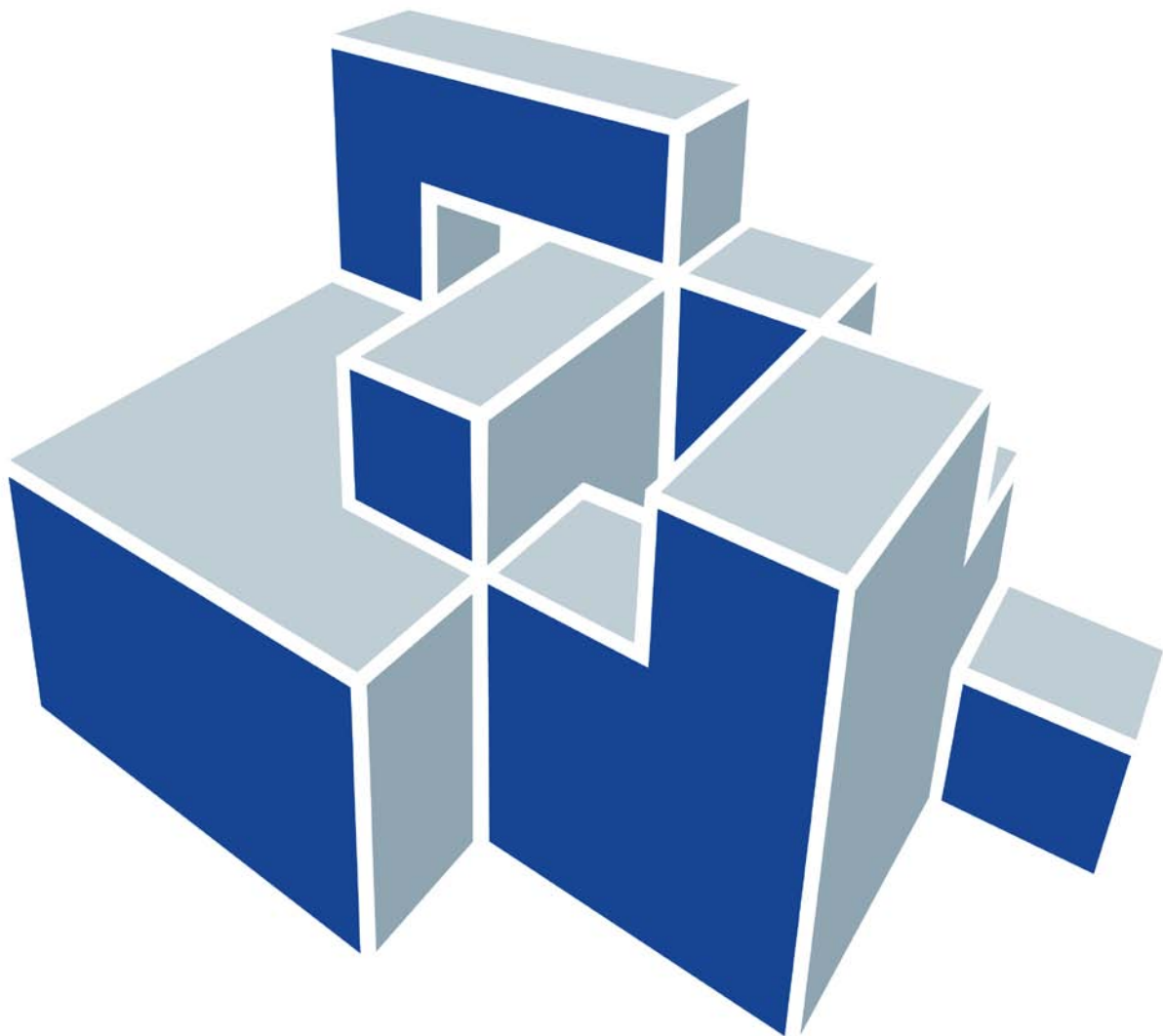


# SIEMENS

## SIMIT 7

*CONTEC Library*

Reference Manual
------------------



## **Edition**

January 2013

Siemens offers simulation software to plan, simulate and optimize plants and machines. The simulation- and optimization- results are only non-binding suggestions for the user. The quality of the simulation and optimizing results depend on the correctness and the completeness of the input data. Therefore, the input data and the results have to be validated by the user.

## **Trademarks**

SIMIT® is a registered trademark of Siemens AG in Germany and in other countries.

Other names used in this document can be trademarks, the use of which by third-parties for their own purposes could violate the rights of the owners.

## **Copyright © Siemens AG 2013 All rights reserved**

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration or a utility model or design, are reserved.

Siemens AG  
Industry Sector  
Industry Automation Division  
Process Automation

SIMIT-HB-V7CONTEC-2013-01-en

## **Exclusion of liability**

We have checked that the contents of this document correspond to the hardware and software described. However, deviations cannot be entirely excluded, and we do not guarantee complete conformance. The information contained in this document is, however, reviewed regularly and any necessary changes will be included in the next edition. We welcome suggestions for improvement.

© Siemens AG 2013

Subject to change without prior notice.

# Contents

<b>1</b>	<b>PREFACE</b>	<b>1</b>
1.1	Target group	1
1.2	Contents	1
1.3	Symbols	1
<b>2</b>	<b>INTRODUCTION</b>	<b>2</b>
<b>3</b>	<b>CONVEYOR TECHNOLOGY SIMULATION</b>	<b>4</b>
3.1	Principles of conveyor technology simulation	5
3.2	Modelling the objects	6
3.3	Modelling the network	8
3.3.1	Modelling segments	8
3.3.2	Modelling branches	11
3.3.3	Modelling boundaries	11
3.4	Special features of conveyor technology simulation	12
3.4.1	Placing and removing objects	12
3.4.2	Simulating the holdup behaviour	12
3.4.2.1	Holdups on straight conveyor sections	12
3.4.2.2	Holdups in curves	13
3.4.3	Time slice allocation for components	13
3.5	Scalability	13
3.5.1	Scalability of conveyor sections	14
3.5.2	Scalability of objects	15
3.6	Generating the simulation of drives and sensors	16
3.6.1	Manual connection of components	17
3.6.2	Using templates	18
<b>4</b>	<b>CONTEC COMPONENT LIBRARY</b>	<b>22</b>
4.1	The Topology connector	22
4.2	Component types for conveyor systems with vehicles	24
4.2.1	<i>Rail-S4</i> – Straight rail with four sensors	24
4.2.2	<i>CurvedRail45-S4</i> – 45° curved rail with four sensors	26
4.2.3	<i>CurvedRail90-S4</i> – 90° curved rail with four sensors	28
4.2.4	<i>RailSwitch-F</i> – Switchable switch with 45° spur	30
4.2.5	<i>RailSwitch-M</i> – Movable switch with 45° spur	32
4.2.6	<i>InOut</i> – Inward and outward section for vehicles	33
4.2.7	<i>RailLifter</i> – Lifter	35

4.2.7.1	<i>RailLifterBase</i> – Lifter (base component)	35
4.2.7.2	<i>RailLifterExtension</i> – Lifter (extension component)	38
<b>4.3</b>	<b>Component types for non-vehicular conveyor systems</b>	<b>39</b>
4.3.1	<i>Conveyor-S4</i> – Straight conveyor with four sensors	41
4.3.2	<i>Conveyor-S4-Stopper</i> – Straight conveyor with four sensors and stopper	44
4.3.3	<i>ConveyorCurve45-R60</i> – 45° curved conveyor (radius 60 pixels)	47
4.3.4	<i>ConveyorCurve45-R100</i> – 45° curved conveyor (radius 100 pixels)	48
4.3.5	<i>ConveyorCurve45-R200</i> – 45° curved conveyor (radius 200 pixels)	50
4.3.6	<i>ConveyorCurve90-R60</i> – 90° curved conveyor (radius 60 pixels)	51
4.3.7	<i>ConveyorCurve90-R100</i> – 90° curved conveyor (radius 100 pixels)	52
4.3.8	<i>ConveyorCurve90-R200</i> – 90° curved conveyor (radius 200 pixels)	54
4.3.9	<i>90DegreeTransfer</i>	55
4.3.10	<i>CrossOver</i>	58
4.3.11	<i>Lifter</i>	59
4.3.11.1	<i>LifterBase</i> – Lifter (base component)	60
4.3.11.2	<i>LifterExtension</i> – Lifter (extension component)	63
4.3.12	<i>Turntable-R60</i>	64
4.3.13	<i>Swiveltable-R120</i> – Swivel table	67
4.3.14	<i>TransferCarriage</i>	69
4.3.15	<i>SpurConveyor-1</i> – Diagonal feed with one spur	72
4.3.16	<i>SpurConveyor-2</i> – Diagonal feed with two spurs	73
<b>4.4</b>	<b>Component types for simulating objects</b>	<b>75</b>
4.4.1	<i>Box</i> – Simple object	75
4.4.2	<i>CBox</i> – Coloured object	75
4.4.3	<i>CBoxDS256</i> – Coloured object with data storage	76
4.4.4	<i>Vehicle</i>	77
4.4.5	<i>CVehicle</i>	78
4.4.6	<i>VehicleDS256</i> – Vehicle with data storage	80
<b>4.5</b>	<b>Component types for simulating identification systems</b>	<b>81</b>
4.5.1	Operating principle	83
4.5.2	<i>ASM452</i> – Interface module for identification systems	86
4.5.3	<i>ASM454</i> – Interface module for identification systems	89
4.5.4	<i>ASM456</i> – Interface module for identification systems	91
4.5.5	<i>ASM473</i> – Interface module for identification systems	94
4.5.6	<i>ASM 475</i> – Interface module for identification systems	97
4.5.7	<i>ASM754</i> – Interface module for identification systems	100
4.5.8	<i>ASM850</i> – Interface module for identification systems	102
4.5.9	<i>ASM854</i> – Interface module for identification systems	104
4.5.10	<i>RF 170C</i> – Interface module for identification systems	106
4.5.11	<i>RF180C</i> – Interface module for identification systems	109
<b>5</b>	<b>CREATING CUSTOM COMPONENT TYPES</b>	<b>113</b>
<b>5.1</b>	<b>Topological properties</b>	<b>113</b>

5.1.1	Topological connectors (connection type <i>MT1</i> )	114
5.1.2	Topology of a segment	115
5.1.3	Topology of a branch	115
5.1.4	Topology of a boundary	116
5.1.5	Determining the next section	116
<b>5.2</b>	<b>Connection to the solver</b>	<b>116</b>
5.2.1	Connection type <i>MT2</i> for segments	117
5.2.2	Connection type <i>MT3</i> for stoppers	118
5.2.3	Connection type <i>MT4</i> for sensors	119
5.2.4	Connector type <i>MT5</i> for placing objects	120
5.2.5	Connection type <i>MT6</i> for positions	121
5.2.6	Connector type <i>MT7</i> for programming objects	122
5.2.7	Connector type <i>MT8</i> for transferring objects at boundary points	123
5.2.8	Errors in the component code	125
<b>5.3</b>	<b>System functions</b>	<b>126</b>
5.3.1	_MT.GetObjectByName	127
5.3.2	_MT.GetObjectByType	127
5.3.3	_MT.Restore	127
5.3.4	_MT.GetWidth	127
5.3.5	_MT.GetHeight	128
5.3.6	_MT.GetDepth	128
5.3.7	_MT.GetAngle	128
5.3.8	_MT.SetPosition	128
5.3.9	_MT.SetAngle	129
5.3.10	_MT.AddAngle	129
5.3.11	_MT.SetHoldup	130
5.3.12	_MT.GetBinaryValue	130
5.3.13	_MT.GetAnalogValue	130
5.3.14	_MT.GetIntegerValue	131
5.3.15	_MT.GetByteValue	131
5.3.16	_MT.GetByteArray	131
5.3.17	_MT.SetBinaryValue	132
5.3.18	_MT.SetAnalogValue	132
5.3.19	_MT.SetIntegerValue	133
5.3.20	_MT.SetByteValue	133
5.3.21	_MT.SetByteArray	133
<b>5.4</b>	<b>System variables</b>	<b>134</b>

## Table of figures

Figure 3-1:	Example of a conveyor technology model	5
Figure 3-2:	Example of a network	5
Figure 3-3:	Creating a new material list	6
Figure 3-4:	Editing the material list	7
Figure 3-5:	Dialog for creating multiple instances	7
Figure 3-6:	Material list once the simulation has started	8
Figure 3-7:	Angle definition	9
Figure 3-8:	Transport of the object over a segment	9
Figure 3-9:	Active/inactive segments	10
Figure 3-10:	Object held up at a stopper	10
Figure 3-11:	Detection of an object at sensor positions	10
Figure 3-12:	Branch at a connector and within a subnetwork	11
Figure 3-13:	Boundary point created by an "open" connector of a component	11
Figure 3-14:	Holdup in the correct position	12
Figure 3-15:	Holdup in the approximate position	12
Figure 3-16:	Holdup in a curve	13
Figure 3-17:	Different sub-models in a conveyor section	13
Figure 3-18:	Diagram scale	14
Figure 3-19:	Default scale	14
Figure 3-20:	Absolute size of the object	15
Figure 3-21:	Copy cell	16
Figure 3-22:	Paste to cell	16
Figure 3-23:	Rail with the name <i>Rail</i>	17
Figure 3-24:	Default value of the implicit connection	17
Figure 3-25:	Trigger component for the rail	17
Figure 3-26:	Editable implicit connection	18
Figure 3-27:	Input with constant value	18
Figure 3-28:	Input with implicit connection	18
Figure 3-29:	Basic templates for the <i>CONTEC</i> library	19
Figure 3-30:	Basic template <i>Rail-S4</i>	19
Figure 3-31:	Calling the CDL function in the Project Manager	20
Figure 3-32:	Preview of the CDL function in the Project Manager	21
Figure 4-1:	Connector component types in the basic library	22
Figure 4-2:	<i>Topology</i> conveyor technology connector	22
Figure 4-3:	Conveyor technology connector on a branch	23
Figure 4-4:	Invalid use of the connector as a branch	23
Figure 4-5:	Error message referring to invalid use of the connector as a branch	23
Figure 4-6:	<i>RAILS</i> library directory	24
Figure 4-7:	Sensor in the component symbol	25
Figure 4-8:	Parameters for the <i>Rail-S4</i> component type	25
Figure 4-9:	Additional parameters for the <i>Rail-S4</i> component type	26
Figure 4-10:	Operating window for the <i>Rail-S4</i> component type	26

Figure 4-11:	Sensor in the component symbol	27
Figure 4-12:	Parameters for the <i>CurvedRail45-S4</i> component type	27
Figure 4-13:	Additional parameters for the <i>CurvedRail45-S4</i> component type	28
Figure 4-14:	Operating window for the <i>CurvedRail45-S4</i> component type	28
Figure 4-15:	Sensor in the component symbol	29
Figure 4-16:	Parameters for the <i>CurvedRail90-S4</i> component type	29
Figure 4-17:	Additional parameters for the <i>CurvedRail90-S4</i> component type	30
Figure 4-18:	Operating window for the <i>CurvedRail90-S4</i> component type	30
Figure 4-19:	Parameters for the <i>RailSwitch-F</i> component type	31
Figure 4-20:	Additional parameters for the <i>RailSwitch-F</i> component type	31
Figure 4-21:	Operating window for the <i>RailSwitch-F</i> component type	31
Figure 4-22:	Parameters for the <i>RailSwitch-M</i> component type	32
Figure 4-23:	Additional parameters for the <i>RailSwitch-M</i> component type	33
Figure 4-24:	Operating window for the <i>RailSwitch-M</i> component type	33
Figure 4-25:	Parameters for the <i>InOut</i> component type	34
Figure 4-26:	Additional parameters for the <i>InOut</i> component type	34
Figure 4-27:	Operating window for the <i>InOut</i> component type	34
Figure 4-28:	Parameters for the <i>RailLifterBase</i> component type	37
Figure 4-29:	Additional parameters for the <i>RailLifterBase</i> component type	37
Figure 4-30:	Operating window for the <i>RailLifterBase</i> component type	38
Figure 4-31:	Parameters for the <i>RailLifterExtension</i> component type	39
Figure 4-32:	Additional parameters for the <i>RailLifterExtension</i> component type	39
Figure 4-33:	Operating window for the <i>RailLifterExtension</i> component type	39
Figure 4-34:	CONVEYOR library directory	40
Figure 4-35:	Indication of the current transport direction in the symbol	41
Figure 4-36:	Parameters for the <i>Conveyor-S4</i> component type	42
Figure 4-37:	Additional parameters for the <i>Conveyor-S4</i> component type	43
Figure 4-38:	Operating window for the <i>Conveyor-S4</i> component type	43
Figure 4-39:	Indication of the current transport direction in the symbol	44
Figure 4-40:	Representation of the stopper in the symbol	45
Figure 4-41:	Parameters for the <i>Conveyor-S4-Stopper</i> component type	45
Figure 4-42:	Additional parameters for the <i>Conveyor-S4-Stopper</i> component type	46
Figure 4-43:	Operating window for the <i>Conveyor-S4-Stopper</i> component type	47
Figure 4-44:	Parameters for the <i>ConveyorCurve45-R60</i> component type	48
Figure 4-45:	Additional parameters for the <i>ConveyorCurve45-R60</i> component type	48
Figure 4-46:	Operating window for the <i>ConveyorCurve45-R60</i> component type	48
Figure 4-47:	Parameters for the <i>ConveyorCurve45-R100</i> component type	49
Figure 4-48:	Additional parameters for the <i>ConveyorCurve45-R100</i> component type	49
Figure 4-49:	Operating window for the <i>ConveyorCurve45-R100</i> component type	49
Figure 4-50:	Parameters for the <i>ConveyorCurve45-R200</i> component type	50
Figure 4-51:	Additional parameters for the <i>ConveyorCurve45-R200</i> component type	50
Figure 4-52:	Operating window for the <i>ConveyorCurve45-R200</i> component type	51

Figure 4-53:	Parameters for the <i>ConveyorCurve90-R60</i> component type	51
Figure 4-54:	Additional parameters for the <i>ConveyorCurve90-R60</i> component type	52
Figure 4-55:	Operating window for the <i>ConveyorCurve90-R60</i> component type	52
Figure 4-56:	Parameters for the <i>ConveyorCurve90-R100</i> component type	53
Figure 4-57:	Additional parameters for the <i>ConveyorCurve90-R100</i> component type	53
Figure 4-58:	Operating window for the <i>ConveyorCurve90-R100</i> component type	53
Figure 4-59:	Parameters for the <i>ConveyorCurve90-R200</i> component type	54
Figure 4-60:	Additional parameters for the <i>ConveyorCurve90-R200</i> component type	55
Figure 4-61:	Operating window for the <i>ConveyorCurve90-R200</i> component type	55
Figure 4-62:	Parameters for the <i>90DegreeTransfer</i> component type	57
Figure 4-63:	Additional parameters for the <i>90DegreeTransfer</i> component type	57
Figure 4-64:	Operating window for the <i>90DegreeTransfer</i> component type	58
Figure 4-65:	Parameters for the <i>CrossOver</i> component type	59
Figure 4-66:	Additional parameters for the <i>CrossOver</i> component type	59
Figure 4-67:	Operating window for the <i>CrossOver</i> component type	59
Figure 4-68:	Parameters for the <i>LifterBase</i> component type	62
Figure 4-69:	Additional parameters for the <i>LifterBase</i> component type	62
Figure 4-70:	Operating window for the <i>LifterBase</i> component type	63
Figure 4-71:	Parameters for the <i>LifterExtension</i> component type	64
Figure 4-72:	Additional parameters for the <i>LifterExtension</i> component type	64
Figure 4-73:	Operating window for the <i>LifterExtension</i> component type	64
Figure 4-74:	Parameters for the <i>Turntable-R60</i> component type	66
Figure 4-75:	Additional parameters for the <i>Turntable-R60</i> component type	66
Figure 4-76:	Operating window for the <i>Turntable-R60</i> component type	67
Figure 4-77:	Parameters for the <i>Swiveltable-R120</i> component type	68
Figure 4-78:	Additional parameters for the <i>Swiveltable-R120</i> component type	69
Figure 4-79:	Operating window for the <i>Swiveltable-R120</i> component type	69
Figure 4-80:	Parameters for the <i>TransferCarriage</i> component type	71
Figure 4-81:	Additional parameters for the <i>TransferCarriage</i> component type	71
Figure 4-82:	Operating window for the <i>TransferCarriage</i> component type	72
Figure 4-83:	Parameters for the <i>SpurConveyor-1</i> component type	72
Figure 4-84:	Additional parameters for the <i>SpurConveyor-1</i> component type	73
Figure 4-85:	Operating window for the <i>SpurConveyor-1</i> component type	73
Figure 4-86:	Parameters for the <i>SpurConveyor-2</i> component type	74
Figure 4-87:	Additional parameters for the <i>SpurConveyor-2</i> component type	74
Figure 4-88:	Operating window for the <i>SpurConveyor-2</i> component type	74
Figure 4-89:	<i>MATERIAL</i> library directory	75
Figure 4-90:	Parameters for the <i>CBoxDS256</i> component type	77
Figure 4-91:	Parameters for the <i>Vehicle</i> component type	78
Figure 4-92:	Additional parameters for the <i>Vehicle</i> component type	78
Figure 4-93:	Parameters for the <i>CVehicle</i> component type	80
Figure 4-94:	Additional parameters for the <i>CVehicle</i> component type	80



