change  $\Delta p = p_b - p_a$  ( $FN.DELTAP$ ),
3. Derivation of the change in pressure based on the mass flow  $d\Delta p/d\dot{M}$  ( $FN.DPDMFL$ ).

### Connector type FLN3 for external nodes

Variables are exchanged between external nodes and the flownet solution procedure via connectors of type *FLN3*. A connector of type *FLN3* is assigned to the external node in the topology description, as follows for a connector with name *FN*:

```
EXTERNAL_NODE e : FN;
```

The connector type *FLN3* provides signals in the forward and reverse direction as listed in the table below.

Table 7-42 Signals of the connector type FLN3

Forward signals		Backward signals	
Name	Variable	Name	Variable
HFW	Specific enthalpy	HBW	Specific enthalpy
PRESSURE	Pressure	MFL	Mass flow rate

The following applies: If the *FN* connector is defined with the direction *OUT*, then the pressure and specific enthalpy are specified via this connector for the external node of the flownet. If the connector is defined with the direction *IN*, the mass flow rate and the specific enthalpy are specified for the external node.

### Connector of type FLN3 with direction OUT

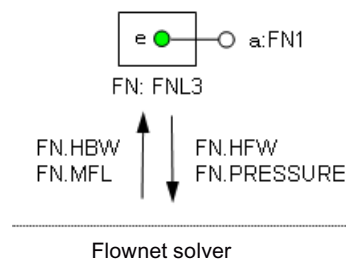
If the connector is defined with direction *OUT*, then the pressure calculated in the component is set for the flownet solution procedure via this connector. This means the external node is a **pressure node**. The following variables can be set at its outputs:

1. Pressure  $p_e$  (*FN.PRESSURE*),
2. Specific enthalpy  $h_e$  (*FN.HFW*).

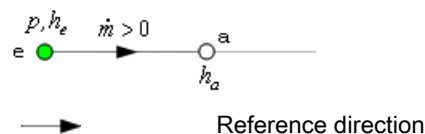
At its inputs, the mass flow supplied to or discharged from the flownet via the external node and calculated by the flownet solution procedure is returned to the components. In the case of discharge from the flownet, the specific enthalpy of the flow is also returned to the component:

1. Mass flow  $\dot{m}$  (*FN.MFL*)
2. Specific enthalpy  $h_a$  (*FN.HBW*)

The figure below illustrates the signal direction for data exchange between the component and the flownet solution procedure.



Regardless of the direction selected in the topology description, the mass flow  $\dot{m}$  is always positive for inflows to the flownet ( $\dot{m} > 0$ ) and negative ( $\dot{m} < 0$ ) for outflows.



### Connector of type FLN3 with direction IN

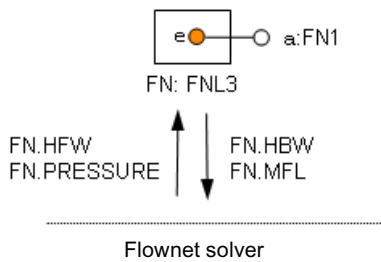
If the connector  $FN$  is defined with direction  $IN$ , then the mass flow set in the component is defined for the flownet via this connector. The external node active in the flownet is therefore a node with mass inflow or a mass flow node. The following variables can be set at its outputs:

1. Mass flow  $\dot{m}$  (FN.MFL)
2. Specific enthalpy  $h$  (FN.HBW)

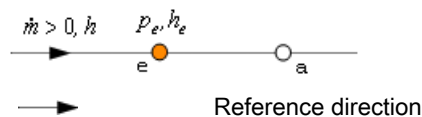
At its inputs, the pressure calculated by the flownet solution procedure for the external node is returned to the components. If discharge from the flownet is set with negative mass flow, then the specific enthalpy of the medium is returned to the component:

1. Pressure  $p_e$  (FN.PRESSURE)
2. Specific enthalpy  $h_e$  (FN.HFW)

The figure below illustrates the signal direction for data exchange between the component and the flownet solution procedure.



Mass flow nodes operate in the flownet like internal nodes with mass inflow or outflow. These nodes are therefore treated as internal nodes in the flownet, which means an internal node with additional inflow or outflow is created in the flownet for an external node. The pressure returned to the component  $p_e$  therefore corresponds to the pressure in this internal node, and the specific enthalpy  $h_e$  corresponds to the specific enthalpy of this internal node.



### Connector type FLN4 for internal nodes

The variables for pressure, specific enthalpy and density calculated by the flownet solution procedure for an internal node can be used in a component via a connector of type  $FLN4$ . The topological description of the component type is completed as follows for a connector with the name  $FN$ :

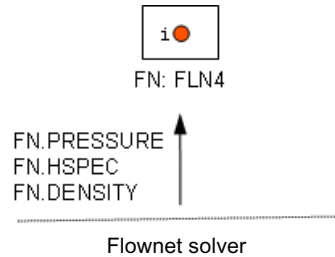
```
INTERNAL_NODE i : FN;
```

The connector must always be defined in the **IN direction**. It connects the component to the flownet to map the variables that are relevant for the internal node  $i$  from the flownet solution procedure to the component.

Name	Connection type	Direction	Dimension
FN	FLN4	IN	1

The variables calculated in the node  $i$  are provided by means of the connector inputs:

1. Pressure  $p_i$  (FN.PRESSURE)
2. Specific enthalpy  $h_i$  (FN.HSPEC)
3. Density  $\rho_i$  (FN.DENSITY)



### Connector type FLN5 for parameters of a flownet

Parameters of a flownet can be exchanged between a component and the flownet solution procedure via a connector of type *FLN5*. This connector can be assigned to any flownet object: a branch object or an internal or external node. The topological definition of the object should be completed, for a connector with name *FN* as per

- FROM a TO b : FN;
- INTERNAL\_NODE i : FN;
- EXTERNAL\_NODE e : FN;

The signals of connector type *FLN5* are listed in the table below. You can find their meaning in the mathematical model of the flownet in the following section: Parameter assignment of flownets (Page 467).

---

#### Note

The relationships used for calculation of the flownet variables, such as temperature, are defined via the *MEDIUM* parameter for the relevant medium. This can only be done during initialization of the simulation. Changes made to the parameters while the simulation is running have no effect.

---

Table 7-43 Signals of connector type FLN5

Signal	Meaning
MEDIUM	Flownet medium: MEDIUM := 0 for water/steam, MEDIUM := 1 for ideal gas, MEDIUM := 2 for liquid; must not be changed while the simulation is running
CG	Specific compression modulus $K/M$ for water/steam medium with density $\rho < 500 \text{ kg/m}^3$ and for ideal gas medium
CL	Specific compression modulus $K/M$ for water/steam medium with density $\rho \geq 500 \text{ kg/m}^3$ and for liquid medium
MG	Thermal factor $1/M$ for water/steam medium with density $\rho < 500 \text{ kg/m}^3$ and for ideal gas medium
ML	Thermal factor $1/M$ for water/steam medium with density $\rho \geq 500 \text{ kg/m}^3$ and for liquid medium
P_INIT	Initial value for the pressure in the internal nodes of the flownet
H_INIT	Initial value for the specific enthalpy in the internal nodes of the flownet
DENSITY	Density of the medium in the flownet when liquid is set as the medium
T_ENV	Ambient temperature
C_ENV	Heat transfer factor $c = \alpha A$ for heat exchange in the internal nodes of the flownet with the environment
L_CR	Specific heat capacity $c_p$ for liquids as medium
IG_R	Specific gas constant $R_s$ for ideal gas as medium
IG_CR	Specific heat capacity $c_p$ for ideal gas as medium
ST	Binary signal; if ST := "True", then a linear transfer is applied for parameters CL, CG and ML, MG for the water/steam medium
AL	Momentum factor A for branches in the flownet

Depending on the direction of the connector *FN* its signals are inputs or outputs. Depending on the direction, flownet parameters can be set or used in component. If the connector is defined with direction *OUT*, the variables set in the component are provided to the flownet as parameters. These parameters are available in the component for evaluation, if the connector is defined with direction *IN*.

### Connector type FLN6 for parameter assignment of a branch

The dynamics of the flow rate can be assigned parameters individually for each branch. To do this, a connector *FN*, via which the momentum factor A is transferred to the flownet solution procedure, must be added to the topological definition of the branch object as follows:

```
FROM a TO b: FN;
```

If a connector of type *FLN6* is defined with direction *OUT*, then the momentum factor A for the branch containing the object is transferred to the flownet solution procedure via this connector.

Name	Connection type	Direction	Dimension
FN	FLN6	OUT	1

If the connector is defined with direction *IN*, then the momentum factor A for the branch is transferred to the branch object by the flownet solution procedure.



### Connector type FLN7 for parameter assignment of an internal node

Each internal node can be assigned parameters individually. A connector via which the parameters are transferred to the flownet solution procedure must be added to the topological definition of an internal node. For a connector *FN* the definition is completed as follows:

```
Internal_Node i: FN;
```

*FN* is always created as an **output** of type *FLN7*.

Name	Connection type	Direction	Dimension
FN	FLN7	OUT	1

The variables listed in the table below are transferred to the flownet solution procedure for the internal node. You can find their meaning in the mathematical model of the flownet in the following section: Parameter assignment of flownets (Page 467).

Table 7-44 Signals of connector type FLN7

Signal	Meaning
CG	Specific compression modulus $K_i / M_i$ for water/steam medium with density $\rho < 500 \text{ kg/m}^3$ and for ideal gas medium
CL	Specific compression modulus $K_i / M_i$ for water/steam medium with density $\rho \geq 500 \text{ kg/m}^3$ and for liquid medium
MG	Thermal factor $1 / M_i$ for water/steam medium with density $\rho < 500 \text{ kg/m}^3$ and for ideal gas medium
ML	Thermal factor $1 / M_i$ for water/steam medium with density $\rho \geq 500 \text{ kg/m}^3$ and for liquid medium
P_INIT	Initial pressure value
H_INIT	Initial value for specific enthalpy
T_ENV	Ambient temperature
C_ENV	Heat transfer factor $c_i = \alpha A$ for heat exchange with the environment
L_CR	Specific heat capacity $c_p$ for liquids as medium
IG_R	Specific gas constant $R_s$ for ideal gas as medium
IG_CR	Specific heat capacity $c_p$ for ideal gas as medium

#### 7.2.4.3 Constants and functions

You can use various constants and functions when creating your own flownet components.

#### Constants

The available constants are listed in the table below.

Table 7-45 Constants for flownet components

Name	Data type	Value	Description
_GRAVITY	analog	9.81	Gravitational constant (gravity)
_T0	analog	273.15	Zero point temperature

Functions

Functions for calculating state variables are available for flownet components with water/steam medium. The available state variables and units are summarized in the table below.

Table 7-46 State variables for water/steam

Variable		Unit
$\rho$	Pressure	bar
$\rho_s$	Saturated steam pressure	bar
$T$	Temperature	°C
$T_s$	Saturated steam temperature	°C
$h$	Specific enthalpy	kJ/kg
$h'$	Specific enthalpy of saturated water	kJ/kg
$h''$	Specific enthalpy of saturated steam	kJ/kg
$\rho$	Density	kg/m <sup>3</sup>
$\rho'$	Density of saturated water	kg/m <sup>3</sup>
$\rho''$	Density of saturated steam	kg/m <sup>3</sup>

All state functions *FNAME* return a state variable *ZVAL* as an analog value. Calls are performed as follows:

$$ZVAL = \_WaterSteam.FNAME(PARAM1, PARAM2).$$

The available state functions are described in the table below.

Table 7-47 State functions for water/steam

ZVAL	FNAME	PARAM1	PARAM2	State function
$\rho$	rph	$\rho$	$h$	$\rho = \rho(\rho, h)$
$T$	trh	$\rho$	$h$	$T = T(\rho, h)$
$T_s$	tvp	$\rho_s$	-	$T_s = T(\rho_s)$
$\rho_s$	pvt	$T_s$	-	$\rho_s = \rho(T_s)$
$\rho'$	rsvt	$T_s$	-	$\rho' = \rho'(T_s)$
$\rho''$	rssvt	$T_s$	-	$\rho'' = \rho''(T_s)$
$\rho'$	rsvp	$\rho_s$	-	$\rho' = \rho'(\rho_s)$
$\rho''$	rssvp	$\rho_s$	-	$\rho'' = \rho''(\rho_s)$
$h'$	hsvt	$T_s$	-	$h' = h'(T_s)$
$h''$	hssvt	$T_s$	-	$h'' = h''(T_s)$
$h'$	hsvp	$\rho_s$	-	$h' = h'(\rho_s)$
$h''$	hssvp	$\rho_s$	-	$h'' = h''(\rho_s)$

The saturated steam pressure  $\rho_s$  and saturated steam temperature  $T_s$  are limited for the saturation functions as follows:

$$0 \leq T_s \leq 373.94 \text{ °C and } 0.0065 \text{ bar} \leq \rho_s \leq 220.64 \text{ bar.}$$

The relevant limit is set for values outside this range.

The values for pressure  $p$  and specific enthalpy  $h$  are limited to the following ranges for the state function  $\rho = \rho(\rho, h)$ :

$0.007 \text{ bar} \leq p \leq 490 \text{ bar}$  and  $20 \text{ kJ/kg} \leq h \leq 3998 \text{ kJ/kg}$

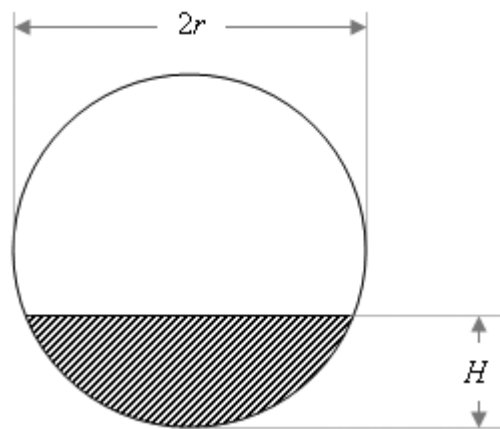
For the state function  $T = T(p, h)$ , the specific enthalpy is limited to the range

$9 \text{ kJ/kg} \leq h \leq 4158$

$\text{kJ/kg}$ . The limits for the density  $\rho$  depend on the enthalpy value: The lowest valid density value is  $0.0012344 \text{ kg/m}^3$  (with  $h = 4160 \text{ kJ/kg}$ ); the highest valid density is  $1045.239 \text{ kg/m}^3$  (with  $h = 96 \text{ kJ/kg}$ ).

The auxiliary function *Cylniv* can be used to calculate fill levels in horizontal cylinders. For a cylinder of length  $L$  and radius  $r$  with specified volume  $V$ , it calculates a fill level of  $H$ . This can be called using:

$H = \text{\_Utilities.Cylniv}(V, L, r)$ .



## User-defined functions

The FLOWNET library provides you with three different media: water/steam, ideal gas and liquid. The medium to be used is set in the signal *MEDIUM* of the connection type *FLN5* with a corresponding ID (0, 1, or 2). You can find additional information on this in the section: Connector type *FLN5* for parameters of a flownet (Page 551).

The state equations for density and temperature that are defined for the relevant medium are accordingly applied by the flownet solution procedure.

With the FLOWNET library you also have the option of performing flownet simulations with other media besides the pre-defined media. You have to provide the media-specific state equations for the calculation of density and temperature in the nodes by means of a “global function”.

Such a global function must consist of a .NET assembly that is named *Userdefined.dll* and which has its “Assembly Name” specified as “Userdefined” in the project settings. It must have the namespace *Userdefined* and a public class named *FlownetFunctions*. This class must contain two public static functions for calculation of the state variables:

- *public static double trh(long m, double r, double h)*
- *public static double rph(long m, double p, double h)*

The function *trh* has to calculate the temperature from the density  $r$  and the specific enthalpy  $h$ ; the function *rph* has to calculate the density from the pressure  $p$  and the specific enthalpy  $h$ . The units for the variables are as follows:

- Temperature in °C,
- Density in kg/m<sup>3</sup>
- Pressure in bar, and
- Specific enthalpy in kJ/kg.

The medium has to be specified by a freely selected negative number. This negative media key number forms the first transfer parameter  $m$  of both functions. In your own functions you only need to consider the user-defined negative media key numbers; if the value given is  $\geq 0$  computation will be performed by SIMIT, and your functions will not be called in this case.

This means you have the option of defining any desired media in these two functions and of using them in a simulation project. Note that in your user-defined functions, you only need to pay attention to negative media key numbers. For non-negative media key numbers, the functions that are included in SIMIT are used. Your user-defined functions are not called by SIMIT in this case.

The scope for your user-defined functions might be set up as follows:

```
namespace Userdefined
{
    public class FlownetFunctions
    {
        public static double trh(long m, double r, double h)
        {
            switch (m)
            {
                case -1:
                    // Calculation for media with key number -1 ...
                    return 0.0;
                case -2:
                    // Calculation for media with key number -2 ...
                    return 0.0;
                default:
                    // Should not occur!
                    return 0.0;
            }
        }

        public static double rph(long m, double p, double h)
        {
            switch (m)
            {
                case -1:
                    // Calculation for media with key number -1 ...
                    return 0.0;
                case -2:
                    // Calculation for media with key number -2 ...
                    return 0.0;
                default:
                    // Should not occur!
                    return 0.0;
            }
        }
    }
}
```

To assign a specific medium to a flownet, set the relevant media key number in the *MEDIUM* signal of the *FLN5* connector type; in the case of user-defined media this means setting the relevant negative value.

---

**Note**

If you set a negative key number for the medium without providing the necessary functions for the calculation of density and temperature, the flownet solution procedure assumes that the density and temperature are zero.

---

You can call these two functions using

`_FlownetFunctions.trh(m, r, p)`

or

`_FlownetFunctions.rph(m, p, h)`

even in your own component types. Moreover you can implement additional functions in the class *FlownetFunctions* and use them by means of a corresponding function call in your component types.

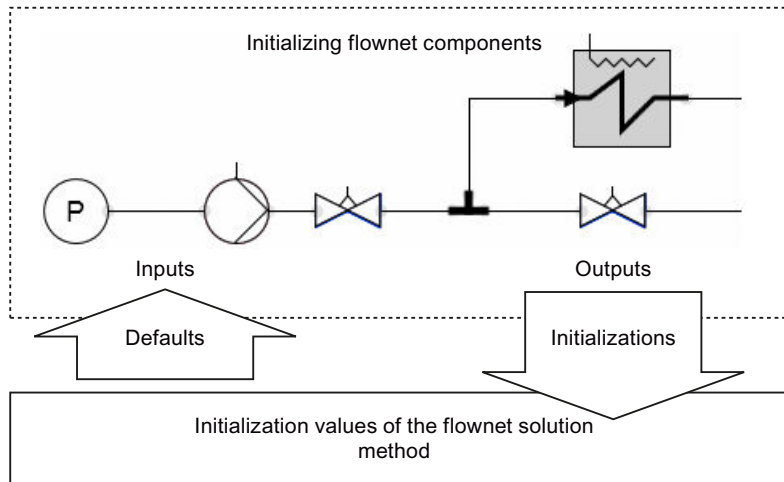
You must store the assembly in the SIMIT workspace, in the subfolder *GlobalFunctions*. The assembly will then automatically be copied to each new project and will therefore be archived with your project.

**Note**

Modifications to the assembly do not affect existing simulation projects. The modified assembly affects newly-created projects. If you want to use the modified assembly in projects that already exist, copy it into the *globalFunctions* folder in these projects.

**7.2.4.4 Initialization of flownet simulations**

The initialization of flownet simulations occurs in two steps. When the simulation is started up, first the components are initialized, which means the simulation model created from the connected components is initialized. Then the flownet solution procedure is initialized. The flownet solution procedure uses the values that are initialized in the components and passed to it via connectors of type FLN2 through FLN7.



As none of the flownet solution procedure variables are available during initialization of the components, it is advisable to pre-assign the inputs of the component types with suitable values. We recommend values that are consistent with the default values of the flownet solution procedure.

## 7.3 The CONTEC library

### 7.3.1 Introduction

The *CONTEC* library is an extension of SIMIT, which provides component types for creating simulations of conveyor systems. By linking components of this library, you can create a model of a conveyor system in SIMIT to simulate the transport of objects in this system. The *CONTEC* library provides a special solution procedure for this purpose for use in SIMIT. Once the simulation has started, the solution procedure continuously calculates the positions of the objects in the modeled conveyor system.

The *CONTEC* library is only suitable for simulating conveyor systems for individual objects; bulk material conveyor systems cannot be simulated. In addition, the simulation is limited to conveyor systems in which the transportation routes for the objects are fixed by the arrangement of the handling equipment. Conveyor systems with autonomously operating vehicles, which means vehicles that determine their own transportation route, are therefore excluded.

SIMIT provides a real-time simulation for the virtual commissioning of user programs for automation systems. Likewise, the simulation of conveyor systems with the *CONTEC* library is always used in a closed circuit with real or simulated automation systems. This sort of simulation with SIMIT is therefore fundamentally different from the material flow simulations used for planning and designing conveyor systems. There, the material flow in a conveyor system is simulated on the basis of a pre-defined control strategy, whereas here the control strategy is translated into an actual automation system, which includes a simulation of the purely mechanical elements of the conveyor system created with SIMIT. To differentiate it, this type of simulation is therefore referred to as material handling simulation in this manual.

As is usual in SIMIT, material handling simulations are easy to create with the *CONTEC* library, using components from the graphical user interface. The emphasis in implementing the component types in the library was on the simple creation and parameter assignment of the simulation model and on the stability of the model in the simulation, rather than on a detailed simulation of the mechanical aspects of conveyor systems. Material handling simulation is therefore based on a movement model that simulates the movement of objects along paths at a pre-defined speed, the paths being determined by the handling equipment. The weight of the object and other factors influencing movement, such as friction, for example, are disregarded.

Two types of conveyor systems can be simulated with the component types provided in the *CONTEC* library:

- Conveyor systems with rail-mounted vehicles  
such as electrical overhead monorails, ground transport systems, etc. and
- Non-vehicular conveyor systems  
such as chain, roller and belt conveyor systems, etc.

*CONTEC* also includes component types for incorporating identification systems such as RFID, Moby, barcodes, etc. into the simulation.

You can use SIMIT Ultimate to create your own library components for material handling simulations and thus expand your material handling library. You can use the special

CONTEC solution procedure for material handling simulation in the component types by means of special connection types.

---

**Note**

When the simulation starts, the system checks whether your SIMIT installation has a license for the CONTEC library. If there are material handling components in your simulation project, which means components that use the material handling solution procedure, then the simulation can only be run if you have a material handling license.

---

### 7.3.2 Material handling simulation

Material handling describes the technology used for moving objects in any direction over limited distances. Objects can generally be moved in any spatial direction. Handling equipment refers to the individual machines that are used to move the object in the conveyor system.

The CONTEC material handling library can be used to create simulation models of a conveyor system and to run simulations with them. The handling equipment is subject to the following restrictions:

- All transportation routes are defined as linear paths.
- Transportation routes and possible branches must be predefined as a structure of the conveyor system for a simulation.

In addition, the simulation of objects is restricted as follows:

- Only individual items can be considered as objects, not bulk goods.
- All objects are represented by a box-shaped outline.
- Other than its dimensions, no other physical properties of the object (for example weight, friction, etc.) are taken into consideration.

The component types contained in the CONTEC library can be divided into two categories:

- Object component types and
- Equipment component types

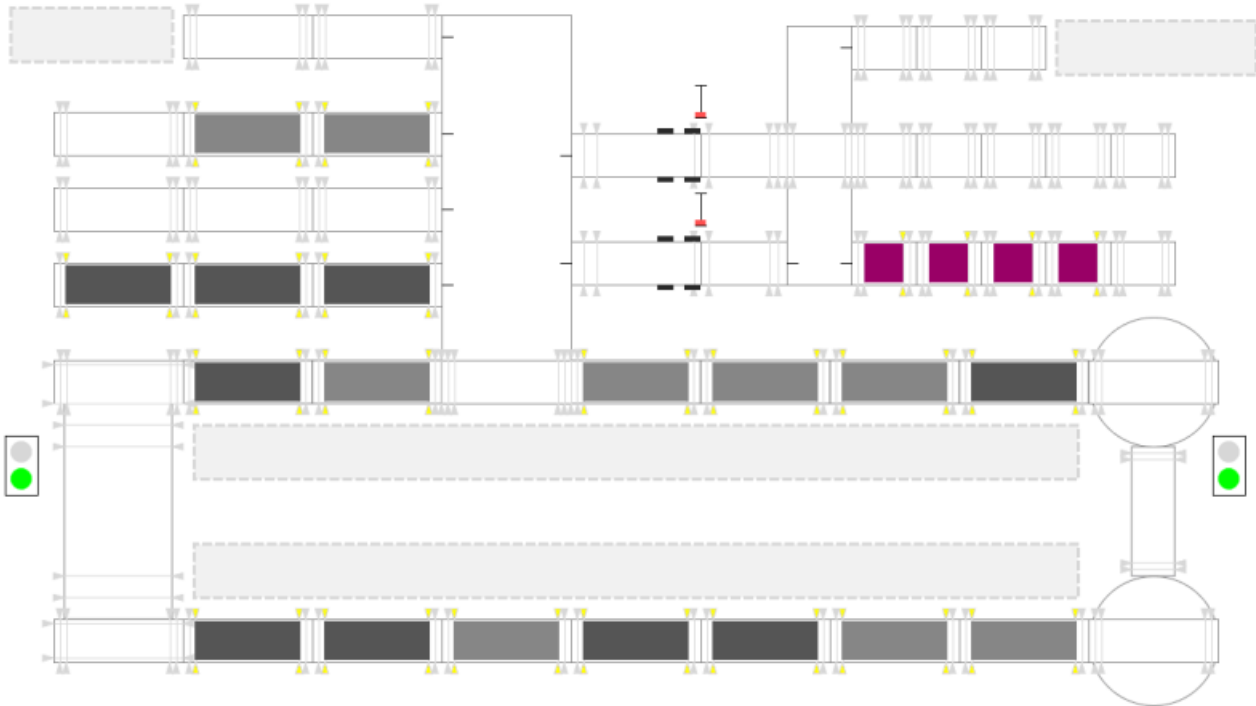
As is usual in SIMIT, equipment component types are added to charts as components using their symbol. The symbols for these component types are designed so that when put them together they create a scaled layout of the conveyor system, as shown for example in the figure below. The resulting system model then only needs to be provided with appropriate parameters and linked to the automation.

Along with the system model, the objects defined for a simulation project are stored in table form. There are symbols for object component types, too, with which the individual objects can be displayed correctly in the conveyor system layout when the simulation is running.

Material handling simulation is based on a special solution procedure that is configured and programmed by means of the individual components of the simulation model. This solution procedure is based on a movement model which allows the position of the objects on the paths defined by the handling equipment to be determined at any moment in the simulation. The solution procedure takes account of the fact that objects can be held up, and it uses the position and dimensions of the objects to activate sensors in the simulation.



The movement of an object on a path can be initiated in the simulation by both the equipment components and the object components. In this way it is possible to simulate both conveyor systems such as roller conveyor systems, in which the drive acts on the equipment, and rail-mounted conveyor systems, in which the drive acts on the vehicle, which means the object.



### 7.3.2.1 Principles of conveyor technology simulation

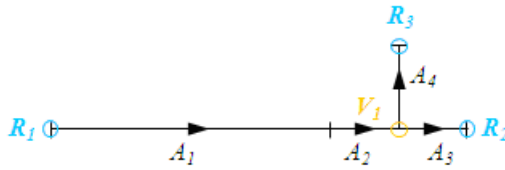
The material handling simulation method is based on a movement model that assumes that the objects move along linear paths, the possible paths being determined by the handling equipment. The way in which the handling equipment is configured defines a conveyor system network that represents the possible transportation routes for the objects in a conveyor system.

Complex motions can be realized by self-defined components and connected to the system in linear pathways. You can find additional information on this in the section: Connector type MT8 for transferring objects at boundary points (Page 672).

A network consists of individual sections and branches. It can be inherently closed or open. In an inherently closed network both ends of each section connect to a branch, whereas in an open network at least one end of one section is not connected to a branch. This forms a boundary point of the network. The smallest possible conveyor system network consists of one section.

A network section is made up of one or more segments. The geometry of a segment can consist of either a straight path of a given length or a circular arc of a given radius and angle. In addition, every segment has a direction. The direction of a segment is the reference direction for the movement of the object over this segment. It is defined in such a way that a positive speed value moves the object in the reference direction, while a negative speed value moves the object against the reference direction.

By way of example, the figure below shows a conveyor system network consisting of three sections or four segments  $A_1$  to  $A_4$  and one branch  $V_1$ .



This section is clearly an open conveyor system network with the three boundary points  $R_1$ ,  $R_2$  and  $R_3$ . If the object crosses the boundary of a conveyor system network it is lost, which means it is removed from the handling equipment and returned to the material list where it is once more available for use. Components can remove objects from or feed them into the conveyor system network at the boundaries by means of suitable connections to the solution procedure.

The special solution procedure of the material handling calculates the position of the objects at equidistant times, as is usual in SIMIT. It supplements the standard SIMIT solution procedure so that material handling component types can be used alongside other component types, for example, component types from the basic library. To enable data to be exchanged, material handling components are connected to the material handling solution procedure by means of specific connections. They receive values calculated by the solution procedure and send variables to the solution procedure via these connections.

For each segment in the conveyor system network, positions can be defined at which sensors detect the object as it passes. The material handling solution procedure then ensures that the relevant sensor signals are activated in the simulation according to the positions and dimensions of the objects.

### 7.3.2.2 Modeling of the objects

Material transport objects are modeled in SIMIT as component types. For each type, at least the size and the graphical representation of the object must be defined. Connectors, parameters, states and behaviors can also be defined for an object in its component type.

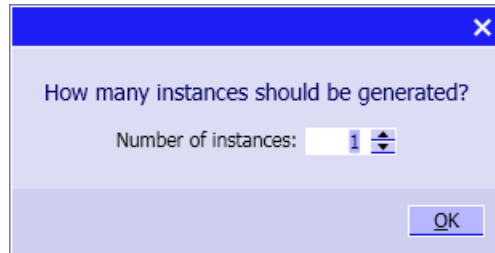
The link symbol is used where possible to represent the object in charts: if a link symbol is defined, this is used for the display; otherwise the basic symbol is used, with any connectors on the symbol being hidden.