Figure 4-95:	Parameters for the <i>VehicleDS256</i> component type	81
Figure 4-96:	Additional parameters for the <i>VehicleDS256</i> component type	81
Figure 4-97:	<i>RFID</i> library directory	82
Figure 4-98:	Correlation between conveyor technology simulation and simulation of interface modules	84
Figure 4-99:	ASM454 in the HW config view	86
Figure 4-100:	Parameters for the <i>ASM452</i> component type	87
Figure 4-101:	Additional parameters for the <i>ASM452</i> component type	88
Figure 4-102:	Operating window for the <i>ASM452</i> component type	89
Figure 4-103:	Parameters for the <i>ASM454</i> component type	90
Figure 4-104:	Operating window for the <i>ASM454</i> component type	91
Figure 4-105:	Parameters for the <i>ASM456</i> component type	92
Figure 4-106:	Additional parameters for the <i>ASM456</i> component type	93
Figure 4-107:	Operating window for the <i>ASM456</i> component type	94
Figure 4-108:	Parameters for the <i>ASM473</i> component type	95
Figure 4-109:	Status values for the read/write device in the <i>ASM473</i>	96
Figure 4-110:	Operating window for the <i>ASM473</i> component type	97
Figure 4-111:	Parameters for the <i>ASM475</i> component type	98
Figure 4-112:	Additional parameters for the <i>ASM475</i> component type	99
Figure 4-113:	Operating window for the <i>ASM475</i> component type	100
Figure 4-114:	Parameters for the <i>ASM754</i> component type	101
Figure 4-115:	Operating window for the <i>ASM754</i> component type	102
Figure 4-116:	Parameters for the <i>ASM850</i> component type	103
Figure 4-117:	Operating window for the <i>ASM850</i> component type	104
Figure 4-118:	Parameters for the <i>ASM854</i> component type	105
Figure 4-119:	Operating window for the <i>ASM854</i> component type	106
Figure 4-120:	Parameters for the <i>RF170C</i> component type	107
Figure 4-121:	Additional parameters for the <i>RF170C</i> component type	108
Figure 4-122:	Operating window for the <i>RF170C</i> component type	109
Figure 4-123:	Parameters for the <i>RF180C</i> component type	110
Figure 4-124:	Additional parameters for the <i>RF180C</i> component type	111
Figure 4-125:	Operating window for the <i>RF180C</i> component type	112
Figure 5-1:	Topology in the component type navigation menu	114
Figure 5-2:	Connectors of type <i>MT1</i>	114
Figure 5-3:	Connecting connectors of type <i>MT1</i> by superposition	114
Figure 5-4:	Connecting connectors of type <i>MT1</i> with a connecting line	115
Figure 5-5:	Topology of a branch	115
Figure 5-6:	Topology of a boundary	116
Figure 5-7:	Implementation at a branch as an internal node	116
Figure 5-8:	Data exchange between conveyor technology components and solver	117
Figure 5-9:	Visibility in the Properties dialog of a connector of type <i>MT2</i> to <i>MT8</i>	117
Figure 5-10:	Definition of a connector of type <i>MT2</i>	117
Figure 5-11:	Signals of a connector of type <i>MT2</i>	118
Figure 5-12:	Definition of a connector of type <i>MT3</i>	119

Figure 5-13:	Signals of a connector of type <i>MT3</i>	119
Figure 5-14:	Definition of a connector of type <i>MT4</i>	119
Figure 5-15:	Signals of a connector of type <i>MT4</i>	120
Figure 5-16:	Definition of a connector of type <i>MT5</i> as an output	120
Figure 5-17:	Signals of a connector of type <i>MT5</i> as an output	121
Figure 5-18:	Definition of a connector of type <i>MT6</i>	121
Figure 5-19:	Signals of a connector of type <i>MT6</i>	122
Figure 5-20:	Orientation of objects	122
Figure 5-21:	Definition of a connector of type <i>MT7</i>	123
Figure 5-22:	Signals of a connector of type <i>MT7</i>	123
Figure 5-23:	Definition of a connector of type <i>MT8</i> as an input	124
Figure 5-24:	Signals of a connector of type <i>MT8</i> as an input	124
Figure 5-25:	Missing calculation of an interface signal	126

## List of tables

Table 4-1:	Selectable width of a conveyor	40
Table 4-2:	Colour chart for <i>CBox</i>	76
Table 4-3:	Colour chart for <i>CBoxDS256</i>	77
Table 4-4:	Colour chart for <i>CVehicle</i>	79
Table 4-5:	Interface modules for Profibus DP	83
Table 4-6:	Interface modules for Profinet IO	83
Table 4-7:	Supported FB45 commands	85
Table 4-8:	Possible error codes of RFID components	85
Table 4-9:	Supported <i>ASM452</i> commands	87
Table 4-10:	Supported <i>ASM454</i> commands	90
Table 4-11:	Supported <i>ASM456</i> commands	92
Table 4-12:	Supported <i>ASM473</i> commands	95
Table 4-13:	Supported <i>ASM475</i> commands	98
Table 4-14:	Supported <i>ASM754</i> commands	101
Table 4-15:	Supported <i>ASM850</i> commands	103
Table 4-16:	Supported <i>ASM854</i> commands	105
Table 4-17:	Supported <i>RF170C</i> commands	107
Table 4-18:	Supported <i>RF180C</i> commands	110
Table 5-1:	Return values of conveyor technology system functions	126
Table 5-2:	System variables	134

# 1 PREFACE

## 1.1 Target group

This manual is intended for anyone who uses the SIMIT simulation system. It describes the process for simulating conveyor systems using the *CONTEC* library, i.e. the component types provided by the library and the simulation method that is used by these types of component. An understanding of the process and component types is essential for creating simulations. This manual provides the necessary information.

In addition to thorough knowledge of the use of personal computers and the Microsoft Windows user interface, it assumes knowledge of SIMIT. An in-depth knowledge of the function of conveyor systems is also assumed.

## 1.2 Contents

This manual describes the component types contained in the *CONTEC* library and the modelling approach on which the library is based. Section 2 is an introduction explaining the basic features of conveyor technology simulation in SIMIT.

Section 3 describes how conveyor systems can be simulated using this library. The modelling approach is described in detail.

The components contained in the *CONTEC* library are described in section 4, and section 3 is useful for understanding their function.

Section 5 explains how to create your own component types for conveyor technology simulation. The topological aspects and the data exchange between the components and the solver are explained in detail. The information on conveyor technology provided in section 3 is needed in order to understand this section.

## 1.3 Symbols

Particularly important information is highlighted in the text as follows:



### **NOTE**

Notes contain important supplementary information about the documentation contents. They also highlight those properties of the system or operator input to which we want to draw particular attention.

---



### **CAUTION**

This means that the system will not respond as described if the specified precautionary measures are not applied.

---



### **WARNING**

This means that the system may suffer irreparable damage or that data may be lost if the relevant precautionary measures are not applied.

---

## 2 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 in order to simulate the transport of objects in this system. The *CONTEC* library provides a special solver for this purpose for use in SIMIT. Once the simulation has been started, the solver continuously calculates the positions of the objects in the modelled 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 configuration of the handling equipment. Conveyor systems with autonomously operating vehicles, i.e. 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. These simulations with SIMIT are 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 predefined 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 in this manual as conveyor technology simulation.

As is usual in SIMIT, conveyor technology 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 parameterisation 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. Conveyor technology simulation is therefore based on a movement model that simulates the movement of objects along paths at a predefined 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 electric overhead monorail systems, floor 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. in the simulation.

The component type editor (CTE) can be used to create your own library components for conveyor technology simulations in order to customise and extend your conveyor technology library. You can use the special *CONTEC* solver for conveyor technology simulation in the component types by means of special connection types. The CTE is included in SIMIT ULTIMATE.

**NOTE**

When the simulation starts, it will check whether your SIMIT installation has a licence for the *CONTEC* library. If there are conveyor technology components in your simulation project, i.e. components that use the conveyor technology solver, then the simulation can only be run if you have a conveyor technology licence.

---

### 3 CONVEYOR TECHNOLOGY SIMULATION

Conveyor technology 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* conveyor technology 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 (e.g. weight, friction, etc.) are taken into consideration.

The component types contained in the *CONTEC* library can be divided into two categories:

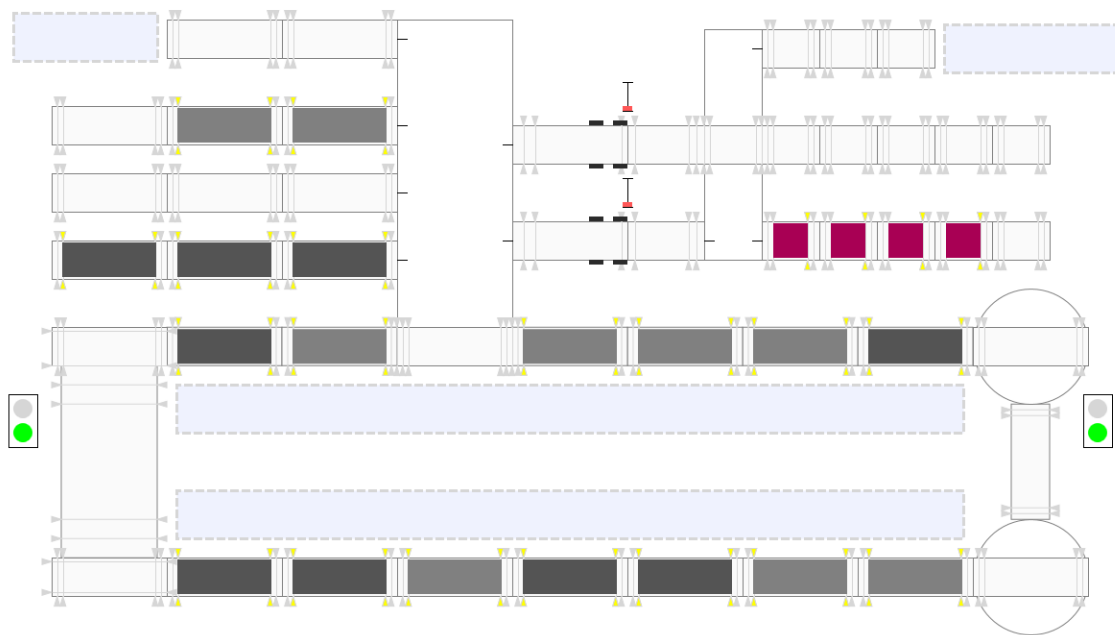
- Object component types and
- Equipment component types

As is usual in SIMIT, equipment component types are added to diagrams as components using their symbol. The symbols for these component types are designed so that when put together they create a scaled layout of the conveyor system, as shown in Figure 3-1 for example. Once created, appropriate parameters are set for the system model and it is 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.

Conveyor technology simulation is based on a special solver that is configured and programmed by means of the individual components of the simulation model. This solver 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 solver 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, i.e. the object.



**Figure 3-1:** Example of a conveyor technology model

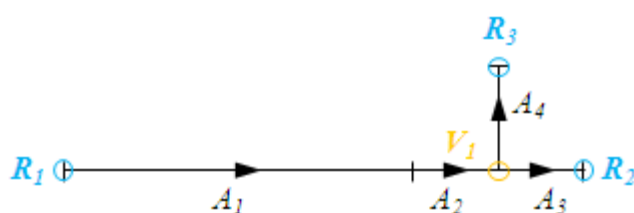
### 3.1 Principles of conveyor technology simulation

The conveyor technology 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 network that represents the possible transportation routes for the objects in a conveyor system.

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 network consists of just 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, Figure 3-2 shows a network consisting of three sections or four segments  $A_1$  to  $A_4$  and one branch  $V_1$ .



**Figure 3-2:** Example of a network



This section is clearly an open network with the three boundary points  $R_1$ ,  $R_2$  and  $R_3$ . If the object crosses the boundary of a network it is lost, i.e. 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 network at the boundaries by means of suitable connections to the solver.

The special conveyor technology solver calculates the position of the objects at equidistant times, as is usual in SIMIT. It supplements the standard SIMIT solver, so conveyor technology component types can be used alongside other component types, for example component types from the basic library. To enable data to be exchanged, conveyor technology components are connected to the conveyor technology solver by means of specific connections. They receive values calculated by the solver and send variables to the solver via these connections.

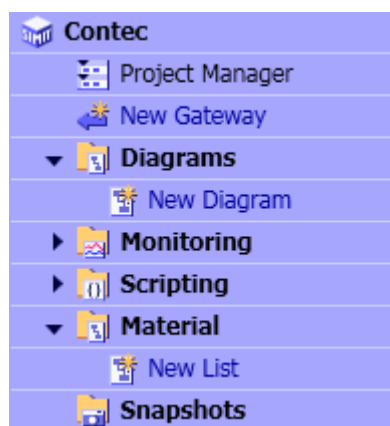
For each segment in the network, positions can be defined at which sensors detect the object as it passes. The conveyor technology solver then ensures that the relevant sensor signals are activated in the simulation according to the positions and dimensions of the objects.

## 3.2 Modelling the objects

Objects are modelled 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 behaviours can also be defined for an object in its component type.

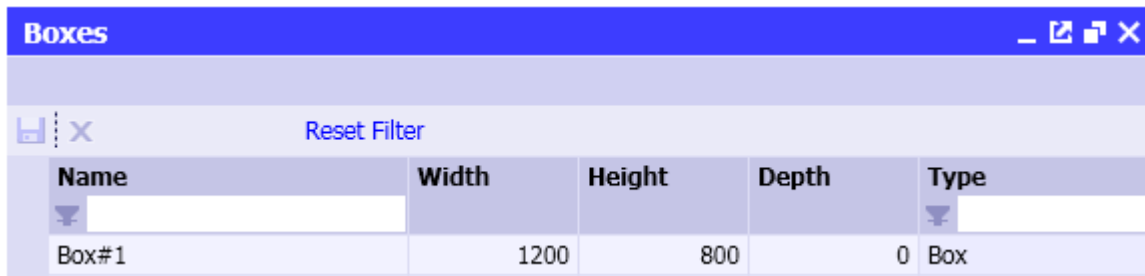
The link symbol is used where possible to represent the object in diagrams: if a link symbol is defined, this is used for the display; otherwise the basic symbol is used, with any connectors on the symbol being hidden.

To simulate objects the available stock of object components has to be defined in the SIMIT project. Use the *New List* command in the *Material* project folder to create a material list, i.e. a list of available object components in the simulation (Figure 3-3).



**Figure 3-3:** Creating a new material list

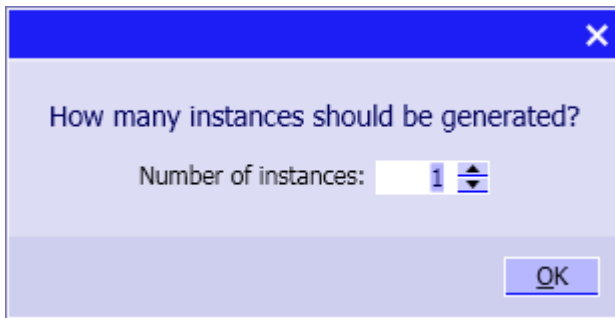
The editor for creating a new material list then opens (Figure 3-4).



Name	Width	Height	Depth	Type
Box#1	1200	800	0	Box

**Figure 3-4:** Editing the material list

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 to 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 Figure 3-5.



**Figure 3-5:** Dialog for creating multiple instances

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. Examples include the *Rail-S4* (section 4.2.1) or the *Conveyor-S4* (section 0).

If object components have a behaviour model (behaviour description), this is executed at the start of the simulation in just the same way as it would be if the object components were placed on a diagram like other components. The simulation of the behaviour of an object defined by that description is entirely independent of the treatment of that object in the conveyor technology simulation solver.

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 (Figure 3-6).

Boxes						
Reset Filter						
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	

**Figure 3-6:** Material list once the simulation has started

If an object is detected at a sensor during the simulation, the solver returns the ID of the detected object component.

### 3.3 Modelling the network

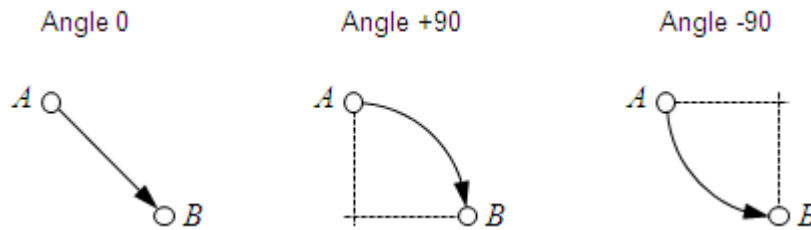
Networks are made up of segments, branches and boundaries. The *CONTEC* library provides various component types for modelling the conveyor system network. These component types represent elementary handling equipment types and therefore encapsulate corresponding structures consisting of segments, branches and boundaries. Thus the complexity of these subnetworks modelled in the component types can differ. For example, in a component type representing a straight conveyor only a single segment is modelled. 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 diagrams, the subnetwork models of the two components are connected accordingly. By then connecting them to other components the complete network of a conveyor system is ultimately constructed.

For every segment the geometry (with coordinates) and the reference direction have to be specified. The positions of stoppers and sensors can also be defined if required. For every branch you must specify the segments it connects and its position in the network.

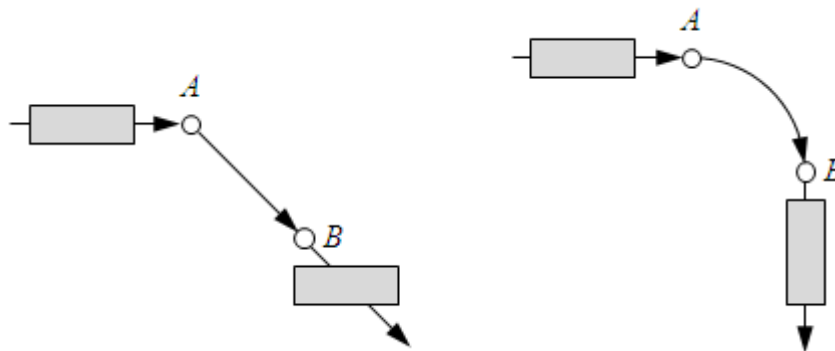
#### 3.3.1 Modelling segments

A segment is modelled 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, then there are two segment options, differentiated by the sign of the angle: a positive angle means that the centre point of the arc is located to the right of the segment in the counting direction, while a negative angle means that the centre point of the arc is located to the left of the segment (see Figure 3-7). The radius of the arc is determined from these three variables. The direction of the segment corresponds to the reference direction for transporting the objects.



**Figure 3-7:** Angle definition

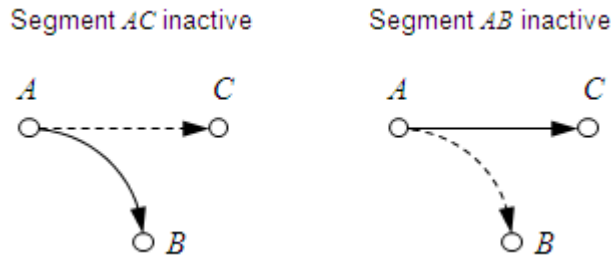
The object being transported over a segment is rotated by the angle of the segment. So on straight segments the object is of course transported without being rotated: the absolute orientation of the object is maintained. Figure 3-8 shows the transport of an object over a straight and a curved segment.



**Figure 3-8:** Transport of the object over a segment

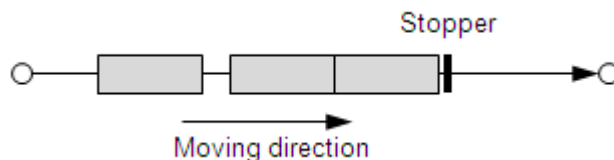
For each segment a speed is defined at which the objects are transported over that segment. Positive speed values cause the object to be transported in the reference direction, while negative speed values cause it to be transported against the reference direction. The inherent speed of an object where applicable is added to the segment speed.

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 disregarded in the conveyor technology solver. 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. Figure 3-9 shows how a switch function can be modelled by alternately activating and deactivating the two segments in the reference direction.



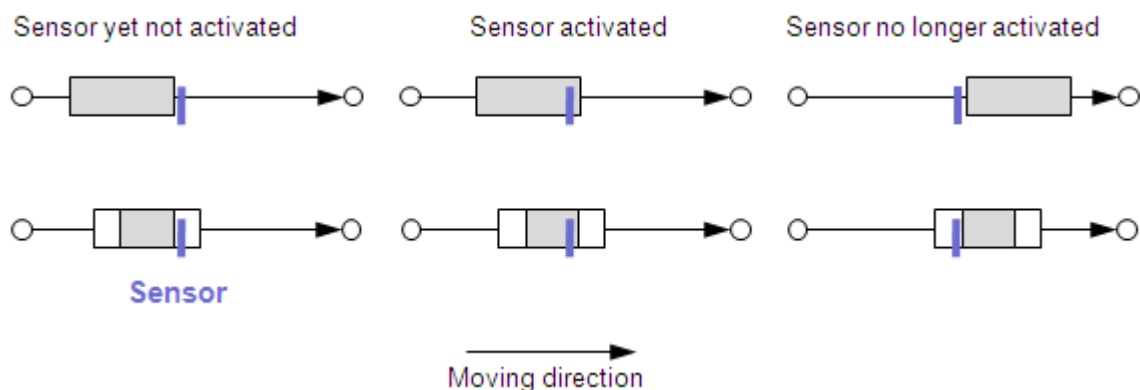
**Figure 3-9:** Active/inactive segments

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, then all objects are stopped at this position and all downstream objects are held up accordingly, as shown in Figure 3-10 by way of example.



**Figure 3-10:** Object held up at a stopper

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 ID of the object covering the position is then recorded in the conveyor technology solver. 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 in respect of the centre of the object. Figure 3-11 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 grey in each case, and movement is assumed to be from left to right.



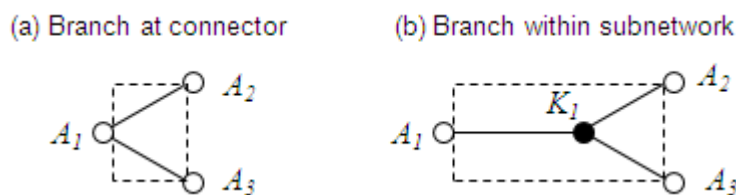
**Figure 3-11:** Detection of an object at sensor positions

Component types in which only one segment with sensors is modelled include the straight rail (*Rail-S4*) and the straight conveyor (*Conveyor-S4*).

### 3.3.2 Modelling branches

As all sections are assumed to be linear, a branch is a point in the network at which more than two segments connect.

In the subnetwork of a component type a branch can either be shown as a connector, i.e. a shared end point of multiple segments (Figure 3-12a), or it can be located within a subnetwork as shown in Figure 3-12b. 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.

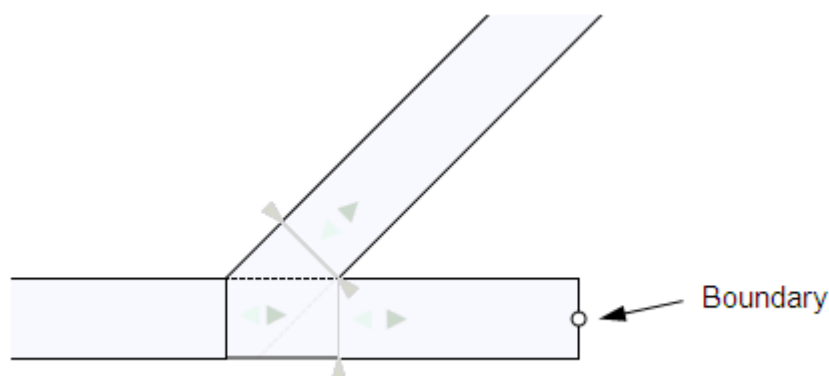


**Figure 3-12:** Branch at a connector and within a subnetwork

At branch points the transportation route must be explicitly defined at every moment of the simulation. In other words, of the segments that connect at a branch, no more than two 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.

### 3.3.3 Modelling 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 network is created if a connector of this component is not connected to other components (Figure 3-13). In the simulation, if an object moves beyond 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.



**Figure 3-13:** Boundary point created by an "open" connector of a component

However, boundary points can also be defined as end points in the subnetwork of component types. The behaviour 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 in order to simulate complex movements. Examples of such components include the turntable (section 4.3.12) and the transfer carriage (section 4.3.14).

## 3.4 Special features of conveyor technology simulation

### 3.4.1 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 centre.

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.

### 3.4.2 Simulating the holdup behaviour

Modelling of the holdup behaviour has been simplified to a great extent. Therefore the positions of held-up objects in the simulation do not necessarily correspond to the actual positions.

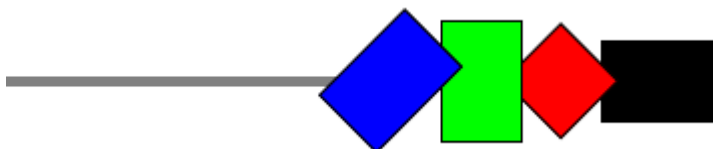
#### 3.4.2.1 Holdups on straight conveyor sections

The holdup behaviour on straight sections is correct for all objects that are either not rotated at all or are rotated in multiples of 90° (see Figure 3-14).



**Figure 3-14:** Holdup in the correct position

Objects with other rotation angles  $\varphi$  are also held up. However, as their position can only be calculated approximately, this leads to overlaps (Figure 3-15).



**Figure 3-15:** Holdup in the approximate position

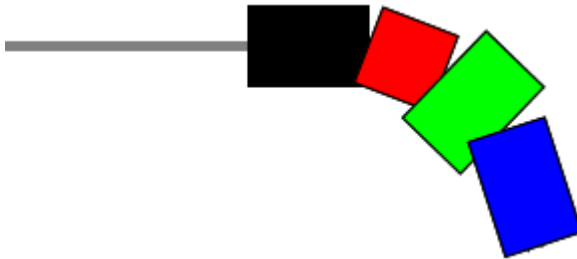
The effective width  $\tilde{b}$  of an object that is used as a simplification to simulate the holdup is calculated using the following formula:

$$\tilde{b} = \frac{\varphi h + (\pi - 2\varphi)b}{\pi}$$

where  $h$  denotes the height and  $b$  the width of the object.

### 3.4.2.2 Holdups in curves

In curves too the position of held-up objects is only an approximation. The held-up objects overlap, as shown in Figure 3-16.

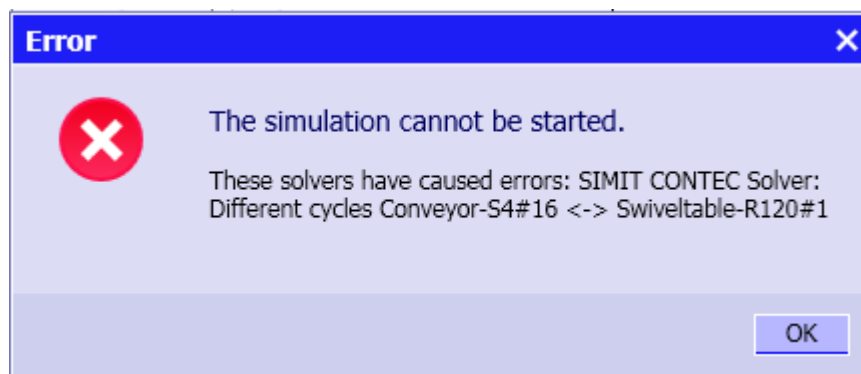


**Figure 3-16:** Holdup in a curve

The overlaps are caused by the fact that the bend of the curve is disregarded in the distance calculation.

### 3.4.3 Time slice allocation for components

All components forming a cohesive network must be allocated the same time slice. Consequently all components that simulate handling equipment and are directly connected to one another via connectors or signal lines must be allocated to the same time slice. Otherwise the simulation model of a cohesive 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.



**Figure 3-17:** Different sub-models in a conveyor section

## 3.5 Scalability

Unlike the case in the standard library, in the conveyor technology library the dimensions of components play a decisive role, as they directly indicate the length of conveyor sections or the dimensions of objects. In accordance with standard engineering practice, millimetres [mm] are used throughout as the length unit, while speeds are given in metres per second [m/s].



### 3.5.1 Scalability of conveyor sections

To enable conveyor systems of differing sizes to be mapped on diagrams with a reasonable resolution, a scale can be assigned to each diagram (Figure 3-18).

Diagramm		
General	Property	Value
	Name	Diagramm
	Width	16000
	Height	14000
	Scale	1 pix : 20 mm
	Background Image	... X

**Figure 3-18:** Diagram scale

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, and
- 1 pix : 100 mm.



#### **NOTE**

If you have licensed the conveyor technology library for your SIMIT installation, the sizes of all components, including components from the basic library, will be 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 diagrams containing base components.

To avoid having to set the scale individually for each diagram, you can choose a default setting for the entire project in the Properties dialog of the Project Manager (Figure 3-19).

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

**Figure 3-19:** Default scale

The scale is used as the default for all new diagrams that you create.



### CAUTION

Setting a default scale in the Project Manager will not change the scale of any existing diagrams.

Note that changing the scale of a diagram does not change the displayed size of a component but rather its effective dimensions in millimetres.

## 3.5.2 Scalability of objects

Objects are shown in the material lists of a SIMIT project with their absolute size (width and height) in millimetres (Figure 3-20). Therefore objects are represented on diagrams in differing sizes, depending on the chosen scale of the diagram.

Boxes				
Reset Filter				
Name	Width	Height	Depth	Type
Box#1	1200	800	0	Box

**Figure 3-20:** Absolute size of the object

The dimensions of an object are predefined 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

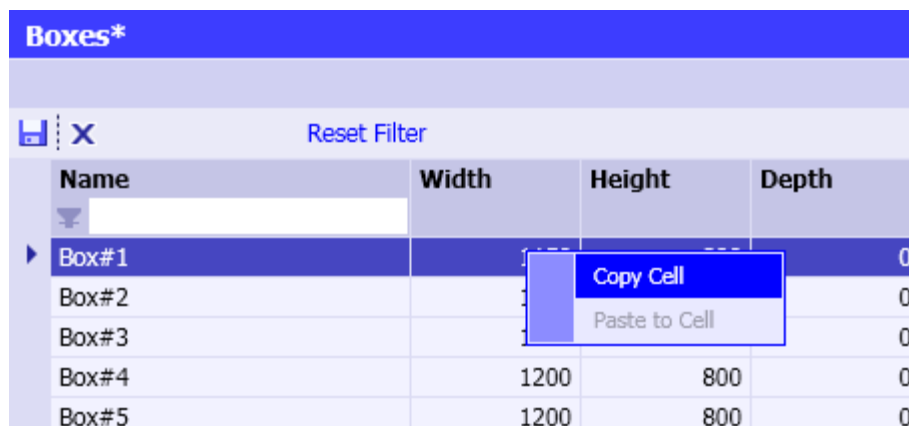
Note that *width* and *height* refer to the dimensions of the object as viewed from above, as in the diagram. The *depth* is the size in the third dimension, which is not shown. This definition was chosen so as to maintain the significance of height and width of components and graphics on a diagram.

The component types provided in the conveyor technology 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. The depth of objects only becomes important if you create a custom conveyor technology component type that evaluates the depth of an object, for example a height check, and use it in your conveyor technology simulation (see section 5).

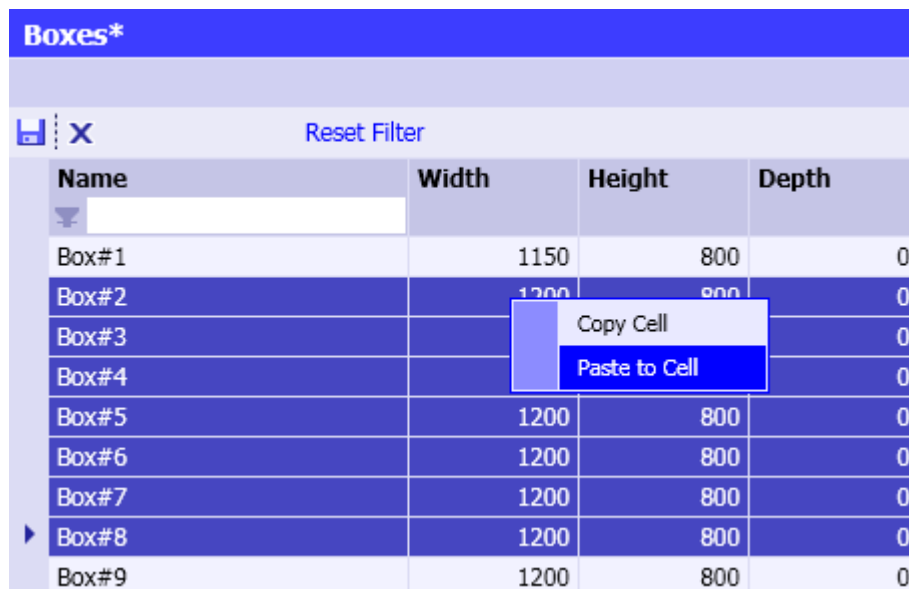
If you want to set the dimensions of several objects to the same value, you can copy the value for one object, in other words from one row of this table, to any number of others. Proceed as follows:

- Select the row containing the desired object value.
- Right-click on the cell containing the value to be copied and select "Copy Cell" in the context menu (see Figure 3-21).
- Select the lines 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 context menu (see Figure 3-22).



Name	Width	Height	Depth
Box#1			0
Box#2			0
Box#3			0
Box#4	1200	800	0
Box#5	1200	800	0

Figure 3-21: Copy cell



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

Figure 3-22: Paste to cell

### 3.6 Generating the simulation of drives and sensors

As conveyor sections can be shown to scale on diagrams 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 diagrams. This would compromise the clarity of the system layout. Therefore the only visible connectors on the component types in the *CONTEC* library are those used to connect them to one another to construct the system layout. Connectors for connecting components to the associated drive and sensor signals, and hence via the gateways 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.

### 3.6.1 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, Figure 3-23 shows a rail with the name *Rail*.



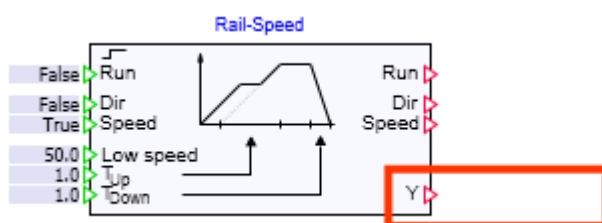
**Figure 3-23:** Rail with the name *Rail*

Therefore its *Speed* input (Figure 3-24) 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




**Figure 3-24:** Default value of the implicit connection

You can therefore create a diagram 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 (see Figure 3-25).






**Figure 3-25:** Trigger component for the rail

Alternatively, you can also enter every other (analog) output of a component in the *Speed* input for the rail by changing the setting of { \$ } to and entering the corresponding output signal, as shown by way of example in Figure 3-26.

Name	Value/Signal
 Speed  	DriveP1#1 Y





**Figure 3-26:** Editable implicit connection

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 (Figure 3-27).

Name	Value/Signal
 Speed  123 	100.0

**Figure 3-27:** Input with constant value

If you want to be able to process output signals for the rail sensors further, you again use implicit connections. You can use any components and assign their (binary) inputs accordingly, as shown by way of example in Figure 3-28.

Name	Value/Signal
 IN  	Rail Sensor1 

**Figure 3-28:** Input with implicit connection

### 3.6.2 Using templates

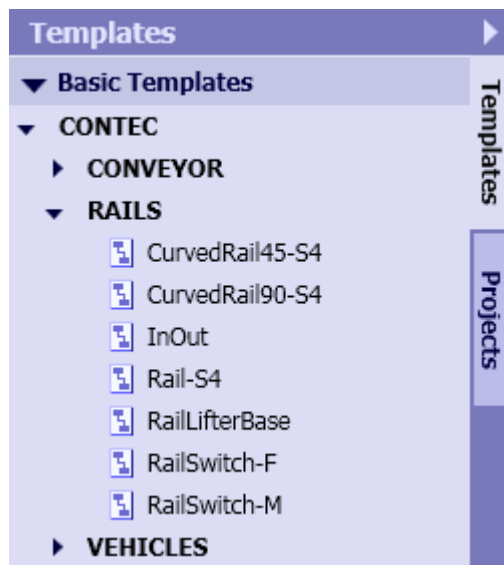
Templates can be used as a way of creating diagrams for drive and sensor simulation with minimal effort.



#### **NOTE**

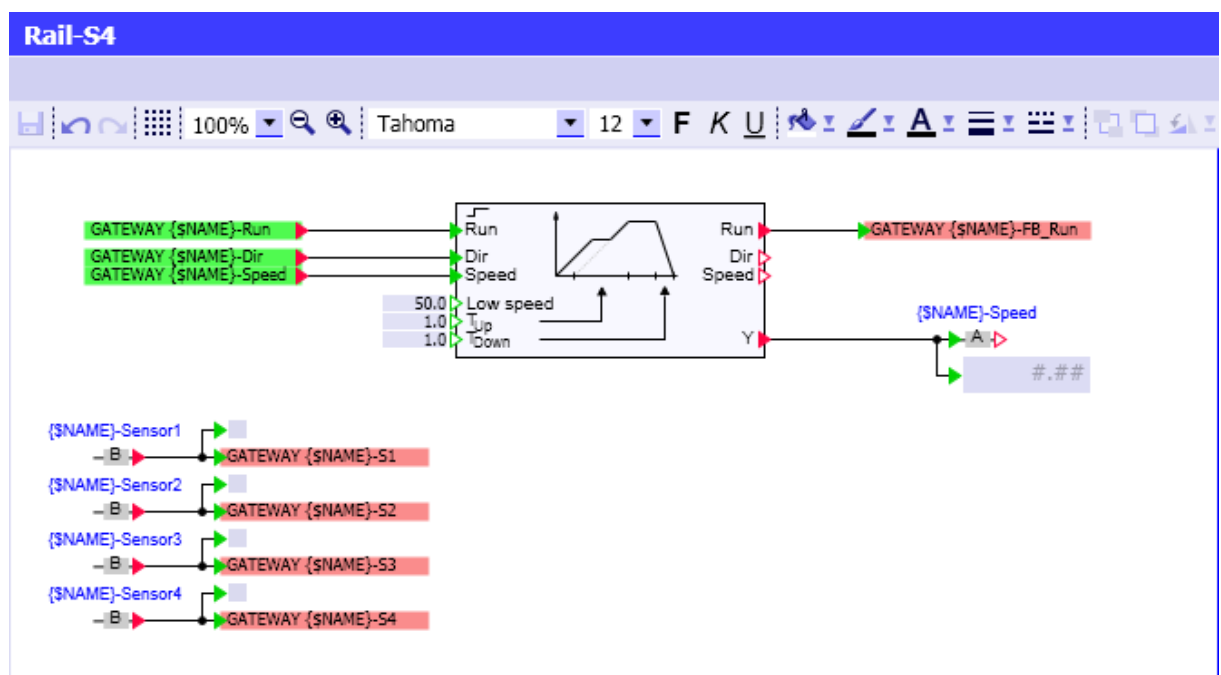
Note that templates can only be used with SIMIT Professional and SIMIT Ultimate.

The conveyor technology library includes templates for connecting the drive and sensor signals for every component type of a conveyor section and for the vehicles. The template names are the same as the names of the component types.



**Figure 3-29:** Basic templates for the *CONTEC* library

All the 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 gateway signals. Figure 3-30 shows the template for the *Rail-S4* component type of a rail.



**Figure 3-30:** Basic template *Rail-S4*

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 gateway simply by instantiating the template.

To connect all conveyor sections in the simulation project to the controller with the minimum possible effort, use the *CDL* (Create Device Level) function in the SIMIT Project Manager (Figure 3-31).

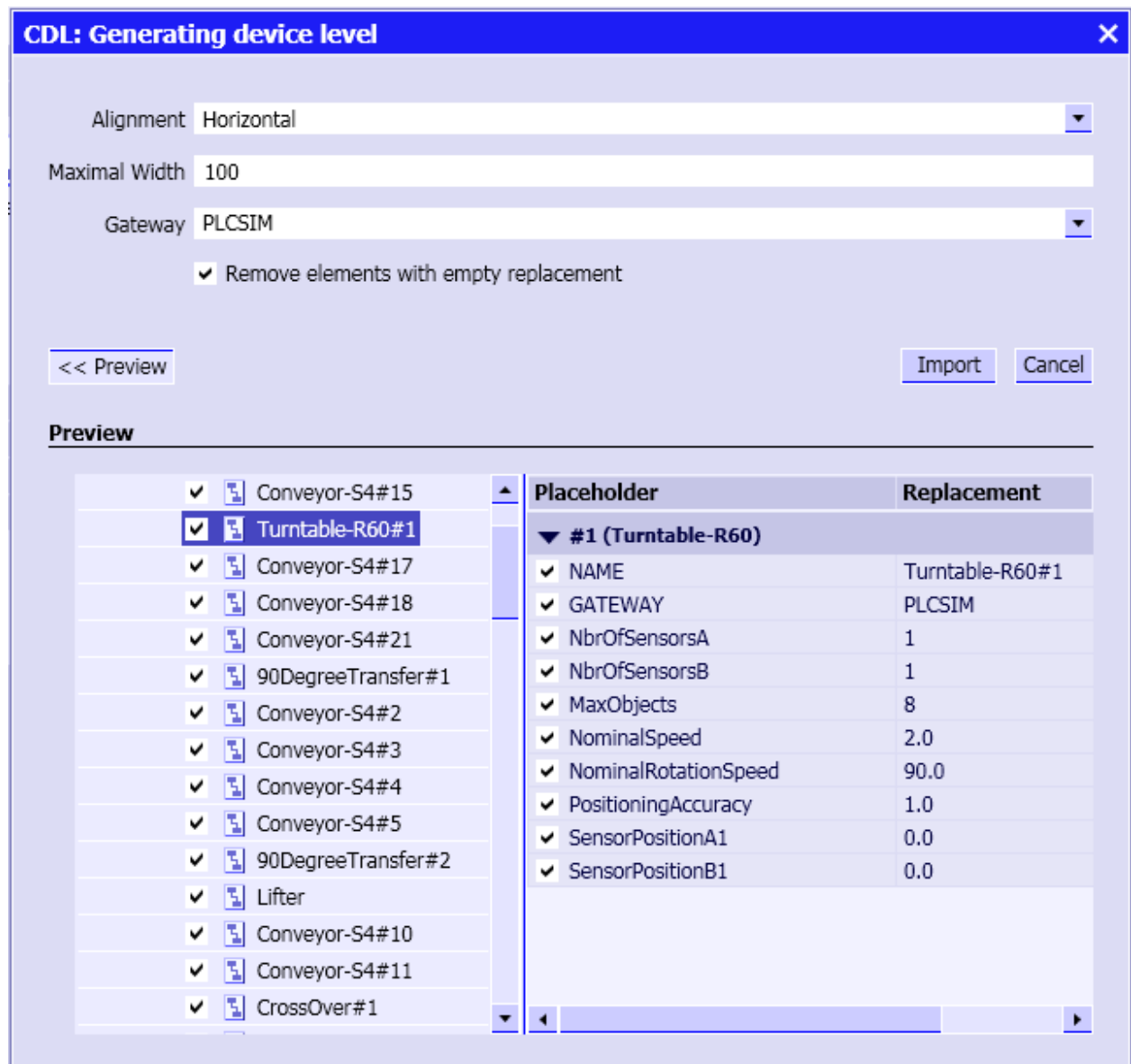


**Figure 3-31:** Calling the CDL function in the Project Manager

To call the function, in the Project Manager select the diagram folder in which you want to instantiate the templates for your simulation project. If you do not select a diagram folder, the CDL command remains disabled. Using the *CDL* function, 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 *CDL* function provides the associated diagrams for connecting the controller.