To simulate objects the available stock of object components has to be defined in the SIMIT project. For this, select "New list" node in the project view under the "Material" project folder. This creates a material list, i.e. a list of the object components available in the simulation.

The editor for creating the material list opens:

Boxes				
Name	Width	Height	Depth	Type
Box#1	1200	800	0	Box

Here you can define which object components are available in the simulation. For each component type you can specify the number of instances to be formed. To create an instance, simply drag the component type you want from the *MATERIAL* directory of the *CONTEC* library into the editor. To create more than one instance, hold down the Alt key as you drag the component type. A query dialog will appear in which you can specify the number of instances, as shown in the figure below:



In the Material directory you can also create multiple material lists and sort them into different folders, if necessary.

In order to place object components on a specific section of the simulated transport network for the start of the simulation, you can assign one or more object components of the same type from the material lists in the project to the component simulating the section.

You can find examples for this in the following sections:

- Rail-S4 – Straight rail with four sensors (Page 578)
- Conveyor-S4 – Straight conveyor with four sensors (Page 594)

If object components have a behavior model (behavior description), this is executed at the start of the simulation in the same way as it would be if the object components were placed on a chart like other components. The simulation of the behavior of an object defined by that description is entirely independent of the treatment of that object in the solution procedure for the material handling simulation.

Once the simulation has started, each object defined in the material list is given a unique identification number which is automatically set by SIMIT and displayed in the material lists.

Boxes						
Obj-ID	Name	Width	Height	Depth	Type	
1	Box#1	1200	800	0	Box	
2	Box#2	1200	800	0	Box	
3	Box#3	1200	800	0	Box	

If an object is detected at a sensor during the simulation, the solution procedure returns the ID of the detected object component.

### 7.3.2.3 Modeling the conveyor system network

Conveyor system networks are made up of segments, branches and boundaries. The *CONTEC* library provides various component types for modeling the conveyor system network. These component types represent elementary handling equipment types and therefore encapsulate corresponding structures consisting of segments, branches and boundaries. Thus the complexity of these subnetworks modeled in the component types can differ. For example, in a component type representing a straight conveyor only a single segment is modeled. By contrast, component types representing more complex handling equipment, like a 90-degree transfer, for example, have a model made up of multiple segments and branches.

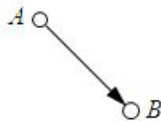
The individual component types have connectors representing end points of segments or branch points. If the connectors of two components of these types are connected on charts, the subnetwork models of the two components are connected accordingly. By connecting them to other components the complete conveyor system network of a conveyor system is ultimately constructed.

For every segment the geometry (with coordinates) and the reference direction have to be specified. Optionally, the positions of stoppers and sensors may also additionally be defined for a segment. For each branch it must be defined which segments it connects, and these must be provided with corresponding information about their positions in the conveyor system network.

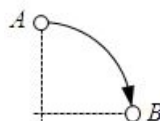
#### Modeling segments

A segment is modeled as a line. It is defined by its two end points, the geometry and the direction of the segment. The geometry can take the form of a straight line or an arc. It is defined by the position of its two end points *A* and *B* and its angle. If the angle is not zero, there are two possible segments which can be differentiated by the sign of the angle: a positive angle means that the center point of the arc is located to the right of the segment in the counting direction, while a negative angle means that the center point of the arc is located to the left of the segment. The radius of the arc is determined by these three variables. The direction of the segment corresponds to the reference direction for transporting the objects.

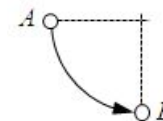
Angle 0



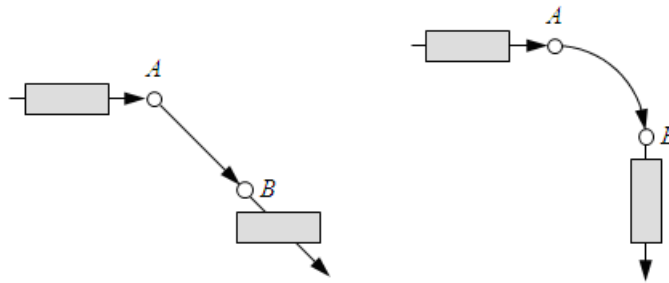
Angle +90



Angle -90



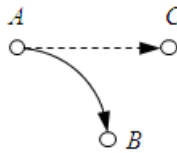
The object being transported over a segment is rotated by the angle of the segment. This means on straight segments the object is transported without being rotated; the absolute orientation of the object is maintained. The figure below shows the transport of an object over a straight and a curved segment:



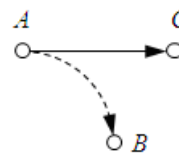
For each segment a speed is defined at which the objects are transported over that segment. Positive speed values cause the object to be transported in the reference direction, while negative speed values cause it to be transported against the reference direction. The inherent speed of an object is added to the segment speed where applicable.

A state is assigned to each segment for movement in the reference direction and for movement against the reference direction. This state identifies a segment as "active" or "inactive" in the corresponding direction. Inactive segments are regarded as non-existent in the material handling solution procedure. This means that objects cannot be moved on these segments, and the positions of objects already located on these segments are not updated. In addition, sensors on inactive segments are not updated. The change in state of segments is used to specify an explicit transportation route at branches. The figure below shows an example of how a switch function can be modeled by alternately activating and deactivating the two segments in the reference direction:

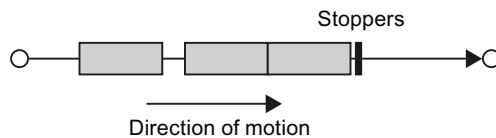
Segment AC inactive



Segment AB inactive

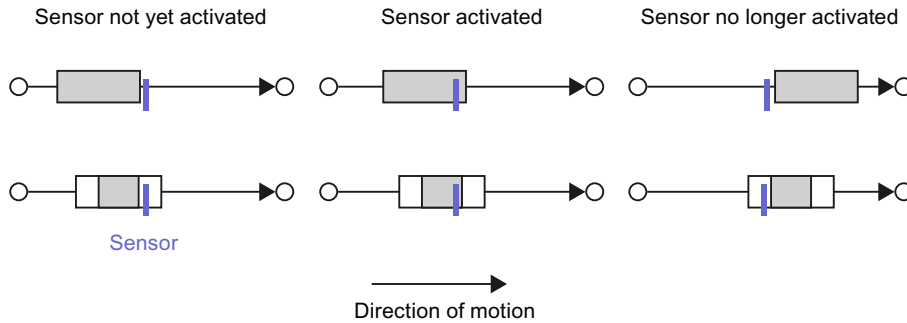


Positions at which the transport of an object can be blocked can be defined on a segment. The state of a stopper position can be set to "active" or "inactive". If a stopper is active, all objects are stopped at this position and all downstream objects are held up accordingly as shown in the figure below by way of example:



In addition, positions at which an object can be detected can be defined on a segment as sensor positions. At each of these sensor positions the identifier (ID) of the object covering the position is then recorded in the material handling solution procedure. The detection range for the objects can also be defined for each sensor position. This range is defined as a percentage of the object size and is assumed to be symmetrical with respect to the center of the object. The figure below illustrates the detection of an object for two different detection ranges: the top row shows the activation of a sensor for the full detection range (100%) of the object; the bottom row shows the activation of a sensor for half the detection range (50%). The detection

range for the object is shown in gray in each case and movement is assumed to be from left to right.



Component types in which only one segment with sensors is modeled include, for example, the straight rail (*Rail-S4*) and the straight conveyor (*Conveyor-S4*).

### Modeling branches

Because all sections are assumed to be linear, a branch is a point in the conveyor system network at which more than two segments connect.

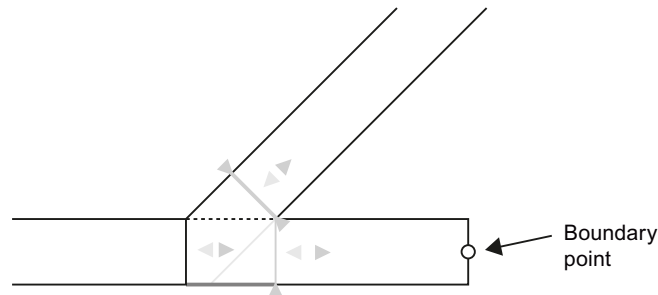
In the subnetwork of a component type a branch can either be shown as a connector, which means a shared end point of multiple segments (as shown in figure below on the left), or it can be located within a subnetwork (as shown in figure below on the right). In the first case the position of the branch point is given directly by the position of the connector, whereas in the second case the branching position is defined relative to the component.

(a) Branch at a connector	(b) Branch in a component
<p>Diagram (a) shows a node <math>A_1</math> on the left. A dashed rectangle encloses a branching point where three lines meet: one horizontal line to <math>A_1</math>, one diagonal line to <math>A_2</math>, and one diagonal line to <math>A_3</math>.</p>	<p>Diagram (b) shows a node <math>A_1</math> on the left. A dashed rectangle encloses a branching point where three lines meet: one horizontal line to <math>A_1</math>, one diagonal line to <math>A_2</math>, and one diagonal line to <math>A_3</math>. A black dot labeled <math>K_1</math> is located at the branching point.</p>

At branch points the transportation route must be explicitly defined at every moment of the simulation. In other words, no more than two of the segments that connect at a branch may be active as a transportation route at any given time. Therefore activation of the segments must be included as a function in the component type.

## Modeling boundaries

Boundaries are end points of a segment that are not connected to other segments. For a component like a straight conveyor, for example, a boundary of the conveyor system network is created if a connector of this component is not connected to other components. In the simulation, if an object moves beyond such a boundary point it is removed from the conveyor section and returned to the material list. A message to that effect appears in the message line.



Boundary points can also be defined as end points in the subnetwork of component types. The behavior of an object passing this boundary point can then be freely defined in the component type using suitable functions. For example, the position of the object can be freely calculated to simulate complex movements. You can find examples for such components in the following sections:

- Turntable-R60 – Turntable (Page 616)
- TransferCarriage – Transfer carriage (Page 621)

### 7.3.2.4 Special features of conveyor technology simulation

#### Placing and removing objects

Objects that are defined in the available material stock for the simulation project can be placed on any segment in the simulation. They are placed at the start of the segment in accordance with the specified reference direction (FROM-TO). Objects are always positioned relative to their geometric center.

Objects can also be removed from the network. There is a corresponding system call for placing and removing objects, which allows this function to be simulated in component types.

#### Simulating the holdup behavior

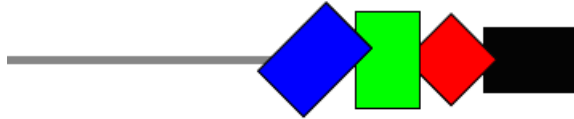
##### Holdups on straight conveyor sections

Modeling of the holdup behavior has been simplified to a great extent. This means the positions of held-up objects in the simulation do not necessarily correspond to the actual positions in all cases.

The holdup behavior on straight sections is correct for all objects that are either not rotated at all or are rotated in multiples of 90°.



Objects with other rotation angles  $\phi$  are also held up. However, as their position can only be calculated approximately, this leads to overlaps.



The effective width  $\tilde{b}$  of an object that is used as a simplification to simulate the holdup is calculated using the following formula:

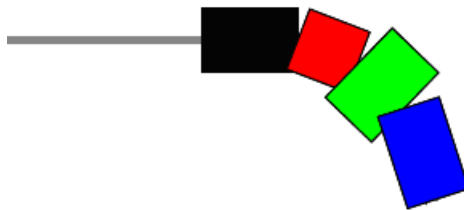
$$\tilde{b} = \frac{\phi h + (\pi - 2\phi) b}{\pi}$$

where  $h$  denotes the height and  $b$  the width of the object.

### Holdup in trends

Modeling of the holdup behavior has been simplified to a great extent. This means the positions of held-up objects in the simulation do not necessarily correspond to the actual positions in all cases.

In trends, too, the position of held-up objects is only an approximation. The held-up objects overlap, as shown in the figure below.

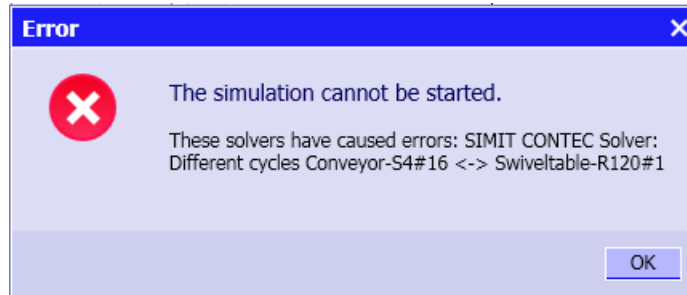


The overlaps are caused by the fact that the bend of the trend is disregarded in the distance calculation.



### Time slice assignment for components

All components forming a cohesive conveyor system network must be assigned to the same time slice. This means all components that simulate handling equipment and are directly connected to one another via connectors or signal lines must be assigned to the same time slice. Otherwise the simulation model of a cohesive conveyor system network will be formed from multiple sub-models with differing cycle times. The simulation start request will then be denied and a corresponding error message displayed.



#### 7.3.2.5 Scalability

Unlike in the standard library, the dimensions of components play a decisive role in the conveyor technology library because they directly indicate the length of conveyor sections or the dimensions of objects. In accordance with standard engineering practice, millimeters [mm] are used throughout as the length unit, while speeds are given in meters per second [m/s].

#### Scalability of conveyor sections

To enable conveyor systems of differing sizes to be mapped on charts with a reasonable resolution, a scale can be assigned to each chart.

Diagramm		
General	Property	Value
	Name	Diagramm
	Width	16000
	Height	14000
	Scale	1 pix : 20 mm
	Background Image	... X

The following scales are possible:

- 1 pix : 1 mm,
- 1 pix : 5 mm,
- 1 pix : 10 mm,
- 1 pix : 20 mm,
- 1 pix : 50 mm,

1 pix : 100 mm.

**Note**

If you have licensed the material transport library for your SIMIT installation, the sizes of all components, including components from the basic library, are reproduced at the specified scale. For basic library components you can retain the intended sizes by choosing a scale of 1 pix : 1 mm for charts containing basic components.

To avoid having to set the scale individually for each chart, you can choose a default setting for the entire project in the property view of the Project Manager.

TEST	
Property	Value
Project Location	E:\40_PROJEKTE\TEST\TEST.simit
Project Version	AA12320-808293-0.87 (*)
Readonly	<input type="checkbox"/>
Default Scale	1 pix : 50 mm
Cycle 1 [ms]	50
Cycle 2 [ms]	100

This scale is used as the default for all new charts that you create.

**Note**

Setting a default scale in the Project manager does not change the scale of any existing charts.

Note that changing the scale of a chart does not change the displayed size of a component, but rather its effective dimensions in millimeters.

**Scalability of objects**

Objects are shown in the material lists of a SIMIT project with their absolute size (width and height) in millimeters. This means objects are represented on charts in differing sizes, depending on the chosen scale of the chart.

Boxes				
Name	Width	Height	Depth	Type
Box#1	1200	800	0	Box

The dimensions of an object are pre-defined in the component type. So when you add an object to the material list, it is added with the width, height and depth defined by the component

type. You can then change the dimensions for each object in the material list, provided that the component is scalable. The object types provided in the CONTEC library are all scalable.

---

**Note**

Note that *width* and *height* refer to the dimensions of the object as viewed from above, as shown in the diagram. The *depth* is the size in the third dimension, which is not shown. This definition was chosen so as to retain the meaning of the height and width of components and graphics on a chart. The component types provided in the material handling library disregard the depth of the object. With the component types available in the library, objects of any depth are detected by sensors solely on the basis of their width and height. For that reason the depth of objects is set to zero by default. The depth of objects only becomes important if you create a custom material handling component type that evaluates the depth of an object, for example a height check, and use it in your material handling simulation.

You can find additional information on this in the section: [Creating custom component types for material handling simulation \(Page 662\)](#).

---

If you want to set the dimensions of several objects to the same value, you can copy the value for one object, in other words from one row of this table to any number of others. Follow the steps outlined below:

- Select the row containing the required object value.
- Right-click on the cell containing the value to be copied and select "Copy Cell" in the shortcut menu.

- Select the rows to which you want to copy the value.
- Right-click on the column to which you want to copy the value and select "Paste to Cell" in the shortcut menu.

The screenshot shows a table titled "Boxes\*" with columns: Name, Width, Height, and Depth. The "Box#1" row is selected. A context menu is open over the "Height" cell of "Box#1", showing "Copy Cell" and "Paste to Cell" options.

Name	Width	Height	Depth
Box#1			0
Box#2			0
Box#3			0
Box#4	1200	800	0
Box#5	1200	800	0

The screenshot shows a table titled "Boxes\*" with columns: Name, Width, Height, and Depth. Rows "Box#2", "Box#3", and "Box#4" are selected. A context menu is open over the "Height" cell of "Box#3", showing "Copy Cell" and "Paste to Cell" options.

Name	Width	Height	Depth
Box#1	1150	800	0
Box#2	1200	800	0
Box#3			0
Box#4			0
Box#5	1200	800	0
Box#6	1200	800	0
Box#7	1200	800	0
Box#8	1200	800	0
Box#9	1200	800	0

### 7.3.2.6 Generating the simulation of drives and sensors

As conveyor sections can be shown to scale on charts as a system layout using appropriate components, there is no point in also showing all additional input and output signals for these components in graphical form in these charts. This would compromise the clarity of the system layout. This means the only visible connectors on the component types in the CONTEC library are those used to connect them to each other to construct the system layout. Connectors for connecting components to the associated drive and sensor signals, and hence via the couplings to the controller signals, are implemented by means of an implicit assignment of the inputs. There are various options available for connecting components. They are explained below by reference to a simple rail.

### Manual connection of components

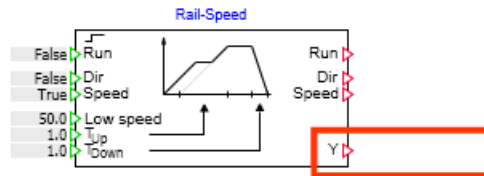
The component type of a rail is created in such a way that the component receives its percentage speed value as an input value from output *Y* of another component whose name is formed from the name of the rail and the suffix *Speed*. For example, the figure below shows a rail with the name *Rail*.



This means its *Speed* input is predefined so that it is implicitly connected to output *Y* of the component with the name *Rail-Speed*.

Name	Value/Signal
Speed  (\$) ▼	Rail-Speed Y

You can therefore create a chart which in this example contains a component with the name *Rail-Speed* and has an output *Y* which then defines the percentage speed for this rail.



Alternatively, you can also enter every other (analog) output of a component in the *Speed* input for the rail by changing the setting from (\$) to and entering the corresponding output signal, as shown by way of example in the figure below.

Name	Value/Signal
Speed   ▼	Rail-Speed Y

If you want the speed defined by the rail to be constant, you can also set the *Speed* input to 123 and enter the desired percentage directly.

Name	Value/Signal
Speed  123 ▼	100.0

If you want to be able to further process output signals for the rail sensors, you would likewise use implicit connections. You can use any components and assign their (binary) inputs accordingly, as shown by way of example in the figure below.

Name	Value/Signal
IN   ▼	Rail Sensor1

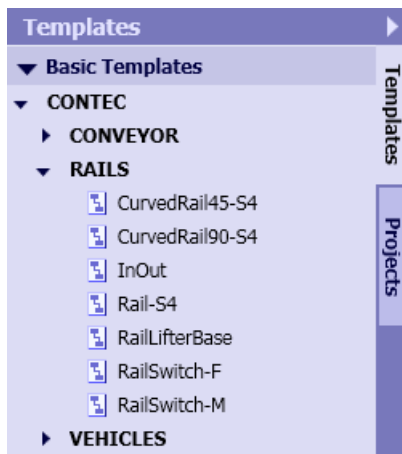
### Using templates

Templates can be used as a way of creating charts for drive and sensor simulation with minimal effort.

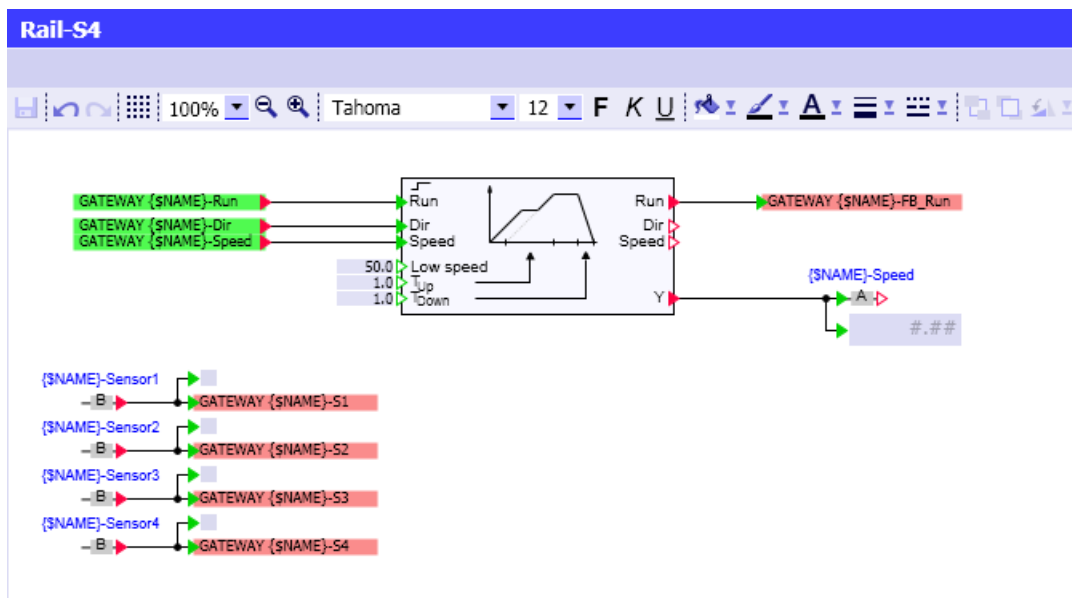
#### Note

Templates can only be used with the product variants SIMIT PROFESSIONAL and SIMIT ULTIMATE.

The material handling library includes suitable templates for connecting the drive and sensor signals for every component type of a conveyor section, as well as for the vehicles. The template names are the same as the names of the component types.



All templates include a component from the basic SIMIT library for simulating the drive. In addition, the template can also be used to connect the simulated sensors in the conveyor section to the controller via coupling signals. The figure below shows the template for the *Rail-S4* component type of a rail.



If you use an identification system that allows the symbolic names of the I/O addresses to be derived from the names of the conveyor sections, you can modify these templates accordingly. You can then obtain the full connection of the corresponding conveyor section to the controller via the coupling by instantiating the template.

To connect all conveyor sections in the simulation project to the controller with the minimum possible effort, use the "*Create device level*" function.

You have 3 options for calling this function:

- From the Portal view, select "Automatic model creation > Create device level".
- From the Project view, select the menu command "Automatic model creation > Create device level".
- From the shortcut menu of chart folder, select "Automatic model creation > Create device level".

You can use the "Create device level" function to create an instance of a template for every component in your simulation project that has a *TEMPLATE* parameter. The name of the template to be instantiated is determined using this *TEMPLATE* parameter. Using its *HIERARCHY* parameter the component can also define a folder hierarchy in which the template is instantiated.

The components of the *CONTEC* library are designed for use with this method. They contain the (additional) parameters *TEMPLATE* and *HIERARCHY*. If you have used *CONTEC* components in your *SIMIT* project, the "Create device level" function provides the associated charts for connecting the controller.

As a reference is usually made to the coupling signals in the templates for generating the device level, you can specify a coupling name in the import dialog. It can be entered under the variable name *GATEWAY*. If your simulation project already includes couplings, they are listed in the drop-down list.



**Note**

You have to adapt the names of the input/output signals in the templates to your identification system to gain the maximum benefit from this method.

The minimum size for a template is 20 x 20 pixels.

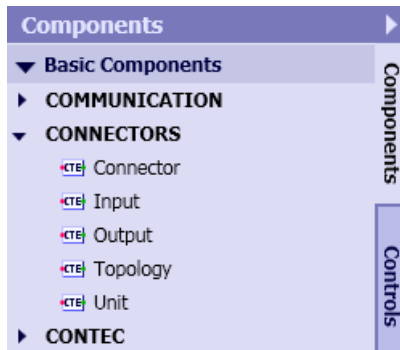


### 7.3.3 Components of the CONTEC library

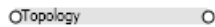
#### 7.3.3.1 The topological connector in the CONTEC library

The *CONNECTORS* directory of the SIMIT basic library includes a connector that can be used to create topological connectors for material handling components that transcend chart limits:

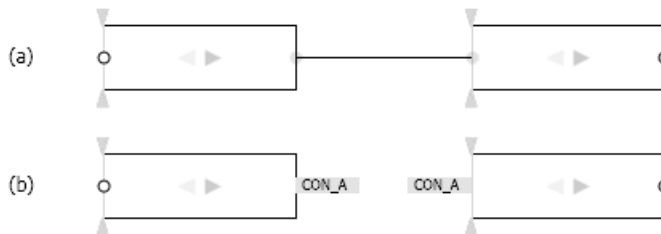
- the *Topology* connector.



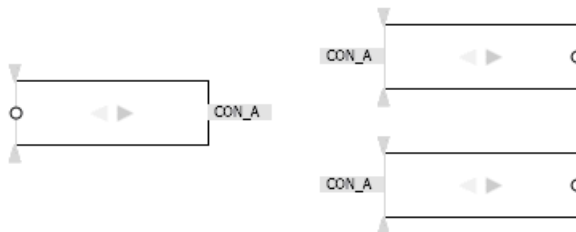
The *Topology* symbol is shown in the figure below:



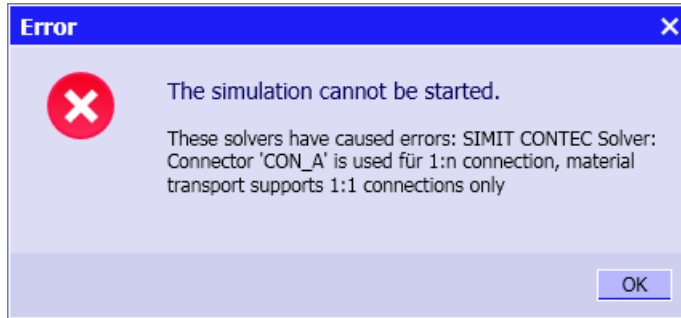
Using the *Topology* connector, a topological connection can be created between two or more material handling components. Under (b) in the figure below, two components are connected by the *CON\_A* connector. The connection is functionally identical to the direct connection of both components via a connecting line, as shown under (a) in the figure below.



The figure below shows three components connected by the *CON\_A* connector. This configuration is not valid, because with material handling components only two topological connectors can ever be connected to one another.

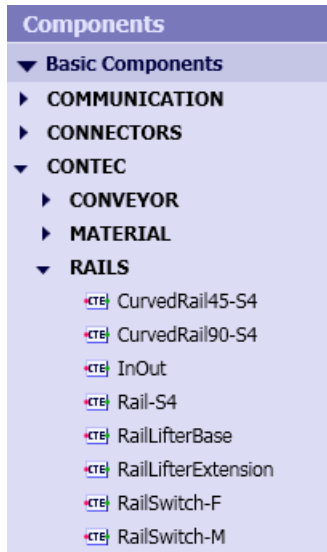


If you start a simulation project containing a configuration like this, an error message will appear as shown in the figure below and the simulation is aborted.



### 7.3.3.2 Component types for conveyor systems with vehicles

The *RAILS* directory of the CONTEC library contains component types for use in vehicular conveyor system simulations. In the broadest sense these types simulate "rails" on which vehicles can be placed as objects. The vehicles are either moved at the speed set by the individual rail segment or the vehicles set the speed themselves. A superposition of both speeds is also possible, although this is not used in practice.



#### Rail-S4 – Straight rail with four sensors

##### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the vehicles. The width of the symbol is scalable. The width corresponds to the length of the rail section.

## Function

The *Rail-S4* component type is used for simulating a straight rail section. The speed at which vehicles are moved over this rail section is set at the hidden *Speed* input of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

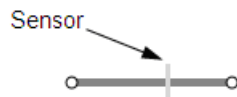
Between one and four sensors can be positioned along the length of the rail. When a vehicle is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the detected vehicle can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

## Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NbrOfSensors*  
Number of sensors used (1 to 4)
- *SensorPosition*  
Position of the corresponding sensor relative to connector *A* of the symbol.

When the simulation is running, every defined sensor is shown in the correct position in the symbol. When a sensor is activated, its color changes to yellow.



The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
NbrOfSensors	1
▼ SensorPosition [1]	...
SensorPositi... [mm]	8000.0

The validity of the parameters is checked while the component is being initialized. The following messages indicate possible parameter assignment errors:

- *Parameter 'SensorPosition' must be less than the component's width*  
The sensor position must not be outside the component.
- *Parameter 'SensorPosition' must not be negative*  
The sensor position must not be negative.

### Additional parameters

The *Rail-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

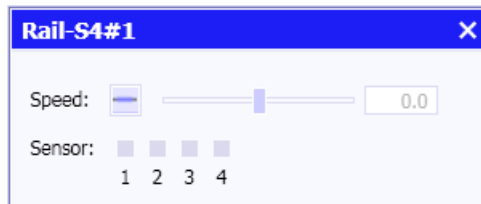
Further additional parameters can be used during initialization to place vehicles on the rail.

- *MaterialType*  
The type of vehicle that is to be placed
- *MaterialList*  
The name of the material list from which the vehicle is to be taken. If this parameter is left blank, all the material lists in the simulation project are searched.
- *InitNbrOfObjects*  
The initial number of vehicles to be placed on the rail
- *Clearance*  
The distance between the vehicles that are initially placed

Name	Value
MaterialType	
MaterialList	
InitNbrOfObjects	0
Clearance [mm]	0.0
TEMPLATE	Rail-S4
HIERARCHY	RAILS

### Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of a maximum of four sensors.



### CurvedRail45-S4 – 45° curved rail with four sensors

#### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the vehicles. The size of the symbol can be scaled proportionally. The size corresponds to the

size of the rail section. Proportional scaling alters the radius of the curve; the angle of 45° does not change.

## Function

The *CurvedRail45-S4* component type is used to simulate a curved rail with a 45° bend. The speed at which vehicles are moved over this rail section is set at the hidden *Speed* input of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

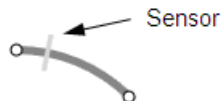
Between one and four sensors can be positioned along the length of the rail. When a vehicle is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the vehicles can be obtained via the hidden integer outputs *SensorId1* to *SensorId4*.