As reference is naturally made to the gateway signals in the templates for generating the device level, you can specify a gateway name in the CDL dialog. It can be entered under the variable name *GATEWAY* (see Figure 3-32). If your simulation project already includes gateways, they are listed in the drop-down list.



**Figure 3-32:** Preview of the CDL function in the Project Manager



#### **NOTE**

You will generally have to adapt the names of the input/output signals in the templates to your identification system in order to gain the maximum benefit from this method.



## 4 CONTEC COMPONENT LIBRARY

The component types in the *CONTEC* library are divided into

- Component types for conveyor systems with vehicles (*RAILS* directory),
- Component types for non-vehicular conveyor systems (*CONVEYOR* directory) and
- Component types for simulating objects (*MATERIAL* directory).

The *SENSORS* library contains a new folder with

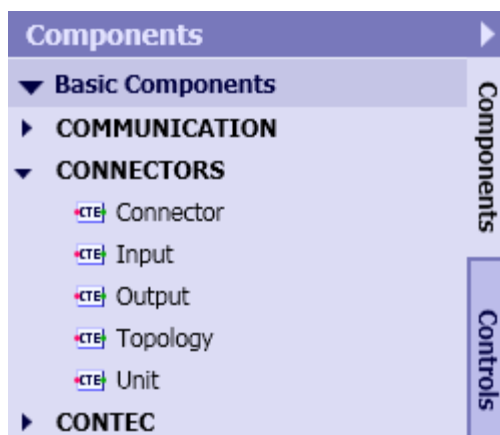
- Components for simulating identification systems (*RFID* directory).

### 4.1 The Topology connector

In the *CONNECTORS* directory of the SIMIT basic library is a connector that can be used to create topological connectors for conveyor technology components that transcend diagram limits:

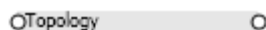
- the *Topology* connector

(see Figure 4-1).



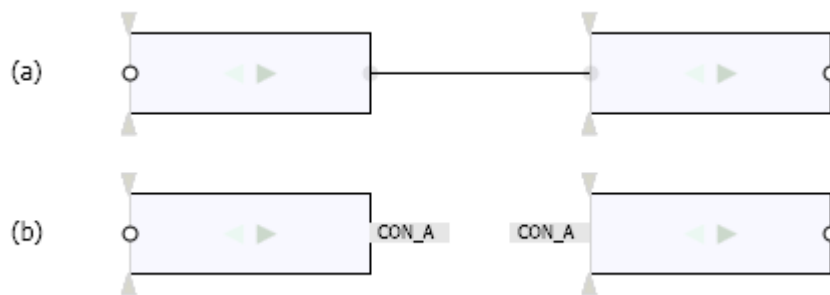
**Figure 4-1:** Connector component types in the basic library

The *Topology* symbol is shown in Figure 4-2.



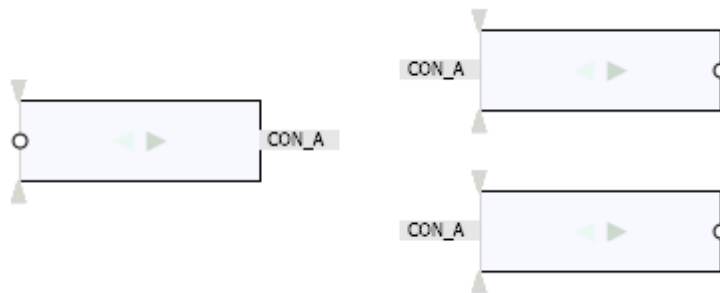
**Figure 4-2:** *Topology* conveyor technology connector

The *Topology* connector can be used to create a topological connection between two conveyor technology components. Figure 4-3b shows two components connected by the *CON\_A* connector. The connection is functionally identical to the direct connection of both components via a connecting line, as shown in Figure 4-3a.



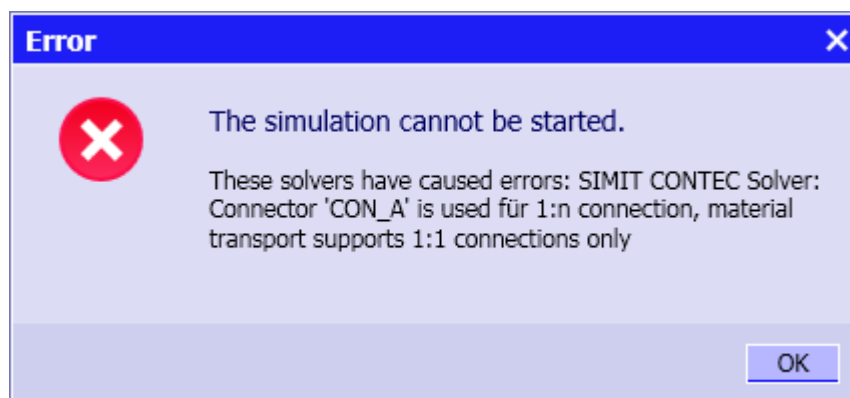
**Figure 4-3:** Conveyor technology connector on a branch

Figure 4-4 shows three components connected by the *CON\_A* connector. This configuration is not valid, because with conveyor technology components only two topological connectors can ever be connected to one another.



**Figure 4-4:** Invalid use of the connector as a branch

If you start a simulation project containing a configuration like this, an error message will appear as shown in Figure 4.5 and the simulation will be aborted.



**Figure 4-5:** Error message referring to invalid use of the connector as a branch

## 4.2 Component types for conveyor systems with vehicles

The *RAILS* directory (Figure 4-6) of the CONTEC library contains component types for use in vehicular conveyor system simulations. In the broadest sense these types are "rails" which can carry vehicles 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.

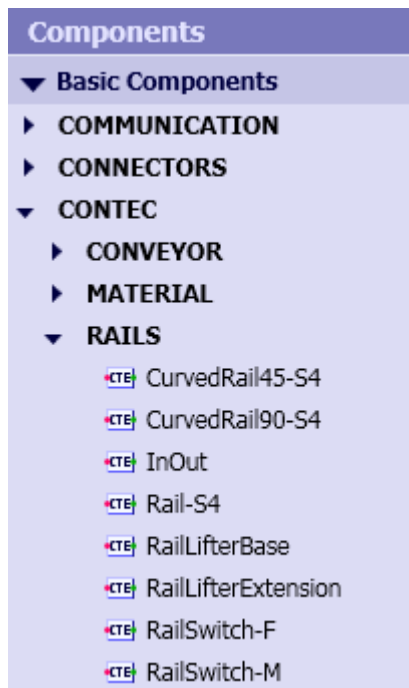


Figure 4-6: *RAILS* library directory

### 4.2.1 Rail-S4 – Straight rail with four sensors

#### Symbol



The grey 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 component's hidden *Speed* input as a percentage of the programmable 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

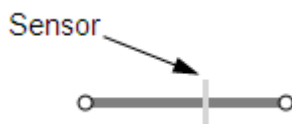
*Sensor4* is set. The ID of the detected vehicle can be obtained via the corresponding hidden integer outputs *SensorId1* to *SensorId4*.

### Parameters

The behaviour 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 in relation to connector A of the symbol.

When the simulation is running, every defined sensor is shown in the correct position in the symbol (Figure 4-7). When a sensor is activated, its colour changes to yellow.



**Figure 4-7:** Sensor in the component symbol

The parameters, their units and default values are shown in Figure 4-8.

Name		Value
NominalSpeed	[m/s]	2.0
NbrOfSensors		1
▼ SensorPosition [1]		...
SensorPositi...	[mm]	8000.0

**Figure 4-8:** Parameters for the *Rail-S4* component type

The validity of the parameters is checked while the component is being initialised. The following messages indicate a possible parameter setting error:

- *Parameter 'SensorPosition' must be less than the component's width*  
The sensor position must be within the component.
- *Parameter 'SensorPosition' 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 (see section 3.6.2).

Further additional parameters can be used during initialisation to place vehicles on the rail.

- *MaterialType*  
The type of vehicle

- *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*

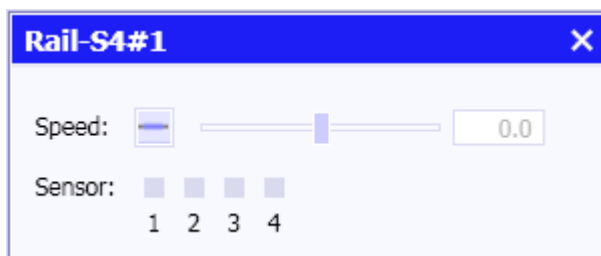
The initial distance between the vehicles

Name	Value
MaterialType	
MaterialList	
InitNbrOfObjects	0
Clearance [mm]	0.0
TEMPLATE	Rail-S4
HIERARCHY	RAILS

**Figure 4-9:** Additional parameters for the *Rail-S4* component type

### Operating window

In the operating window (Figure 4-10) 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 four sensors.



**Figure 4-10:** Operating window for the *Rail-S4* component type

## 4.2.2 *CurvedRail45-S4* – 45° curved rail with four sensors

### Symbol



The grey 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 only 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 component's hidden *Speed* input as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter).

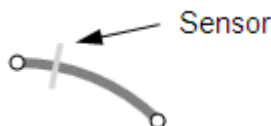
Between one and four sensors can be positioned along the length of the rail. When a vehicle is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the vehicles can be obtained via the hidden integer outputs *SensorId1* to *SensorId4*.

### Parameters

The behaviour 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 in relation to connector A of the symbol.

When the simulation is running, every defined sensor is shown in the correct position in the symbol (Figure 4-11). When a sensor is activated, its colour changes to yellow.



**Figure 4-11:** Sensor in the component symbol

The parameters, their units and default values are shown in Figure 4-12.

Name	Value
NominalSpeed [m/s]	2.0
NbrOfSensors	1
▼ SensorPosition [1]	...
SensorPositi... [mm]	8000.0

**Figure 4-12:** Parameters for the *CurvedRail45-S4* component type

The validity of the parameters is checked while the component is being initialised. The following messages indicate a possible parameter setting error:

- *Parameter 'SensorPosition' must be less than the component's width*  
The sensor position must be within the component.

- Parameter 'SensorPosition' must not be negative

### Additional parameters

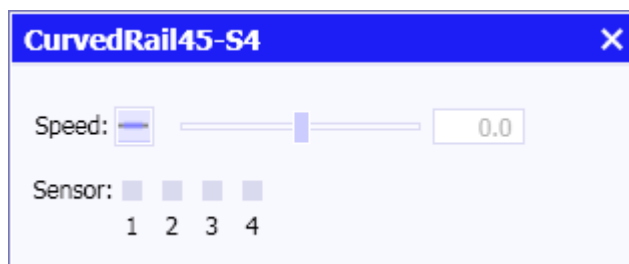
The *CurvedRail45-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	CurvedRail45-S4
HIERARCHY	RAILS

**Figure 4-13:** Additional parameters for the *CurvedRail45-S4* component type

### Operating window

In the operating window (Figure 4-14) 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 four sensors.



**Figure 4-14:** Operating window for the *CurvedRail45-S4* component type

## 4.2.3 *CurvedRail90-S4* – 90° curved rail with four sensors

### Symbol



The grey 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 only 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 component's hidden *Speed* input as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter).

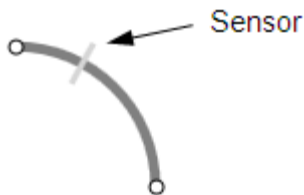
Between one and four sensors can be positioned along the length of the rail. When a vehicle is within the detection range of a sensor, the corresponding hidden binary output *Sensor1* to *Sensor4* is set. The ID of the vehicles can be obtained via the hidden integer outputs *SensorId1* to *SensorId4*.

### Parameters

The behaviour 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 in relation to connector A of the symbol.

When the simulation is running, every defined sensor is shown in the correct position in the symbol (Figure 4-15). When a sensor is activated, its colour changes to yellow.



**Figure 4-15:** Sensor in the component symbol

The parameters, their units and default values are shown in Figure 4-16.

Name		Value
NominalSpeed	[m/s]	2.0
NbrOfSensors		1
▼ SensorPosition [1]		...
SensorPositi...	[mm]	8000.0

**Figure 4-16:** Parameters for the *CurvedRail90-S4* component type

The validity of the parameters is checked while the component is being initialised. The following messages indicate a possible parameter setting error:

- *Parameter 'SensorPosition' must be less than the component's length*  
The sensor position must be within the component.
- *Parameter 'SensorPosition' must not be negative*



### Additional parameters

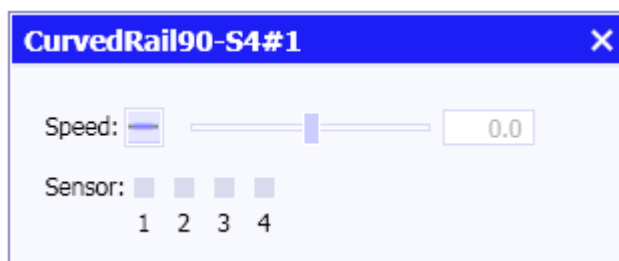
The *CurvedRail90-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	CurvedRail90-S4
HIERARCHY	RAILS

**Figure 4-17:** Additional parameters for the *CurvedRail90-S4* component type

### Operating window

In the operating window (Figure 4-18) 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 four sensors.



**Figure 4-18:** Operating window for the *CurvedRail90-S4* component type

## 4.2.4 RailSwitch-F – Switchable switch with 45° spur

### Symbol



The grey 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), then 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 component's hidden *Speed* input as a percentage of the programmable 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 behaviour 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 Figure 4-19.

Name	Value
NominalSpeed [m/s]	2.0

**Figure 4-19:** Parameters for the *RailSwitch-F* component type

### Additional parameters

The *RailSwitch-F* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	RailSwitch-F
HIERARCHY	RAILS

**Figure 4-20:** Additional parameters for the *RailSwitch-F* component type

### Operating window

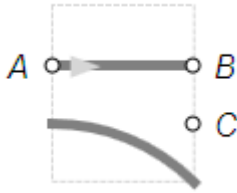
In the operating window (Figure 4-21) you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider and switch the sections.



**Figure 4-21:** Operating window for the *RailSwitch-F* component type

### 4.2.5 *RailSwitch-M* – Movable switch with 45° spur

#### Symbol



The grey 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 programmable (*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 programmable 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 component's hidden analog input *Speed* as a percentage of the programmable 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 behaviour 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 Figure 4-22.

Name		Value
NominalSpeed	[m/s]	2.0
SwitchingTime	[s]	1.0

**Figure 4-22:** Parameters for the *RailSwitch-M* component type

### Additional parameters

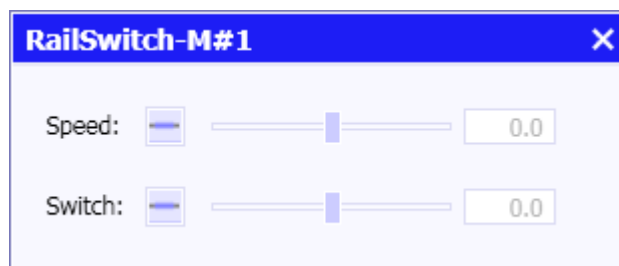
The *RailSwitch-M* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	RailSwitch-M
HIERARCHY	RAILS

**Figure 4-23:** Additional parameters for the *RailSwitch-M* component type

### Operating window

In the operating window (Figure 4-24) 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 programmable switching time (*SwitchingTime*).



**Figure 4-24:** Operating window for the *RailSwitch-M* component type

## 4.2.6 InOut – Inward and outward section for vehicles

### Symbol



The width of the symbol is scalable. The 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 network. The *InOut* component is connected to an open section of the network at connector *A* to extend the network.

To move them into the network, individual vehicles are placed on the unconnected end of the inward/outward section (left-hand 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 network. The speed at which

vehicles are moved over the inward/outward section is set at the component's hidden *Speed* input as a percentage of the programmable 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 programmed by means of parameters.

The parameters, their units and default values are shown in Figure 4-25.

Name	Value
NominalSpeed [m/s]	2.0
MaterialType	
MaterialList	

**Figure 4-25:** Parameters for the *InOut* component type

### Additional parameters

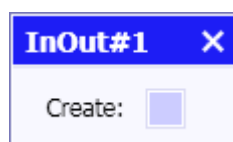
The *InOut* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	InOut
HIERARCHY	RAILS

**Figure 4-26:** Additional parameters for the *InOut* component type

### Operating window

In the operating window (Figure 4-27) you can place a new vehicle on the inward/outward section by clicking the button.



**Figure 4-27:** Operating window for the *InOut* component type

### 4.2.7 *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 extension components of the *RailLifterExtension* type. The base component simulates the lifter in the base level, while each additional level is simulated by an extension 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 extension components on the same diagram. You can create a separate diagram for each level, for example, and distribute the lifter components over the individual diagrams.

The following points apply to the programming 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 extension components used.
- In the base component you have to specify the position (*LevelPosition*) for each level, i.e. the level above the base level, in millimetres. 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 setting the parameters for the extension components:

- The name of the extension 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 indicator. The number indicating the level starts with 1.
- The *BaseName* parameter is the name of the base component.
- The length of the lifter is determined by the width of the base component.

Therefore all extension components must be exactly the same width as the base component.



#### **NOTE**

Please note, all extension components must be exactly the same width as the base component. Otherwise, when starting the simulation a message will be shown pointing to this inconsistency.

The rail drive and the lifter sensors are fully implemented in the base component. Therefore extension components do not work without the base component.

#### 4.2.7.1 *RailLifterBase* – Lifter (base component)

##### **Symbol**



The grey 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 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 component's hidden analog input *Speed* as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter). Similarly, the lifting speed, i.e. 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 programmable nominal lifting speed (*NominalLifterSpeed* parameter).

In order to be able 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 one and four 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*.



### CAUTION

The base component has further hidden inputs for exchanging signals with extension components. The corresponding signals are assigned to these inputs. Do not change these default settings, otherwise the lifter components will not behave in the intended way in the simulation.

### Parameters

The behaviour of the component can be programmed.

- *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 in relation to connector A of the symbol

- *NbrOfSensorsB*  
Number of sensors relative to connector B (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor in relation 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 colour changes to yellow.

The parameters, their units and default values are shown in Figure 4-28.

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

**Figure 4-28:** Parameters for the *RailLifterBase* component type

### Additional parameters

The *RailLifterBase* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

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

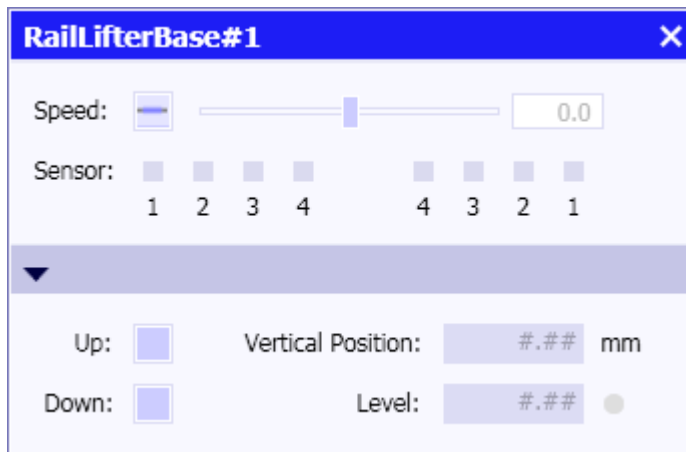
**Figure 4-29:** Additional parameters for the *RailLifterBase* component type

### Operating window

In the operating window (Figure 4-30) 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 buttons. The current lifter position and level are displayed.





**Figure 4-30:** Operating window for the *RailLifterBase* component type

#### 4.2.7.2 *RailLifterExtension* – Lifter (extension component)

##### **Symbol**



The grey 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 lifter rail.



##### **CAUTION**

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 behaviour of the component can be programmed:

- *Level*  
Level at which the component is located. The numbers start at 1.
- *BaseName*  
Name of the corresponding base component

The parameters, their units and default values are shown in Figure 4-31.

Name	Value
Level	1
BaseName	

**Figure 4-31:** Parameters for the *RailLifterExtension* component type

### ***Additional parameters***

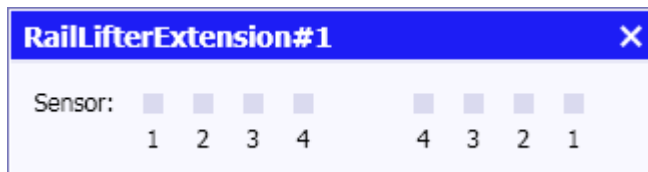
You have to specify how many vehicles can be present on the section (lifter rail) defined by the component at any one time. This parameter should be set to the same value as for the base component.

Name	Value
MaxObjects	8

**Figure 4-32:** Additional parameters for the *RailLifterExtension* component type

### ***Operating window***

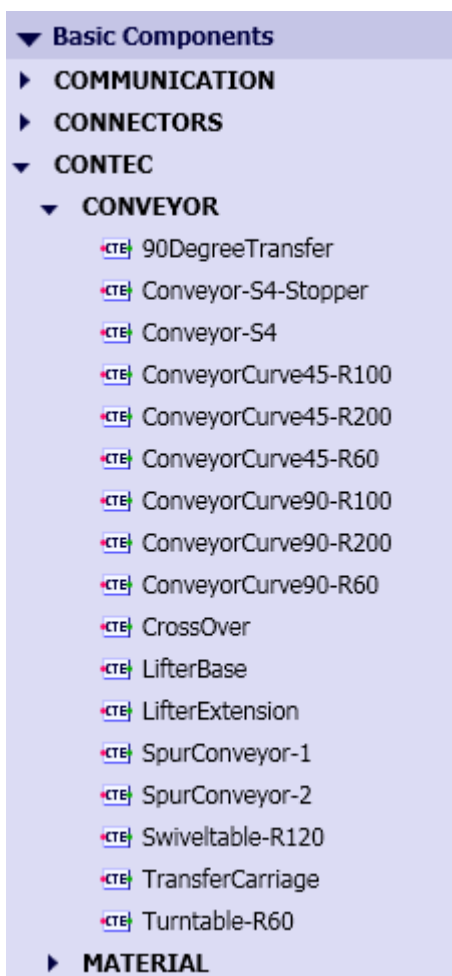
In the operating window (Figure 4-33) you can only monitor the status of the sensors.