## Parameters

The behavior of the component can be set by means of a parameter:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NbrOfSensors*  
Number of sensors used (1 to 4)
- *SensorPosition*  
Position of the corresponding sensor relative to connector *A* of the symbol.

When the simulation is running, every defined sensor is shown in the correct position in the symbol. When a sensor is activated, its color changes to yellow.



The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
NbrOfSensors	1
▼ SensorPosition [1]	...
SensorPositi... [mm]	8000.0

The validity of the parameters is checked while the component is being initialized. The following messages indicate possible parameter assignment errors:

- *Parameter 'SensorPosition' must be less than the component's length*  
The sensor position must not be outside the component.
- *Parameter 'SensorPosition' must not be negative*  
The sensor position must not be negative.

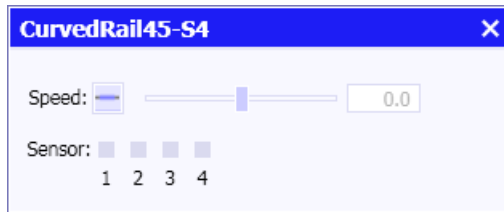
**Additional parameters**

The *CurvedRail45-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	CurvedRail45-S4
HIERARCHY	RAILS

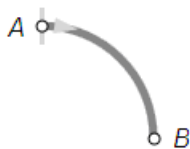
**Operating window**

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of a maximum of four sensors.



**CurvedRail90-S4 – 90° curved rail with four sensors**

**Symbol**



The gray arrow head in the symbol indicates the reference direction for movement of the vehicles. The size of the symbol can be scaled proportionally. The size corresponds to the size of the rail section. Proportional scaling alters the radius of the curve; the angle of 90° does not change.

**Function**

The *CurvedRail90-S4* component type is used to simulate a curved rail with a 90° bend. The speed at which vehicles are moved over this rail section is set at the hidden *Speed* input of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

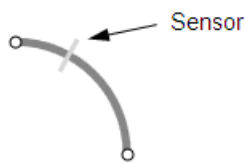
Between one and four sensors can be positioned along the length of the rail. When a vehicle is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the vehicles can be obtained via the hidden integer outputs *SensorId1* to *SensorId4*.

## Parameters

The behavior of the component can be set by means of parameters:

- *MaxSpeed*  
Nominal conveyor speed; adjustable online
- *NbrOfSensors*  
Number of sensors used (1 to 4)
- *SensorPosition*  
Position of the corresponding sensor relative to connector *A* of the symbol.

When the simulation is running, every defined sensor is shown in the correct position in the symbol. When a sensor is activated, its color changes to yellow.



The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
NbrOfSensors	1
▼ SensorPosition [1]	...
SensorPositi... [mm]	8000.0

The validity of the parameters is checked while the component is being initialized. The following messages indicate possible parameter assignment errors:

- *Parameter 'SensorPosition' must be less than the component's length*  
The sensor position must not be outside the component.
- *Parameter 'SensorPosition' must not be negative*  
The sensor position must not be negative.

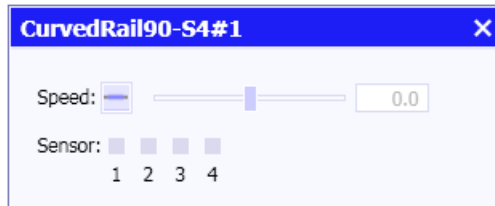
## Additional parameters

The *CurvedRail90-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	CurvedRail45-S4
HIERARCHY	RAILS

## Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of a maximum of four sensors.



## RailSwitch-F – Switchable switch with 45° spur

### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the vehicles. The size of the symbol can be scaled proportionally to adjust the size of the switch. Proportional scaling alters the length of the switch and the radius of the spur curve, but the spur angle of 45° does not change.

### Function

The *RailSwitch-F* component type is used to simulate a branch (switch) or junction with a 45° spur. It functions as a switch when objects are moved in the reference direction and as a junction when they are moved against the reference direction.

If the hidden binary input *Switch* is set to one (True), transport is switched from the straight section (*A–B*) to the spur (*A–C*). Note that transport against the reference direction is likewise only possible over an active segment.

The speed at which vehicles are moved over the rails is set at the hidden *Speed* input of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

The switch cannot be operated if it is occupied by vehicles. If an attempt is made to operate an occupied switch, the following error message appears: "Switch cannot be operated while being occupied".

### Parameters

The behavior of the component can be set by means of a parameter:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online

The parameter, its unit and default value are shown in the figure below.



Name	Value
NominalSpeed [m/s]	2.0
SwitchingTime [s]	1.0

### Additional parameters

The *RailSwitch-F* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	RailSwitch-F
HIERARCHY	RAILS

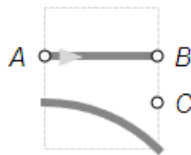
### Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and switch the sections.



### RailSwitch-M – Movable switch with 45° spur

#### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the vehicles. The size of the symbol can be scaled proportionally to adjust the size of the switch. Proportional scaling alters the length of the switch and the radius of the spur curve, but the spur angle of 45° does not change.

#### Function

The *RailSwitch-M* component type is used to simulate a branch (switch) or junction with a 45° spur. It functions as a switch when objects are moved in the reference direction and as a junction when they are moved against the reference direction.

The hidden analog input *Switch* is used to move the switch from transport over the straight section (*A – B*) to transport over the spur (*A – C*). The switching time is configurable (*SwitchingTime* parameter). Positive values at the *Switch* input move the switch to the *A – C* section, while negative values move it to the *A – B* section. The absolute values are percentages of the configurable switching speed. The moving operation for a component is shown in animated form in the symbol.

The speed at which vehicles are moved over the rails is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

The switch cannot be moved if it is occupied by vehicles. If an attempt is made to move an occupied switch, the following error message appears: "Switch cannot be operated while being occupied".

### Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *SwitchingTime*  
Switching time for the switch when triggered with +/- 100% values at the *Switch* input.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
SwitchingTime [s]	1.0

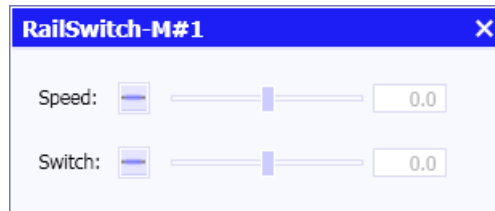
### Additional parameters

The *RailSwitch-M* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	RailSwitch-M
HIERARCHY	RAILS

## Operating window

In the operating window, you can use a slider to set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) and the switching time as a percentage of the configurable switching time (*SwitchingTime*).



## InOut – Inward and outward section for vehicles

### Symbol



The width of the symbol is scalable. The set width corresponds to the length of the inward/outward section.

### Function

The *InOut* component type represents a section over which vehicles are moved into or out of a conveyor system network. The InOut component is connected to an open section of the network at connector *A* to extend the conveyor system network.

To move them into the network, individual vehicles are placed on the unconnected end of the inward/outward section (left edge of the symbol) by setting the hidden binary input *CreateObject* and moved from there to connector *A*. Correspondingly, vehicles to be moved out of the network are moved from connector *A* to the unconnected end of the inward/outward section, from where they are removed from the conveyor system network. The speed at which vehicles are moved over the inward/outward section is set at the hidden *Speed* input of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

The inward/outward section can only be occupied by one vehicle at a time. If the section is occupied by a vehicle, no other vehicle can be positioned to be moved into the network or removed from the network.

### Parameters

The type of vehicles to be moved into the network, the material list from which they are taken and the speed at which the vehicles are moved over the inward/outward section can be set by means of parameters.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
MaterialType	
MaterialList	

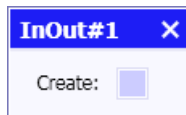
### Additional parameters

The *InOut* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	InOut
HIERARCHY	RAILS

### Operating window

In the operating window, you can place a new vehicle on the inward/outward section by clicking the pushbutton.



### RailLifter – Lifter

A lifter is a rail that can be moved vertically through several levels. In SIMIT a lifter is made up of the base component of the *RailLifterBase* type and up to eight expansion components of the *RailLifterExtension* type. The base component simulates the lifter in the base level, while each additional level is simulated by an expansion component. A lifter that can be moved through up to eight additional levels can be simulated in this way.

You do not have to position the base component and the associated expansion components on the same chart. You can create a separate chart for each level, for example, and distribute the lifter components over the individual charts.

The following points apply to the parameter assignment of the base component:

- There are no restrictions on the name of the base component.
- The *NbrOfExtensions* parameter indicates the number of additional levels and hence also the number of expansion components used.
- In the base component you have to specify the position (*LevelPosition*) for each level, which means the level above the base level in millimeters. The base level is by definition at level zero. All level values must be positive and increase with the level number.

The following points should be borne in mind when assigning the parameters for the expansion components:

- The name of the expansion component is formed from the name of the base component, the '#' character, and a number indicating the level.
- The *Level* parameter must correspond to the level number. The numbering of the levels starts with one.
- The *BaseName* parameter is the name of the base component.
- The length of the lifter is determined by the width of the base component.

---

### Note

All expansion components must be exactly the same width as the base component. Otherwise, a message will be displayed pointing to this inconsistency when the simulation is started.

---

The rail drive and the lifter sensors are fully implemented in the base component. This means expansion components do not work without the base component.

## RailLifterBase – Lifter (base component)

### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the vehicles. The width of the symbol is scalable. The set width corresponds to the length of the lifter rail.

### Function

The *RailLifterBase* component type is used to simulate the base station of a lifter. The speed at which vehicles are moved over the lifter rail is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). Similarly, the lifting speed, which means the speed at which the rail is moved to the different levels, is set at the hidden analog input *LifterSpeed* as a percentage of the configurable nominal lifting speed (*NominalLifterSpeed* parameter).

To move vehicles in and out at the various lifter levels, the lifter rail must be moved to the appropriate level and stop flush with that level. Flushness must be set with a positioning accuracy  $\Delta$  that is the same for all levels (*PositioningAccuracy* parameter). The rail stops flush with a level  $H$  if the following applies for its level  $h$ :

$$H - \Delta \leq h \leq H + \Delta$$

Between 1 and 4 sensors can be positioned on the rail relative to each of the two connectors  $A$  and  $B$ . When a vehicle is within the detection range of a sensor, the corresponding hidden

binary output *Sensor1* to *Sensor4* is set. The ID of the detected vehicle can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

---

**Note**

The base component has further hidden inputs for exchanging signals with extension components. The corresponding signals are assigned to these inputs by default. Do not change these default settings, otherwise the lifter components will not behave in the intended way in the simulation.

---

**Parameters**

The behavior of the component is configurable.

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NominalLifterSpeed*  
Nominal lifting speed; adjustable online
- *NbrOfExtensions*  
Number of additional levels (1 to 8)
- *LevelPosition*  
Level of the corresponding additional level
- *PositioningAccuracy*  
Positioning accuracy within which the level is deemed to have been reached.
- *NbrOfSensorsA*  
Number of sensors relative to connector A (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor relative to connector A of the symbol
- *NbrOfSensorsB*  
Number of sensors relative to connector B (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor relative to connector B of the symbol

When the simulation is running, the defined sensors are shown in the correct position in the symbol. When a sensor is activated, its color changes to yellow.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
NominalLifterSp... [m/s]	1.5
NbrOfExtensions	1
▼ LevelPosition [1]	...
LevelPosition1 [mm]	0.0
PositioningAccu... [mm]	100.0
NbrOfSensorsA	1
▼ SensorPositionA [1]	...
SensorPositi... [mm]	0.0
NbrOfSensorsB	1
▼ SensorPositionB [1]	...
SensorPositi... [mm]	0.0

### Additional parameters

The *RailLifterBase* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

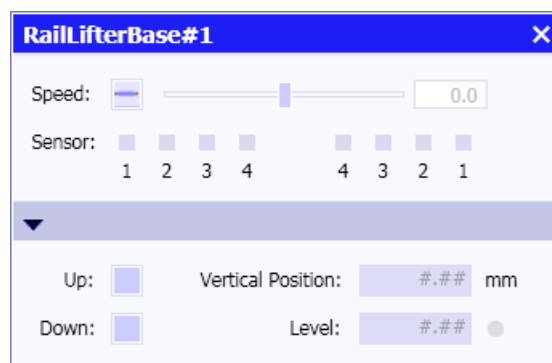
You also need to specify how many vehicles can be present on this conveyor section at any one time.

Name	Value
MaxObjects	8
TEMPLATE	RailLifterBase
HIERARCHY	RAILS

### Operating window

In the operating window, you can use a slider to set the transport speed as a percentage of the nominal conveyor speed (*NominalSpeed*) and you can monitor the status of the sensors.

In the extended operating window you can move the lifter up or down by means of pushbuttons. The current lifter position and level are displayed.



### RailLifterExtension – Lifter (extension component)

#### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the vehicles. The width of the symbol is scalable. The set width corresponds to the length of the lifter rail.

---

#### Note

The width of the extension component must be the same as the width of the base component.

---

#### Function

The *RailLifterExtension* component type is used to simulate a lifter at one of the accessible levels. A component of this type can only be used in combination with a component of the *RailLifterBase* type.

#### Parameters

The behavior of the component is configurable:

- *Level*  
Level at which the component is located. The numbering starts at one.
- *BaseName*  
Name of the corresponding base component

The parameters, their units and default values are shown in the figure below.

Name	Value
Level	1
BaseName	

#### Additional parameters

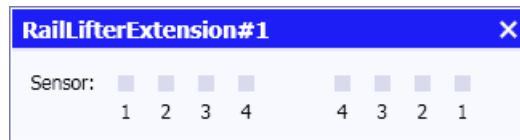
Specify how many objects can be present on the section (lifter rail) defined by the component at any one time. This parameter should be set to the same value as for the base component.

Name	Value
MaxObjects	8



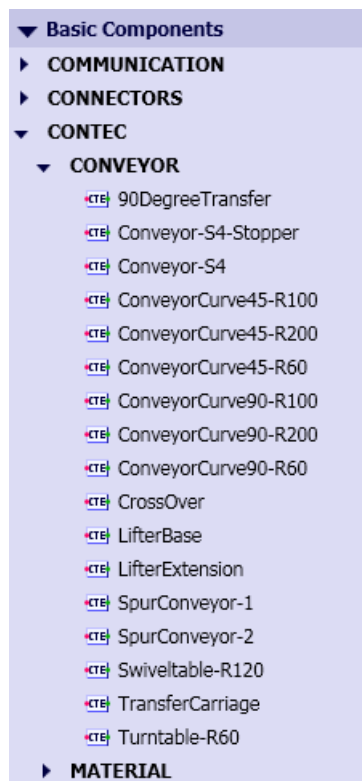
## Operating window

In the operating window, you can monitor the status of the sensors.



### 7.3.3.3 Component types for non-vehicular conveyor systems

The *CONVEYOR* directory of the *CONTEC* library contains component types for use in simulating conveyor systems such as roller, chain and belt conveyor systems. All of these components determine the speed at which the object is moved. The object is passive in respect of these components, in other words it generally does not have its own drive. These components are typically used to transport pallets, containers or capital goods.



All component types of the *CONVEYOR* library have a fixed symbol height of 40 pixels. According to the selected scale the width of the conveyors can be adjusted to the values listed in the table below.

Table 7-48 Selectable widths for conveyors

Conveyor width	Scale
40mm	1pix : 1mm
200mm	1pix : 5mm
400mm	1pix : 10mm
800mm	1pix : 20mm
2000mm	1pix : 50mm
4000mm	1pix : 100mm

### Conveyor-S4 – Straight conveyor with four sensors

#### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

Table 7-49 Indication of the current transport direction in the symbol

	Transport in the reference direction
	No transport
	Transport against the reference direction

The width of the symbol is scalable. The set width corresponds to the length of the conveyor.

## Function

The *Conveyor-S4* component type simulates a straight conveyor section of a given length. The speed at which objects are moved over this conveyor is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

Between 1 and 4 sensors can be positioned on the conveyor relative to each of the two connectors *A* and *B*. When an object is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the detected object can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

## Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NbrOfSensorsA*  
Number of sensors relative to connector *A* (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor relative to connector *A* of the symbol
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor relative to connector *B* of the symbol

When the simulation is running, the defined sensors are shown in the correct position in the symbol. When a sensor is activated, its color changes to yellow.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
NbrOfSensorsA	1
▼ SensorPositionA [1]	...
SensorPositionA1 [mm]	0.0
NbrOfSensorsB	1
▼ SensorPositionB [1]	...
SensorPositionB1 [mm]	0.0

## Additional parameters

The *Conveyor-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Further additional parameters can be used to place objects on the conveyor for the simulation. These are placed on the conveyor after the start of the simulation, starting at end A:

- *MaterialType*  
The type of object initially placed on the conveyor
- *MaterialList*  
The name of the material list from which the object is to be taken. If this parameter is left blank, all the material lists in the simulation project are searched.
- *InitNbrOfObjects*  
The number of objects that are to be placed
- *Clearance*  
The distance between the objects that are to be placed

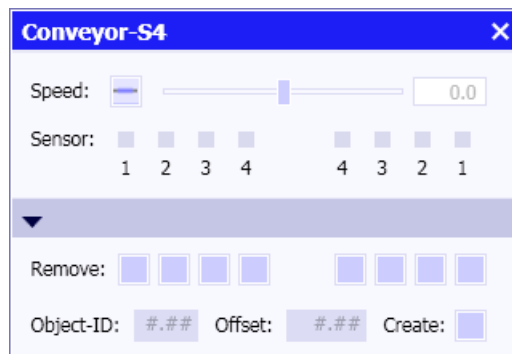
Name	Value
MaterialType	
MaterialList	
InitNbrOfObjects	0
Clearance [mm]	0.0
TEMPLATE	Conveyor-S4
HIERARCHY	CONVEYOR

### Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of the maximum eight sensors.

In the extended operating window you can remove objects that are currently detected by one of the eight sensors from the conveyor section (*Remove*). In addition, you can place new objects on the conveyor section using the *Create* pushbutton (*Create*). The object with the specified object ID (*Object-ID*) is then placed on the conveyor at the specified distance (*Offset*) from connector A. If no object ID is specified, the system searches the material lists (*MaterialList* additional parameter) for an object of the type specified by the *MaterialType* additional parameter.

This simulates manual interventions in the conveyor system such as the removal and placement of objects.



## Conveyor-S4-Stopper – Straight conveyor with four sensors and stopper

### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

Table 7-50 Indication of the current transport direction in the symbol

	Transport in the reference direction
	No transport
	Transport against the reference direction

The width of the symbol is scalable. The set width corresponds to the length of the conveyor.

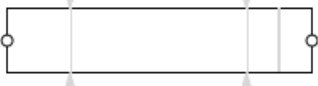
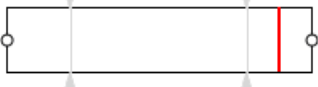
### Function

The *Conveyor-S4-Stopper* component type simulates a straight conveyor section of a given length with a stopper at a given position on the conveyor section. The speed at which objects are moved over this conveyor is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

Between 1 and 4 sensors can be positioned on the conveyor relative to each of the two connectors *A* and *B*. When an object is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the detected object can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

The stopper can be activated and deactivated to stop objects at the stopper position in both conveyor directions. When a stopper is activated, its color in the symbol changes to red.

Table 7-51 Representation of the stopper in the symbol

	Deactivated stopper
	Activated stopper

### Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *StopperPosition*  
Position of the stopper relative to connector *A*
- *NbrOfSensorsA*  
Number of sensors relative to connector *A* (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor relative to connector *A* of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor relative to connector *B* of the symbol.

When the simulation is running, the defined sensors are shown in the correct position in the symbol. When a sensor is activated, its color changes to yellow.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
StopperPosition [mm]	0.0
NbrOfSensorsA	1
▼ SensorPositionA [1]	...
SensorPositionA1 [mm]	0.0
NbrOfSensorsB	1
▼ SensorPositionB [1]	...
SensorPositionB1 [mm]	0.0

## Additional parameters

The *Conveyor-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Further additional parameters can be used to place objects on the conveyor for the simulation. These are placed on the conveyor after the start of the simulation, starting at end *A*:

- *MaterialType*  
The type of object initially placed on the conveyor
- *MaterialList*  
The name of the material list from which the object is to be taken. If this parameter is left blank, all the material lists in the simulation project are searched.
- *InitNbrOfObjects*  
The number of objects that are to be placed
- *Clearance*  
The distance between the objects that are to be placed:

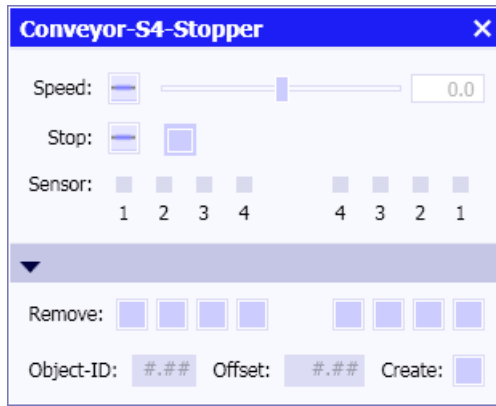
Name	Value
MaterialType	
MaterialList	
InitNbrOfObjects	0
Clearance	0.0
TEMPLATE	Conveyor-S4-Stopper
HIERARCHY	CONVEYOR

## Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of the maximum eight sensors. The stopper can be activated and deactivated by means of a switch.

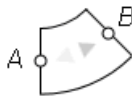
In the extended operating window you can remove objects that are currently detected by one of the eight sensors from the conveyor section (*Remove*). In addition, you can place new objects on the conveyor section using the *Create* pushbutton (*Create*). The object with the specified object ID (*Object-ID*) is then placed on the conveyor at the specified distance (*Offset*) from connector *A*. If no object ID is specified, the system searches the material lists (*MaterialList* additional parameter) for an object of the type specified by the *MaterialType* additional parameter.

This simulates manual interventions in the conveyor system such as the removal and placement of objects.



**ConveyorCurve45-R60 – 45° curved conveyor (radius 60 pixels)**

**Symbol**



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

**Function**

The *ConveyorCurve45-R60* component type is used to simulate a curved conveyor. The conveyor section forms a 45° arc. The component symbol cannot be scaled. The radius of the arc is set to 60 pixels and cannot be changed. The effective radius of the component is determined by the scale set for the chart. For example, a scale of 1 pix : 100 mm gives an effective radius of 6 meters.

The speed at which objects are moved over this conveyor is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

**Parameter**

The nominal conveyor speed can be set by means of a parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0



## Additional parameters

The *ConveyorCurve45-R60* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	ConveyorCurve45-R60
HIERARCHY	CONVEYOR

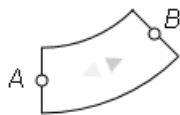
## Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



## ConveyorCurve45-R100 – 45° curved conveyor (radius 100 pixels)

### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

### Function

The *ConveyorCurve45-R100* component type is used to simulate a curved conveyor. The conveyor section forms a 45° arc. The component symbol cannot be scaled. The radius of the arc is set to 100 pixels and cannot be changed. The effective radius of the component is determined by the scale set for the chart. For example, a scale of 1 pix : 100 mm gives an effective radius of 10 meters.

The speed at which objects are moved over this conveyor is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

**Parameters**

The nominal conveyor speed can be set by means of a parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0

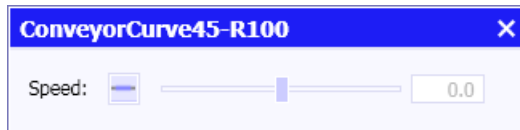
**Additional parameters**

The *ConveyorCurve45-R100* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	ConveyorCurve45-R100
HIERARCHY	CONVEYOR

**Operating window**

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider



**ConveyorCurve45-R200 – 45° curved conveyor (radius 200 pixels)**

**Symbol**



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

## Function

The *ConveyorCurve45-R200* component type is used to simulate a curved conveyor. The conveyor section forms a 45° arc. The component symbol cannot be scaled. The radius of the arc is set to 200 pixels and cannot be changed. The effective radius of the component is determined by the scale set for the chart. For example, a scale of 1 pix : 100 mm gives an effective radius of 20 meters.

The speed at which objects are moved over this conveyor is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

## Parameters

The nominal conveyor speed can be set by means of a parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0

## Additional parameters

The *ConveyorCurve45-R200* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	ConveyorCurve45-R200
HIERARCHY	CONVEYOR

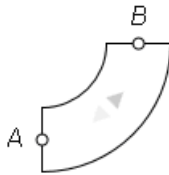
## Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider



**ConveyorCurve90-R60 – 90° curved conveyor (radius 60 pixels)**

**Symbol**



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

**Function**

The *ConveyorCurve45-R60* component type is used to simulate a curved conveyor. The conveyor section forms a 90° arc. The component symbol cannot be scaled. The radius of the arc is set to 60 pixels and cannot be changed. The effective radius of the component is determined by the scale set for the chart. For example, a scale of 1 pix : 100 mm gives an effective radius of 6 meters.

The speed at which objects are moved over this conveyor is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

**Parameters**

The nominal conveyor speed can be set by means of a parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0

**Additional parameters**

The *ConveyorCurve90-R60* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	ConveyorCurve90-R60
HIERARCHY	CONVEYOR

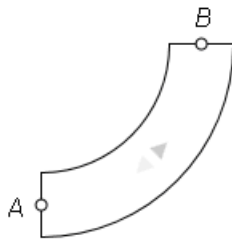
## Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



## ConveyorCurve90-R100 – 90° curved conveyor (radius 100 pixels)

### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

### Function

The *ConveyorCurve90-R100* component type is used to simulate a curved conveyor. The conveyor section forms a 90° arc. The component symbol cannot be scaled. The radius of the arc is set to 100 pixels and cannot be changed. The effective radius of the component is determined by the scale set for the chart. For example, a scale of 1 pix : 100 mm gives an effective radius of 10 meters.

The speed at which objects are moved over this conveyor is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

### Parameters

The nominal conveyor speed can be set by means of a parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0

**Additional parameters**

The *ConveyorCurve90-R100* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	ConveyorCurve90-R100
HIERARCHY	CONVEYOR

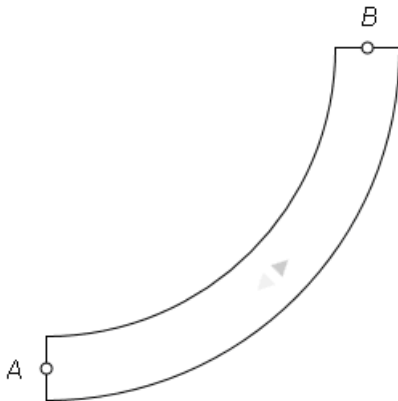
**Operating window**

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



**ConveyorCurve90-R200 – 90° curved conveyor (radius 200 pixels)**

**Symbol**



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

**Function**

The *ConveyorCurve90-R200* component type is used to simulate a curved conveyor. The conveyor section forms a 90° arc. The component symbol cannot be scaled. The radius of the arc is set to 200 pixels and cannot be changed. The effective radius of the component is determined by the scale set for the chart. For example, a scale of 1 pix : 100 mm gives an effective radius of 20 meters.

The speed at which objects are moved over this conveyor is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

## Parameters

The nominal conveyor speed can be set by means of a parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0

## Additional parameters

The *ConveyorCurve90-R200* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	ConveyorCurve90-R200
HIERARCHY	CONVEYOR

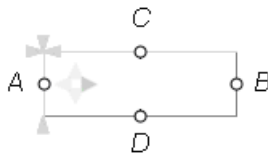
## Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



## 90DegreeTransfer – 90-degree transfer

### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the objects on the conveyor section *A – B*. When the simulation is running the current transport direction is indicated by green arrows in the symbol. The direction of transfer (to *D* or *C*) is also indicated by green arrows.

The width of the symbol is scalable. The set width corresponds to the length of the 90-degree transfer. As with all component types in the *CONVEYOR* library, the height of the symbol is fixed at 40 pixels. The effective width of the 90-degree transfer is therefore determined by the scale set for the chart.

The location of positions C and D for the transfer of objects can be moved. To do so, hold down the ALT key and drag the connection point C or D with the mouse (with the left mouse button held down) to the desired position.

## Function

The *90DegreeTransfer* component type is used to simulate a 90-degree transfer for moving objects from conveyor section *A – B* to a conveyor connected at connector *C* or *D* without rotating them.

The speed at which objects are moved over the conveyor section *A – B* is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). Correspondingly, the speed at which the objects are transferred to the transfer section *C – D* is set at the hidden analog input *TransferSpeed* as a percentage of the configurable nominal transfer speed (*NominalTransferSpeed* parameter). The reference direction of the transfer sections is set from connector C to the transport section or from the transport section to connector *D*.

Between one and four sensors can be positioned on the transfer relative to each of the four connectors *A*, *B*, *C* and *D*. When an object is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the detected object can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

## Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NominalTransferSpeed*  
Nominal transfer speed; adjustable online
- *PositioningAccuracy*  
Accuracy with which the object must be positioned for transfer to the transfer section connecting to *C* or *D*
- *NbrOfSensorsA*  
Number of sensors relative to connector *A* (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor relative to connector *A* of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor relative to connector *B* of the symbol.
- *NbrOfSensorsC*  
Number of sensors relative to connector *C* (1 to 4)



- *SensorPositionC*  
Position of the corresponding sensor relative to connector *C* of the symbol.
- *NbrOfSensorsD*  
Number of sensors relative to connector *D* (1 to 4)
- *SensorPositionD*  
Position of the corresponding sensor relative to connector *D* of the symbol.

When the simulation is running, the defined sensors are shown in the correct position in the symbol. When a sensor is activated, its color changes to yellow.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
PositioningAccuracy [mm]	100.0
NbrOfSensorsA	1
▼ SensorPositionA [1]	...
SensorPositionA1 [mm]	0.0
NbrOfSensorsB	1
▼ SensorPositionB [1]	...
SensorPositionB1 [mm]	0.0
NbrOfSensorsC	1
▼ SensorPositionC [1]	...
SensorPositionC1 [mm]	0.0
NbrOfSensorsD	1
▼ SensorPositionD [1]	...
SensorPositionD1 [mm]	0.0

## Additional parameters

The *90DegreeTransfer* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

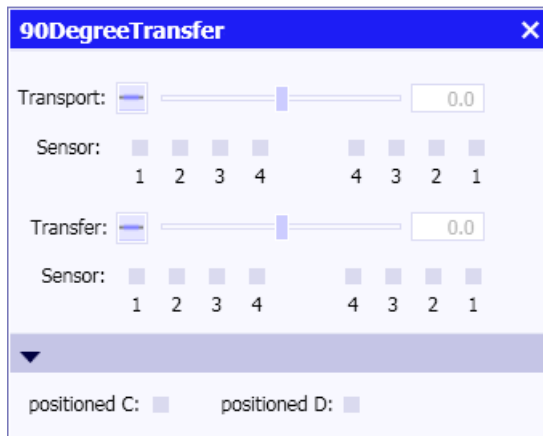
You also need to specify how many objects can be present on the conveyor section at any one time.