**Figure 4-33:** Operating window for the *RailLifterExtension* component type

## **4.3 Component types for non-vehicular conveyor systems**

The *CONVEYOR* directory (Figure 4-34) 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.



**Figure 4-34:** CONVEYOR library directory

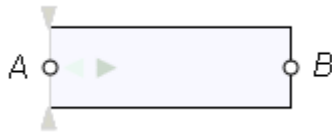
All component types of the CONVEYOR library have a symbol of a fixed height of 40 pixel. According to the selected scale the width of the conveyors can be adjusted to the values listed in Table 4-1.

Conveyor width	Scale
40mm	1pix : 1mm
200mm	1pix : 5mm
400mm	1pix : 10mm
800mm	1pix : 20mm
2000mm	1pix : 50mm
4000mm	1pix : 100mm

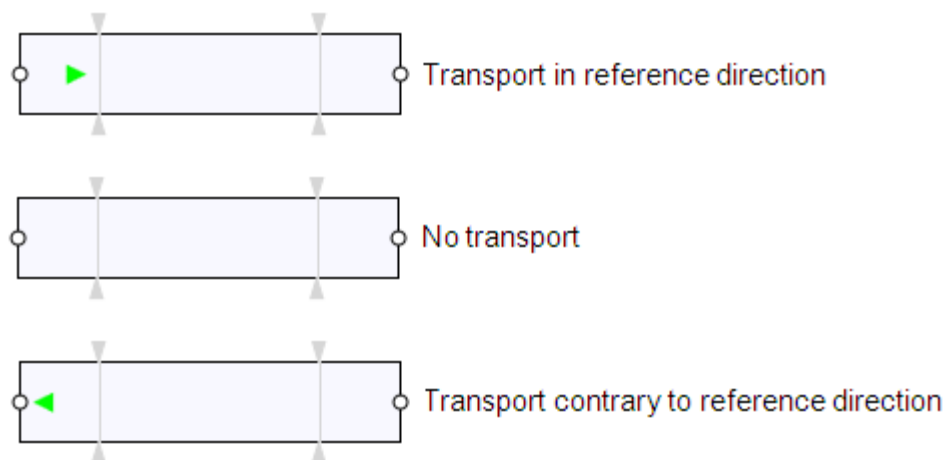
**Table 4-1:** Selectable width of a conveyor

### 4.3.1 Conveyor-S4 – Straight conveyor with four sensors

#### Symbol



The grey 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 (see Figure 4-35).



**Figure 4-35:** Indication of the current transport direction in the symbol

The width of the symbol is scalable. The 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 component's hidden analog input *Speed* as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter).

Between one and four 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 behaviour 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 in relation to connector *A* of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor in relation 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 colour changes to yellow.

The parameters, their units and default values are shown in Figure 4-36.

Name		Value
NominalSpeed	[m/s]	2.0
NbrOfSensorsA		1
▼ SensorPositionA [1]		...
SensorPositionA1	[mm]	0.0
NbrOfSensorsB		1
▼ SensorPositionB [1]		...
SensorPositionB1	[mm]	0.0

**Figure 4-36:** Parameters for the *Conveyor-S4* component type

### ***Additional parameters***

The *Conveyor-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

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 to be 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 to be placed on the conveyor
- *Clearance*  
The distance between the objects

Name	Value
MaterialType	
MaterialList	
InitNbrOfObjects	0
Clearance [mm]	0.0
TEMPLATE	Conveyor-S4
HIERARCHY	CONVEYOR

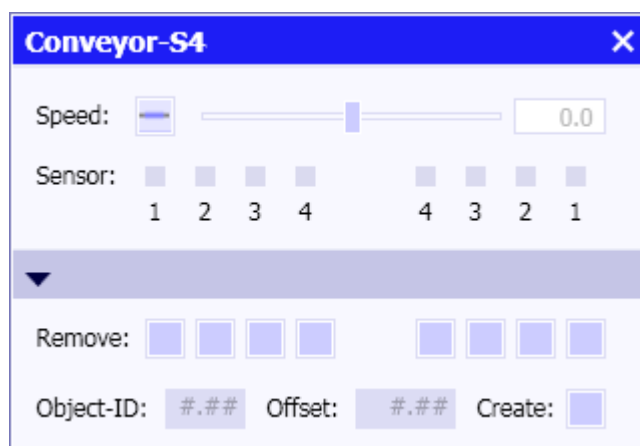
**Figure 4-37:** Additional parameters for the *Conveyor-S4* component type

### Operating window

In the operating window (Figure 4-38) 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* button (*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.

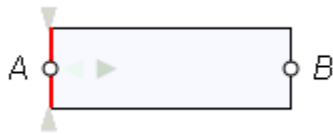
This simulates manual interventions in the conveyor system such as the removal and placement of objects.



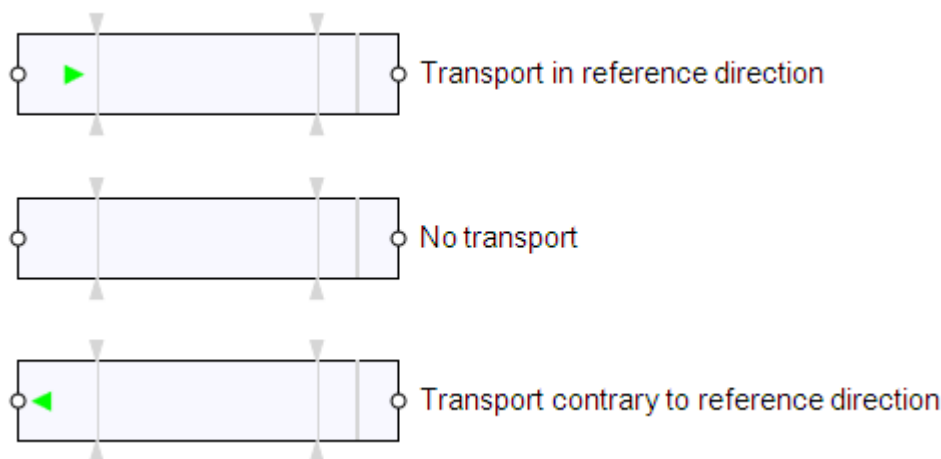
**Figure 4-38:** Operating window for the *Conveyor-S4* component type

### 4.3.2 Conveyor-S4-Stopper – Straight conveyor with four sensors and stopper

#### Symbol



The grey 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 (see Figure 4-35).



**Figure 4-39:** Indication of the current transport direction in the symbol

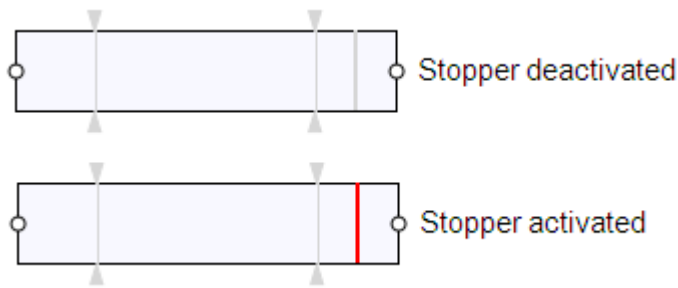
The width of the symbol is scalable. The width corresponds to the length of the conveyor.

#### 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 component's hidden analog input *Speed* as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter).

Between one and four 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 colour in the symbol changes to red (see Figure 4-40).



**Figure 4-40:** Representation of the stopper in the symbol

### Parameters

The behaviour of the component can be set by means of parameters:

- *NominalSpeed*  
Nominal conveyor speed; adjustable online
- *StopperPosition*  
Position of the stopper in relation to connector A
- *NbrOfSensorsA*  
Number of sensors relative to connector A (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor in relation to connector A of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector B (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor in relation 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 colour changes to yellow.

The parameters, their units and default values are shown in Figure 4-41.

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

**Figure 4-41:** Parameters for the *Conveyor-S4-Stopper* component type



### Additional parameters

The *Conveyor-S4* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

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 to be 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 to be placed on the conveyor
- *Clearance*  
The distance between the objects

Name	Value
MaterialType	
MaterialList	
InitNbrOfObjects	0
Clearance	0.0
TEMPLATE	Conveyor-S4-Stopper
HIERARCHY	CONVEYOR

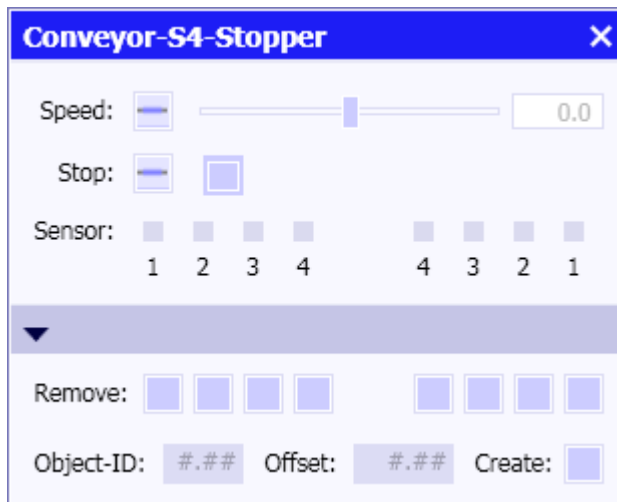
**Figure 4-42:** Additional parameters for the *Conveyor-S4-Stopper* component type

### Operating window

In the operating window (Figure 4-43) 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* button (*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.



**Figure 4-43:** Operating window for the *Conveyor-S4-Stopper* component type

### 4.3.3 *ConveyorCurve45-R60* – 45° curved conveyor (radius 60 pixels)

#### **Symbol**



The grey 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 diagram. For example, a scale of 1 pix : 100 mm gives an effective radius of 6 metres.

The speed at which objects are moved over this conveyor is set at the component's hidden analog input *Speed* as a percentage of the programmable 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 Figure 4-44.

Name	Value
NominalSpeed	[m/s] 2.0

**Figure 4-44:** Parameters for the *ConveyorCurve45-R60* component type

### Additional parameters

The *ConveyorCurve45-R60* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	ConveyorCurve45-R60
HIERARCHY	CONVEYOR

**Figure 4-45:** Additional parameters for the *ConveyorCurve45-R60* component type

### Operating window

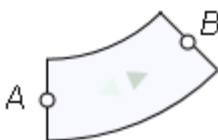
In the operating window (Figure 4-46) you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



**Figure 4-46:** Operating window for the *ConveyorCurve45-R60* component type

## 4.3.4 *ConveyorCurve45-R100* – 45° curved conveyor (radius 100 pixels)

### Symbol



The grey 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 diagram. For example, a scale of 1 pix : 100 mm gives an effective radius of 10 metres.

The speed at which objects are moved over this conveyor is set at the component's hidden analog input *Speed* as a percentage of the programmable 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 Figure 4-47.

Name	Value
NominalSpeed	[m/s] 2.0

**Figure 4-47:** Parameters for the *ConveyorCurve45-R100* component type

### Additional parameters

The *ConveyorCurve45-R100* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	ConveyorCurve45-R100
HIERARCHY	CONVEYOR

**Figure 4-48:** Additional parameters for the *ConveyorCurve45-R100* component type

### Operating window

In the operating window (Figure 4-49) you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



**Figure 4-49:** Operating window for the *ConveyorCurve45-R100* component type

### 4.3.5 ConveyorCurve45-R200 – 45° curved conveyor (radius 200 pixels)

#### Symbol



The grey 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 diagram. For example, a scale of 1 pix : 100 mm gives an effective radius of 20 metres.

The speed at which objects are moved over this conveyor is set at the component's hidden analog input *Speed* as a percentage of the programmable 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 Figure 4-50.

Name	Value
NominalSpeed	[m/s] 2.0

**Figure 4-50:** Parameters for the *ConveyorCurve45-R200* component type

#### Additional parameters

The *ConveyorCurve45-R200* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	ConveyorCurve45-R200
HIERARCHY	CONVEYOR

**Figure 4-51:** Additional parameters for the *ConveyorCurve45-R200* component type

### Operating window

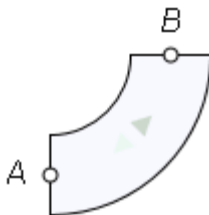
In the operating window (Figure 4-52) you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



**Figure 4-52:** Operating window for the *ConveyorCurve45-R200* component type

### 4.3.6 *ConveyorCurve90-R60* – 90° curved conveyor (radius 60 pixels)

#### Symbol



The grey 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-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 diagram. For example, a scale of 1 pix : 100 mm gives an effective radius of 6 metres.

The speed at which objects are moved over this conveyor is set at the component's hidden analog input *Speed* as a percentage of the programmable 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 Figure 4-53.

Name	Value
NominalSpeed	[m/s] 2.0

**Figure 4-53:** Parameters for the *ConveyorCurve90-R60* component type

### Additional parameters

The *ConveyorCurve90-R60* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	ConveyorCurve90-R60
HIERARCHY	CONVEYOR

**Figure 4-54:** Additional parameters for the *ConveyorCurve90-R60* component type

### Operating window

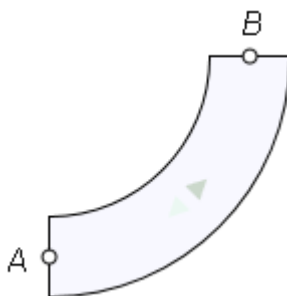
In the operating window (Figure 4-55) you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



**Figure 4-55:** Operating window for the *ConveyorCurve90-R60* component type

## 4.3.7 *ConveyorCurve90-R100* – 90° curved conveyor (radius 100 pixels)

### Symbol



The grey 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 diagram. For example, a scale of 1 pix : 100 mm gives an effective radius of 10 metres.

The speed at which objects are moved over this conveyor is set at the component's hidden analog input *Speed* as a percentage of the programmable 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 Figure 4-56.

Name	Value
NominalSpeed	[m/s] 2.0

**Figure 4-56:** Parameters for the *ConveyorCurve90-R100* component type

### Additional parameters

The *ConveyorCurve90-R100* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	ConveyorCurve90-R100
HIERARCHY	CONVEYOR

**Figure 4-57:** Additional parameters for the *ConveyorCurve90-R100* component type

### Operating window

In the operating window (Figure 4-58) you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.

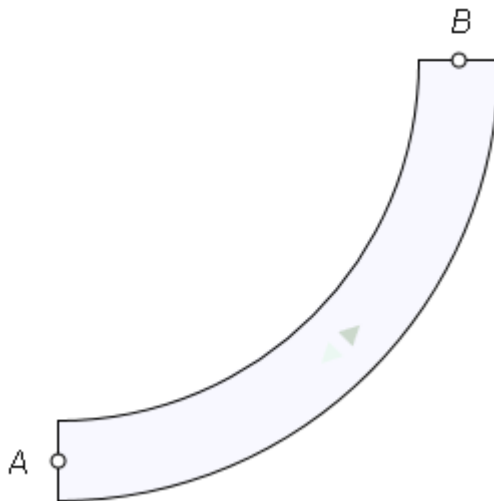


**Figure 4-58:** Operating window for the *ConveyorCurve90-R100* component type



### 4.3.8 *ConveyorCurve90-R200* – 90° curved conveyor (radius 200 pixels)

#### **Symbol**



The grey 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 diagram. For example, a scale of 1 pix : 100 mm gives an effective radius of 20 metres.

The speed at which objects are moved over this conveyor is set at the component's hidden analog input *Speed* as a percentage of the programmable 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 Figure 4-59.

Name		Value
NominalSpeed	[m/s]	2.0

**Figure 4-59:** Parameters for the *ConveyorCurve90-R200* component type

#### **Additional parameters**

The *ConveyorCurve90-R200* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	ConveyorCurve90-R200
HIERARCHY	CONVEYOR

**Figure 4-60:** Additional parameters for the *ConveyorCurve90-R200* component type

### Operating window

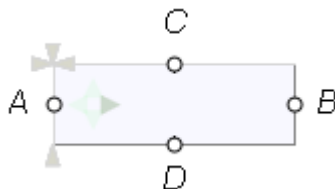
In the operating window (Figure 4-61) you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.



**Figure 4-61:** Operating window for the *ConveyorCurve90-R200* component type

## 4.3.9 90DegreeTransfer

### Symbol



The grey 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 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 set at 40 pixels. The effective width of the 90-degree transfer is therefore determined by the scale set for the diagram.

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 component's hidden analog input *Speed* as a percentage of the programmable 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 programmable 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 behaviour 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 in relation to connector *A* of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor in relation to connector *B* of the symbol.
- *NbrOfSensorsC*  
Number of sensors relative to connector *C* (1 to 4)
- *SensorPositionC*  
Position of the corresponding sensor in relation to connector *C* of the symbol.
- *NbrOfSensorsD*  
Number of sensors relative to connector *D* (1 to 4)
- *SensorPositionD*  
Position of the corresponding sensor in relation 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 colour changes to yellow.

The parameters, their units and default values are shown in Figure 4-62.

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

**Figure 4-62:** Parameters for the *90DegreeTransfer* component type

### Additional parameters

The *90DegreeTransfer* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

You also need to specify how many objects can be present on the conveyor section at any one time.

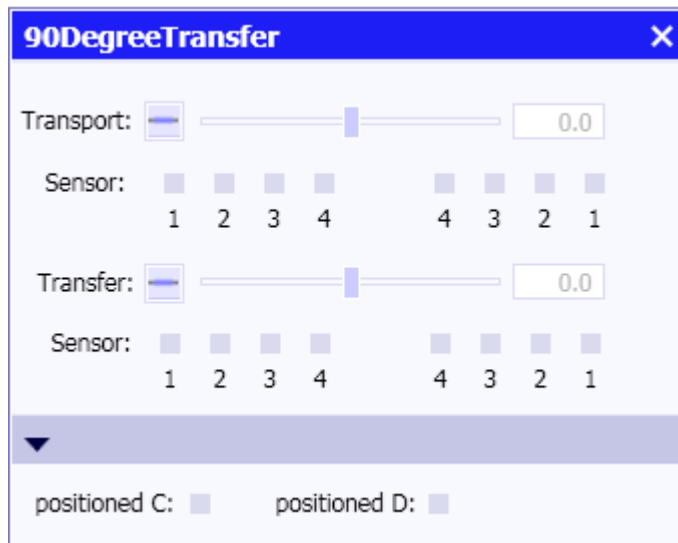
Name	Value
MaxObjects	8
TEMPLATE	90DegreeTransfer
HIERARCHY	CONVEYOR

**Figure 4-63:** Additional parameters for the *90DegreeTransfer* component type

### Operating window

In the operating window (Figure 4-64) 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.

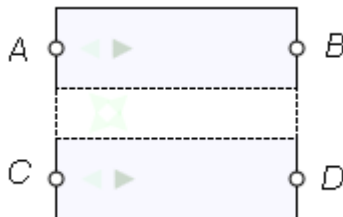
The extended operating window shows whether the object is close enough to connector C or D to be transferred.



**Figure 4-64:** Operating window for the *90DegreeTransfer* component type

### 4.3.10 CrossOver

#### Symbol



The two grey 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 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 component's hidden analog input *Speed* as a percentage of the programmable 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), then 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), then 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 Figure 4-65.

Name	Value
NominalSpeed	[m/s] 2.0

**Figure 4-65:** Parameters for the *CrossOver* component type

### Additional parameters

The *CrossOver* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

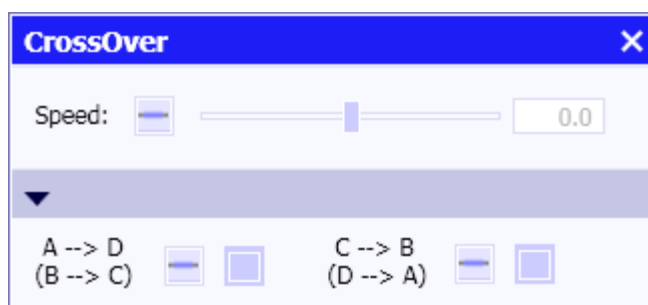
Name	Value
TEMPLATE	CrossOver
HIERARCHY	CONVEYOR

**Figure 4-66:** Additional parameters for the *CrossOver* component type

### Operating window

In the operating window (Figure 4-67) 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 in order to set the transport direction of the crossover manually.



**Figure 4-67:** Operating window for the *CrossOver* component type

## 4.3.11 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 extension components of the *LifterExtension* type. The base component simulates the lifter in the base

level, while each additional level is simulated by an extension 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 extension components on the same diagram. You can create a separate diagram for each level, for example, and distribute the lifter components over the individual diagrams.

The following points apply to the programming 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 extension components used.
- In the base component you have to specify the position (*LevelPosition*) for each level, i.e. the level above the base level, in millimetres. 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 setting the parameters for the extension components:

- The name of the extension 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 indicator. The number indicating the level starts with 1.
- The *BaseName* parameter is the name of the base component.
- The length of the lifter is determined by the width of the base component.

Therefore all extension components must be exactly the same width as the base component.



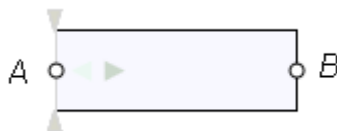
#### **NOTE**

Please note, all extension components must be exactly the same width as the base component. Otherwise, when starting the simulation a message will be shown pointing to this inconsistency.

The drives and the lifter sensors are fully implemented in the base component. Therefore extension components do not work without the base component.