Name	Value
MaxObjects	8
TEMPLATE	90DegreeTransfer
HIERARCHY	CONVEYOR

## Operating window

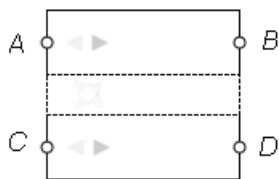
In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) and the transfer speed as a percentage of the nominal transfer speed (*NominalTransferSpeed*) using a slider and monitor the status of the maximum eight sensors.

The extended operating window shows whether the object is close enough to connector C or D to be transferred.



CrossOver – Crossover

Symbol



The two gray arrow heads in the symbol indicate the reference direction for movement of the objects over sections A – B and C – D. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

The width of the symbol is scalable. The set width corresponds to the length of the crossover.

Function

The *CrossOver* component type simulates a crossover. The speed at which objects are moved over this switch is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter).

The switch is operated by means of the two binary inputs *Switch\_CB\_DA* and *Switch\_AD\_BC*. If the *Switch\_CB\_DA* input is set (True), the object is no longer transported over section A – B but over section A – D (transport in reference direction) or section B – C (transport against reference direction), depending on the transport direction. Correspondingly, if the *Switch\_AD\_BC* input is set (True), the object is no longer transported over section C – D but over section C – B (transport in reference direction) or over section A – D (transport against reference direction), depending on the transport direction. Setting both inputs at the same time deactivates the switch, which means that transport is stopped.

## Parameters

The nominal speed at which the object is transported is set by means of the *NominalSpeed* parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0

## Additional parameters

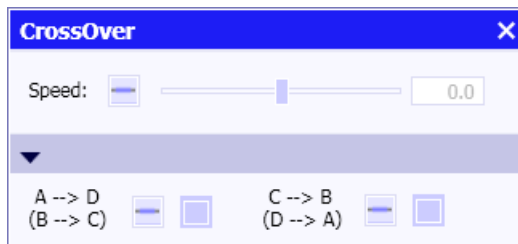
The *CrossOver* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	CrossOver
HIERARCHY	CONVEYOR

## Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of the maximum eight sensors.

In the extended operating window you can set the two binary inputs *Switch\_CB\_DA* and *Switch\_AD\_BC* by means of a slider to set the transport direction of the crossover manually.



## Lifter

A lifter is a conveyor section that can be moved vertically through several levels. In SIMIT a lifter is made up of the base component of the *LifterBase* type and up to eight expansion components of the *LifterExtension* type. The base component simulates the lifter in the base level, while each additional level is simulated by an expansion component. A lifter that can be moved through up to eight additional levels can be simulated in this way.

You do not have to position the base component and the associated expansion components on the same chart. You can create a separate chart for each level, for example, and distribute the lifter components over the individual charts.

The following points apply to the parameter assignment of the base component:

- There are no restrictions on the name of the base component.
- The *NbrOfExtensions* parameter indicates the number of additional levels and hence also the number of expansion components used.
- In the base component you have to specify the position (*LevelPosition*) for each level, which means the level above the base level in millimeters. The base level is by definition at level zero. All level values must be positive and increase with the level number.

The following points should be borne in mind when assigning the parameters for the expansion components:

- The name of the expansion component is formed from the name of the base component, the '#' character, and a number indicating the level.
- The *Level* parameter must correspond to the level number. The numbering of the levels starts with one.
- The *BaseName* parameter is the name of the base component.
- The length of the lifter is determined by the width of the base component.

---

**Note**

All expansion components must be exactly the same width as the base component. Otherwise, a message will be displayed pointing to this inconsistency when the simulation is started.

---

The drives and the lifter sensors are fully implemented in the base component. This means expansion components do not work without the base component.

**LifterBase – Lifter (base component)**

**Symbol**



The gray arrow head in the symbol indicates the reference direction for movement of the objects. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

The width of the symbol is scalable. The set width corresponds to the length of the lifter.

## Function

The *LifterBase* component type is used to simulate the base station of a lifter. The speed at which objects are moved over the conveyor section is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). Similarly, the lifting speed, which means the speed at which the lifter is moved to the different levels, is set at the hidden analog input *LifterSpeed* as a percentage of the configurable nominal lifting speed (*NominalLifterSpeed* parameter).

To move objects in and out at the various lifter levels, the lifter must be moved to the appropriate level and stop flush with that level. Flushness must be set with a positioning accuracy  $\Delta$  that is the same for all levels (*PositioningAccuracy* parameter). The lifter stops flush with a level  $H$  if the following applies for its level  $h$ :

$$H - \Delta \leq h \leq H + \Delta$$

Between 1 and 4 sensors can be positioned on the lifter relative to each of the two connectors  $A$  and  $B$ . When an object is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the detected object can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

---

### Note

The base component has further hidden inputs for exchanging signals with extension components. The corresponding signals are assigned to these inputs by default. Do not change these default settings, otherwise the lifter components will not behave in the intended way in the simulation.

---

## Parameters

The behavior of the component is configurable.

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NominalLifterSpeed*  
Nominal lifting speed; adjustable online
- *NbrOfExtensions*  
Number of additional levels (1 to 8)
- *LevelPosition*  
Level of the corresponding additional level
- *PositioningAccuracy*  
Positioning accuracy within which the level is deemed to have been reached.
- *NbrOfSensorsA*  
Number of sensors relative to connector  $A$  (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor relative to connector  $A$  of the symbol.

- *NbrOfSensorsB*  
Number of sensors relative to connector B (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor relative to connector *B* of the symbol.

When the simulation is running, the defined sensors are shown in the correct position in the symbol. When a sensor is activated, its color changes to yellow.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0
NominalLifterSpeed	[m/s] 1.5
NbrOfExtensions	1
▼ LevelPosition [1]	...
LevelPosition1	[mm] 0.0
PositioningAccuracy	[mm] 100.0
NbrOfSensorsA	1
▼ SensorPositionA [1]	...
SensorPositionA1	[mm] 0.0
NbrOfSensorsB	1
▼ SensorPositionB [1]	...
SensorPositionB1	[mm] 0.0

### Additional parameters

The *LifterBase* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

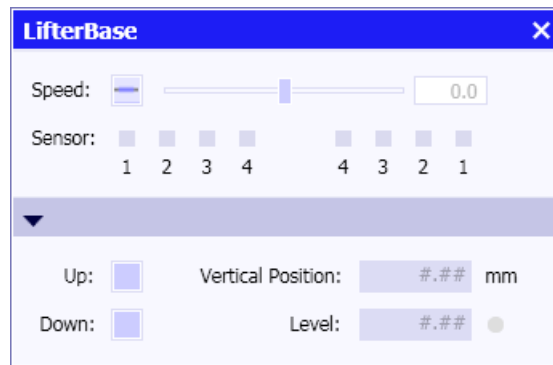
You also need to specify how many objects can be present on this conveyor section at any one time.

Name	Value
MaxObjects	8
TEMPLATE	LifterBase
HIERARCHY	CONVEYOR

### Operating window

In the operating window, you can use a slider to set the transport speed as a percentage of the nominal conveyor speed (*NominalSpeed*) and you can monitor the status of the sensors.

In the extended operating window you can move the lifter up or down by means of pushbuttons. The current lifter position and level are displayed.



## LifterExtension – Lifter (extension component)

### Symbol



The gray arrow head in the symbol indicates the reference direction for movement of the objects. The width of the symbol is scalable. The set width corresponds to the length of the lifter rail.

### Note

The width of the extension component must be the same as the width of the base component.

### Function

The *LifterExtension* component type is used to simulate a lifter at one of the accessible levels. A component of this type can only be used in combination with a component of the *LifterBase* type.

### Parameters

The behavior of the component is configurable:

- *Level*  
Level at which the component is located. The numbering starts at one.
- *BaseName*  
Name of the corresponding base component

The parameters, their units and default values are shown in the figure below.

Name	Value
BaseName	
Level	1

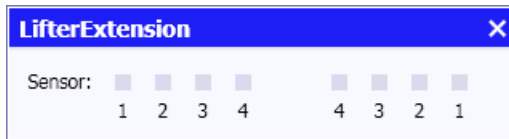
**Additional parameters**

Specify how many objects can be present on the section defined by the component at any one time. This parameter should be set to the same value as for the base component.

Name	Value
MaxObjects	8

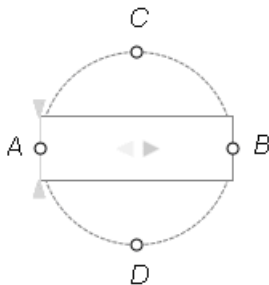
**Operating window**

In the operating window, you can monitor the status of the maximum eight sensors.



**Turntable-R60 – Turntable**

**Symbol**



The gray arrow head in the symbol indicates the reference direction for transport of the object on the turntable. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

**Function**

The *Turntable-R60* component type simulates a turntable. A turntable rotates a conveyor section through multiples of 90°. Using a turntable, objects can be transported onwards in the original conveyor direction or at right angles to that direction.

The speed at which objects are moved on the turntable is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed



(*NominalSpeed* parameter). Correspondingly, the rotation speed is set at the hidden analog input *RotationSpeed* as a percentage of the configurable nominal rotation speed (*NominalRotationSpeed* parameter).

The component symbol cannot be scaled. The radius of the turntable is thus fixed at 60 pixels and cannot be changed. Correspondingly, the length of the turntable is fixed at 120 pixels. The effective turntable length of the component is determined by the scale set for the chart. For example, a scale of 1 pix : 100 mm gives an effective length of 12 meters.

Between 1 and 4 sensors can be positioned on the turntable relative to each of the two connectors *A* and *B*. When an object is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the detected object can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

## Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NominalRotationSpeed*  
Nominal rotation speed; adjustable online
- *PositioningAccuracy*  
Accuracy with which the turntable must be positioned for the object to be transferred
- *NbrOfSensorsA*  
Number of sensors relative to connector *A* (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor relative to connector *A* of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor relative to connector *B* of the symbol.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
NominalRotationSpeed [°/s]	90.0
PositioningAccuracy [°]	1.0
NbrOfSensorsA	1
▼ SensorPositionA [1]	...
SensorPositionA1 [mm]	0.0
NbrOfSensorsB	1
▼ SensorPositionB [1]	...
SensorPositionB1 [mm]	0.0

### Additional parameters

The *TurnTable-R60* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

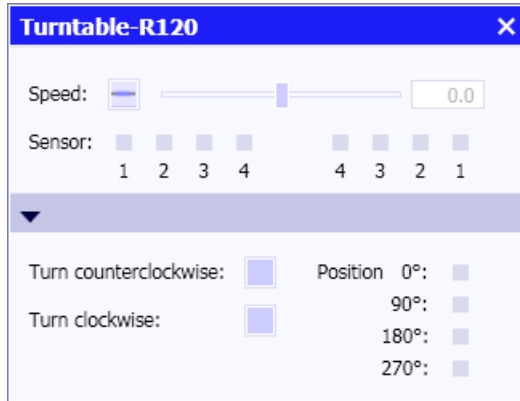
Specify how many objects can be located on the turntable at the same time.

Name	Value
MaxObjects	8
TEMPLATE	Turntable-R60
HIERARCHY	CONVEYOR

### Operating window

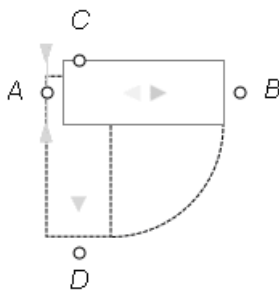
In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of the maximum 8 sensors.

In the extended operating window you can rotate the turntable manually by 90° in both directions by means of pushbuttons. The end position in each case is displayed.



### Swiveltable-R120 – Swivel table

#### Symbol



The gray arrow head in the symbol indicates the reference direction for transport of the object on the swivel table. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

## Function

The *Swiveltable-R120* component type simulates a swivel table that can be used to rotate a conveyor section through 90°. Using a swivel table, objects can be transported onwards in the original conveyor direction or at right angles to that direction.

The speed at which objects are moved on the swivel table is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). Correspondingly, the rotation speed is set at the hidden analog input *RotationSpeed* as a percentage of the configurable nominal rotation speed (*NominalRotationSpeed* parameter).

The component symbol cannot be scaled. The radius of the swivel table, which means the length of the swivel table, is fixed at 120 pixels and cannot be changed. The effective swivel table length of the component is determined by the scale set for the chart. For example, a scale of 1 pix : 100 mm gives an effective length of 12 meters.

Between one and four sensors can be positioned on the swivel table relative to each of the two connectors *A* and *B*. When an object is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the detected object can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

## Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NominalRotationSpeed*  
Nominal rotation speed; adjustable online
- *PositioningAccuracy*  
Accuracy with which the turntable must be positioned for the object to be transferred
- *NbrOfSensorsA*  
Number of sensors relative to connector *A* (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor relative to connector *A* of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor relative to connector *B* of the symbol.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
NominalRotationSpeed [°/s]	90.0
PositioningAccuracy [°]	1.0
NbrOfSensorsA	1
▼ SensorPositionA [1]	...
SensorPositionA1 [mm]	0.0
NbrOfSensorsB	1
▼ SensorPositionB [1]	...
SensorPositionB1 [mm]	0.0

### Additional parameters

The Swiveltable-R120 component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Specify how many objects can be located on the swivel table at the same time.

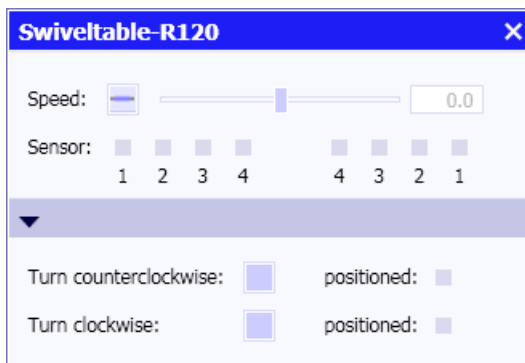
Name	Value
MaxObjects	8
TEMPLATE	Swiveltable-R120
HIERARCHY	CONVEYOR

### Operating window

You can also swivel the swivel table manually in both directions. The display shows whether the turntable is positioned sufficiently accurately in one of its two positions.

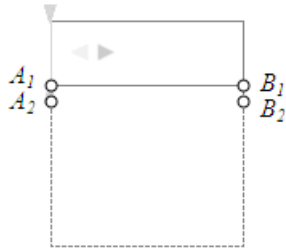
In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of the maximum 8 sensors.

In the extended operating window you can rotate the swivel table manually through 90° by means of a pushbutton. Another pushbutton allows you to rotate it back to its original position. Indicators show when it has reached the end position.



## TransferCarriage – Transfer carriage

### Symbol



The height and width of this component are scalable. The width sets the length of the transfer carriage, the height sets the length of the transfer path.

The gray arrow head in the symbol indicates the reference direction for transport of the object on the swivel table. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

### Function

The *TransferCarriage* component type is used for simulating a transfer carriage. Objects can be moved onto the transfer carriage from both sides. It can then be moved to another position at right angles to the transport direction and the objects can then be transported onwards via connectors  $A_i$  and  $B_i$ .

The number of connectors  $A_i$ ,  $B_i$  present on the left and right of the component is configurable. A maximum of 16 connectors on each side is possible; by default there are two connectors on each side.  $A_1$ ,  $A_2$ ,  $B_1$ ,  $B_2$ . Each connector marks a position that can be approached by the transfer carriage at right angles to the transport direction.

The speed at which objects are transported on the transfer carriage is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). Correspondingly, the transfer speed is set at the hidden analog input *TransferSpeed* as a percentage of the configurable nominal transfer speed (*NominalTransferSpeed* parameter).

Between one and four sensors can be positioned on the transfer carriage relative to each of the two connector sides  $A$  and  $B$ . When an object is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the detected object can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

### Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *NominalTransferSpeed*  
Nominal transfer speed; adjustable online

- *PositioningAccuracy*  
Accuracy with which the transfer carriage must be positioned for the object to be transferred.
- *NbrOfConnectorsA*  
Number of connectors on the left side (A)
- *NbrOfConnectorsB*  
Number of connectors on the right side (B)
- *NbrOfSensorsA*  
Number of sensors relative to connector side A (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor relative to connector side A of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector side B (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor relative to connector side B of the symbol.

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
NominalTransferSpeed [m/s]	1.5
PositioningAccuracy [mm]	50.0
NbrOfConnectorsA	2
NbrOfConnectorsB	2
NbrOfSensorsA	1
▼ SensorPositionA [1]	...
SensorPositionA1 [mm]	0.0
NbrOfSensorsB	1
▼ SensorPositionB [1]	...
SensorPositionB1 [mm]	0.0

### Additional parameters

The *TransferCarriage* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

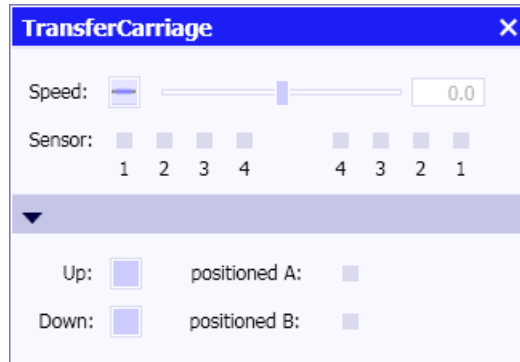
Specify how many objects can be located on the transfer carriage at the same time.

Name	Value
MaxObjects	8
TEMPLATE	TransferCarriage
HIERARCHY	CONVEYOR

### Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and monitor the status of the maximum 8 sensors.

In the extended operating window you can move the transfer carriage manually in both directions to the next position by means of pushbuttons (*Up*, *Down*). The end position in each case is displayed. Indicators show when the transfer carriage has reached a connector position on one or the other side.



## SpurConveyor-1 – Diagonal feed with one spur

### Symbol



The gray arrow head in the symbol indicates the reference direction for transport of the object on the diagonal feed. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

### Function

The *SpurConveyor-1* component type simulates a diagonal feed with one spur. Objects can be transported along section *A – C* or along the spur section *A – B*.

The speed at which objects are moved on the diagonal feed is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). To switch conveying to the spur section, the hidden binary input *Switch* is set to one (True).

### Parameters

The nominal conveyor speed can be set by means of a parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0

### Additional parameters

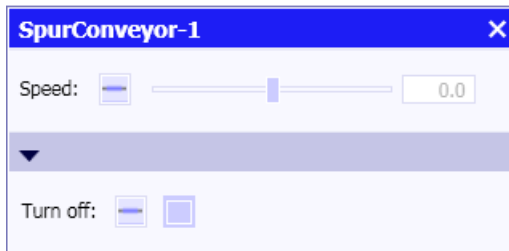
The *SpurConveyor-1* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	SpurConveyor-1
HIERARCHY	CONVEYOR

### Operating window

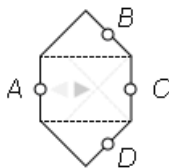
In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.

In the extended operating window you can change to the spur section by means of a switch.



### SpurConveyor-2 – Diagonal feed with two spurs

#### Symbol



The gray arrow head in the symbol indicates the reference direction for transport of the object on the diagonal feed. When the simulation is running the current transport direction is indicated by green arrows in the symbol.

#### Function

The *SpurConveyor-2* component type simulates a diagonal feed with two alternative spurs. Objects can be transported along section *A – C* or along one of the two spur sections *A – B* or *A – D*.

The speed at which objects are moved on the diagonal feed is set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). To switch conveying to one of the two spur sections, one of the



two hidden binary inputs *Switch\_AB* or *Switch\_AD* is set to one (True): *Switch\_AB* switches to spur section *A – B* and *Switch\_AD* switches to spur section *A – D*. If one of the two inputs is set, setting the other one has no effect. There will be no switching to either spur section if both inputs are set at the same time.

## Parameters

The nominal conveyor speed can be set by means of a parameter. Its unit and default value are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0

## Additional parameters

The *SpurConveyor-2* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	SpurConveyor-2
HIERARCHY	CONVEYOR

## Operating window

In the operating window, you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.

In the extended operating window you can change to the corresponding spur section *A – B* or *A – D* by means of a switch.



### 7.3.3.4 Component types for simulating objects

The *MATERIAL* directory of the *CONTEC* library contains component types for simulating objects. Vehicles (*Vehicle*) can be used with component types in the *RAILS* directory to simulate vehicular conveyor systems. Boxes (*Box*) can be used to simulate objects for component types in the *CONVEYOR* directory.



You create material lists with these components. You can find more information on the topic in the section: Scalability of objects (Page 570).

The size of a component (width and height of the symbol) is defined in the material list in millimeters.

#### Box – Simple object

##### Symbol



##### Function

The *Box* component type is used to simulate an object with a rectangular outline, such as a pallet or box. The component type defines the dimensions of the object. In the resolution 1:20 the standard dimensions correspond to the dimensions of a Euro pallet (1200 x 800 mm).

#### CBox – Colored object

##### Symbol








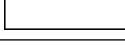


## Function

The *CBox* component type is used to simulate an object with a rectangular outline, such as a pallet or box. The component type defines the dimensions of the object. In the resolution 1:20 the standard dimensions correspond to the dimensions of a Euro pallet (1200 x 800 mm).

When the simulation is running the symbol of a component of this type can be displayed in one of 8 different colors. The color is determined with the three binary inputs *R*, *G* and *B* of the component by mixing together the 3 primary colors red, green and blue.

Table 7-52 Color chart for CBox

Input R	Input G	Input B	Display
False	False	False	
False	False	True	
False	True	False	
False	True	True	
True	False	False	
True	False	True	
True	True	False	
True	True	True	

## CBoxDS256 – Colored object with data storage

### Symbol




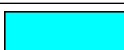


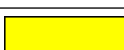
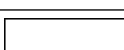


### Function

The *CBoxDS256* component type is used to simulate an object with a rectangular outline, such as a pallet or box. The component type defines the dimensions of the object. In the resolution 1:20 the standard dimensions correspond to the dimensions of a Euro pallet (1200 x 800 mm).

When the simulation is running the symbol of a component of this type can be displayed in one of 8 different colors. The color is determined with the three binary inputs *R*, *G* and *B* of the component by mixing together the 3 primary colors red, green and blue.

Table 7-53 Color chart for *CBoxDS256*

Input R	Input G	Input B	Display
False	False	False	
False	False	True	
False	True	False	
False	True	True	
True	False	False	
True	False	True	
True	True	False	
True	True	True	

In addition, a data storage with a configurable size (*SizeOfStorage* parameter) is defined for components of this type. This memory represents the mobile data storage for an object. It is created as a state vector *MDS* with the data type *byte* and can be up to 256 bytes in size. Component types for writing to and reading this memory are located in the *SENSORS* library in the *RFID* directory.

**Parameters**

The size of the data storage can be set by means of the *SizeOfStorage* parameter. The default setting is one.

Name	Value
SizeOfStorage	1

**Vehicle**

**Symbol**



## Function

The *Vehicle* component type is used for simulating vehicles, such as those for electric overhead monorail systems. The vehicle is assumed to have a rectangular outline. The component type defines the dimensions of the outline. In the resolution 1:20 the standard dimensions correspond to the dimensions 1000 x 300 mm.

The speed at which these vehicles move in the rail network in the simulation is generally determined by the individual rail sections. However, the speed can also be set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). The two speed settings for a component are added together.

A component of this type can also be configured so that the vehicle starts off again with a delay if it has previously been stopped by a holdup. This function is only effective if the speed is set by the conveyor section.

## Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal speed of the vehicle; adjustable online
- *StartUpDelay*  
Start-up delay
- *SensorRange*  
Detection range of the sensors

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
StartUpDelay [s]	0.0
SensorRange [%]	100.0

## Additional parameters

The *Vehicle* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	Vehicle
HIERARCHY	VEHICLES

**CVehicle**

**Symbol**



**Function**








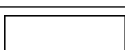
The *CVehicle* component type is used for simulating vehicles, such as those for electric overhead monorail systems. The vehicle is assumed to have a rectangular outline. The component type defines the dimensions of the outline. In the resolution 1:20 the standard dimensions correspond to the dimensions 1000 x 300 mm.

The speed at which these vehicles move in the rail network in the simulation is generally determined by the individual rail sections. However, the speed can also be set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). The two speed settings for a component are added together.

A component of this type can also be configured so that the vehicle starts off again with a delay if it has previously been stopped by a holdup. This function is only effective if the speed is set by the conveyor section.

When the simulation is running the symbol of a component of this type can be displayed in one of 8 different colors. The color is determined with the 3 binary inputs *R*, *G* and *B* of the component by mixing together the 3 primary colors red, green and blue.

Table 7-54 Color chart for *CVehicle*

Input R	Input G	Input B	Display
False	False	False	
False	False	True	
False	True	False	
False	True	True	
True	False	False	
True	False	True	
True	True	False	
True	True	True	

## Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal speed; adjustable online
- *StartUpDelay*  
Start-up delay
- *SensorRange*  
Detection range for sensors

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed [m/s]	2.0
StartUpDelay [s]	0.0
SensorRange [%]	100.0

## Additional parameters

The *Vehicle* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

Name	Value
TEMPLATE	CVehicle
HIERARCHY	VEHICLES

## VehicleDS256 – Vehicle with data storage

### Symbol



### Function

The *VehicleDS256* component type is used for simulating vehicles, such as those for electric overhead monorail systems. The vehicle is assumed to have a rectangular outline. The component type defines the dimensions of the outline. In the resolution 1:20 the standard dimensions correspond to the dimensions 1000 x 300 mm.

The speed at which these vehicles move in the rail network in the simulation is generally determined by the individual rail sections. However, the speed can also be set at the hidden analog input *Speed* of the component as a percentage of the configurable nominal conveyor speed (*NominalSpeed* parameter). The two speed settings for a component are added together.

A component of this type can also be configured so that the vehicle starts off again with a delay if it has previously been stopped by a holdup. This function is only effective if the speed is set by the conveyor section.

In addition, a data storage with a configurable size (*SizeOfStorage* parameter) is defined for components of this type. This memory represents the mobile data storage for a vehicle. It is created as a state vector *MDS* with the data type *byte* and can be up to 256 bytes in size. Component types for writing to and reading this memory are located in the *SENSORS* library in the *RFID* directory.

## Parameters

The behavior of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal speed; adjustable online
- *StartUpDelay*  
Start-up delay
- *SensorRange*  
Detection range for sensors
- *SizeOfStorage*  
Size of the mobile data storage

The parameters, their units and default values are shown in the figure below.

Name	Value
NominalSpeed	[m/s] 2.0
StartUpDelay	[s] 0.0
SensorRange	[%] 100.0
SizeOfStorage	1

## Additional parameters

The *VehicleDS256* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level. You can find additional information on this in the section: Using templates (Page 574).

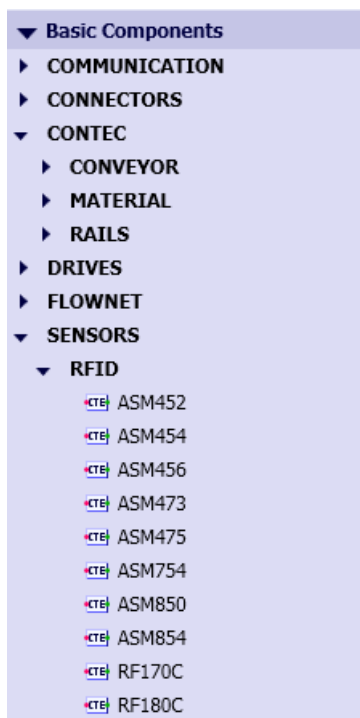
Name	Value
TEMPLATE	VehicleDS256
HIERARCHY	VEHICLES

### 7.3.3.5 Component types for simulating identification systems

The *RFID* directory of the *SENSORS* library contains component types for simulating identification systems that can be activated by the controller via the FC/FB45 in normal operation. Other operating modes are not simulated. Components of these types enables data exchange between the controller and the corresponding simulation components of objects. You can find additional information on this in the section: Component types for simulating objects (Page 626).



Both fixed data such as barcodes and write/read data in the case of a Moby can be exchanged.



All component types in the *RFID* directory have the same structure, but they differ in terms of the number of channels available and the commands that are supported.

As communication between interface modules (ASM) and the controller takes place by means of cyclical I/O data and non-cyclical records, the component types in the *RFID* directory can only be used with couplings that support both methods of communication. These components can therefore only be used with the PROFIBUS DP and PROFINET IO coupling. The first table shows the interface modules supported for PROFIBUS DP, while the second table shows the interface modules supported for PROFINET IO.

Table 7-55 Interface modules for PROFIBUS DP

Interface module	MLFB	Head-end station supported for modular slaves
ASM 452	6GT2002-0EB20	
ASM 454	6GT2002-2EE00	
ASM 456	6GT2002-0ED00	
ASM 473	6GT2002-0HA00 6GT2002-0HA10	ET200X 6ES7 141-1BF00-0AB0 (80D2) ET200X 6ES7 141-1BF40-0AB0 (80D3) ET200X 6ES7 141-1BF11-0XB0 (803D) ET200X 6ES7 141-1BF12-0XB0 (803D) ET200X 6ES7 142-1BD21-0XB0 (803C) ET200X 6ES7 142-1BD22-0XB0 (803C) ET200X 6ES7 143-1BF00-0AB0 (809A) ET200X 6ES7 143-1BF00-0XB0 (809A)

Interface module	MLFB	Head-end station supported for modular slaves
ASM 475	6GT2002-0GA10	ET200M 6ES7 153-2BA00-0XB0 (801E) ET200M 6ES7 153-2BB00-0XB0 (8071)
ASM 754	6GT2302-2EE00	
ASM 850	6GT2402-2EA00	
ASM 854	6GT2402-2BB00	
RF170C	6GT2002-0HD00	ET200pro 6ES7 154-1AA00-0AB0 (8118) ET200pro 6ES7 154-1AA01-0AB0 (8118) ET200pro 6ES7 154-2AA00-0AB0 (8119) ET200pro 6ES7 154-2AA01-0AB0 (8119)

Table 7-56 Interface modules for PROFINET IO

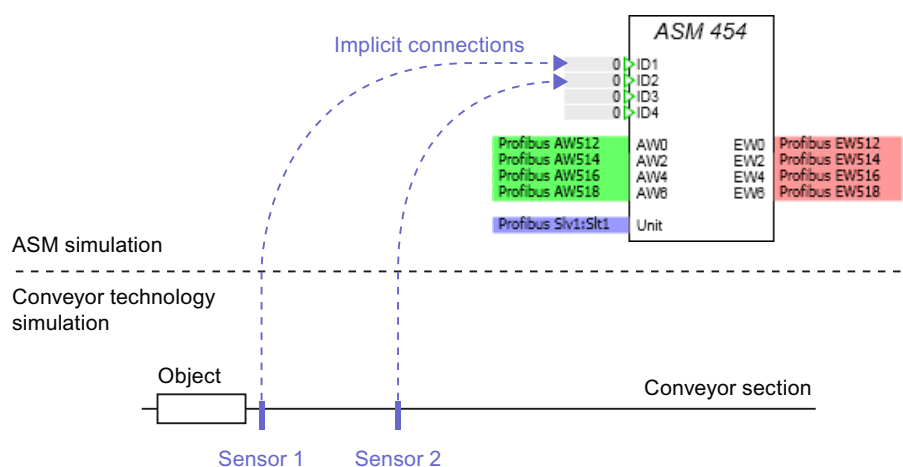
Interface module	MLFB	Head-end station supported for modular slaves
ASM 475	6GT2002-0GA10	ET200M 6ES7 153-4BA00-0XB0 ET200M 6ES7 153-4AA00-0XB0
RF170C	6GT2002-0HD00	ET200pro 6ES7 154-4AB10-0AB0 ET200pro 6ES7 154-6AB00-0AB0 ET200pro 6ES7 154-6AB50-0AB0

### Operating principle

Simulated objects are managed in SIMIT as object components in material lists. Every object component is automatically assigned a unique identifier (ID) at the start of the simulation, which can be accessed via the simulated sensors of a conveyor section component. In addition, the content of a mobile data storage (MDS) can be represented in object components. Accordingly, the two component types *CBoxDS256* and *VehicleDS256* in the *MATERIAL* directory of the *CONTEC* library have been created with data storage. In order to be able to exchange the contents of a simulated data storage with the controller, components are needed in the simulation that simulate the function of the corresponding interface module.

In the real system an interface module (ASM) is connected to a write/read device (SLG), which is positioned at a certain point in the conveyor system. An object transported past that point is detected by the SLG, and the SLG can read and in some cases also write to the object data storage.

In SIMIT, the read/write devices (SLG) that are actually connected to the ASM and their behavior are not simulated; only their functionality in terms of converting the data from the mobile data storages in the ASM component types is simulated. As shown schematically in the figure below, the identifiers (ID) of the objects as supplied by the simulated sensors on the conveyor section are transmitted to the ASM components. Therefore the corresponding inputs of the ASM components have to be implicitly connected to the simulated sensors.



This ID is used to notify the controller via the cyclical input signal that an object has been detected. Correspondingly, when the ASM component receives an acyclical command, it is executed with the data from the MDS of the detected object. The data storage is accessed via the ID.

All of the RFID component types contained in the *CONTEC* library include a simulation of the "SIMATIC sensors – RFID systems" for communication with the STEP 7 function FB45 in normal operation. Other operating modes of the RFID/Moby systems are not simulated and are therefore not supported in the simulation.

The RFID components support the FB45 commands listed in the figure below.

Table 7-57 Supported FB45 commands

Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Status SLG	0x04	Reads the status of the SLG
Antenna On/Off	0x0A	Switches the antenna on/off (if present)
MDS Status	0x0B	Reads the MDS status from an existing MDS

#### Note

Not every RFID component type supports all the commands listed in the table. A description of the commands supported by an RFID component type can be found in the description of that component type.

A mobile data storage (MDS) is simulated in SIMIT in the form of a state array of the *byte* type, which is created within the individual conveyor objects. This data is then accessed via the ID of the detected conveyor object reported to the ASM component and the *MDS* parameter configured in the ASM component. This parameter specifies the name of the state array to be accessed. In this way you can define multiple and also different MDS for a conveyor object, simply by storing details in the ASM component regarding which MDS is to be read or written to. However, for the component types of objects with MDS provided in the *CONTEC* library

7.3 The CONTEC library

(component types *CBoxDS256*, *VehicleDS256*), only one MDS is modeled. Because it is named "MDS", the correct default setting of parameters of ASM components ensures that access is available.

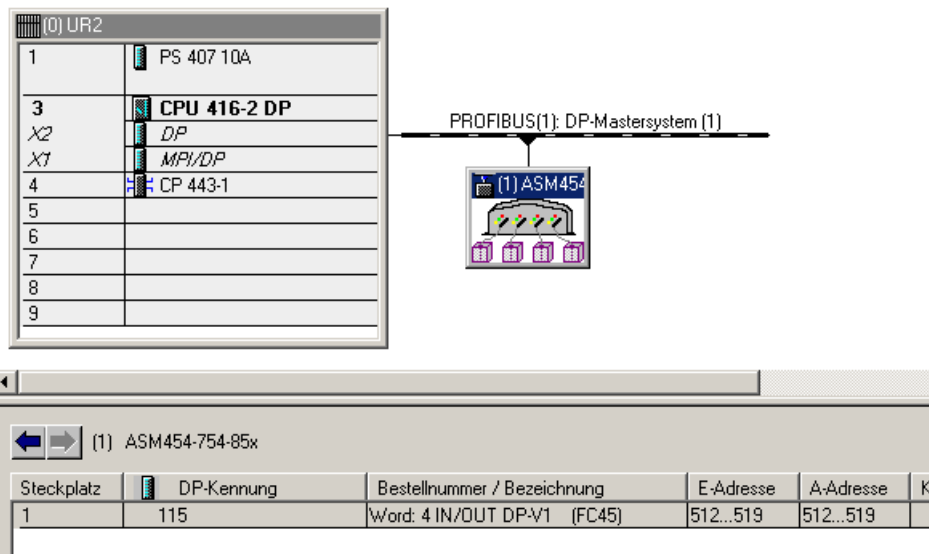
A missing MDS, missing state arrays or range violations are reported to the controller by an *RFID* component with the appropriate error messages.

Table 7-58 Possible error codes of RFID components

Command	Code	Meaning
Status OK:	0x00	No error (default)
Presence error	0x01	No MDS (ID==0) at the SLG
Unknown command:	0x05	The SIMIT component does not support the command sent to the ASM.
Address error:	0x0	<ul style="list-style-type: none"> <li>• Error when accessing the data in the MDS</li> <li>• Range violation when accessing the MDS</li> </ul>
Antenna error:	0x1C	<ul style="list-style-type: none"> <li>• Write/read commands when antenna is switched off</li> <li>• Incorrect antenna command (2x On / 2x Off)</li> </ul>

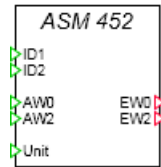
Signals are exchanged between RFID components and the controller both via input and output words from the coupling for cyclical communication and acyclically via corresponding communication functions implemented in the component type. For acyclical communication the Unit input of the ASM component has to be connected to the Unit connector. The Unit connector includes the slave and slot addresses or the device and slot addresses of the ASM from the hardware configuration. Using drag-and-drop, it can be copied to the diagram from the Hardware view in the property view of the PROFIBUS or PROFINET coupling and connected to the ASM component. To configure the Unit connector manually, use the notation "*SlvX:SlY*" for PROFIBUS DP, whereby *X* denotes the corresponding slave and *Y* the corresponding module, and "*DevX:SlY*" for PROFINET IO, whereby *X* denotes the device and *Y* the slot.

The figure below shows an example of a hardware configuration with a four-channel interface module that matches the view in the figure above.



## ASM452 – Interface module for identification systems

### Symbol



### Function

The *ASM452* component type simulates a 2-channel interface module for identification systems on PROFIBUS DP. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data for each channel is exchanged with the controller via the *AW0*, *EW0* and *AW2*, *EW2* inputs/outputs. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signals. The *ID1* and *ID2* inputs should be implicitly connected to the sensors of the material handling simulation.

Table 7-59 Supported *ASM452* commands

Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Status SLG	0x04	Reads the status of the SLG
Antenna On/Off	0x0A	Switches the antenna on/off (if present)
MDS Status	0x0B	Reads the MDS status from an existing MDS

### Parameters

Parameters *MDS1* and *MDS2* indicate which memory area of the MDS should be accessed by object components via sensor *ID1* or *ID2*.

The *AntennaInitStatus* parameter specifies the initial antenna status with which the SLG should be initialized in the ASM at the start of simulation: True := antenna on, False := antenna off.

The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True

### Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online modifiable parameters for each of the two simulated read/write devices (SLG).

Name	Value
SLG1_UDT110_hardware	0
SLG1_UDT110_hardware_version	0
SLG1_UDT110_loader_version	0
SLG1_UDT110_firmware	0
SLG1_UDT110_firmware_version	0
SLG1_UDT110_driver	0
SLG1_UDT110_driver_version	0
SLG1_UDT110_interface	0
SLG1_UDT110_baud	0
SLG1_UDT110_reserved1	0
SLG1_UDT110_reserved2	0
SLG1_UDT110_reserved3	0
SLG1_UDT110_distance_limiting_SLG	0
SLG1_UDT110_multitag_SLG	0
SLG1_UDT110_field_ON_control_SLG	0
SLG1_UDT110_field_ON_time_SLG	0
SLG1_UDT110_sync_SLG	0
SLG1_UDT110_stand_by	0
SLG1_UDT110_MDS_control	0

### Operating window

The operating window shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for both channels.

The screenshot shows a window titled "ASM452" with a close button. It displays the following data:

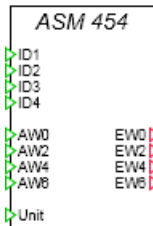
Channel	OUT		IN		
	1	2	1	2	
DO	■	■	■	■	#MDS
DO	■	■	■	■	#MDS
DO	■	■	■	■	#MDS
DO	■	■	■	■	#MDS
NU	■	■	■	■	SLG error
NU	■	■	■	■	ASM busy
repeat command	■	■	■	■	comand repeat active
cancel	■	■	■	■	ANZ cancel
Reserved	■	■	■	■	ANZ MDS present
Reserved	■	■	■	■	MDS presence changed
Reserved	■	■	■	■	Reserved
Reserved	■	■	■	■	Command counter (BL)
Reserved	■	■	■	■	Command counter (BH)
Reserved	■	■	■	■	Acknowledgement counter (QL)
Reserved	■	■	■	■	Acknowledgement counter (QH)
init run	■	■	■	■	ANZ reset

Channel	1	2	Command	Command Status
Antenna On/Off	■	■	00 Reset	00 OK
Command	■.■.■	■.■.■	01 Write	01 Presence error
Command Status	■.■.■	■.■.■	02 Read	05 Unknown Cmd
			03 Init MDS	13 Address error
			04 Status SLG	28 Antenna error
			10 Ant On/Off	
			11 Status MDS	

### ASM454 – Interface module for identification systems

#### Symbol



**Function**

The *ASM454* component type simulates a 4-channel interface module for identification systems on PROFIBUS DP. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data for each channel is exchanged with the controller via the *AW0, EW0* to *AW6, EW6* inputs/outputs. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signals. The inputs *ID1* to *ID4* should be implicitly connected to the sensors of the material handling simulation.

Table 7-60 Supported ASM454 commands

Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS

**Parameters**

Parameters *MDS1* to *MDS4* indicate which area of the MDS should be accessed by object components via sensor *ID1* to *ID4*. The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
MDS3	MDS
MDS4	MDS



### Operating window

The operating window shows the current states of the input and output words of the four channels that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the most recently executed command and the associated command status for each of the four channels.

The screenshot shows the ASM454 operating window with the following data:

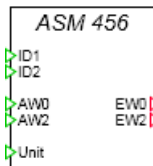
Channel	OUT				IN				Command	Command Status
	1	2	3	4	1	2	3	4		
DO	■	■	■	■	■	■	■	■	#MDS	
DO	■	■	■	■	■	■	■	■	#MDS	
DO	■	■	■	■	■	■	■	■	#MDS	
DO	■	■	■	■	■	■	■	■	#MDS	
NU	■	■	■	■	■	■	■	■	SLG error	
NU	■	■	■	■	■	■	■	■	ASM busy	
repeat command	■	■	■	■	■	■	■	■	comand repeat active	
cancel	■	■	■	■	■	■	■	■	ANZ cancel	
Reserved	■	■	■	■	■	■	■	■	ANZ MDS present	
Reserved	■	■	■	■	■	■	■	■	MDS presence changed	
Reserved	■	■	■	■	■	■	■	■	Reserved	
Reserved	■	■	■	■	■	■	■	■	Command counter (BL)	
Reserved	■	■	■	■	■	■	■	■	Command counter (BH)	
Reserved	■	■	■	■	■	■	■	■	Acknowledgement counter (QL)	
Reserved	■	■	■	■	■	■	■	■	Acknowledgement counter (QH)	
init run	■	■	■	■	■	■	■	■	ANZ reset	

Channel	1	2	3	4	Command	Command Status
Command	###	###	###	###	00 Reset	00 OK
Command Status	###	###	###	###	01 Write	01 Presence error
					02 Read	05 Unknown Cmd
					03 Init MDS	13 Address error
					04 Status SLG	28 Antenna error
					10 Ant On/Off	
					11 Status MDS	

### ASM456 – Interface module for identification systems

#### Symbol



#### Function

The *ASM456* component type simulates a 2-channel interface module for identification systems on PROFIBUS DP. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data for each channel is exchanged with the controller via the *AW0*, *EW0* and *AW2*, *EW2* inputs/outputs. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signals. The *ID1* and *ID2* inputs should be implicitly connected to the sensors of the material handling simulation.

Table 7-61 Supported ASM456 commands


Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Status SLG	0x04	Reads the status of the SLG
Antenna On/Off	0x0A	Switches the antenna on/off (if present)
MDS Status	0x0B	Reads the MDS status from an existing MDS

### Parameters

Parameters *MDS1* and *MDS2* indicate which memory area of the MDS should be accessed by object components via sensor *ID1* or *ID2*.

The *AntennaInitStatus* parameter specifies the initial antenna status with which the SLG should be initialized in the ASM at the start of simulation: True := antenna on, False := antenna off.

The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True 

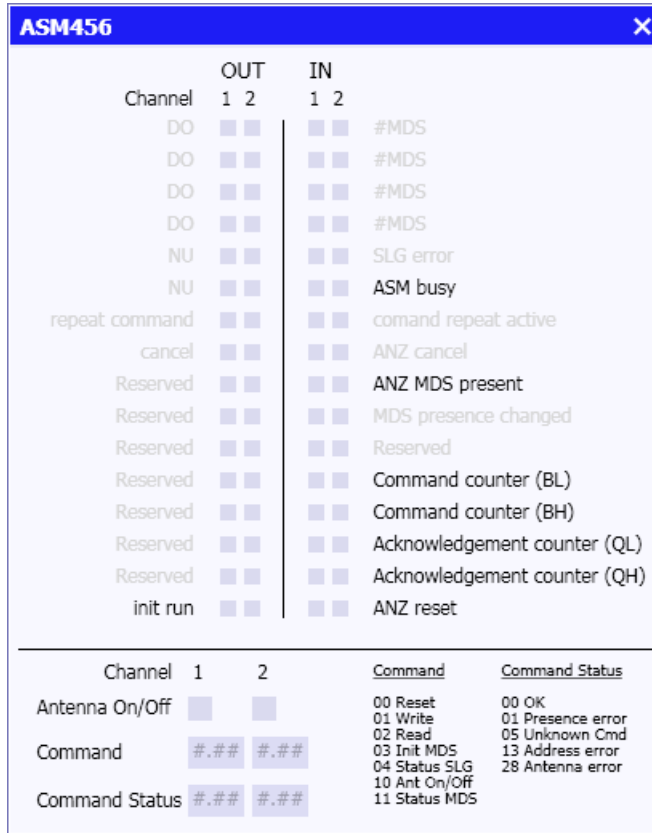
## Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online modifiable parameters for each of the two simulated read/write devices (SLG).

Name	Value
SLG1_UDT110_hardware	0
SLG1_UDT110_hardware_version	0
SLG1_UDT110_loader_version	0
SLG1_UDT110_firmware	0
SLG1_UDT110_firmware_version	0
SLG1_UDT110_driver	0
SLG1_UDT110_driver_version	0
SLG1_UDT110_interface	0
SLG1_UDT110_baud	0
SLG1_UDT110_reserved1	0
SLG1_UDT110_reserved2	0
SLG1_UDT110_reserved3	0
SLG1_UDT110_distance_limiting_SLG	0
SLG1_UDT110_multitag_SLG	0
SLG1_UDT110_field_ON_control_SLG	0
SLG1_UDT110_field_ON_time_SLG	0
SLG1_UDT110_sync_SLG	0
SLG1_UDT110_stand_by	0
SLG1_UDT110_MDS_control	0

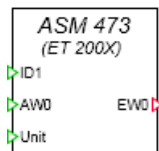
**Operating window**

The operating window shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for both channels.



**ASM473 – Interface module for identification systems**

**Symbol**



**Function**

The *ASM473* component type simulates a 1-channel interface module for identification systems on PROFIBUS DP. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data is exchanged with the controller via the *AW0*, *EWO* input/output. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signal. The *ID1* input should be implicitly connected to the sensor of the material handling simulation.

Table 7-62 Supported *ASM473* commands


Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Status SLG	0x04	Reads the status of the SLG
Antenna On/Off	0x0A	Switches the antenna on/off (if present)
MDS Status	0x0B	Reads the MDS status from an existing MDS

## Parameters

Parameter *MDS1* indicates which memory area of the MDS should be accessed by object components via sensor *ID1*.

The *AntennaInitStatus* parameter specifies the initial antenna status with which the SLG should be initialized in the ASM at the start of simulation: True := antenna on, False := antenna off.

The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
AntennaInitStatus	True 

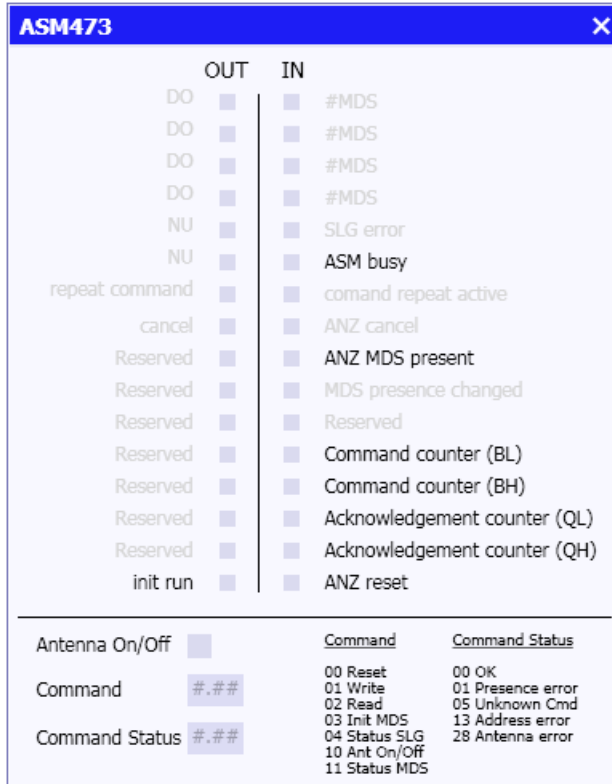
### Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online-editable parameters for the simulated read/write device (SLG).

Name	Value
SLG1_UDT110_hardware	0
SLG1_UDT110_hardware_version	0
SLG1_UDT110_loader_version	0
SLG1_UDT110_firmware	0
SLG1_UDT110_firmware_version	0
SLG1_UDT110_driver	0
SLG1_UDT110_driver_version	0
SLG1_UDT110_interface	0
SLG1_UDT110_baud	0
SLG1_UDT110_reserved1	0
SLG1_UDT110_reserved2	0
SLG1_UDT110_reserved3	0
SLG1_UDT110_distance_limiting_SLG	0
SLG1_UDT110_multitag_SLG	0
SLG1_UDT110_field_ON_control_SLG	0
SLG1_UDT110_field_ON_time_SLG	0
SLG1_UDT110_sync_SLG	0
SLG1_UDT110_stand_by	0
SLG1_UDT110_MDS_control	0

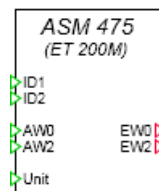
### Operating window

The operating window shows the current states of the input and output word that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status.



### ASM475 – Interface module for identification systems

#### Symbol



#### Function

The *ASM475* component type simulates a 2-channel interface module for identification systems on PROFIBUS DP or PROFINET IO. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data for each channel is exchanged with the controller via the *AW0*, *EW0* and *AW2*, *EW2* inputs/outputs. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signals. The *ID1* and *ID2* inputs should be implicitly connected to the sensors of the material handling simulation.

Table 7-63 Supported *ASM475* commands


Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Status SLG	0x04	Reads the status of the SLG
Antenna On/Off	0x0A	Switches the antenna on/off (if present)
MDS Status	0x0B	Reads the MDS status from an existing MDS

### Parameters

Parameters *MDS1* and *MDS2* indicate which memory area of the MDS should be accessed by object components via sensor *ID1* or *ID2*.

The *AntennaInitStatus* parameter specifies the initial antenna status with which the SLG should be initialized in the ASM at the start of simulation: True := antenna on, False := antenna off.

The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True 



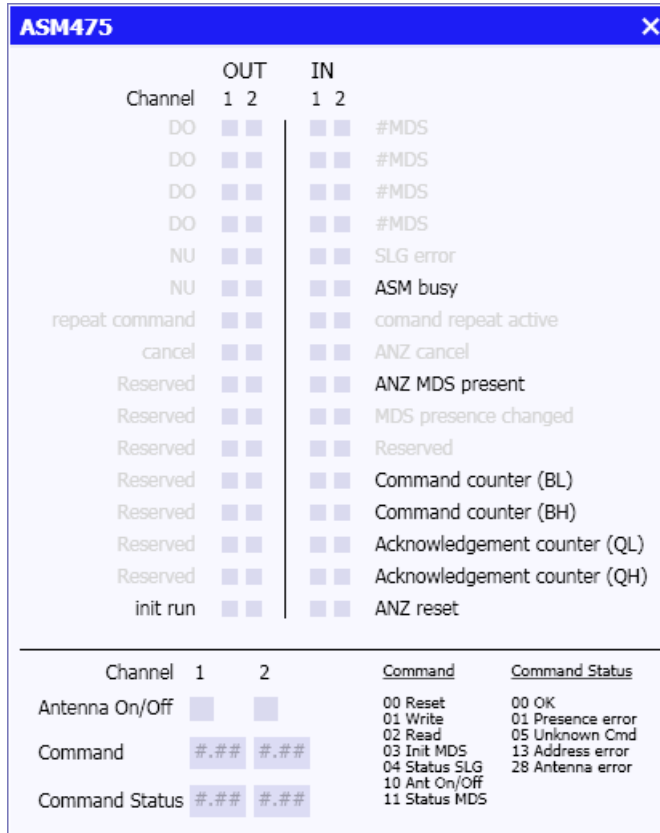
## Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online modifiable parameters for each of the two simulated read/write devices (SLG).

Name	Value
SLG1_UDT110_hardware	0
SLG1_UDT110_hardware_version	0
SLG1_UDT110_loader_version	0
SLG1_UDT110_firmware	0
SLG1_UDT110_firmware_version	0
SLG1_UDT110_driver	0
SLG1_UDT110_driver_version	0
SLG1_UDT110_interface	0
SLG1_UDT110_baud	0
SLG1_UDT110_reserved1	0
SLG1_UDT110_reserved2	0
SLG1_UDT110_reserved3	0
SLG1_UDT110_distance_limiting_SLG	0
SLG1_UDT110_multitag_SLG	0
SLG1_UDT110_field_ON_control_SLG	0
SLG1_UDT110_field_ON_time_SLG	0
SLG1_UDT110_sync_SLG	0
SLG1_UDT110_stand_by	0
SLG1_UDT110_MDS_control	0

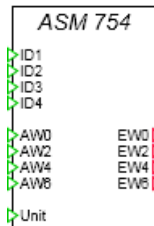
Operating window

The operating window shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for both channels.



ASM754 – Interface module for identification systems

Symbol



## Function

The *ASM754* component type simulates a 4-channel interface module for identification systems on PROFIBUS DP. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data for each channel is exchanged with the controller via the *AW0*, *EW0* to *AW6*, *EW6* inputs/outputs. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signals. The inputs *ID1* to *ID4* should be implicitly connected to the sensors of the material handling simulation.

Table 7-64 Supported ASM754 commands

Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS

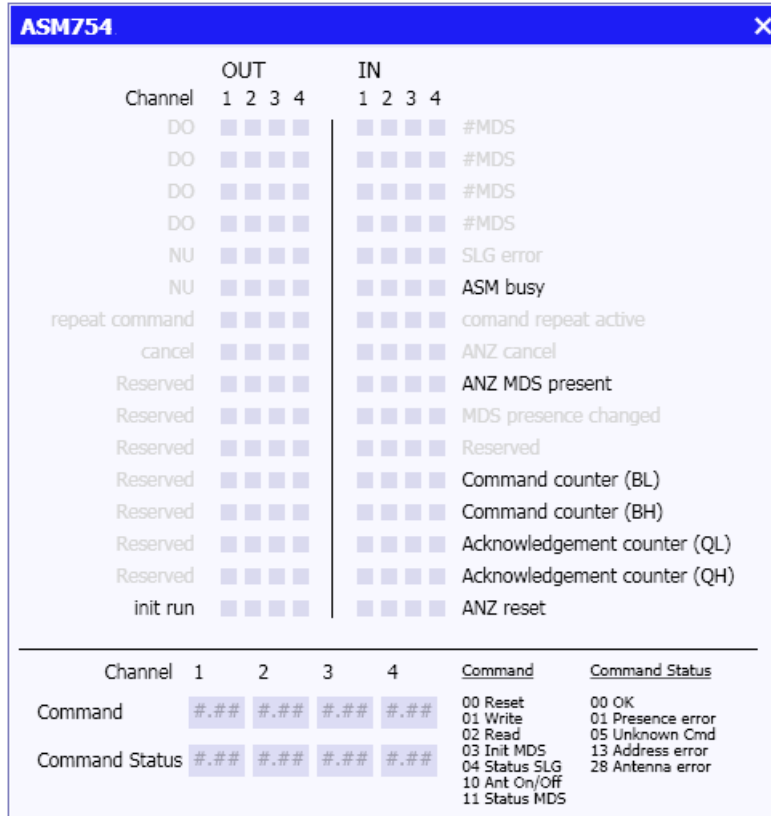
## Parameters

Parameters *MDS1* to *MDS4* indicate which memory area of the MDS should be accessed by object components via sensor *ID1* to *ID4*. The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
MDS3	MDS
MDS4	MDS

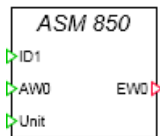
Operating window

The operating window shows the current states of the input and output words of the four channels that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for each of the four channels.



ASM850 – Interface module for identification systems

Symbol



Function

The *ASM850* component type simulates a 1-channel interface module for identification systems on PROFIBUS DP. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data is exchanged with

the controller via the *AW0*, *EW0* input/output. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signal. The *ID1* input should be implicitly connected to the sensor of the material handling simulation.

Table 7-65 Supported ASM850 commands


Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Antenna On/Off	0x0A	Switches the antenna on/off (if present)

## Parameters

Parameter *MDS1* indicates which memory area of the MDS should be accessed by object components via sensor *ID1*.

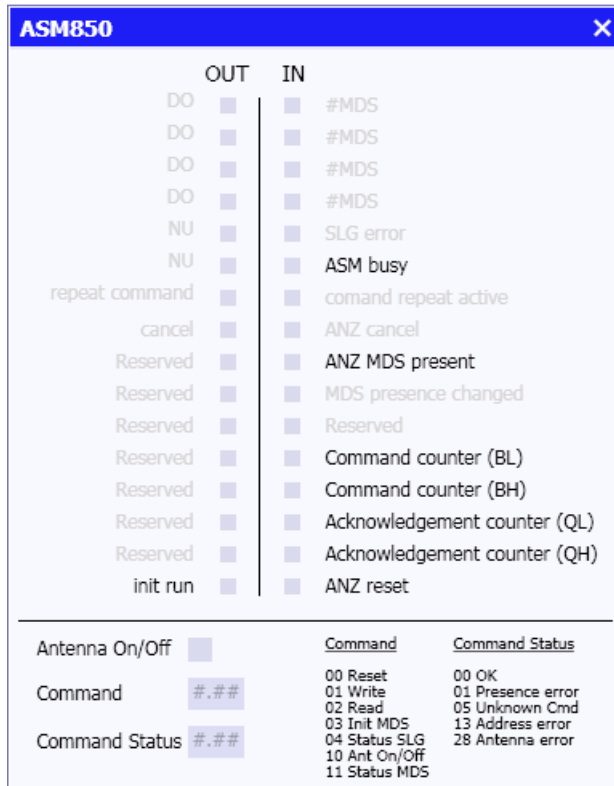
The *AntennaInitStatus* parameter specifies the initial antenna status with which the SLG should be initialized in the ASM at the start of simulation: True := antenna on, False := antenna off.

The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
AntennaInitStatus	True 

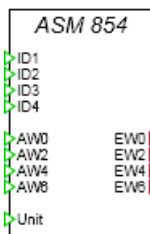
**Operating window**

The operating window shows the current states of the input and output word that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status.



**ASM854 – Interface module for identification systems**

**Symbol**



**Function**

The *ASM854* component type simulates a 4-channel interface module for identification systems on PROFIBUS DP. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data for each channel is exchanged with the controller via the *AW0*, *EW0* to *AW6*, *EW6* inputs/outputs. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signals. The inputs *ID1* to *ID4* should be implicitly connected to the sensors of the material handling simulation.

Table 7-66 Supported *ASM854* commands


Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Antenna On/Off	0x0A	Switches the antenna on/off (if present)

## Parameters

Parameters *MDS1* to *MDS4* indicate which memory area of the MDS should be accessed by object components via sensor *ID1* to *ID4*.