#### **4.3.11.1 LifterBase – Lifter (base component)**

##### **Symbol**



The grey arrow head in the symbol indicates the reference direction for movement of the vehicles. 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 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 component's hidden analog input *Speed* as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter). Similarly, the lifting speed, i.e. 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 programmable nominal lifting speed (*NominalLifterSpeed* parameter).

In order to be able 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 one and four 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*.



### CAUTION

The base component has further hidden inputs for exchanging signals with extension components. The corresponding signals are assigned to these inputs. Do not change these default settings, otherwise the lifter components will not behave in the intended way in the simulation.

## Parameters

The behaviour of the component can be programmed.

- *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 in relation to connector  $A$  of the symbol
- *NbrOfSensorsB*  
Number of sensors relative to connector  $B$  (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor in relation 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 colour changes to yellow.

The parameters, their units and default values are shown in Figure 4-68.

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

**Figure 4-68:** Parameters for the *LifterBase* component type

### Additional parameters

The *LifterBase* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

You also need to specify how many objects can be present on this conveyor section at any one time.

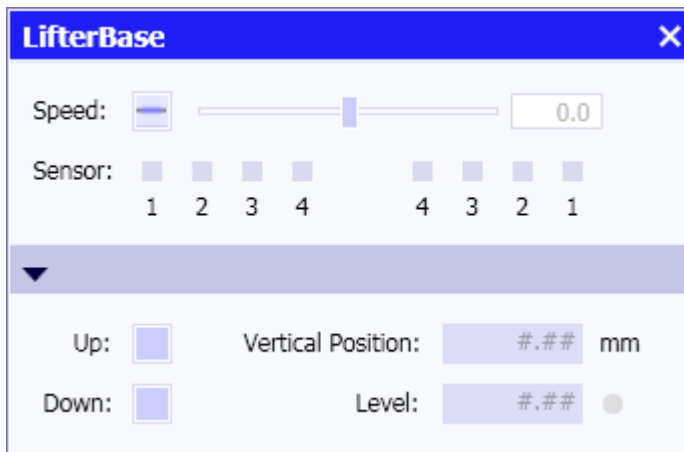
Name	Value
MaxObjects	8
TEMPLATE	LifterBase
HIERARCHY	CONVEYOR

**Figure 4-69:** Additional parameters for the *LifterBase* component type

### Operating window

In the operating window (Figure 4-70) 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.

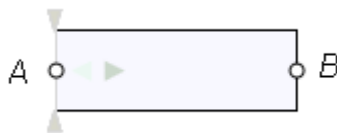
In the extended operating window you can move the lifter up or down by means of buttons. The current lifter position and level are displayed.



**Figure 4-70:** Operating window for the *LifterBase* component type

#### 4.3.11.2 *LifterExtension* – Lifter (extension component)

##### Symbol



The grey 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 lifter rail.



##### **CAUTION**

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 behaviour of the component can be programmed:

- *Level*  
Level at which the component is located. The numbers start at 1.
- *BaseName*  
Name of the corresponding base component

The parameters, their units and default values are shown in Figure 4-71.

Name	Value
BaseName	
Level	1

**Figure 4-71:** Parameters for the *LifterExtension* component type

### **Additional parameters**

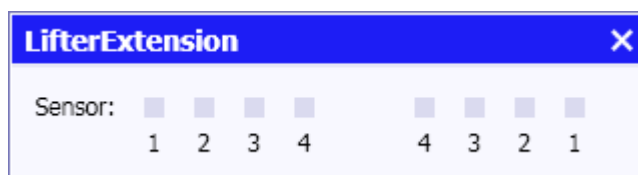
You have to specify how many vehicles can be present on the section defined by the component at any one time. This parameter should be set to the same value as for the base component.

Name	Value
MaxObjects	8

**Figure 4-72:** Additional parameters for the *LifterExtension* component type

### **Operating window**

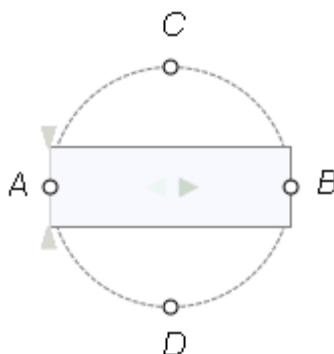
In the operating window (Figure 4-73) you can only monitor the status of the maximum eight sensors.



**Figure 4-73:** Operating window for the *LifterExtension* component type

## **4.3.12 Turntable-R60**

### **Symbol**



The grey 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 component's hidden analog input *Speed* as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter). Correspondingly, the rotation speed is set at the hidden analog input *RotationSpeed* as a percentage of the programmable nominal rotation speed (*NominalRotationSpeed* parameter).

The component symbol cannot be scaled. The radius of the turntable is set to 60 pixels and cannot be changed. Correspondingly, the length of the turntable is set to 120 pixels. The effective turntable length of the component is determined by the scale set for the diagram. For example, a scale of 1 pix : 100 mm gives an effective length of 12 metres.

Between one and four 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 behaviour 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 in order for the object to be transferred.
- *NbrOfSensorsA*  
Number of sensors relative to connector *A* (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor in relation to connector *A* of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor in relation to connector *B* of the symbol.

The parameters, their units and default values are shown in Figure 4-74.

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

**Figure 4-74:** Parameters for the *Turntable-R60* component type

### Additional parameters

The *Turntable-R60* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

You also need to specify how many objects can be present on the turntable at any one time.

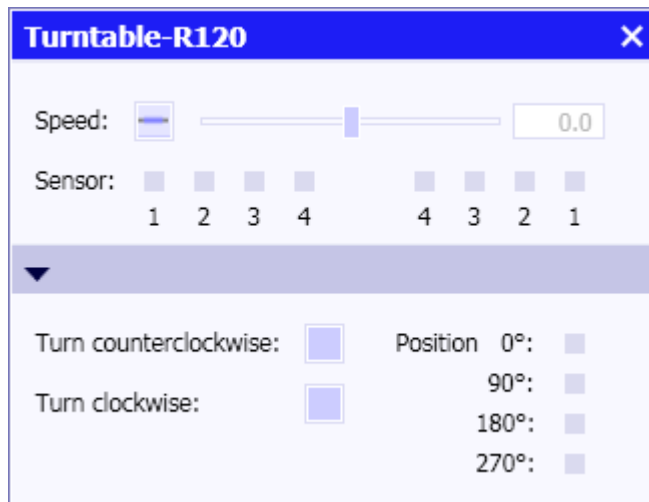
Name	Value
MaxObjects	8
TEMPLATE	Turntable-R60
HIERARCHY	CONVEYOR

**Figure 4-75:** Additional parameters for the *Turntable-R60* component type

### Operating window

In the operating window (Figure 4-76) 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.

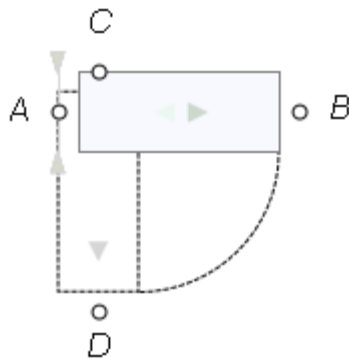
In the extended operating window you can rotate the turntable manually by 90° in both directions by means of buttons. The end position in each case is displayed.



**Figure 4-76:** Operating window for the *Turntable-R60* component type

### 4.3.13 Swiveltable-R120 – Swivel table

#### Symbol



The grey 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 component's hidden analog input *Speed* as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter). Correspondingly, the rotation speed is set at the hidden analog input *RotationSpeed* as a percentage of the programmable nominal rotation speed (*NominalRotationSpeed* parameter).

The component symbol cannot be scaled. The radius of the swivel table, i.e. the length of the swivel table, is set to 120 pixels and cannot be changed. The effective swivel table length of the component is determined by the scale set for the diagram. For example, a scale of 1 pix : 100 mm gives an effective length of 12 metres.

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 behaviour 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 in order for the object to be transferred.
- *NbrOfSensorsA*  
Number of sensors relative to connector *A* (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor in relation to connector *A* of the symbol.
- *NbrOfSensorsB*  
Number of sensors relative to connector *B* (1 to 4)
- *SensorPositionB*  
Position of the corresponding sensor in relation to connector *B* of the symbol.

The parameters, their units and default values are shown in Figure 4-77.

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

**Figure 4-77:** Parameters for the *Swiveltable-R120* component type

### Additional parameters

The *Swiveltable-R120* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

You also need to specify how many objects can be present on the swivel table at any one time.

Name	Value
MaxObjects	8
TEMPLATE	Swiveltable-R120
HIERARCHY	CONVEYOR

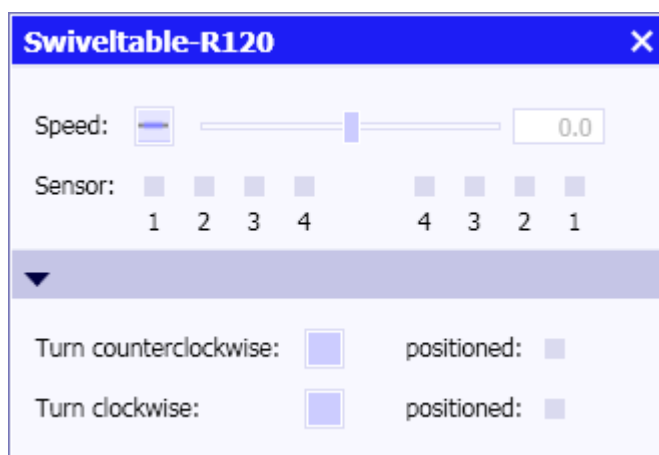
**Figure 4-78:** Additional parameters for the *Swiveltable-R120* component type

### 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 (Figure 4-79) 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.

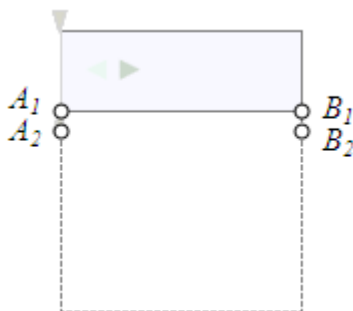
In the extended operating window you can rotate the swivel table manually through 90° by means of a button. Another button allows you to rotate it back to its original position. Indicators show when it has reached the end position.



**Figure 4-79:** Operating window for the *Swiveltable-R120* component type

### 4.3.14 TransferCarriage

#### Symbol





The height and width of this component are scalable. The width sets the length of the transfer carriage, the height the length of the transfer path.

The grey 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 can be programmed. Up to 16 connectors on each side are possible. The default setting is 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 component's hidden analog input *Speed* as a percentage of the programmable nominal conveyor speed (*NominalSpeed* parameter). Correspondingly, the transfer speed is set at the hidden analog input *TransferSpeed* as a percentage of the programmable 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 behaviour 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 in order for the object to be transferred.
- *NbrOfConnectorsA*  
Number of connectors on the left-hand side (A)
- *NbrOfConnectorsB*  
Number of connectors on the right-hand side (B)
- *NbrOfSensorsA*  
Number of sensors relative to connector side A (1 to 4)
- *SensorPositionA*  
Position of the corresponding sensor in relation 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 in relation to connector side *B* of the symbol.

The parameters, their units and default values are shown in Figure 4-80.

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

**Figure 4-80:** Parameters for the *TransferCarriage* component type

### Additional parameters

The *TransferCarriage* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

You also need to specify how many objects can be present on the transfer carriage at any one time.

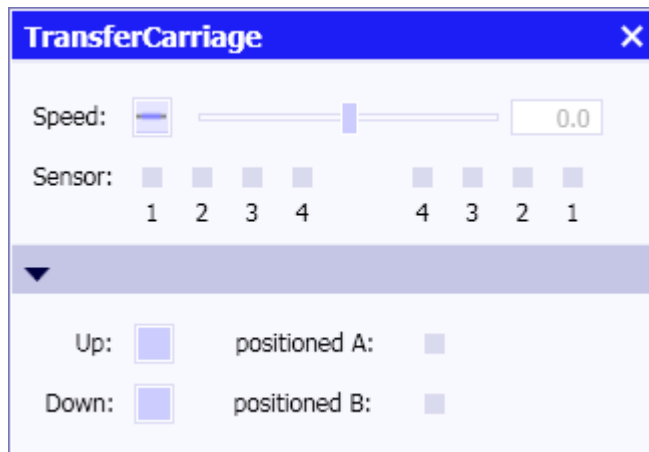
Name	Value
MaxObjects	8
TEMPLATE	TransferCarriage
HIERARCHY	CONVEYOR

**Figure 4-81:** Additional parameters for the *TransferCarriage* component type

### Operating window

In the operating window (Figure 4-82) 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 move the transfer carriage manually in both directions to the next position by means of buttons (*Up*, *Down*). The end position in each case is displayed. Indicators show when the transfer carriage has reached a connector position on one or other side.



**Figure 4-82:** Operating window for the *TransferCarriage* component type

### 4.3.15 *SpurConveyor-1* – Diagonal feed with one spur

#### **Symbol**



The grey 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 *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 component's hidden analog input *Speed* as a percentage of the programmable 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 Figure 4-83.

Name	Value
NominalSpeed	[m/s] 2.0

**Figure 4-83:** Parameters for the *SpurConveyor-1* component type

### Additional parameters

The *SpurConveyor-1* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

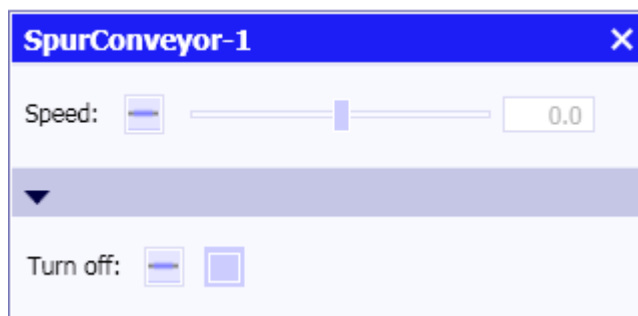
Name	Value
TEMPLATE	SpurConveyor-1
HIERARCHY	CONVEYOR

**Figure 4-84:** Additional parameters for the *SpurConveyor-1* component type

### Operating window

In the operating window (Figure 4-85) you can set the conveyor speed as a percentage of the nominal conveyor speed (*NominalSpeed*) using a slider.

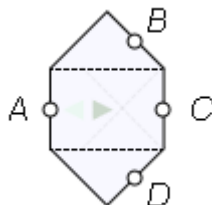
In the extended operating window you can change to the spur section by means of a switch.



**Figure 4-85:** Operating window for the *SpurConveyor-1* component type

## 4.3.16 *SpurConveyor-2* – Diagonal feed with two spurs

### Symbol



The grey 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 *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 component's hidden analog input *Speed* as a percentage of the programmable 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 both inputs is set, setting the other one is without effect. There will be no switching to a 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 Figure 4-86.

Name	Value
NominalSpeed	[m/s] 2.0

**Figure 4-86:** Parameters for the *SpurConveyor-2* component type

### Additional parameters

The *SpurConveyor-2* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	SpurConveyor-2
HIERARCHY	CONVEYOR

**Figure 4-87:** Additional parameters for the *SpurConveyor-2* component type

### Operating window

In the operating window (Figure 4-88) 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.



**Figure 4-88:** Operating window for the *SpurConveyor-2* component type

## 4.4 Component types for simulating objects

The *MATERIAL* directory (Figure 4-89) 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.

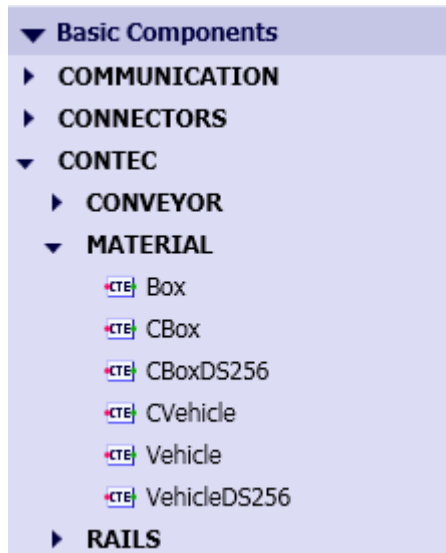


Figure 4-89: *MATERIAL* library directory

These component types are used to create material lists – as described in section 3.5.2. The size of a component (width and height of the symbol) is defined in the material list in millimetres.

### 4.4.1 *Box* – Simple object

#### *Symbol*



#### *Function*

The *Box* component type is used to simulate an object with a rectangular outline, such as a pallet or case. 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).

### 4.4.2 *CBox* – Coloured object









#### *Symbol*



**Function**

The *Box* component type is used to simulate an object with a rectangular outline, such as a pallet or case. 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 eight different colours. The colour is determined from the component's three binary inputs *R*, *G* and *B*, by mixing together the three primary colours red, green and blue (see Table 4-2).

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	







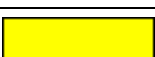
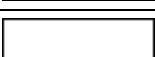
**Table 4-2:** Colour chart for *CBox*

#### 4.4.3 *CBoxDS256* – Coloured 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 case. 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 eight different colours. The colour is determined from the component's three binary inputs *R*, *G* and *B*, by mixing together the three primary colours red, green and blue (see Table 4-3).

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	

**Table 4-3:** Colour chart for *CBoxDS256*

In addition, a data store with a programmable size (*SizeOfStorage* parameter) is defined for components of this type. This store 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 store are located in the *SENSORS* library in the *RFID* directory.

### Parameters

The size of the data store can be set by means of the *SizeOfStorage* parameter. The default setting is 1 (Figure 4-90).

Name	Value
SizeOfStorage	1

**Figure 4-90:** Parameters for the *CBoxDS256* component type

## 4.4.4 Vehicle

### Symbol



### Function

The *Vehicle* component type is used for simulating vehicles on electric overhead monorail systems, for example. The vehicle is assumed to have a square 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



component's hidden analog input *Speed* as a percentage of the programmable 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 behaviour 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 Figure 4-91.

Name		Value
NominalSpeed	[m/s]	2.0
StartUpDelay	[s]	0.0
SensorRange	[%]	100.0

**Figure 4-91:** Parameters for the *Vehicle* component type

### Additional parameters

The *Vehicle* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name	Value
TEMPLATE	Vehicle
HIERARCHY	VEHICLES

**Figure 4-92:** Additional parameters for the *Vehicle* component type

## 4.4.5 CVehicle

### Symbol











### Function

The *CVehicle* component type is used for simulating vehicles on electric overhead monorail systems, for example. 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 component's hidden analog input *Speed* as a percentage of the programmable 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 eight different colours. The colour is determined from the component's three binary inputs *R*, *G* and *B*, by mixing together the three primary colours red, green and blue (see Table 4-4).

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	

**Table 4-4:** Colour chart for *CVehicle*

### Parameters

The behaviour 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 Figure 4-93.

Name		Value
NominalSpeed	[m/s]	2.0
StartUpDelay	[s]	0.0
SensorRange	[%]	100.0

**Figure 4-93:** Parameters for the *CVehicle* component type

### ***Additional parameters***

The *CVehicle* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name		Value
TEMPLATE		CVehicle
HIERARCHY		VEHICLES

**Figure 4-94:** Additional parameters for the *CVehicle* component type

## **4.4.6 VehicleDS256 – Vehicle with data storage**

### ***Symbol***



### ***Function***

The *VehicleDS256* component type is used for simulating vehicles on electric overhead monorail systems, for example. 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 component's hidden analog input *Speed* as a percentage of the programmable 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 store with a programmable size (*SizeOfStorage* parameter) is defined for components of this type. This store 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 store are located in the *SENSORS* library in the *RFID* directory.

### Parameters

The behaviour 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 store

The parameters, their units and default values are shown in Figure 4-95.

Name		Value
NominalSpeed	[m/s]	2.0
StartUpDelay	[s]	0.0
SensorRange	[%]	100.0
SizeOfStorage		1

**Figure 4-95:** Parameters for the *VehicleDS256* component type

### Additional parameters

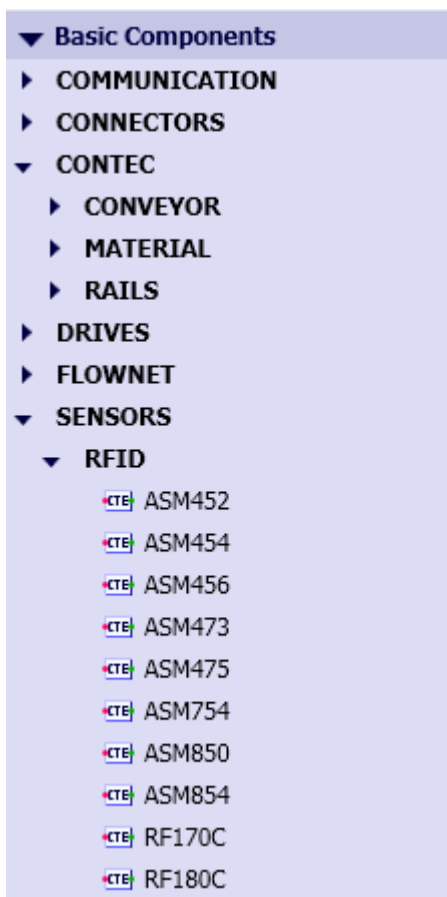
The *VehicleDS256* component includes the additional parameters *TEMPLATE* and *HIERARCHY*, which can be used to control generation of the drive level (see section 3.6.2).

Name		Value
TEMPLATE		VehicleDS256
HIERARCHY		VEHICLES

**Figure 4-96:** Additional parameters for the *VehicleDS256* component type

## 4.5 Component types for simulating identification systems

The *RFID* directory (Figure 4-97) 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. With components of these types it is possible to exchange data between the controller and the corresponding simulation components of objects (see section 4.4). Both fixed data such as barcodes and write/read data in the case of a Moby can be exchanged.



**Figure 4-97:** *RFID* library directory

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 gateways that support both methods of communication. These components can therefore only be used with the Profibus DP and Profinet IO gateway. Table 4-5 shows the interface modules supported for Profibus DP, while Table 4-6 shows the interface modules supported for Profinet IO.

Interface module	MLFB	Head end supported for modular slaves
<b>ASM 452</b>	6GT2002-0EB20	
<b>ASM 454</b>	6GT2002-2EE00	
<b>ASM 456</b>	6GT2002-0ED00	
<b>ASM 473</b>	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)
<b>ASM 475</b>	6GT2002-0GA10	ET200M 6ES7 153-2BA00-0XB0 (801E) ET200M 6ES7 153-2BB00-0XB0 (8071)
<b>ASM 754</b>	6GT2302-2EE00	
<b>ASM 850</b>	6GT2402-2EA00	
<b>ASM 854</b>	6GT2402-2BB00	
<b>RF170C</b>	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 4-5:** Interface modules for Profibus DP

Interface module	MLFB	Head end supported for modular slaves
<b>ASM 475</b>	6GT2002-0GA10	ET200M 6ES7 153-4BA00-0XB0 ET200M 6ES7 153-4AA00-0XB0
<b>RF170C</b>	6GT2002-0HD00	ET200pro 6ES7 154-4AB10-0AB0 ET200pro 6ES7 154-6AB00-0AB0 ET200pro 6ES7 154-6AB50-0AB0

**Table 4-6:** Interface modules for Profinet IO

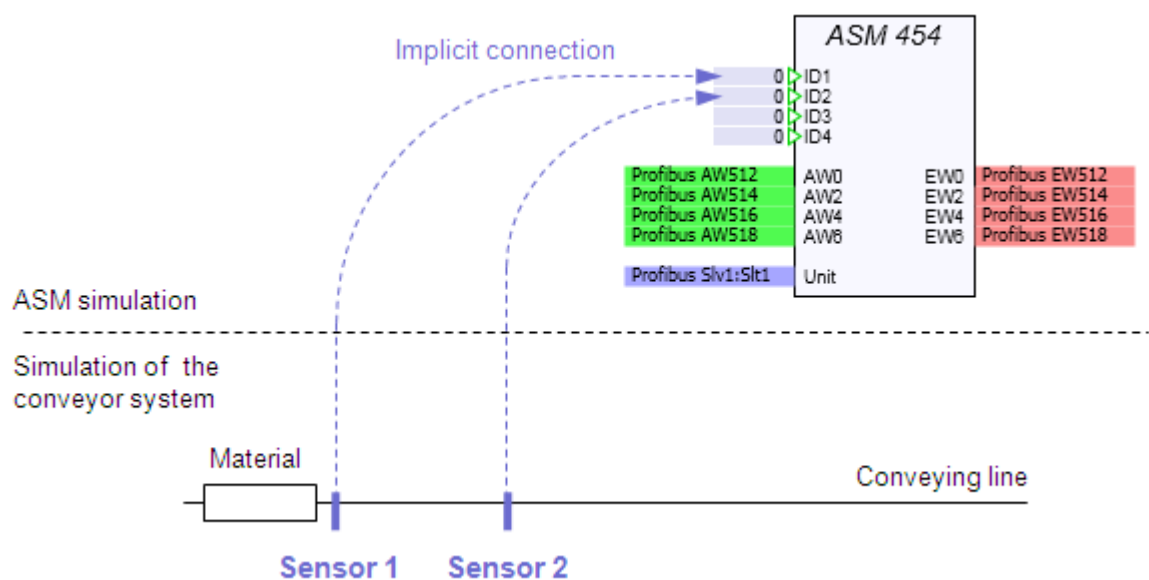
### 4.5.1 Operating principle

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 store (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 a data store. In order to be able to

exchange the contents of a simulated data store 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's data store.

In SIMIT the read/write device (SLG) that is actually connected to the ASM and its behaviour is not simulated; instead its functionality in terms of converting the data from the mobile data stores in the ASM component types is simulated. As shown schematically in Figure 4-98, 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.



**Figure 4-98:** Correlation between conveyor technology simulation and simulation of interface modules

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 store 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 so are not supported in the simulation.

The RFID components support the FB45 commands listed in Table 4-7.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises 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 MDS

**Table 4-7:** Supported FB45 commands



### CAUTION

Not every RFID component type supports all the commands listed in Table 4-7. A description of the commands supported by an RFID component type can be found in the description of that component type.

A mobile data store (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 of which MDS is to be read or written to. However, for the component types of objects with MDS provided in the CONTEC library (component types *CBoxDS256*, *VehicleDS256*), only one MDS is modelled. As it has the name "*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 (see Table 4-8).

Command	Code	Description
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	- Error accessing data in the MDS - Range violation during MDS access
Antenna error:	0x1C	- Write/read commands with antenna switched off - Wrong antenna command (2x On/2x Off)

**Table 4-8:** Possible error codes of RFID components

Signals are exchanged between RFID components and the controller both via input and output words from the gateway 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. It can be copied to the diagram from the Hardware



view in the Properties dialog of the Profibus or Profinet gateway using drag and drop and connected to the ASM component. To configure the Unit connector manually, use the notation "SlvX:SlY" for Profibus DP, where X denotes the corresponding slave and Y the corresponding module, and "DevX:SlY" for Profinet IO, where X denotes the device and Y the slot.

Figure 4-99 shows an example of a suitable hardware configuration for display in Figure 4-98, with a four-channel interface module.

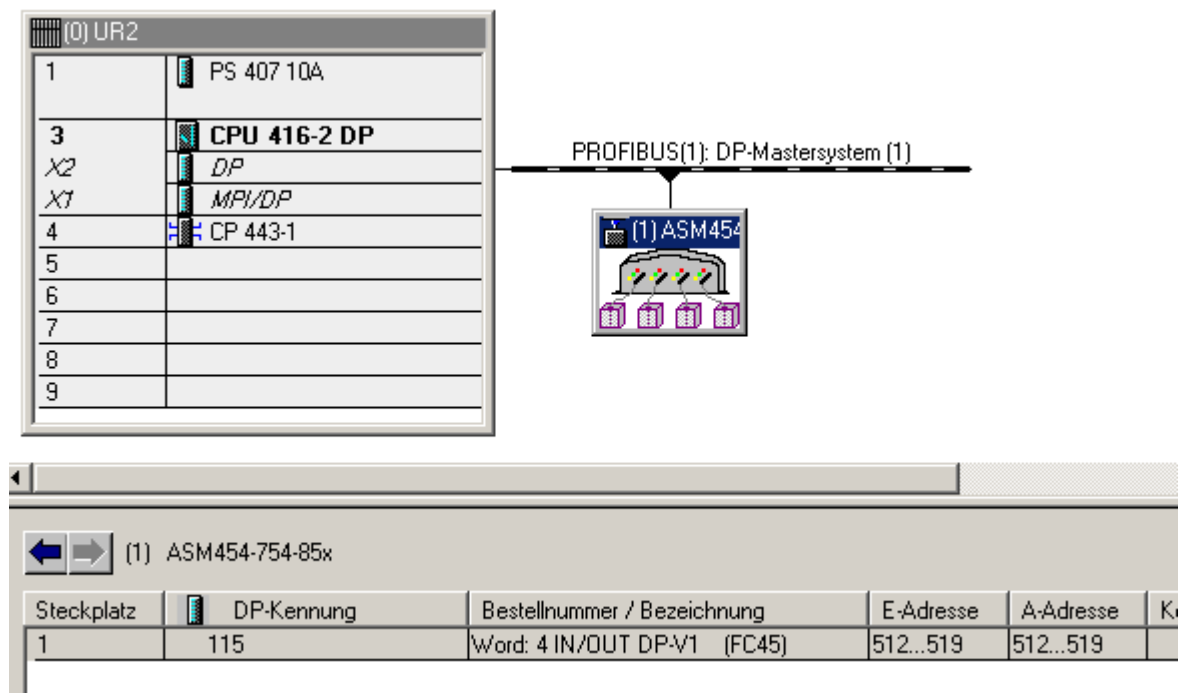
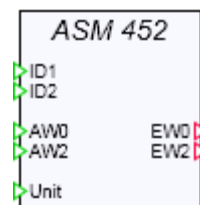


Figure 4-99: ASM454 in the HW config view

## 4.5.2 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 Table 4-9 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are linked to the Profibus DP gateway via the corresponding input/output signals. The *ID1* and

*ID2* inputs should be implicitly connected to the sensors of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises 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 MDS


**Table 4-9:** Supported *ASM452* commands

### 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 initialised in the ASM at the start of simulation: True:= Antenna on, False:= Antenna off.

Figure 4-100 shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True 

**Figure 4-100:** Parameters for the *ASM452* component type

### Additional parameters

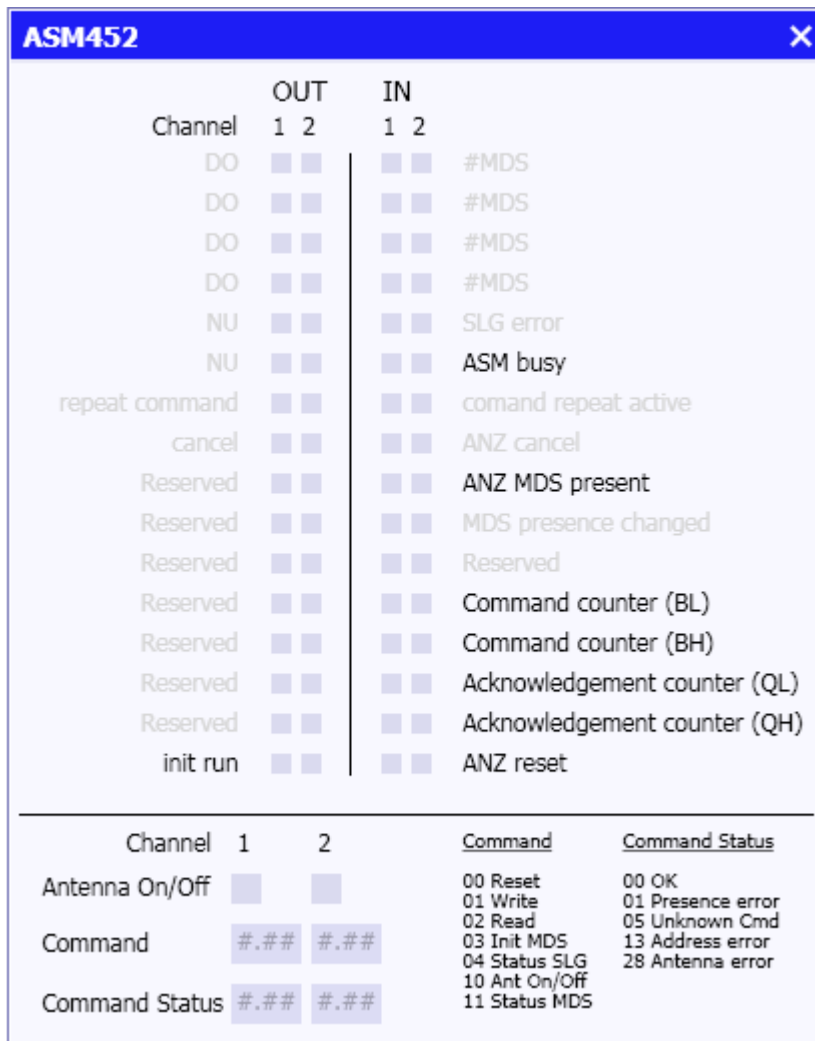
The additional parameters comprise status values corresponding to the UDT110 as online-editable parameters for each of the two simulated read/write devices (SLGs) (see Figure 4-101).

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

**Figure 4-101:** Additional parameters for the *ASM452* component type

### ***Operating window***

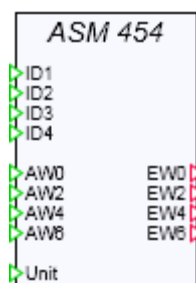
The operating window (Figure 4-102) shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in grey are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for both channels.



**Figure 4-102:** Operating window for the ASM452 component type

### 4.5.3 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 Table 4-10 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are linked to the Profibus DP gateway via the corresponding input/output signals. The *ID1* to *ID4* inputs should be implicitly connected to the sensors of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises the data area of an MDS

**Table 4-10:** Supported ASM454 commands

### Parameters

Parameters *MDS1* to *MDS4* indicate which area of the MDS should be accessed by object components via sensor *ID1* to *ID4*. Figure 4-103 shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
MDS3	MDS
MDS4	MDS

**Figure 4-103:** Parameters for the ASM454 component type

### Operating window

The operating window (Figure 4-104) shows the current states of the input and output words of the four channels that are accessed by FB45. The bits shown in grey 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.

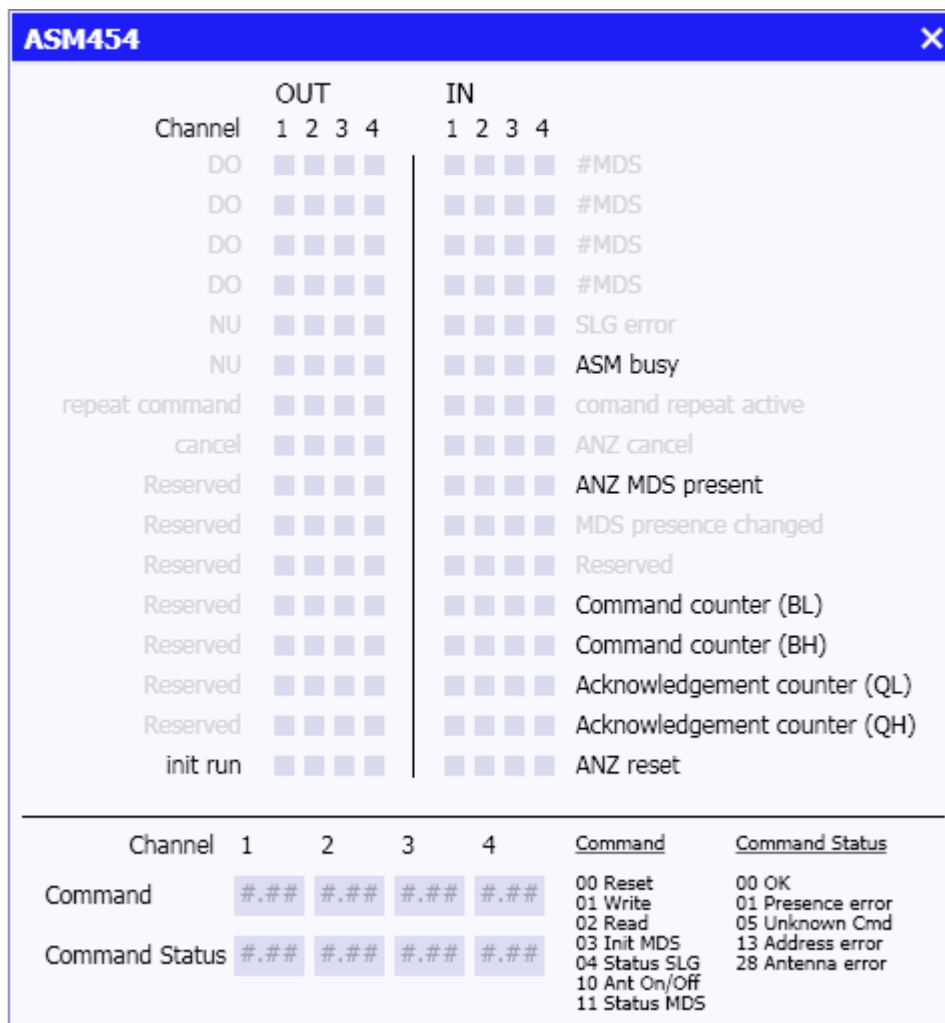
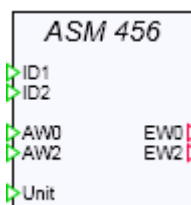


Figure 4-104: Operating window for the ASM454 component type

#### 4.5.4 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 Table 4-11 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are linked to the Profibus DP gateway via the corresponding input/output signals. The *ID1* and

*ID2* inputs should be implicitly connected to the sensors of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises 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 MDS

**Table 4-11:** Supported *ASM456* commands

### 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 initialised in the ASM at the start of simulation: True:= Antenna on, False:= Antenna off.

Figure 4-105 shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True

**Figure 4-105:** Parameters for the *ASM456* component type

### Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online-editable parameters for each of the two simulated read/write devices (SLGs) (see Figure 4-106).

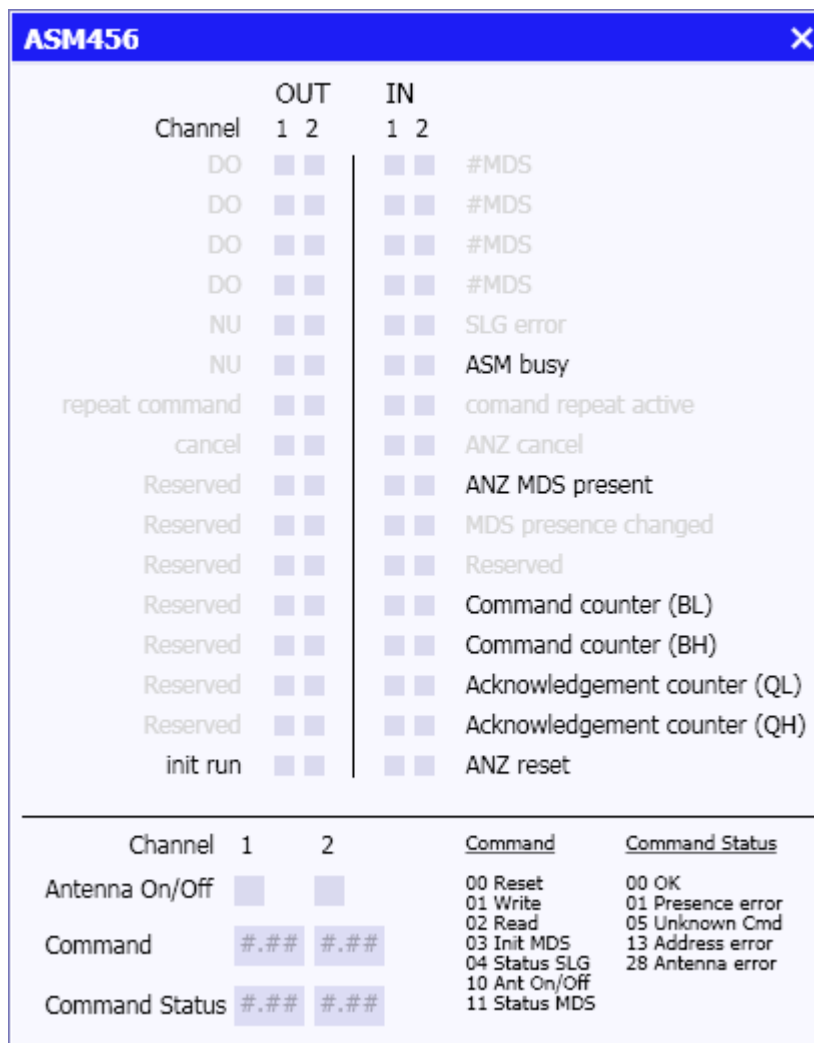
Name	Value
SLG1_UDT110_hardware	0
SLG1_UDT110_hardware_version	0
SLG1_UDT110_loader_version	0
SLG1_UDT110_firmware	0
SLG1_UDT110_firmware_version	0
SLG1_UDT110_driver	0
SLG1_UDT110_driver_version	0
SLG1_UDT110_interface	0
SLG1_UDT110_baud	0
SLG1_UDT110_reserved1	0
SLG1_UDT110_reserved2	0
SLG1_UDT110_reserved3	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

**Figure 4-106:** Additional parameters for the *ASM456* component type

### ***Operating window***

The operating window (Figure 4-107) shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in grey 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.

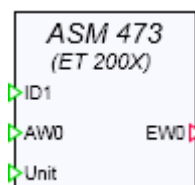




**Figure 4-107:** Operating window for the ASM456 component type

### 4.5.5 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 Table 4-12 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are linked to the Profibus DP gateway via

the corresponding input/output signal. The *ID1* input should be implicitly connected to the sensor of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises 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 MDS

**Table 4-12:** Supported *ASM473* commands

### 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 initialised in the ASM at the start of simulation: True:= Antenna on, False:= Antenna off.

Figure 4-108 shows the parameters and their default settings.

Name	Value
MDS1	MDS
AntennaInitStatus	True

**Figure 4-108:** Parameters for the *ASM473* component type

### Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online-editable parameters for the simulated read/write device (SLG) (see Figure 4-110).

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

**Figure 4-109:** Status values for the read/write device in the *ASM473*

### ***Operating window***

The operating window (Figure 4-110) shows the current states of the input and output word that are accessed by FB45. The bits shown in grey 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.