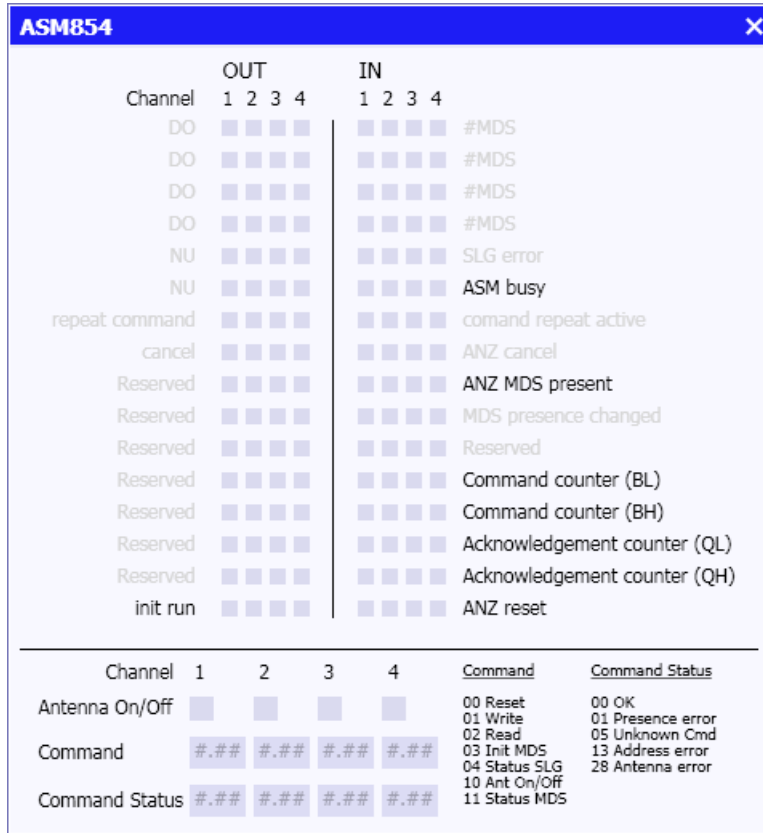
The *AntennaInitStatus* parameter specifies the initial antenna status with which the SLG should be initialized in the ASM at the start of simulation: True := antenna on, False := antenna off.

The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
MDS3	MDS
MDS4	MDS
AntennaInitStatus	True 

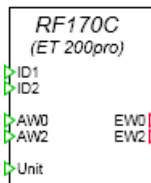
Operating window

The operating window shows the current states of the input and output words of the four channels that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for each of the four channels.



RF170C – Interface module for identification systems

Symbol



Function

The RF170C component type simulates a 2-channel interface module for identification systems on PROFIBUS DP or PROFINET IO. The commands listed in the table below are supported.



All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data for each channel is exchanged with the controller via the *AW0*, *EW0* and *AW2*, *EW2* inputs/outputs. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signals. The *ID1* and *ID2* inputs should be implicitly connected to the sensors of the material handling simulation.

Table 7-67 Supported *RF170C* commands


Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Status SLG	0x04	Reads the status of the SLG
Antenna On/Off	0x0A	Switches the antenna on/off (if present)
MDS Status	0x0B	Reads the MDS status from an existing MDS

## Parameters

Parameters *MDS1* and *MDS2* indicate which memory area of the MDS should be accessed by object components via sensor *ID1* or *ID2*.

The *AntennaInitStatus* parameter specifies the initial antenna status with which the SLG should be initialized in the ASM at the start of simulation: True := antenna on, False := antenna off.

The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True 

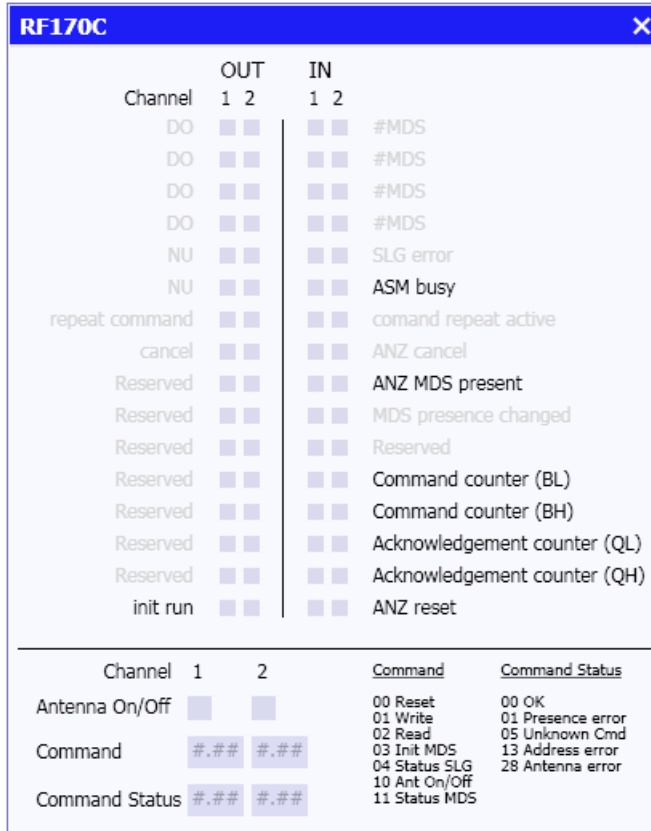
### Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online modifiable parameters for each of the two simulated read/write devices (SLG).

Name	Value
SLG1_UDT110_hardware	0
SLG1_UDT110_hardware_version	0
SLG1_UDT110_loader_version	0
SLG1_UDT110_firmware	0
SLG1_UDT110_firmware_version	0
SLG1_UDT110_driver	0
SLG1_UDT110_driver_version	0
SLG1_UDT110_interface	0
SLG1_UDT110_baud	0
SLG1_UDT110_reserved1	0
SLG1_UDT110_reserved2	0
SLG1_UDT110_reserved3	0
SLG1_UDT110_distance_limiting_SLG	0
SLG1_UDT110_multitag_SLG	0
SLG1_UDT110_field_ON_control_SLG	0
SLG1_UDT110_field_ON_time_SLG	0
SLG1_UDT110_sync_SLG	0
SLG1_UDT110_stand_by	0
SLG1_UDT110_MDS_control	0

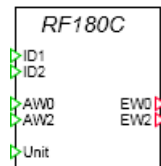
### Operating window

The operating window shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for both channels.



### RF180C – Interface module for identification systems

#### Symbol



#### Function

The *RF180C* component type simulates a 2-channel interface module for identification systems on PROFINET IO. The commands listed in the table below are supported.

All inputs and outputs of the component types are integers. The ASM slave and module address is set at the *Unit* input using the Unit connector. The cyclical data for each channel is exchanged with the controller via the *AW0*, *EW0* and *AW2*, *EW2* inputs/outputs. They have to be linked to the PROFIBUS DP coupling via the corresponding input/output signals. The *ID1* and *ID2* inputs should be implicitly connected to the sensors of the material handling simulation.

Table 7-68 Supported *RF180C* commands


Command	Code	Meaning
Reset	0x00	Reset of the SLG
Write	0x01	Writes data to an existing MDS
Read	0x02	Reads data from an existing MDS
Init MDS	0x03	Initializes the data area of an MDS
Status SLG	0x04	Reads the status of the SLG
Antenna On/Off	0x0A	Switches the antenna on/off (if present)
MDS Status	0x0B	Reads the MDS status from an existing MDS

### Parameters

Parameters *MDS1* and *MDS2* indicate which memory area of the MDS should be accessed by object components via sensor *ID1* or *ID2*.

The *AntennaInitStatus* parameter specifies the initial antenna status with which the SLG should be initialized in the ASM at the start of simulation: True := antenna on, False := antenna off.

The figure below shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True 

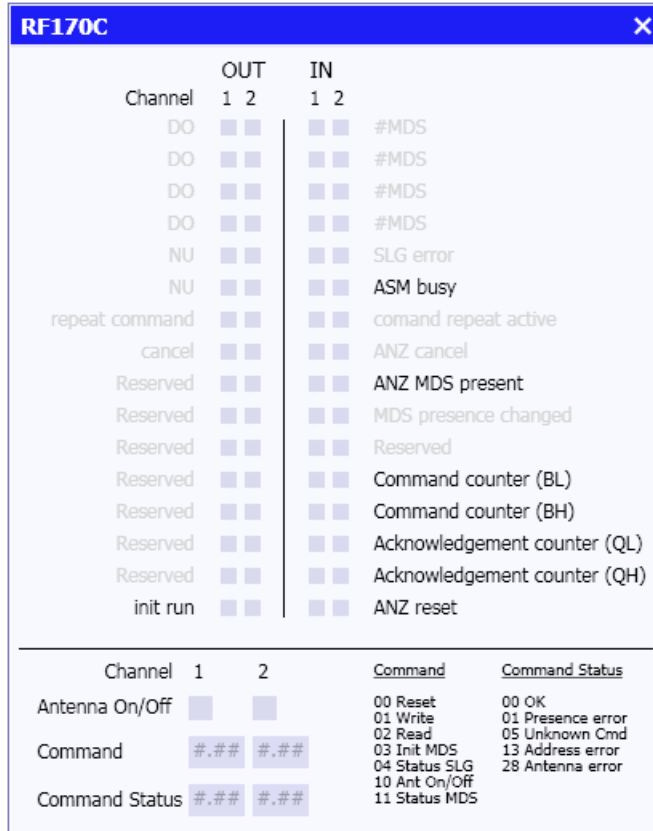
## Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online modifiable parameters for each of the two simulated read/write devices (SLG).

Name	Value
SLG1_UDT110_hardware	0
SLG1_UDT110_hardware_version	0
SLG1_UDT110_loader_version	0
SLG1_UDT110_firmware	0
SLG1_UDT110_firmware_version	0
SLG1_UDT110_driver	0
SLG1_UDT110_driver_version	0
SLG1_UDT110_interface	0
SLG1_UDT110_baud	0
SLG1_UDT110_reserved1	0
SLG1_UDT110_reserved2	0
SLG1_UDT110_reserved3	0
SLG1_UDT110_distance_limiting_SLG	0
SLG1_UDT110_multitag_SLG	0
SLG1_UDT110_field_ON_control_SLG	0
SLG1_UDT110_field_ON_time_SLG	0
SLG1_UDT110_sync_SLG	0
SLG1_UDT110_stand_by	0
SLG1_UDT110_MDS_control	0

Operating window

The operating window shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in gray are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for both channels.



7.3.4 Creating custom component types for material handling simulation

The SIMIT component type editor (*CTE*) included in SIMIT Ultimate allows you to create your own component types, which use the mechanisms of the conveyor technology simulation solution procedure. You can thus expand the functions of the components supplied with the *CONTEC* library, for example, by implementing more complex transport processes that are not covered by the supplied library components and thus enhance your conveyor technology simulations. You can also create your own component types from scratch to extend the *CONTEC* library and adapt it to your specific conditions.

Three aspects must be considered when creating component types:

- The topological aspects,
- The connection to the solution procedure, and
- The specific behavior of the component.

The topological aspect is covered by appropriate extensions to the component type definition. Special connection types are provided for the connection to the solution procedure.

In addition, the general descriptions of component properties in the "SIMIT - Component Type Editor" manual form the basis for creating component types for material handling simulation. The general properties also apply in full to these component types.

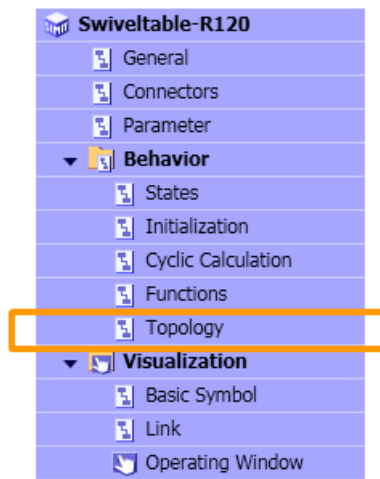
### 7.3.4.1 Topological properties

The topology of the modeled conveyor system is automatically determined when compiling a simulation project from interconnected material handling components. Each component must therefore provide relevant topological information about itself, which means information about how it is to be treated topologically in the system. From a topological point of view it is necessary to know how the reference directions for variables in a material handling system are defined for a component and how data exchange between the component and the solution procedure is set up.

Elements of the material handling system with topological information are:

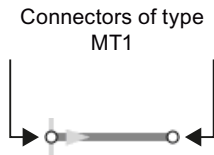
- Segments of conveyor sections
- Branches
- Boundaries

Relationships with the topological connectors of the material handling component must be defined for each of these elements. To do this, open the topology editor by double-clicking on the *Topology* entry in the component type navigation menu.



### Topological connectors (connection type MT1)

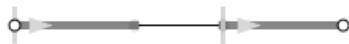
The *MT1* connection type indicates the connectors that are used to connect material handling components to one another. The topology of the conveyor system is derived from interconnected connectors of this type and the topological information about the individual handling equipment components. *MT1* is purely a topological connector type, which means it does not carry any signals. Connectors of this type are simply referred to below as **topological connectors**. Topological connectors are represented by circles.



They can be joined by superimposing the connectors. Once linked, both connectors are hidden.



If topological connectors are connected with each other by a connecting line in the chart editor, then the connected connectors are hidden. They are still visible only as pale shadows. Connectors of the *MT1* type cannot be joined more than once; they can only ever establish a 1:1 connection.



### Topology of a segment

The topology description of a segment is used to set its reference direction. For example, the definition

```
FROM a TO b;
```

defines a segment with a reference direction from the starting point *a* to the end point *b*. The starting point and end point can be defined in the component type as a connector of type *MT1* or as a branch or boundary.

Speeds are positive in the reference direction. The reference direction is also the preferred direction, which is used by the solution procedure to determine the calculation order.

### Topology of a branch

A branch as an internal node in a component type is defined as follows in the topology description:

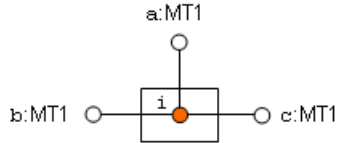
```
INTERNAL_NODE i;
```

The topological connectors of the component must also be assigned to this branch. Three connectors can be assigned in the topological description as per the example below:

```
FROM i TO a;  
FROM i TO b;  
FROM i TO c;
```



The three connectors a, b and c must be defined as type *FLN1* connectors in the component type. The figure below shows a diagram of the resulting topological structure of the component type:



A branch can also be defined as multiple segments having a common starting or end point that is defined as an *MT1* topological connector.

### Topology of a boundary

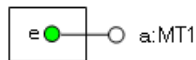
In the topological description of a component type a boundary point as an external node *e* is defined as follows:

```
EXTERNAL_NODE e;
```

This external node must be also be linked to at least one topological connector of the component. The topology description is, for example, to be completed as

```
FROM e TO a;
```

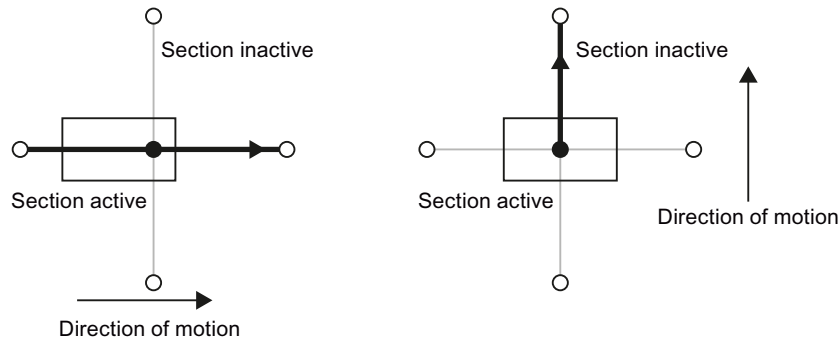
. The topological connector a must be defined in the component type as a connector of type *MT1*. The figure below shows a diagram of the resulting topological structure of the component type.



### Determining the next section

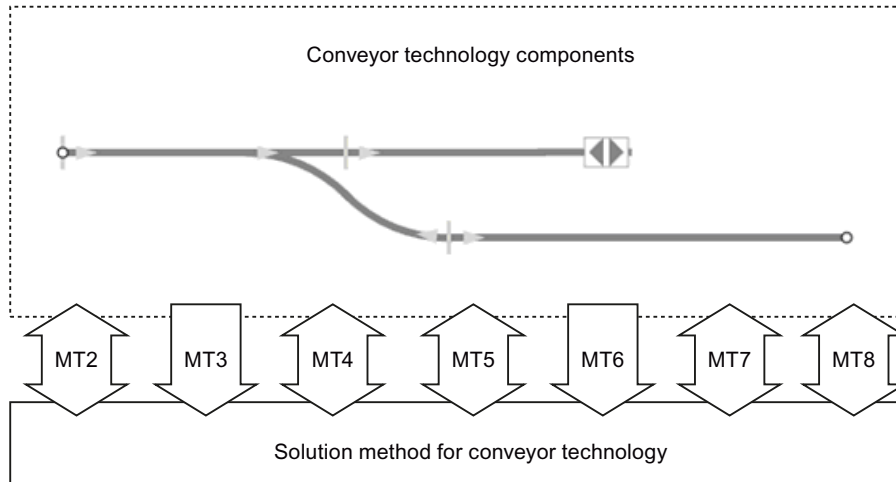
An object is only ever assigned to one segment in the conveyor system network. Transport can only take place if the transition to the next segment is unambiguously defined. This means only one possible subsequent segment can be identified as active at branching points; all other alternative segments must be inactive.

An exception occurs at a branch as an internal node of a component, if the outline of an object covers the branch and only one of the segments connected to the branch is active. In this case the object is assigned to this active segment.



### 7.3.4.2 Connection to the solution procedure

The segments, branches and boundary points of a component and the solution procedure for the material handling exchange data via special connectors. There are seven different connection types *MT2* to *MT8* available for this purpose. Their use in material handling components is explained in the sections below.



Connectors of connection types *MT2* to *MT8* establish a connection to the solution procedure and must therefore be set as **hidden** on the component symbol. In the connector properties, the usage "*Only in property view*" or "*Only in CTE*" must be set.

Property	Value
Usage	Property view only
Visibility Default	

Connectors of connection type *MT2* to *MT7* should be defined in the OUT direction, a connector of connection type *MT8* in the IN direction.

#### See also

- Connection type *MT2* for segments (Page 666)
- Connection type *MT3* for stoppers (Page 667)
- Connection type *MT4* for sensors (Page 668)
- Connector type *MT5* for placing objects (Page 669)
- Connection type *MT6* for positions (Page 670)
- Connector type *MT7* for parameter assignment of objects (Page 671)

#### Connection type *MT2* for segments

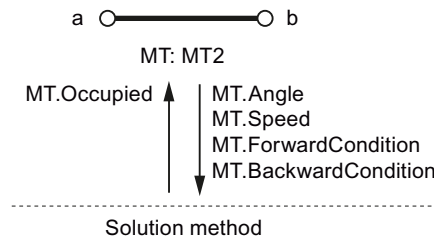
A segment can exchange variables with the solution procedure via a connector of type *MT2*. For a connector with the name *MT* the topological description of the segment is completed as follows:

```
FROM a TO b: MT;
```

The connector must always be defined in the **OUT** direction. It connects the component to the solution procedure to exchange various variables that are relevant to the segment between the solution procedure and the component.

Name	Connection type	Direction	Dimension
MT	MT2	OUT	1

The figure below shows the input and output signals with the direction of data exchange between the component and the solution procedure.



The output signals for the segment are set or calculated in the component and sent to the solution procedure:

1. **Angle** (integer)  
The angle in degrees of the segment of a circle that describes this segment. Positive angles rotate in a clockwise direction, negative angles rotate in a counterclockwise direction. For a straight connection the angle should be set to zero.  
This setting is only evaluated when the simulation is initialized; changing the value during the cyclical calculation has no effect.
2. **Speed** (analog)  
The speed in m/s at which objects are to be moved over this segment. Positive values move it in the preferred direction, negative values move it against the preferred direction.
3. **ForwardCondition** (integer)  
Specifies whether this segment is active in the preferred direction (0) or not (-1).
4. **BackwardCondition** (integer)  
Specifies whether this segment is active against the preferred direction (0) or not (-1).

The component receives information about the conveyor section from the solution procedure via the input signal:

1. **Occupied** (bool)  
Specifies whether the outline of one or more objects is in contact with this segment.

Connection type *MT2* is used in the *Rail-S4* and *Conveyor-S4* library components, for example.

### Connection type MT3 for stoppers

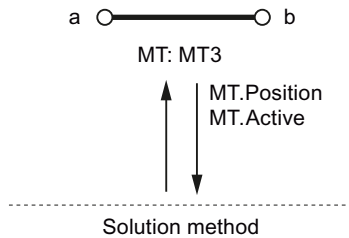
A segment can exchange variables with the solution procedure via a connector of type *MT3*. For a connector with the name *MT* the topological description of the segment is completed as follows:

```
FROM a TO b: MT;
```

The connector must always be defined in the **OUT** direction. It connects the component to the solution procedure to exchange various variables between the solution procedure and the component.

Name	Connection type	Direction	Dimension
MT	MT3	OUT	1

The figure below shows the input and output signals with the direction of data exchange between the component and the solution procedure.



The output signals for the segment are set or calculated in the component and sent to the solution procedure:

- Position** (analog)  
 The position of the stopper on the segment as a percentage of the total length of the segment.  
 This setting is only evaluated when the simulation is initialized; changing the value during the cyclical calculation has no effect.
- Active** (binary)  
 Indicates whether the stopper is active (True) or inactive (False).

Connector type *MT3* is used in the *Conveyor-S4-Stopper* library component, for example.

### Connection type MT4 for sensors

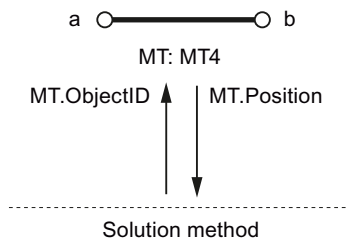
A segment can exchange variables with the solution procedure via a connector of type *MT4*. For a connector with the name *MT* the topological description of the segment is completed as follows:

FROM a TO b: MT;

The connector must always be defined in the OUT direction. It connects the component to the solution procedure to exchange various variables between the solution procedure and the component.

Name	Connection type	Direction	Dimension
MT	MT4	OUT	1

The figure below shows the input and output signals with the direction of data exchange between the component and the solution procedure.



The output signals for the segment are set or calculated in the component and sent to the solution procedure:

1. **Position** (analog)

The position of the sensor on the segment as a percentage of the total length of the segment.

This setting is only evaluated when the simulation is initialized; changing the value during the cyclical calculation has no effect.

The component receives information about the segment from the solution procedure via the input signal:

1. **ObjectID** (integer)

The ID of the object that is currently detected by the sensor. If no object is detected, this value is zero.

Connection type *MT4* is used in the *Rail-S4* and *Conveyor-S4* library components, for example.

### Connector type MT5 for placing objects

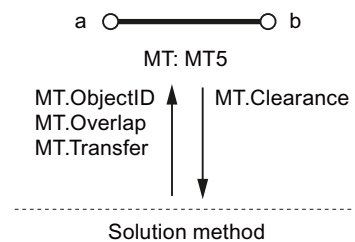
A connector of connection type *MT5* can be used for placing objects on a segment. For a connector with the name *MT*, the topological definition of the segment should be completed as follows:

```
FROM a TO b: MT;
```

The connector must always be defined in the OUT direction. It connects the component to the solution procedure to exchange various variables between the solution procedure and the component.

Name	Connection type	Direction	Dimension
MT	MT5	OUT	1

The figure below shows the input and output signals with the direction of data exchange between the component and the solution procedure.



The output signals for the segment are set or calculated in the component and sent to the solution procedure:

1. **ObjectID** (integer)  
The ID of the object to be placed on this segment. The component generally obtains this ID by means of appropriate function calls. You can find information on this in the following sections:
  - `_MT.GetObjectByName` (Page 675)
  - `_MT.GetObjectByType` (Page 675)
2. **Overlap** (analog)  
Overlap in millimeters in the preferred direction. This can be used to move the placement position in relative terms. The placement position must always be on the segment, however.
3. **Transfer** (bool)  
When this signal is set to *True*, the solution procedure starts the placement process. The component can only set the signal for exactly one cycle, after which it must return to *False*.

The component receives information about the segment from the solution procedure via the input signals:

1. **Clearance** (analog)  
The available length of segment in millimeters viewed in the preferred direction.

Connection type *MT5* is used in the *Rail-S4* and *Conveyor-S4* library components, for example.

### Connection type MT6 for positions

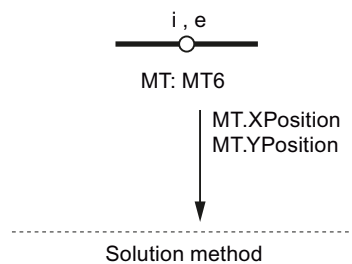
The positions of branches as internal nodes and boundary points as external nodes have to be configured individually. The topological definition of a node must be supplemented with a connector via which the position is sent to the solution procedure, for a connector *MT* as follows:

```
EXTERNAL_NODE E: MT;
INTERNAL_NODE I: MT;
```

The connector must always be defined in the **OUT** direction. It connects the component to the solution procedure to exchange various variables that are relevant to the node between the solution procedure and the component.

Name	Connection type	Direction	Dimension
MT	MT6	OUT	1

The figure below shows the input and output signals with the direction of data exchange between the component and the solution procedure.



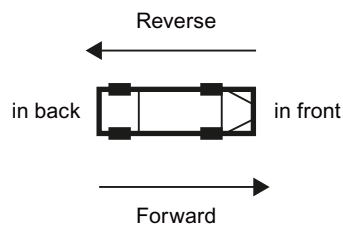
The output signals for the node are set or calculated in the component and sent to the solution procedure:

1. **XPosition** (analog)  
The X position of the node in millimeters relative to the top left corner of the component. This setting is only evaluated when the simulation is initialized; changing the value during the cyclical calculation has no effect.
2. **YPosition** (analog)  
The Y position of the node in millimeters relative to the top left corner of the component. This setting is only evaluated when the simulation is initialized; changing the value during the cyclical calculation has no effect.

### Connector type MT7 for parameter assignment of objects

An object can exchange variables with the solution procedure via a connector of type *MT7*. No topological description is necessary.

The orientation of the object is dependent on its configuration and corresponds to the way the basic symbol is displayed in the CTE.

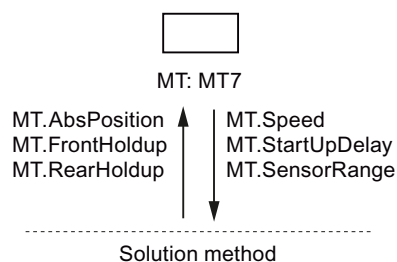


The connector of type *MT7* must always be defined in the **OUT** direction. It connects the component to the solution procedure to exchange various variables that are relevant to the object between the solution procedure and the component.

A component can have no more than one *MT7* type connector.

Name	Connection type	Direction	Dimension
MT	MT7	OUT	1

The figure below shows the input and output signals with the direction of data exchange between the component and the solution procedure.



The output signals for the object are set or calculated in the component and sent to the solution procedure:

1. **Speed** (analog)  
The speed of the object established by its drive in m/s. A positive speed moves the vehicle forwards, a negative speed moves it backwards.  
This speed is replaced by a speed set by the rail if applicable.
2. **StartUpDelay** (analog)  
The start-up delay in seconds. If the object is held up by the solution procedure, the object waits for this time before moving again once the hold-up is cleared.
3. **SensorRange** (analog)  
The percentage of the object length that is detected by a sensor. This value must be between 0 and 100%.

The component receives information about the object from the solution procedure via the input signals:

1. **AbsPosition** (analog)  
Reserved for subsequent development.
2. **FrontHoldup** (binary)  
If this signal is set to *True*, there is a stopper or another object directly in front of the object.
3. **RearHoldup** (binary)  
If this signal is set to *True*, there is a stopper or another object directly behind the object.

Connector type *MT7* is used in the *Vehicle* library component, for example.

### Connector type MT8 for transferring objects at boundary points

Objects can be transferred from the solution procedure to a component and vice versa at boundary points as external nodes. Once a component rather than the solution procedure has control over the object, the position of the object cannot be changed by the solution procedure. The object remains visible on the chart, however, and can now be positioned by the component by means of corresponding system functions. You can find additional information on this in the section: `_MT.SetPosition` (Page 677).

This allows more complex transport operations, which cannot be simulated with a path-based solution procedure, to be implemented in conveyor components.

A boundary point as an external node can exchange variables with the solution procedure via a connector of type *MT8*. For a connector with the name *MT* the topological description of the external node is completed as follows:

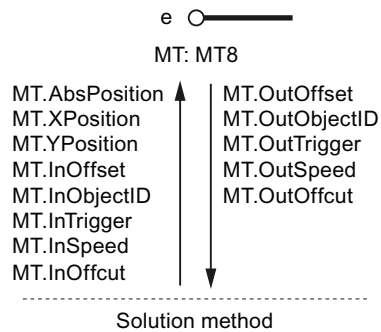
```
EXTERNAL_NODE E: MT;
```

The connector must always be defined in the **IN direction**. It connects the component to the solution procedure to exchange various variables between the solution procedure and the component.

Name	Connection type	Direction	Dimension
MT	MT8	IN	1

The figure below shows the input and output signals with the direction of data exchange between the component and the solution procedure.





The output signals for the external node are set or calculated in the component and sent to the solution procedure:

1. **OutOffset** (analog)  
The length of the available section, viewed from the connector into the component. A negative value means that viewed from the component, an object is projecting onto the adjacent section controlled by the solution procedure; in this case the value corresponds to the length of the projecting part of the object. All values are given in millimeters.
2. **OutObjectID** (integer)  
The object ID of the object that viewed from the component is projecting onto the adjacent section controlled by the solution procedure. The value is zero if no object is projecting.
3. **OutTrigger** (binary)  
The component sets this signal to *True* for exactly one cycle to show that the adjacent section controlled by the solution procedure has to take over the object.
4. **Out.Speed** (analog)  
The speed in m/s at which the object is transferred.
5. **OutOffcut** (analog)  
The time that the current cycle could not include because the object had reached the transfer node, measured in seconds. The solution procedure has to take this time into account in the next cycle to calculate an additional movement.

The component receives information from the solution procedure via the input signals:

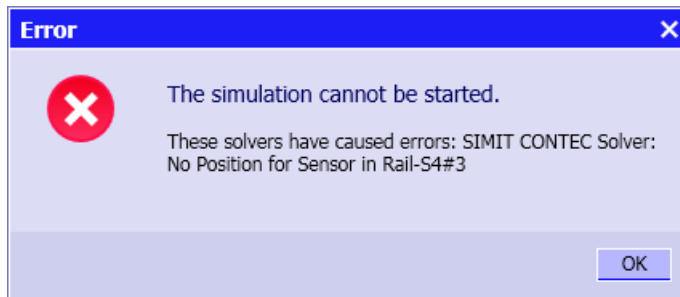
1. **AbsPosition** (analog)  
Reserved for subsequent developments.
2. **XPosition** (analog)  
The relative X position of the connector linked to the node relative to the top left corner of the component in millimeters.
3. **YPosition** (binary)  
The relative Y position of the connector linked to the node relative to the top left corner of the component in millimeters.
4. **InOffset** (analog)  
The length of the available section, viewed from the component connector, in the direction of the adjacent section outside the component. A negative value means that viewed from the adjacent section controlled by the solution procedure, an object is projecting into this component; in this case the value corresponds to the length of the projecting part of the object. All values are given in millimeters.

5. **InObjectID** (integer)  
The object ID of the object that viewed from the section controlled by the solution procedure is projecting into the component. The value is zero if no object is projecting.
6. **InTrigger** (binary)  
The solution procedure sets this signal to *True* for exactly one cycle to show that the component has to take over the object.
7. **InSpeed** (analog)  
The speed in m/s at which the object was last moved by the solution procedure.
8. **InOffcut** (analog)  
The time that the current cycle could not include because the object had reached the transfer node, measured in seconds. The component has to take this time into account in the next cycle in order to calculate an additional movement.

Connector type *MT8* is used in the *TransferCarriage* library component, for example.

### Errors in the component code

If interfaces to the solution procedure are created in material handling components but not all necessary signals are made available, the simulation start request is denied and a corresponding error message displayed.



#### 7.3.4.3 System functions

If you create your own component types you can utilize various system functions to access objects in the simulation.

All functions described below return an error code as the return value. In order for a function to be able to supply return values in a variable that is passed, this variable must be defined as a field (vector), in this case with the dimension one. Refer to the table below for the meaning of this return value.

Table 7-69 Return values of conveyor technology system functions

Value	Meaning
0	No error
-2	The object cannot be positioned directly because it is currently under solution procedure control.
-3	The object cannot be positioned or rotated because it is not currently assigned to a conveyor section.

Value	Meaning
-10	There is no object with the specified ID.
-11	The object with the specified ID is not available. <sup>1)</sup>
-12	There is no object with the specified name.
-13	The object with the specified name is not available.
-14	There is no object of the specified type.
-15	An object of the specified type is not available.
-20	The component has no signal with the specified name.
-21	The specified signal type cannot be used in this context.
-22	The address range of the byte array to be read or written has been exceeded.
-1000	General error

<sup>1)</sup> An object is not available if it is reserved for being assigned to a conveyor section or if it is assigned to a conveyor section.

### **\_MT.GetObjectByName**

The *\_MT.GetObjectByName* function returns the ID of the object with the instance name *name* if it is available as an object, otherwise it returns the value -1. The return value type is *long*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- name (string)  
The instance name of the object.

### **\_MT.GetObjectByType**

The *\_MT.GetObjectByType* function returns the ID of an object of component type *type* if it is available as an object, otherwise it returns the value -1. The return value type is *long*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- type (string)  
The type of object required.
- store (string)  
The name of the material list in which the object is located. If an empty string is passed here, all existing lists are searched.

### **\_MT.Restore**

The *\_MT.Restore* function releases an object. The return value is *void*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be released.
- delay (bool)  
Indicates whether the return should be delayed by one cycle so that the object is available for immediate reuse.

### **\_MT.GetWidth**

The *\_MT.GetWidth* function returns the width of an object in millimeters. The return value type is *double*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.

### **\_MT.GetHeight**

The *\_MT.GetHeight* function returns the height of an object in millimeters. The return value type is *double*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.

### **\_MT.GetDepth**

The *\_MT.GetDepth* function returns the depth of an object in millimeters. The return value type is *double*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.

### **\_MT.GetAngle**

The *\_MT.GetAngle* function returns the angle in degrees of an object relative to the tangent of the segment on which it is located (-180° to +180°). The return value type is *integer*.

Note that this function can only be called for objects that are assigned to a conveyor section.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.

### **\_MT.SetPosition**

The *\_MT.SetPosition* function sets the position of an object relative to the position of the conveyor section component to which this object is assigned. The position is specified in millimeters relative to the top left corner of the component (in its starting position). The return value type is *void*.

Note that this function can only be called for objects that are assigned to a conveyor section but are not under solution procedure control.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be changed.
- x (analog)  
X position in millimeters relative to the conveyor section component.
- y (analog)  
Y position in millimeters relative to the conveyor section component.

### **\_MT.SetAngle**

The *\_MT.SetAngle* function sets the angle in degrees of an object relative to the tangent of the segment on which it is located (-180° to +180°). The return value type is *void*.

Note that this function can only be called for objects that are assigned to a conveyor section.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be changed.
- angle (integer)  
Angle in degrees relative to the tangent of the segment.

### **\_MT.AddAngle**

The *\_MT.AddAngle* function increases the angle in degrees of an object relative to the tangent of the segment on which it is located (-180° to +180°). The return value type is *void*.

Note that this function can only be called for objects that are assigned to a conveyor section.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be changed.
- angle (integer)  
Angle in degrees relative to the tangent of the segment.

### **\_MT.SetHoldup**

The *\_MT.SetHoldup* function transfers to an object information about whether it is positioned directly in front of or behind a stopper or another object. The return value type is *void*.

Note that this function can only be called for objects that are assigned to a conveyor section but are not under solution procedure control.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be changed.
- front (binary)  
Information as to whether the object is blocked in front.
- rear (binary)  
Information as to whether the object is blocked to the rear.

### **\_MT.GetBinaryValue**

The *\_MT.GetBinaryValue* function is used to query a binary variable (state, input, output) of an object. The return value type is *bool*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the variable to be queried

### **\_MT.GetAnalogValue**

The *\_MT.GetAnalogValue* function is used to query an analog variable (discrete state, input, output) of an object. The return value type is *double*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the variable to be queried.

### **\_MT.GetIntegerValue**

The *\_MT.GetIntegerValue* function is used to query an integer variable (state, input, output) of an object. The return value type is *long*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the variable to be queried.

### **\_MT.GetByteValue**

The *\_MT.GetByteValue* function is used to query a byte (state) of an object. The return value type is *byte*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the state to be queried.

### **\_MT.GetByteArray**

The *\_MT.GetByteArray* function is used to query a byte array (state) of an object. The return value type is *void*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the state vector to be read.
- stateoffset (long)  
The offset in bytes from which the state vector is to be read.
- buffer (byte[])  
Byte array in which the read byte is to be entered. If you specify a state vector for the component, enter it in the notation for new state values, which means prefixed with "@".
- bufferoffset (long)  
The offset in bytes from which the read bytes are to be entered.
- count (long)  
The number of bytes to be read.

---

#### **Note**

All arrays must be large enough to execute this operation; otherwise, unpredictable errors could occur in the simulation process.

---

### **\_MT.SetBinaryValue**

The *\_MT.SetBinaryValue* function is used to write a binary variable (state, input) of an object. The return value type is *void*.



Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be described.
- property (string)  
The name of the state to be written.
- value (bool)  
The value to be written.

### **\_MT.SetAnalogValue**

The *\_MT.SetAnalogValue* function is used to write an analog variable (discrete state, input) of an object. The return value type is *void*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be described.
- property (string)  
The name of the variable to be written.
- value (double)  
The value to be written.

### **\_MT.SetIntegerValue**

The *\_MT.SetIntegerValue* function is used to write an integer variable (state, input) of an object. The return value type is *void*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be described.
- property (string)  
The name of the variable to be written.
- value (long)  
The value to be written.

### **\_MT.SetByteValue**

The *\_MT.SetByteValue* function is used to write a byte (state) of an object. The return value type is *void*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be described.
- property (string)  
The name of the state to be written.
- value (byte)  
The value to be written.

### **\_MT.SetByteArray**

The *\_MT.SetByteArray* function is used to write a byte array (state) of an object. The return value type is *void*.

Specify values for the following parameters:

- error (long[])  
Error code  
You can find information about the error code in the table in the following section: System functions (Page 674).
- id (long)  
The ID of the object to be described.
- property (string)  
The name of the state vector of the object to be described.
- stateoffset (long)  
The offset in bytes from which the state vector is to be written.
- buffer (byte[])  
Byte array containing the byte to be written.
- bufferoffset (long)  
The offset in bytes from which the byte array is to be transferred.
- count (long)  
The number of bytes to be written.

---

#### **Note**

All arrays must be large enough to execute this operation; otherwise, unpredictable errors could occur in the simulation process.

---

#### 7.3.4.4 System variables

The system variables listed in the table below are available for modeling conveyor technology components. These variables are available in the component behavior description with read access only.

Table 7-70 System variables

<b>Variable name</b>	<b>Meaning</b>
_WIDTH	Width of the component in pixels
_HEIGHT	Height of the component in pixels
_SCALEX	Horizontal scaling of the component. The factor one is the default setting.
_SCALEY	Vertical scaling of the component. The factor one is the default setting.
_TECHSCALE	Scale of the chart on which the component is located. The number of millimeters corresponding to one pixel is specified.



## Appendix

### 8.1 Automatic modeling > Table Import

Use this menu command to open the "Table import" dialog window:

You can find additional information on this in the section: "Table import" dialog box (Page 694).

### 8.2 Automatic modeling > IEA import

Use this menu command to open the "IEA import" dialog window:

You can find additional information on this in the section: "IEA import" dialog box (Page 693).

### 8.3 Automatic modeling > CMT import

Use this menu command to open the "CMT import" dialog window:

You can find additional information on this in the section: "CMT import" dialog box (Page 699).

### 8.4 Automatic modeling > Create device level

Use this menu command to open the "Create device level" dialog window:

You can find additional information on this in the section: "Create device level" dialog box (Page 695).

### 8.5 Automatic modeling > Generic import

Use this menu command to open the "Generic import" dialog window:

You can find additional information on this in the section: Generic import (Page 235).

## 8.6 Automatic model creation > Automatic parameter assignment

Use this menu command to open the "Automatic parameter assignment" dialog window:

You can find additional information on this in the section: "Automatic parameter assignment" dialog box (Page 695).

## 8.7 Portal view > Start

The "Start" button provides access to the following functions:

- **Open existing project**  
Here, you can open an existing project. You can find additional information on this in the section: Project > Open... (Page 703)
- **Create new project**  
Here, you can create a new project. You can find additional information on this in the section: Project > New project ... (Page 702)
- **Retrieve project**  
Here, you can retrieve a project. You can find additional information on this in the section: Project > Retrieve (Page 705)
- **Retrieve sample project**  
You retrieve one of the supplied sample projects here. Select the desired sample project from the drop-down list. You can freely select the storage location of the retrieved sample project.
- **Close project**  
Closes the current project.
- **Getting started**  
This explains the first steps with SIMIT.
- **Installed software**  
Here, you are shown the currently installed SIMIT software. You can find additional information on this in the section: "Info" dialog box (Page 697)
- **Help**  
You can open the help file for SIMIT here.
- **User interface language**  
You can change the user interface language of SIMIT here. The change takes effect when SIMIT is restarted.

## 8.8 Portal view > Couplings

The "Couplings" button provides access to the following functions:

- **Add new coupling**  
Select a communication path by which data should be exchanged in SIMIT.
- **Assign coupling signals**  
If the signals of the connectors can be uniquely assigned to a coupling, they can be updated with this function. This is required for name changes, for example.
- **IM configuration**  
If PROFINET IO or PROFIBUS DP is selected as a coupling, enter a name and the IP address of the simulation unit here.
- **Help**  
Here you access the SIMIT online help.

## 8.9 Portal view > Simulation model

The "Simulation model" button provides access to the following functions:

- **Add new chart**  
Add a new chart to your project. SIMIT switches to the project view in the chart editor.
- **Create new macro**  
Create a new macro. SIMIT switches to the project view in the Macro Editor.
- **Create new component type**  
Create a new component type. SIMIT switches to the project view in the Component Type Editor.
- **Help**  
Here you access the SIMIT online help.

## 8.10 Portal view > Automatic model creation

The "Automatic model creation" button provides access to the following functions:

- **Table import**  
Import a table. A chart is created in SIMIT for every row of the table. You can find additional information on this in the section: The table import (Page 226).
- **IEA import**  
Import an IEA file. A chart is created in SIMIT for every line of the IEA file. You can find additional information on this in the section: The IEA import (Page 229).
- **CMT import**  
Import a CMT file. A chart is created in SIMIT for every control module type of the CMT file. You can find additional information on this in the section: The CMT import (Page 232).
- **Create new template**  
Create a new template. SIMIT switches to the project view in the Template Editor. You can find additional information on this in the section: Templates (Page 218).

- **Generic import**  
Import an XML file. SIMIT automatically creates charts and instantiates templates based on this XML file. You can find additional information on this in the section: Generic import (Page 235).
- **Create device level**  
Create instances of templates. The components in the project must have the "TEMPLATE" and "HIERARCHY" parameters for this. You can find additional information on this in the section: "Create device level" dialog box (Page 695).
- **Automatic parameter assignment**  
Transfer parameters and input defaults from a table into existing charts. You can find additional information on this in the section: Automatic parameter assignment (Page 243).
- **Help**  
Here you access the SIMIT online help.

## 8.11 Portal view > Diagnostics & visualization

The "Diagnostics & visualization" button provides access to the following functions:

- **Consistency check**  
Check your project for formal errors. The result of the consistency check is displayed in the project view. You can find additional information on this in the section: Consistency check (Page 273).
- **Find & replace**  
Execute the "Find & replace" function in the current project. You can find additional information on this in the section: Find & replace (Page 265).
- **Add new trend**  
Create a new trend. SIMIT switches to the project view in the Trend Editor. You can find additional information on this in the section: Trends (Page 255).
- **Edit archive**  
Create an archive. A trend can be created from an archive. SIMIT switches to the project view in the Archive Editor.
- **Help**  
Here you access the SIMIT online help.

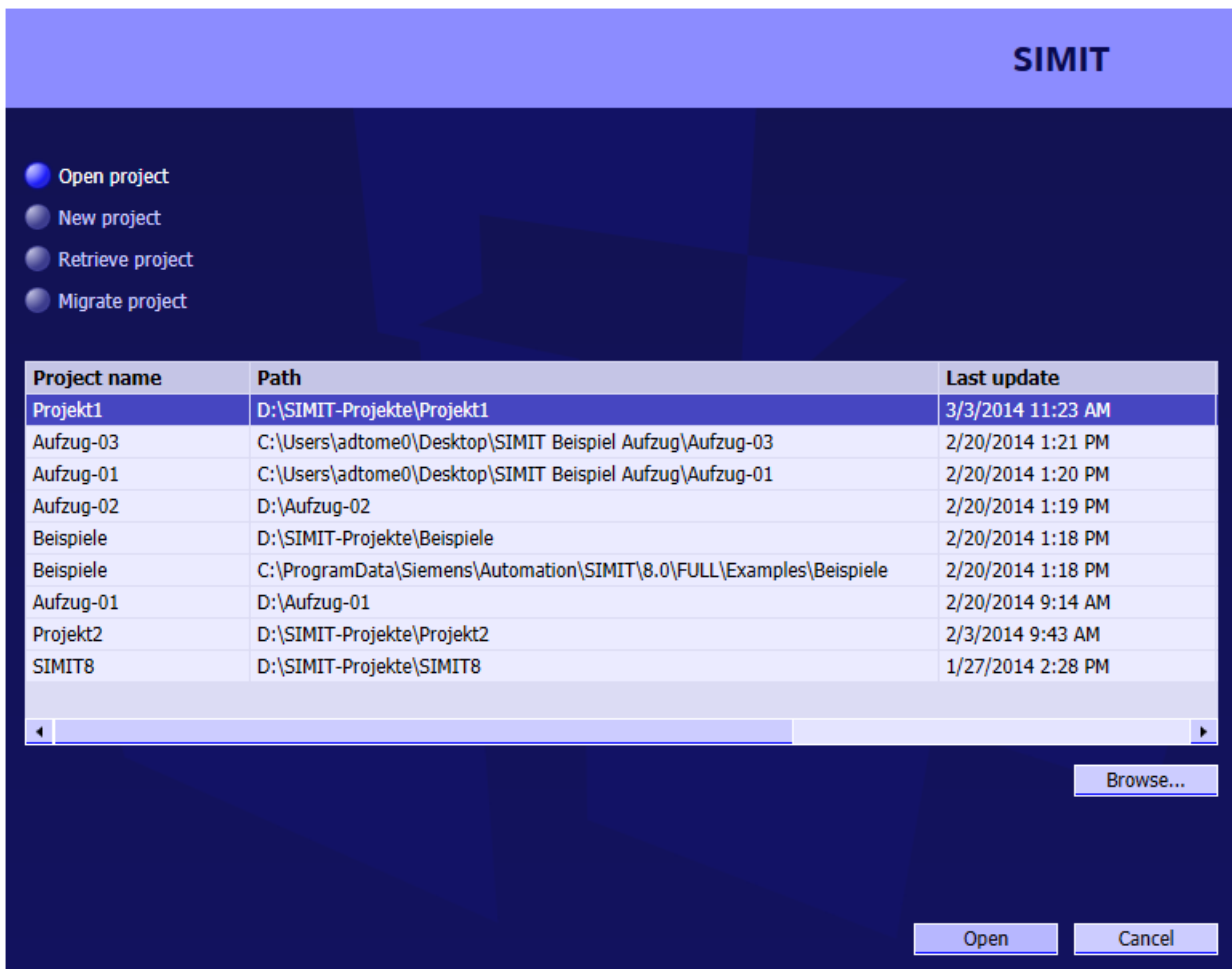
## 8.12 "Select project" dialog box

This dialog box opens with following menu commands:

- "Project > New project ..."
- "Project > Open"
- "Project > Retrieve"
- "Project > Migrate"

The appearance of the dialog box varies depending on the selection.



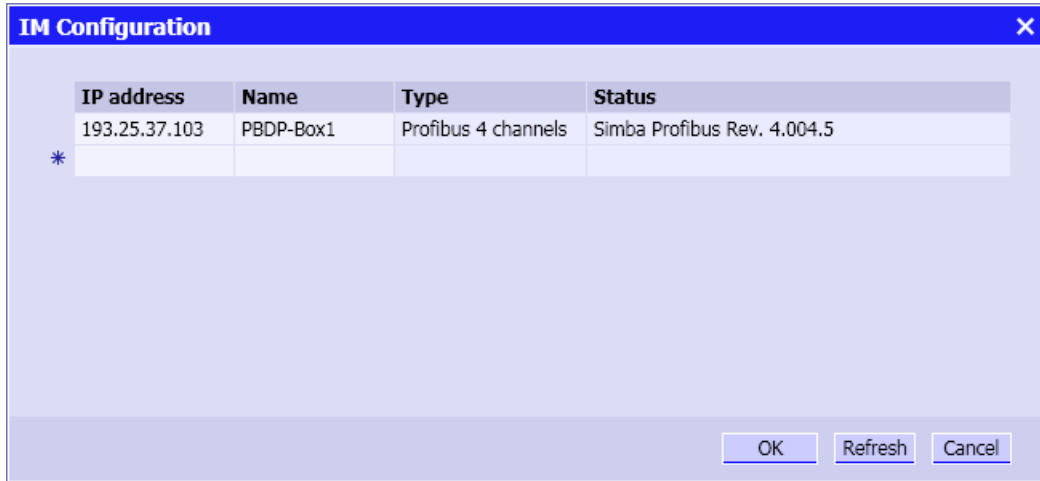


The following functions can be executed:

- "Open project" opens an existing project. You can find additional information on this in the section: Project > Open... (Page 703). SIMIT projects have the file extension ".simit".
- Create a new project with "New project". You can find additional information on this in the section: Project > New project ... (Page 702).
- You can retrieve a project with "Retrieve project". You can find additional information on this in the section: Project > Retrieve (Page 705). Archived SIMIT projects have the file extension ".simarc".
- With "Migrate project", you migrate a project from a previous version of SIMIT.

SIMIT projects can be stored in any folder in the file system. To select a folder, click "Browse...".

### 8.13 "IM configuration" dialog box

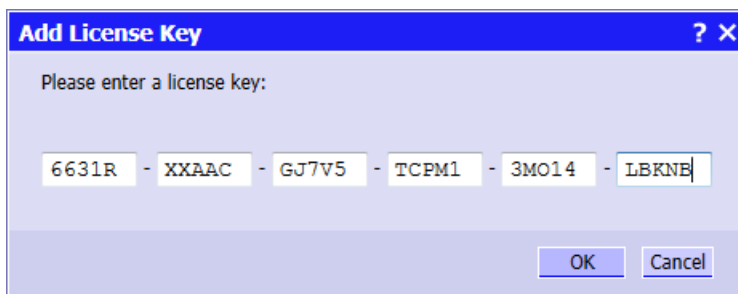


Use this dialog box to configure a PROFIBUS or PROFINET interface.

- IP address: Enter the "IP address" of the interface. You can find additional information about the IP address in the following sections:
  - Configuring the PROFIBUS DP interface module (Page 104)
  - Configuring the PROFINET IO interface module (Page 125)
- Name: Enter a name of your choice for the interface.
- Type: The type is determined by SIMIT when you click the "Update" button.
- Status: The status is determined by SIMIT when you click the "Update" button.

Click "OK" to apply the changes and close the dialog box.

### 8.14 "Add license key" dialog box



Enter a valid license key here to unlock a new SIMIT version, a different product version or another library. The license key can be transferred with copy and paste. Select any text input field for pasting.

Then click "OK" in order to activate the license key and close the dialog box.

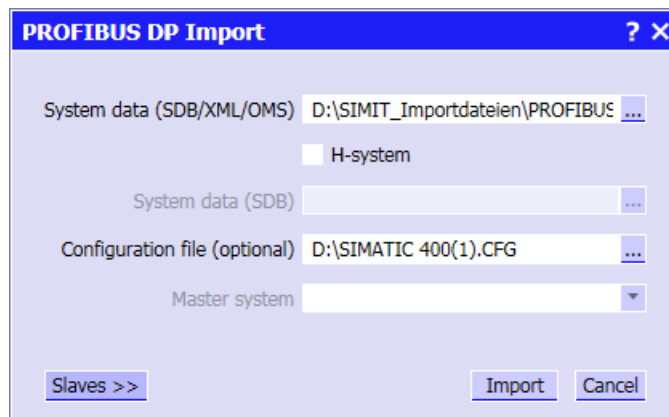
You can find information about license keys in the section: Installation and update (Page 17)

## 8.15 "No license key found" dialog box

This dialog box opens when the license key for SIMIT was not found.

You can re-enter a license key here or cancel the action.

## 8.16 "PROFIBUS DP import" dialog box



In this dialog box, make the following settings for importing a PROFIBUS DP interface:

- **System data (SDB/XML/OMS)**  
Here, you can select the storage location of the system data or system data block.
- **"H-system" check box**  
Select this check box if you are using an H-system
- **System data (SDB)**  
Here, you can select the storage location of a system data block of the H-system. This input box is only enabled if you have selected the "H-system" check box.
- **Configuration file (optional)**  
Here, you can select the storage location of the configuration file. This entry is optional.
- **Master system**  
Select the master system of the PROFIBUS DP connection from the drop-down list here. The drop-down list is only active when multiple master systems are available.

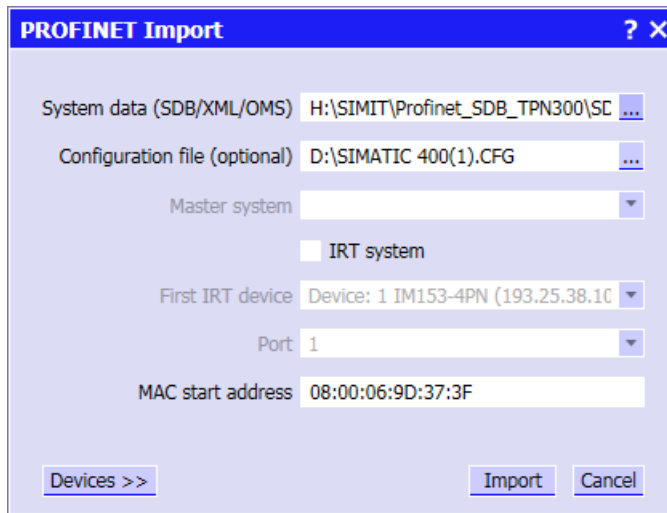
Click "..." to select the respective storage location.

Click "Slaves >>" to display the associated slaves, select them and possibly import them.

Click "Import" to start the import and close the dialog box.

You can find additional information about importing a PROFIBUS DP interface in the section: PROFIBUS DP coupling (Page 78)

## 8.17 "PROFINET import" dialog box



In this dialog box, make the following settings for importing a PROFINET IO interface:

- **System data (SDB/XML)**  
Here, you can select the storage location of the system data block.
- **Symbol table (optional)**  
Here, you can select the storage location of the symbol table. This entry is optional.
- **Configuration file (optional)**  
Here, you can select the storage location of the configuration file. This entry is optional.
- **"IRT-System" check box**  
Select this check box if the IRT protocol is to be supported.
- **First IRT device**  
Select the first IRT device of the PROFINET controller from the drop-down list. This input box is only enabled if you have selected the "IRT system" check box.
- **Port**  
Select the port to the first IRT device from the drop-down list. This input box is only enabled if you have selected the "IRT system" check box.
- **MAC start address**  
The MAC start address is assigned automatically by SIMIT during the import. You need to change this address if you have multiple PROFINET interfaces from the same subnet. You can find additional information on this in the section: Assigning a MAC start address (Page 116).

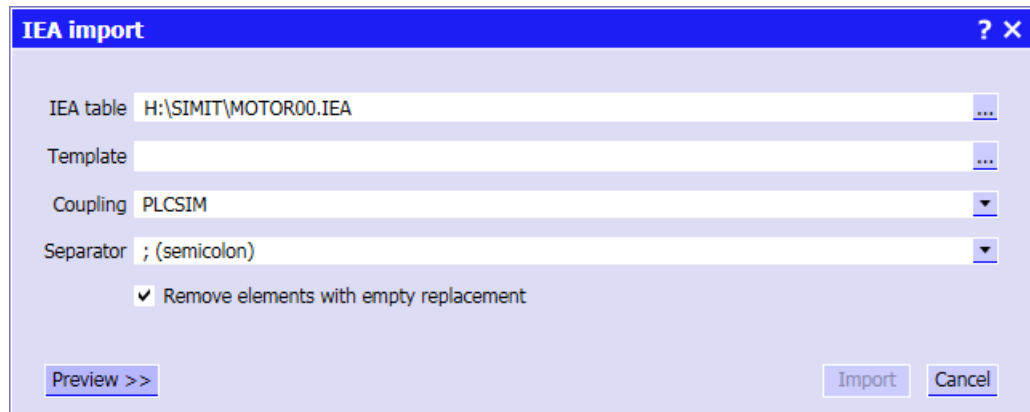
Click "..." to select the respective storage location.

Click "Devices >>" to display the associated slaves, select them and possibly import them.

Click "Import" to start the import and close the dialog box.

You can find additional information about importing a PROFINET IO interface in the section: PROFINET IO coupling (Page 106).

## 8.18 "IEA import" dialog box



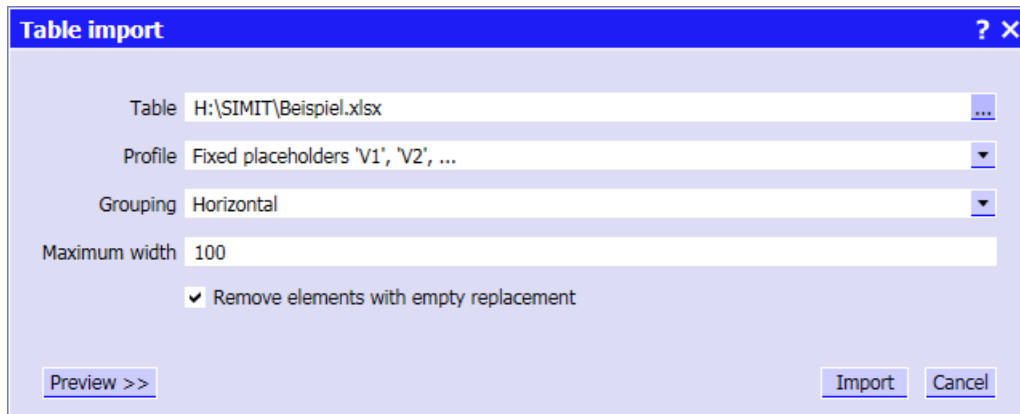
In this dialog box, make the following settings for importing an IEA file.

- **IEA table**  
Select the storage location of the IEA file here by clicking the "..." button.
- **Template**  
Select the storage location of the template here by clicking the "..." button.
- **Coupling**  
Select the coupling used from the drop-down list.
- **Separator**  
Select the desired separator from the drop-down list.
- **"Remove elements with empty replacement"** check box  
Select this check box if you want to remove empty replacements from the file.

Click "Preview>>" to display a preview of the import. On the left in the preview, you can see the newly created folders and charts from the import. Individual folders and charts can be deselected here with a check box. If you select a chart on the left, you will see the replacements to be performed on the right. The template used in each case is entered in the second header on the right and an indication is given if a document has been found.

Click "Import" to start the import and close the dialog box.

## 8.19 "Table import" dialog box



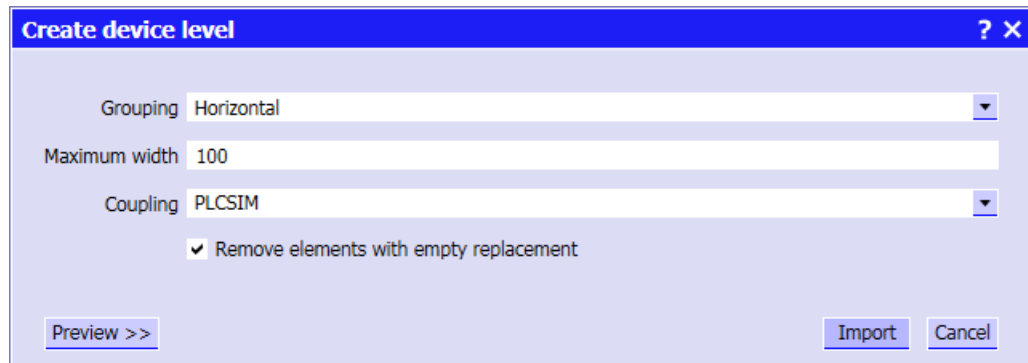
In this dialog box, make the following settings for importing a table.

- **Table**  
Select the storage location of the table here by clicking the "..." button.
- **Profile**  
Select the desired profile from the drop-down list.
- **Grouping**  
Select the desired grouping (horizontal or vertical) from the drop-down list.
- **Maximum width**  
Enter the desired width in pixels here.
- **"Remove elements with empty replacement"** check box  
Activate this check box if you want to remove empty replacements from the file.

Click "Preview>>" to display a preview of the import. On the left in the preview, you can see the newly created folders and charts from the import. Individual folders and charts can be deselected here with a check box. If you select a chart on the left, you will see the replacements to be performed on the right. The template used in each case is entered in the second header on the right and an indication is given if a document has been found.

Click "Import" to start the import and close the dialog box.

## 8.20 "Create device level" dialog box



In this dialog box, make the following settings for the "Create device level" function:

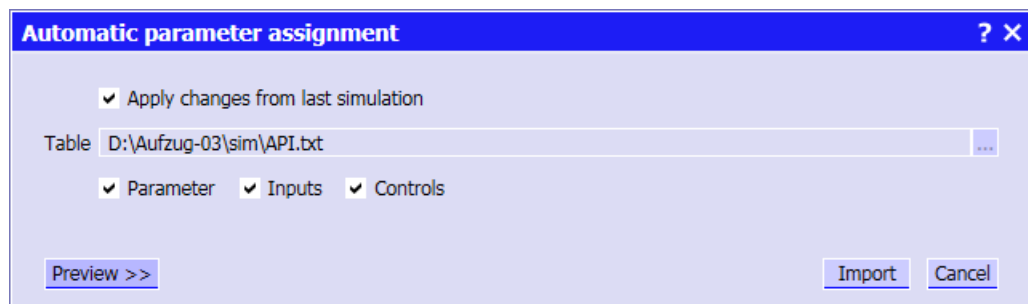
- **Grouping**  
Select the desired grouping (horizontal or vertical) from the drop-down list.
- **Maximum width**  
Enter the desired width in pixels here.
- **Coupling**  
Select the coupling used from the drop-down list.
- **"Remove elements with empty replacement"** check box  
Activate this check box if you want to remove empty replacements from the file.

Click "Preview>>" to display a preview with the current settings. In the preview, you can select or deselect individual objects by clicking the appropriate check box.

Click "Import" to start the creation of the device level and close the dialog box.

You can find additional information on this in the section: Using templates (Page 574).

## 8.21 "Automatic parameter assignment" dialog box



In this dialog box, make the following settings for the automatic parameter assignment:

- **"Apply changes from last simulation"** check box  
Select the check box to automatically apply the changes from the last active simulation to this project. The check box is only enabled if at least one simulation has already been run for the project.
- **Table**  
Here you can select the storage location of the table. The table must have a specific format.
- **"Parameters"** check box  
Select the check box to apply the parameters from the table.
- **"Inputs"** check box  
Select the check box to apply the input defaults from the table.
- **"Controls"** check box  
Select the check box to apply the control defaults from the table.

Click "Preview>>" to display the preview with automatic parameter assignment.

Click "Import" to perform the automatic parameter assignment and close the dialog box.

You can find additional information in the section Automatic parameter assignment (Page 243).

## 8.22 "Characteristic" editor

The characteristic editor is used to create a characteristic curve by specifying interpolation points and interpolation between these points.

You can find additional information in the section: Characteristic (Page 320)



## 8.23 "Info" dialog box



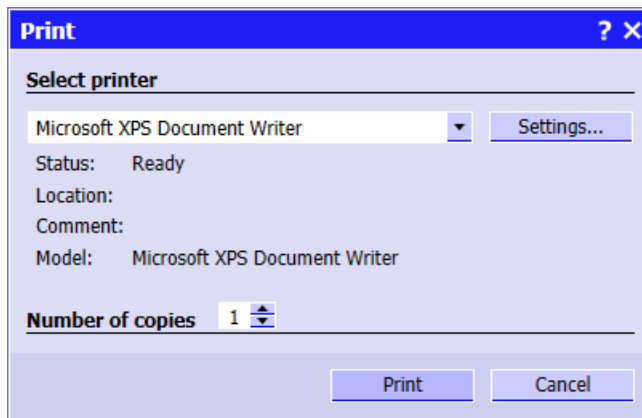
This menu command displays the following information about your SIMIT installation:

- Product variant of SIMIT (STANDARD, PROFESSIONAL or ULTIMATE)
- Version
- License number
- Version of the SU software (if installed)
- Version of SIMIT Virtual Controller (VC) (if installed)
- List of licensed libraries

## 8.24 "Print" dialog box

Charts and templates can be printed. The following properties are provided for this:

- If a folder is highlighted in the project tree, all charts in this folder and in all subfolders are generated as a single print job.
- If a chart does not fit on the selected paper format, it is automatically scaled.
- The size of the chart is indicated by a black frame in the printout.
- The following information is printed as well:
  - For charts, the chart name and the corresponding folder hierarchy
  - For templates, the absolute file name with path

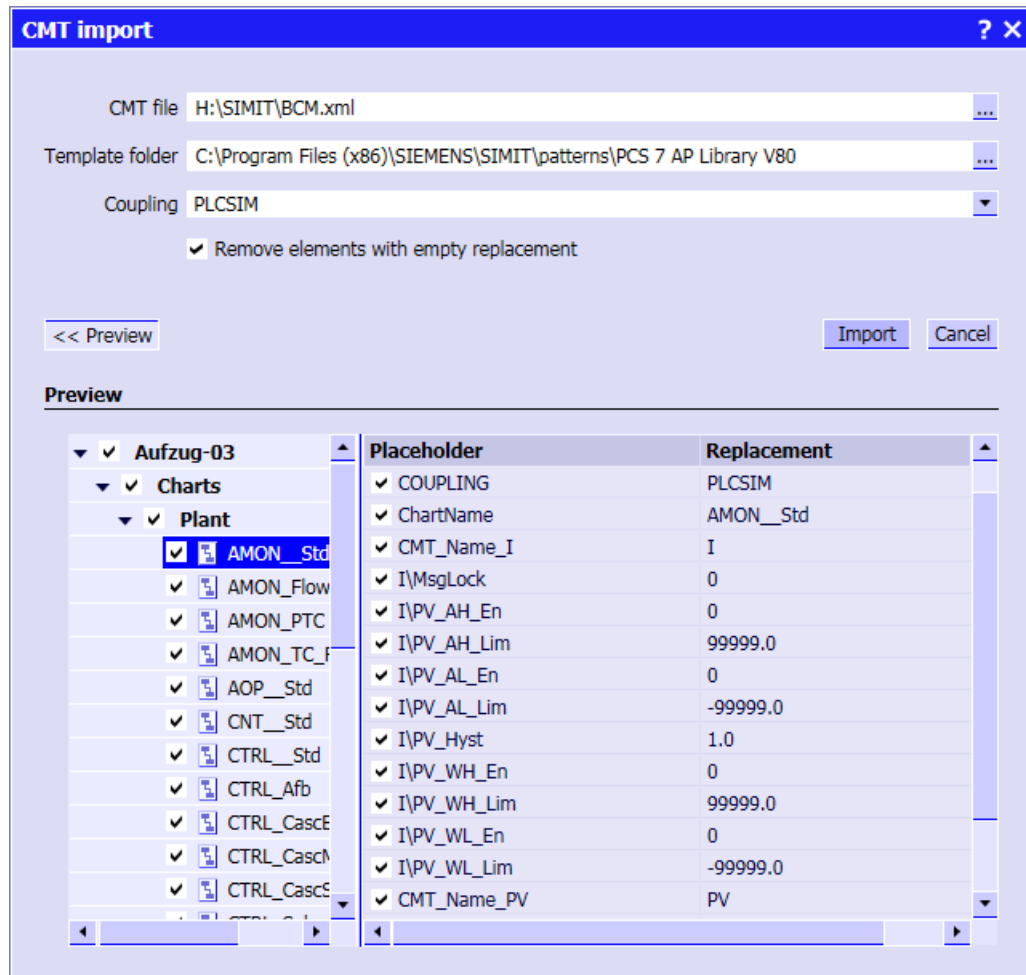


Select a printer from the drop-down list and set the number of copies.

Click the "Settings ..." button to change the settings of the selected printer.

Click "Print" to start printing and close the dialog box.

## 8.25 "CMT import" dialog box



In this dialog box you make the following settings for importing an CMT file:

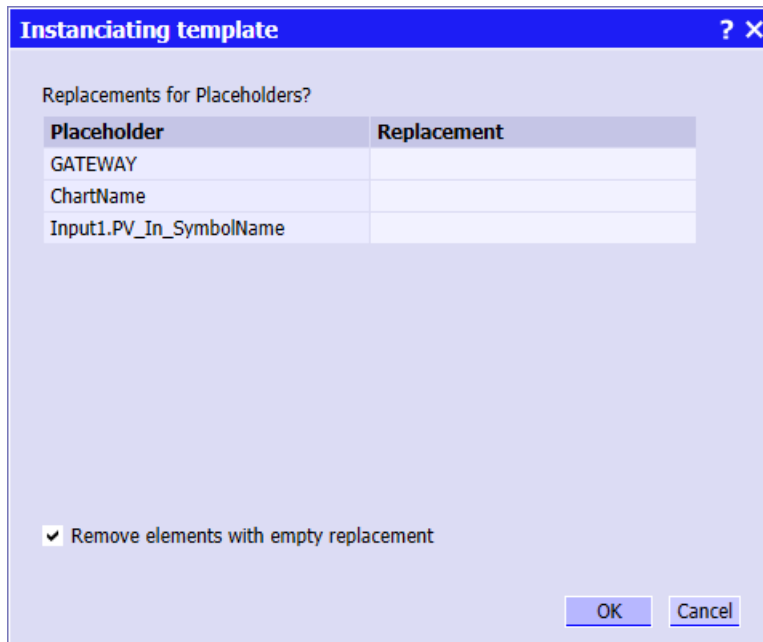
- CMT file**  
 The CMT file is an XML file that was exported from PCS 7. Here you select the storage location of the CMT file by clicking the "..." button.
- Template folder**  
 The template folder specifies the folder in which a search is performed for the SIMIT templates referenced in the CMT file. The name of the CMT type specified in the CMT file must exactly match the name of the template used in SIMIT. Select the storage location of the template folder here by clicking the "..." button.
- Coupling**  
 An available SIMIT coupling must be selected as the coupling to assign I/O signals to the correct communication partner (coupling). Select a coupling from the drop-down list.
- Remove elements with empty replacement** check box  
 Select this check box if elements (e.g. components, macros, controls) should be deleted if they have no replacement for a placeholder in the XML file.

Click "Preview>>" to display a preview of the import. On the left in the preview, you can see the newly created folders and charts from the import. Individual folders and charts can be deselected here with a check box. If you select a chart on the left, you will see the replacements to be performed on the right. The template used in each case is entered in the second header on the right and an indication is given if a document has been found.

Click "Import" to start the import and close the dialog box.

You can find additional information on this in the section: The CMT import (Page 232).

## 8.26 "Instantiate template" dialog box

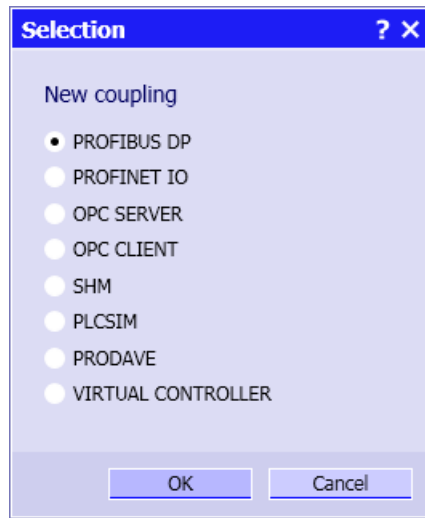


In this dialog box, the parameters of the selected template are listed in the "Placeholder" column. Arbitrary values can be entered as replacements in the "Replacement" column.

Select the "Remove elements with empty replacement" check box to remove parameter in the template for which you have not entered a name in the "Replacement" column.

Click "OK" to apply the changes and close the dialog box.

## 8.27 "Selection of new coupling" dialog box



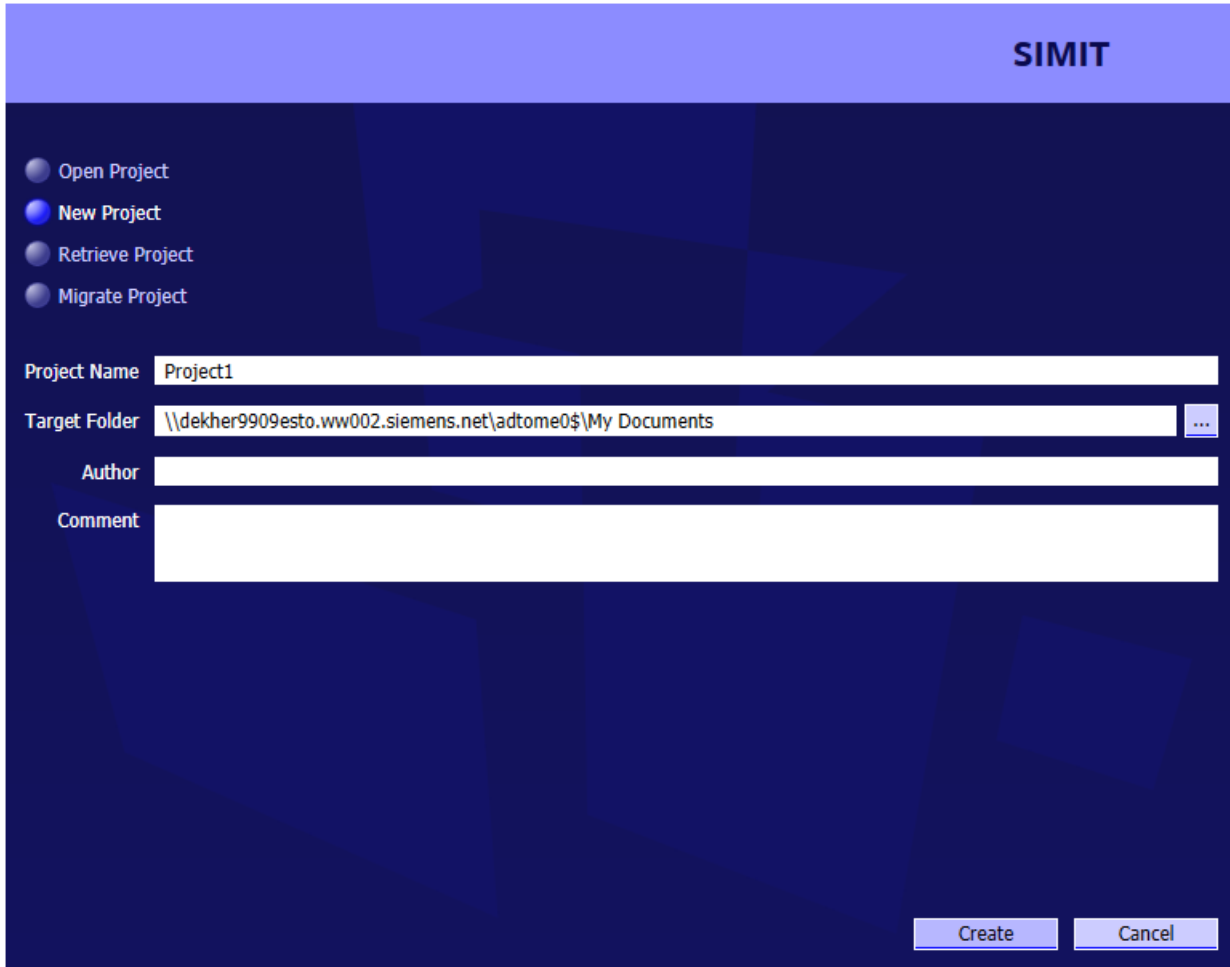
Select a new coupling in this dialog box. The following couplings are available for selection:

- **PROFIBUS DP**  
You can find additional information on the coupling in the section: PROFIBUS DP coupling (Page 78).
- **PROFINET IO**  
You can find additional information on the coupling in the section: PROFINET IO coupling (Page 106)
- **OPC server**  
The "OPC Server" coupling is only available in the product versions SIMIT PROFESSIONAL and SIMIT ULTIMATE. You can find additional information on the coupling in the section: The OPC Server coupling (Page 139).
- **OPC client**  
The "OPC Client" coupling is only available in the product versions SIMIT PROFESSIONAL and SIMIT ULTIMATE.
- **SHM**  
The "SHM" coupling is only available in the product version SIMIT ULTIMATE. You can find additional information on the coupling in the section: Shared Memory coupling (Page 152).
- **PLCSIM**  
The "PLCSIM" coupling is only available in the product versions SIMIT PROFESSIONAL and SIMIT ULTIMATE. You can find additional information on the coupling in the section: PLCSIM coupling (Page 131).
- **PRODAVE**  
You can find additional information on the coupling in the section: PRODAVE coupling (Page 127).
- **Virtual Controller**  
SIMIT VC software must also be installed for the "Virtual Controller" coupling to work. You can find additional information on the coupling in the section: "Virtual controller" coupling (Page 135).

Click "OK" to confirm the selection and close the dialog box.

## 8.28 Project > New project ...

Use this menu command to open the following dialog box:



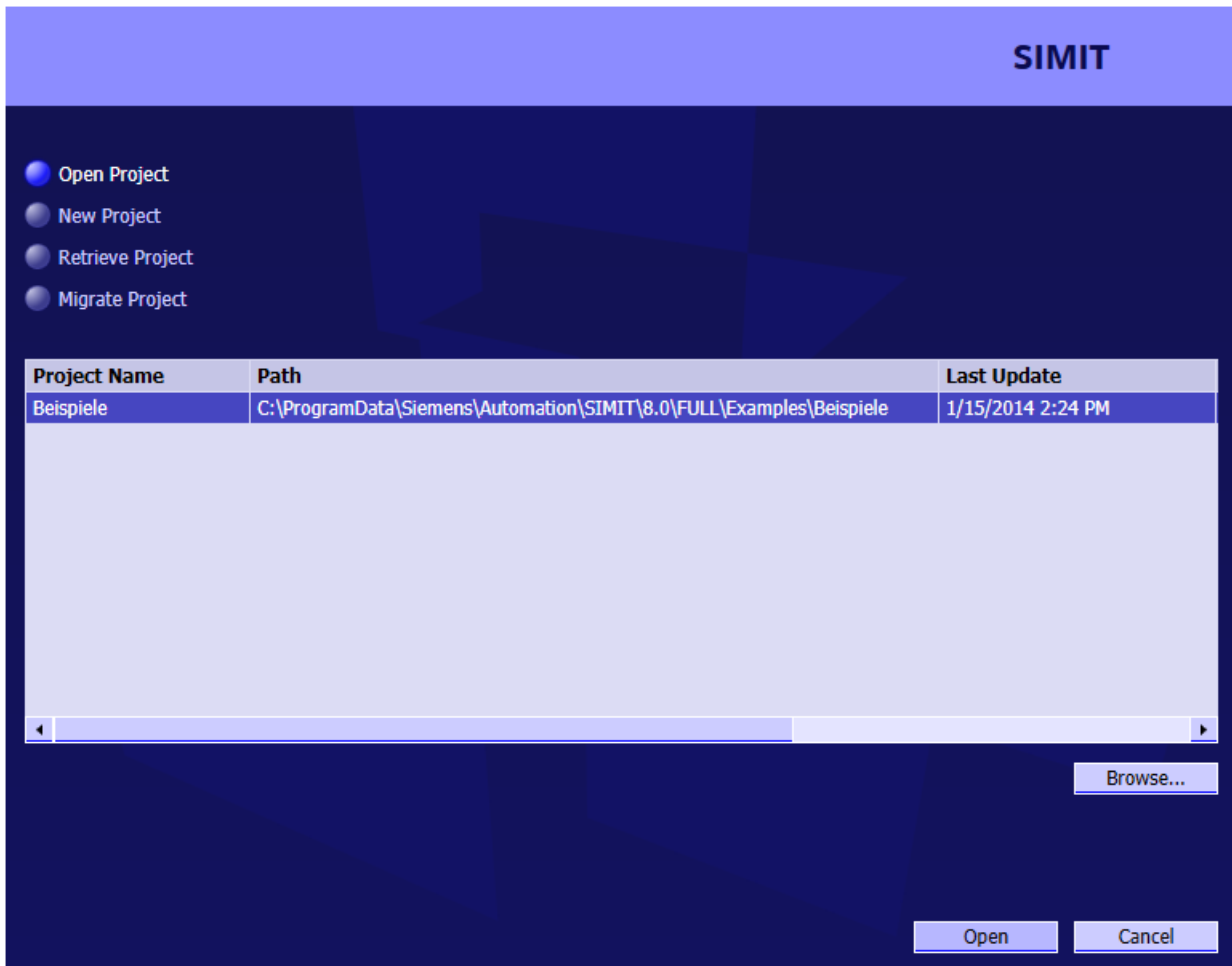
Here, you can create a new project. Enter the required data for this and make the settings in the input fields.

- **Project name**  
Enter a name for your project or accept the default name here.
- **Target folder**  
The storage location of the new project is preset here. Click "... " to select a different storage location.
- **Author**  
You can change the name of the author here. This entry is optional.
- **Comment**  
You can enter a comment of your choice about the project here. This entry is optional.

Click "Create" to create the new project and close the dialog box.

## 8.29 Project > Open...

Use this menu command to open the following dialog box:



The most recent projects and the sample project are listed here. The project name, location and date of the last update of the project are also displayed for each project. This information cannot be changed here.

If the desired project is not displayed here, you can search for it by clicking "Browse ...".

Open the project by selecting it and clicking the "Open" button.

## 8.30 Project > Close

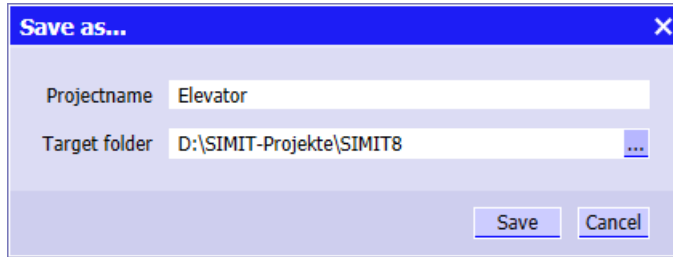
Use this menu command to close the current project.

### 8.31 Project > Save all

Use this menu command to save the current project.

### 8.32 Project > Save as...

Use this menu command to open the following dialog box:



Project name and target folder are preset.

- **Project name**  
Accept the preset name or enter another name for your project.
- **Target folder**  
Accept the preset storage location for your project or select another one by clicking "...".

Click "Save" to save the project and close the dialog box.

### 8.33 Project > Archive

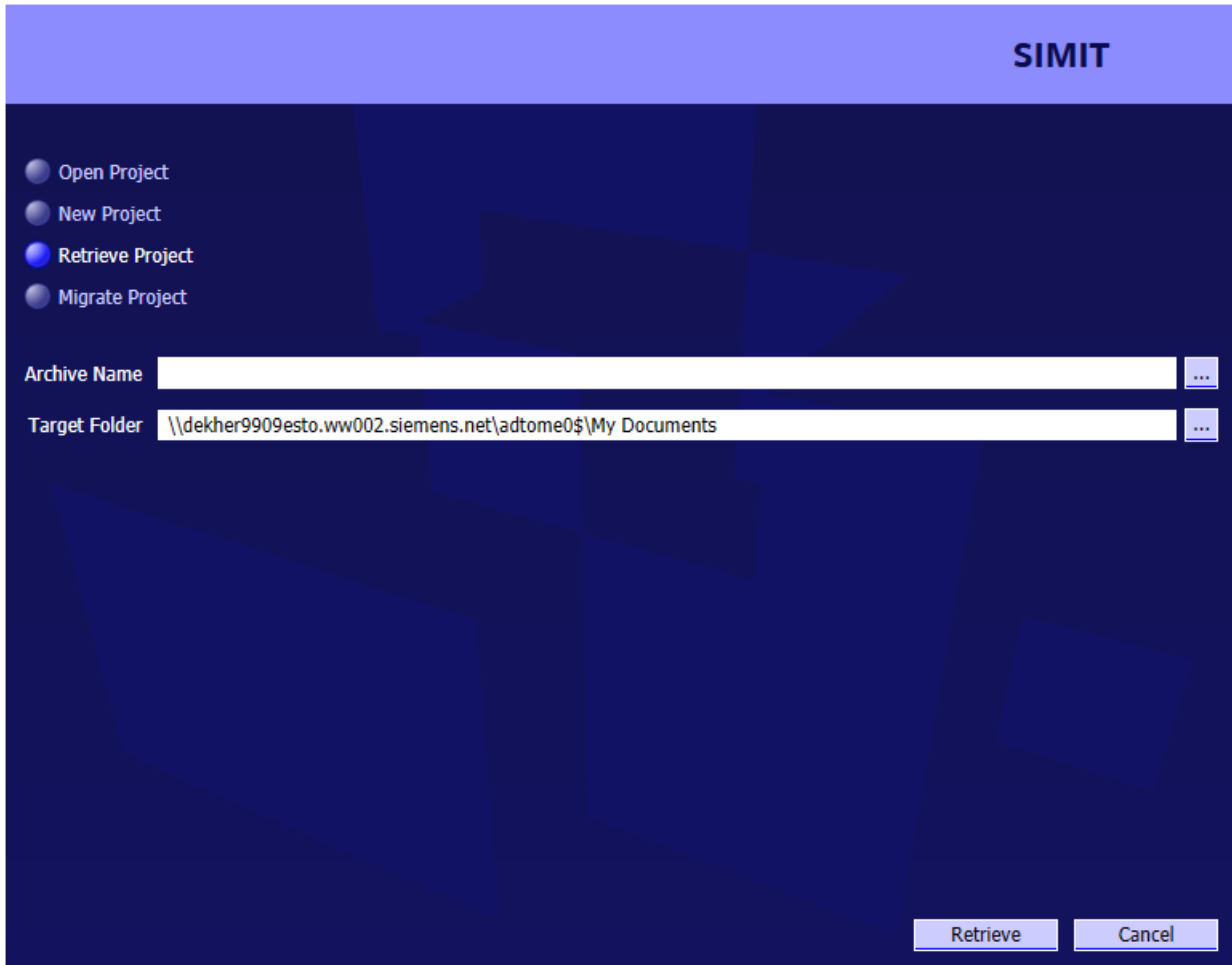
This menu command opens the "Save as" dialog box, where you can enter a name and storage location for the current project. Then click the "Save" button to perform the archiving of the project.

The archived project receives the file extension \*.simarc



## 8.34 Project > Retrieve

Use this menu command to open the following dialog box:



Here, you can retrieve an archived project.

- **Archive name**  
Click "...". to select an archived SIMIT project. Archived SIMIT projects have the file extension ".simarc".
  - **Target folder**  
Click "...". to select the folder in which the project is to be stored after retrieval.
- Click "Retrieve" to retrieve the selected project.

## 8.35 Project > Migrate

This menu command opens a dialog box where you can choose a project that is to be migrated.

### 8.36 Project > Exit

Use this menu command to close the current project.

If changes have not been saved in the project, a dialog box opens in which you are asked whether you want to save before exiting.

### 8.37 Edit > Cut

Use this menu command to remove the selected object.

### 8.38 Edit > Copy

Use this menu command to copy the selected object to the clipboard.

### 8.39 Edit > Paste

Use this menu command to paste the copied object from the clipboard to the current editor.

### 8.40 Simulation > Initialize

Use this menu command to start the consistency check. The result of the consistency check is displayed.

---

**Note**

Initialization does not start simulation.

---

### 8.41 Simulation > Start

Use this menu command to start the simulation.

You can find additional information on this in the section: Running simulations (Page 28).

Before the simulation starts, you are asked whether you want to save the changes to the project and the consistency check is performed.

### 8.42 Simulation > Pause

Use this menu command to pause the simulation.

**8.43 Simulation > Single Step**

This menu command executes a single step in the simulation.

You can find additional information on this in the section: Executing a single step (Page 287).

**8.44 Simulation > Exit**

This menu command stops the current simulation and switches to project view.

**8.45 Simulation > Snapshot**

Use this menu command to make a snapshot of the simulation.

You can find additional information on this in the section: Running simulations (Page 28).

**8.46 Window > Tile Horizontally**

Use this menu command to split the selected editor window horizontally.

**8.47 Window > Tile Vertically**

Use this menu command to split the selected editor window vertically.

**8.48 Window > Unsplit**

This menu command cancels horizontal or vertical splits of the window.

**8.49 Window > Close all**

This menu command closes all open windows.

**8.50 Options > Zoom**

Use this menu command to zoom the selected object. Use the slider control to do this.

**8.51 Options > IM configuration**

Use this menu command to open the "IM configuration" dialog box:

You can find information on this in the following sections:

- "IM configuration" dialog box (Page 690)
- Configuring the PROFIBUS DP interface module (Page 104)
- Configuring the PROFINET IO interface module (Page 125)

## 8.52 Options > Assign coupling signals

With this menu command, the entries in input or output connectors are updated on the charts of the entire project, when the signal registered in the connector is clearly associated with a coupling.

Examples of applications:

- A coupling was subsequently renamed
- Coupling signals were subsequently assigned symbolic names

You can find additional information on this in the section: Indirect addressing (Page 224).

## 8.53 Help > About

This menu command displays the information about the SIMIT installation.

You can find additional information on this in the section: "Info" dialog box (Page 697).

## 8.54 Help > Add License Key

Use this menu command to open the "Add license key" dialog box

You can find additional information in the section: "Add license key" dialog box (Page 690).