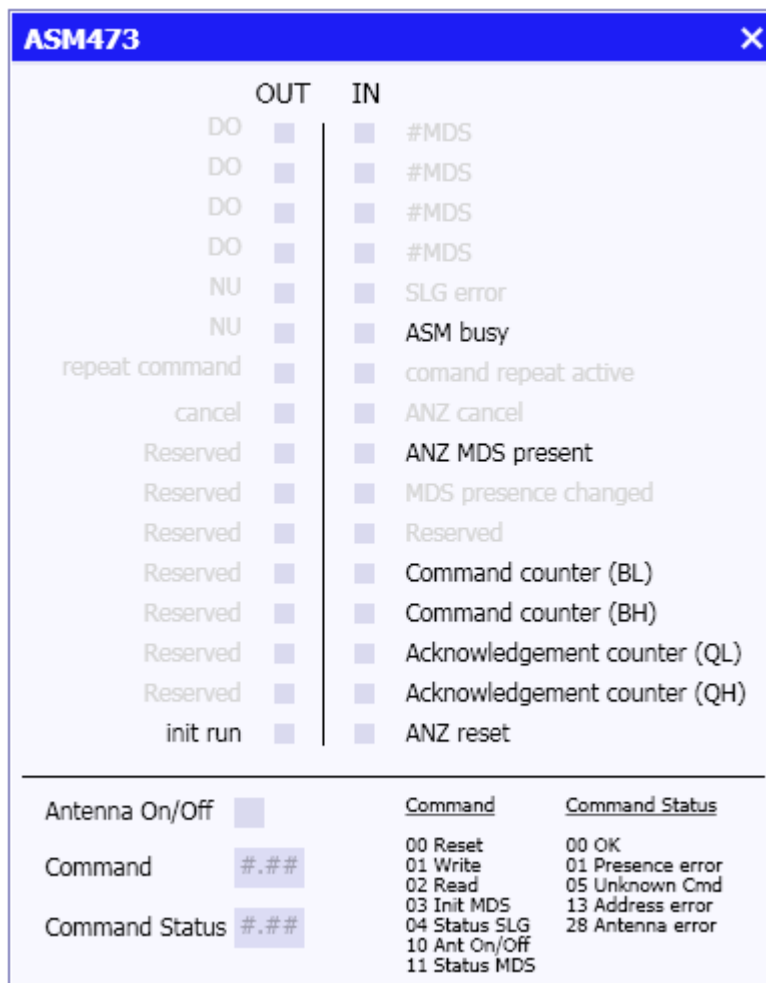
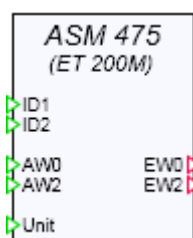


Figure 4-110: Operating window for the ASM473 component type

## 4.5.6 ASM 475 – 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 Table 4-13 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are linked to the Profibus DP gateway via the corresponding input/output signals. The *ID1* and

*ID2* inputs should be implicitly connected to the sensors of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises 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 MDS

**Table 4-13:** Supported ASM475 commands

### 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 initialised in the ASM at the start of simulation: True:= Antenna on, False:= Antenna off.

Figure 4-111 shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True

**Figure 4-111:** Parameters for the ASM475 component type

### Additional parameters

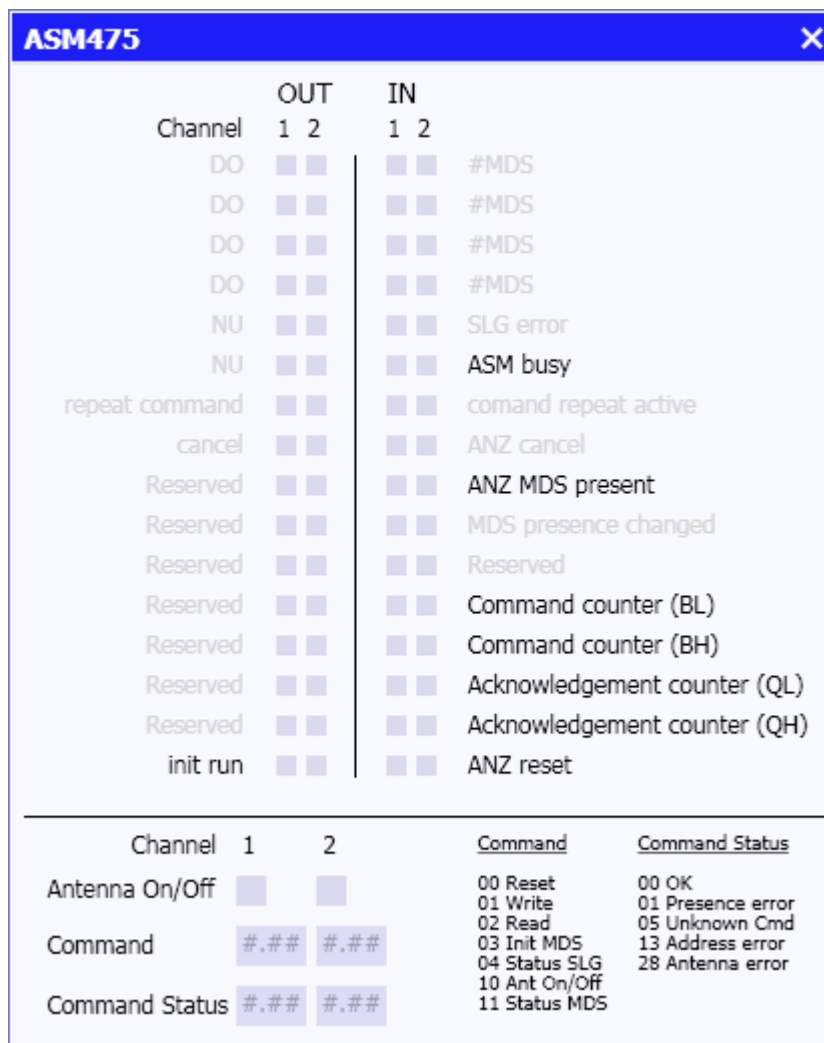
The additional parameters comprise status values corresponding to the UDT110 as online-editable parameters for each of the two simulated read/write devices (SLGs) (see Figure 4-112).

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

**Figure 4-112:** Additional parameters for the *ASM475* component type

### ***Operating window***

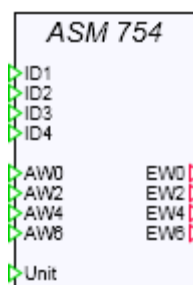
The operating window (Figure 4-113) shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in grey are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for both channels.



**Figure 4-113:** Operating window for the ASM475 component type

#### 4.5.7 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 Table 4-14 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are linked to the Profibus DP gateway via the corresponding input/output signals. The *ID1* to *ID4* inputs should be implicitly connected to the sensors of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises the data area of an MDS

**Table 4-14:** Supported *ASM754* commands

### Parameters

Parameters *MDS1* to *MDS4* indicate which memory area of the MDS should be accessed by object components via sensor *ID1* to *ID4*. Figure 4-114 shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
MDS3	MDS
MDS4	MDS

**Figure 4-114:** Parameters for the *ASM754* component type

### Operating window

The operating window (Figure 4-115) shows the current states of the input and output words of the four channels that are accessed by FB45. The bits shown in grey 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.



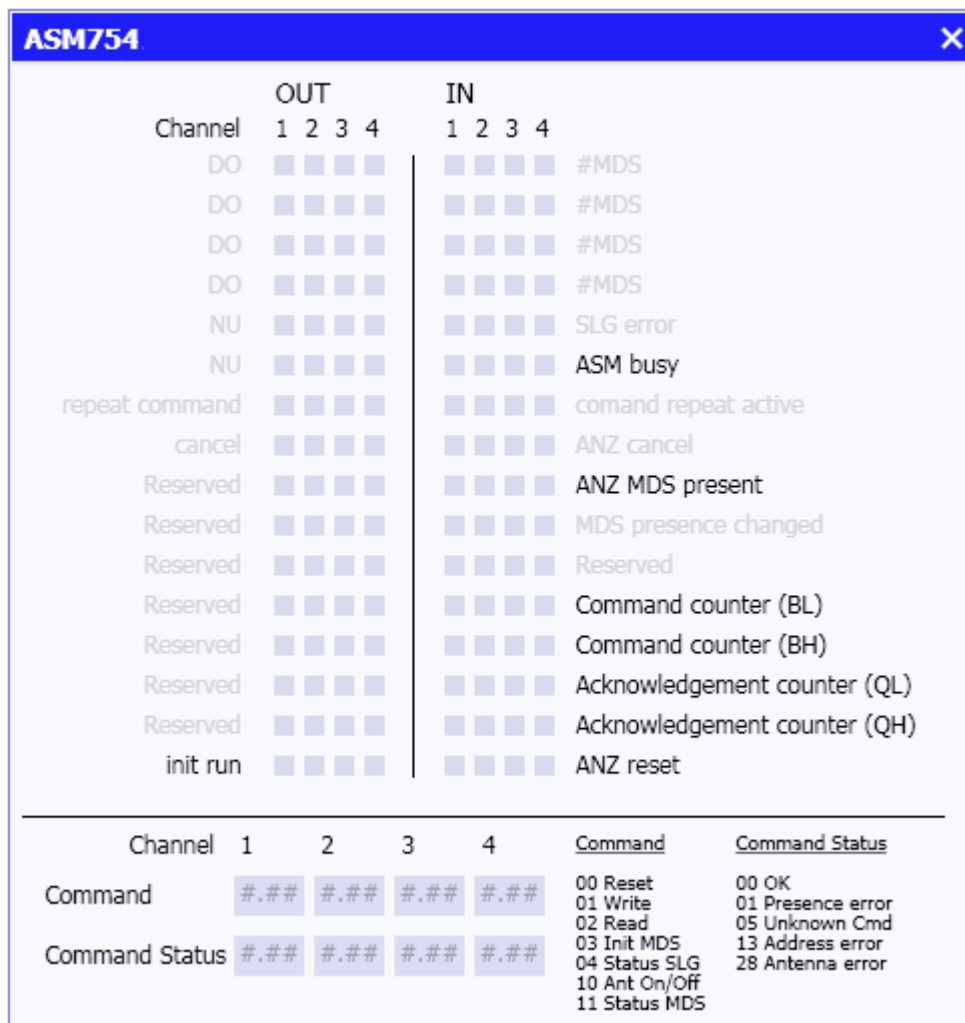
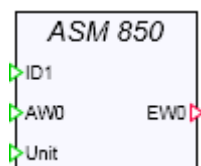


Figure 4-115: Operating window for the ASM754 component type

## 4.5.8 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 Table 4-9 are supported.

All inputs and outputs of the component types are integers. The ASM's slave and module address is set at the *Unit* input using the Unit connector. The cyclical data is exchanged with the controller via the *AWD*, *EWO* input/output. They are linked to the Profibus DP gateway via the corresponding input/output signal. The *ID1* input should be implicitly connected to the sensor of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises the data area of an MDS
Antenna On/Off	0x0A	Switches the antenna on/off (if present)


**Table 4-15:** Supported *ASM850* commands

### 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 initialised in the ASM at the start of simulation: True:= Antenna on, False:= Antenna off.

Figure 4-116 shows the parameters and their default settings.

Name	Value
MDS1	MDS
AntennaInitStatus	True 

**Figure 4-116:** Parameters for the *ASM850* component type

### Operating window

The operating window (Figure 4-117) shows the current states of the input and output word that are accessed by FB45. The bits shown in grey 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.

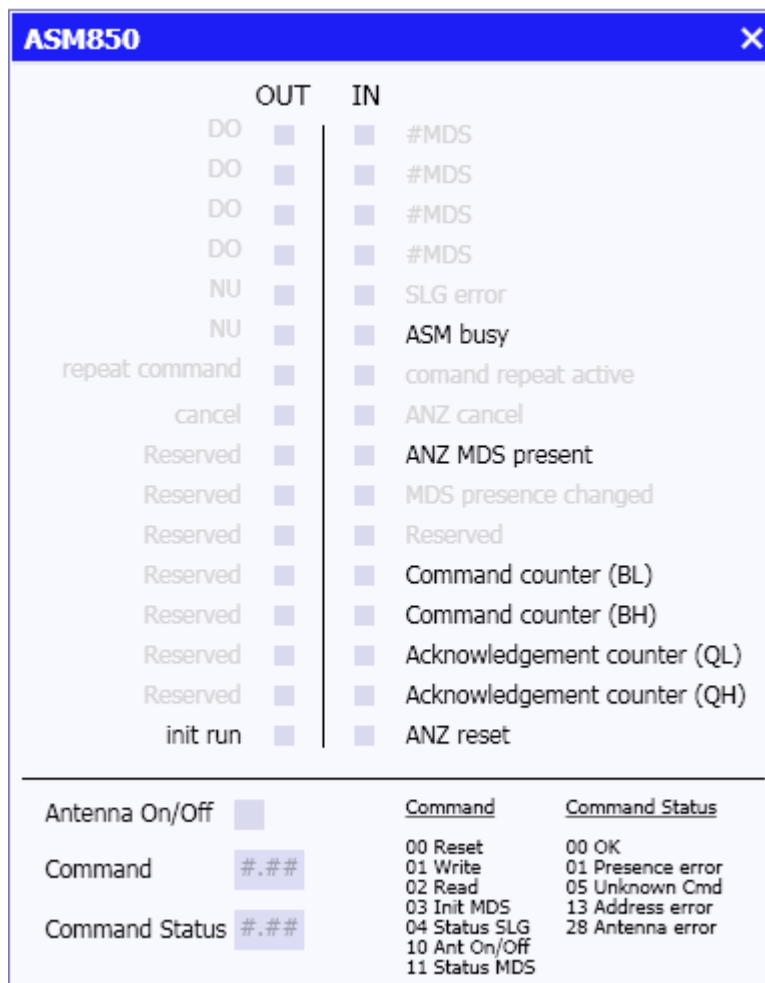
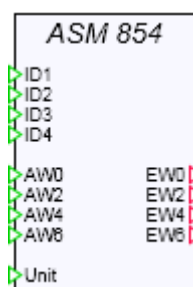


Figure 4-117: Operating window for the ASM850 component type

## 4.5.9 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 Table 4-10 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are

linked to the Profibus DP gateway via the corresponding input/output signals. The *ID1* to *ID4* inputs should be implicitly connected to the sensors of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises the data area of an MDS
Antenna On/Off	0x0A	Switches the antenna on/off (if present)


**Table 4-16:** Supported *ASM854* commands

### 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 initialised in the ASM at the start of simulation: True:= Antenna on, False:= Antenna off.

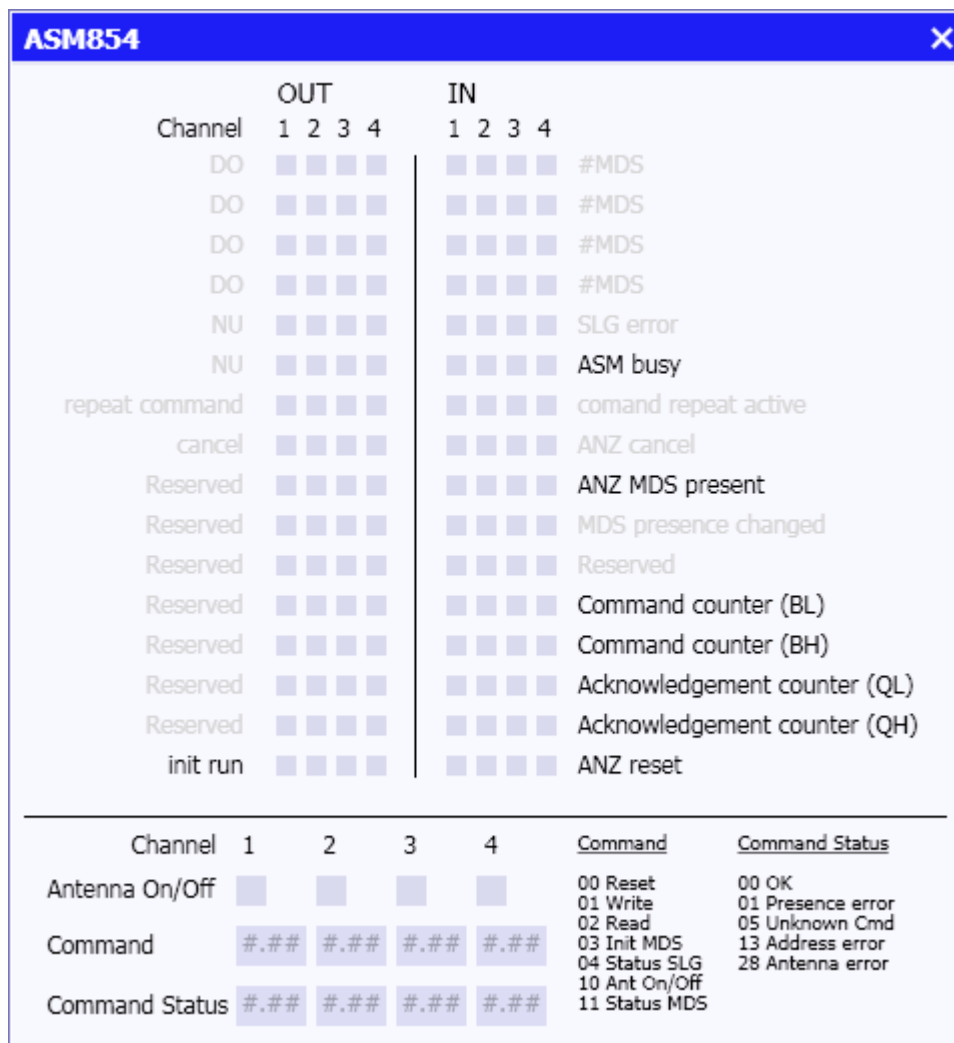
Figure 4-118 shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
MDS3	MDS
MDS4	MDS
AntennaInitStatus	True 

**Figure 4-118:** Parameters for the *ASM854* component type

### Operating window

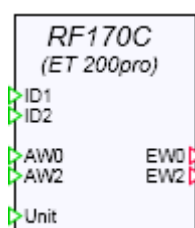
The operating window (Figure 4-119) shows the current states of the input and output words of the four channels that are accessed by FB45. The bits shown in grey are not evaluated by an ASM component. The extended operating window shows the antenna status, the most recently executed command and the associated command status for each of the four channels.



**Figure 4-119:** Operating window for the ASM854 component type

#### 4.5.10 RF 170C – 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 Table 4-17 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are

linked to the Profibus DP gateway via the corresponding input/output signals. The *ID1* and *ID2* inputs should be implicitly connected to the sensors of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises 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 MDS


**Table 4-17:** Supported *RF170C* commands

### 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 initialised in the ASM at the start of simulation: True:= Antenna on, False:= Antenna off.

Figure 4-120 shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True 

**Figure 4-120:** Parameters for the *RF170C* component type

### Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online-editable parameters for each of the two simulated read/write devices (SLGs) (see Figure 4-121).

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

**Figure 4-121:** Additional parameters for the *RF170C* component type

### ***Operating window***

The operating window (Figure 4-122) shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in grey 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.

		OUT		IN		
Channel		1	2	1	2	
DO		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	#MDS
DO		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	#MDS
DO		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	#MDS
DO		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	#MDS
NU		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SLG error
NU		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ASM busy
repeat command		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	comand repeat active
cancel		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ANZ cancel
Reserved		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ANZ MDS present
Reserved		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	MDS presence changed
Reserved		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Reserved
Reserved		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Command counter (BL)
Reserved		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Command counter (BH)
Reserved		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Acknowledgement counter (QL)
Reserved		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Acknowledgement counter (QH)
init run		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ANZ reset

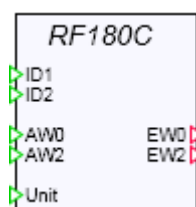
  

Channel	1	2	Command	Command Status
Antenna On/Off	<input type="checkbox"/>	<input type="checkbox"/>	00 Reset	00 OK
Command	#.##	#.##	01 Write	01 Presence error
Command Status	#.##	#.##	02 Read	05 Unknown Cmd
			03 Init MDS	13 Address error
			04 Status SLG	28 Antenna error
			10 Ant On/Off	
			11 Status MDS	

Figure 4-122: Operating window for the *RF170C* component type

#### 4.5.11 *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 Table 4-18 are supported.

All inputs and outputs of the component types are integers. The ASM's 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 are



linked to the Profibus DP gateway via the corresponding input/output signals. The *ID1* and *ID2* inputs should be implicitly connected to the sensors of the conveyor technology simulation.

Command	Code	Description
Reset	0x00	Resets the SLG
Write	0x01	Writes data to an MDS
Read	0x02	Reads data from an MDS
Init MDS	0x03	Initialises 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 MDS

**Table 4-18:** Supported *RF180C* commands

### 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 initialised in the ASM at the start of simulation: True:= Antenna on, False:= Antenna off.

Figure 4-123 shows the parameters and their default settings.

Name	Value
MDS1	MDS
MDS2	MDS
AntennaInitStatus	True

**Figure 4-123:** Parameters for the *RF180C* component type

### Additional parameters

The additional parameters comprise status values corresponding to the UDT110 as online-editable parameters for each of the two simulated read/write devices (SLGs) (see Figure 4-124).

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

**Figure 4-124:** Additional parameters for the *RF180C* component type

### ***Operating window***

The operating window (Figure 4-125) shows the current states of the input and output words of the two channels that are accessed by FB45. The bits shown in grey 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.

RF170C					
	OUT		IN		
Channel	1	2	1	2	
DO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	#MDS
DO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	#MDS
DO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	#MDS
DO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	#MDS
NU	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SLG error
NU	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ASM busy
repeat command	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	comand repeat active
cancel	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ANZ cancel
Reserved	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ANZ MDS present
Reserved	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	MDS presence changed
Reserved	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Reserved
Reserved	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Command counter (BL)
Reserved	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Command counter (BH)
Reserved	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Acknowledgement counter (QL)
Reserved	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Acknowledgement counter (QH)
init run	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	ANZ reset

Channel	1	2	Command	Command Status
Antenna On/Off	<input type="checkbox"/>	<input type="checkbox"/>	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	

**Figure 4-125:** Operating window for the *RF180C* component type

## 5 CREATING CUSTOM COMPONENT TYPES

The component type editor (*CTE*)<sup>1</sup> SIMIT allows you to create your own component types, utilising the mechanisms of the conveyor technology simulation solver. You can expand the functions of the components supplied with the *CONTEC* library, e.g. implement 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 solver, and
- the specific behaviour of the component.

The topological aspect is covered by the corresponding additions to the component type definition. Special connection types are provided for the connection to the solver.

In addition, the general descriptions of component properties in the *CTE* manual form the basis for creating conveyor technology simulation component types. The general properties also apply in full to these component types.

### 5.1 Topological properties

The topology of the modelled conveyor system is automatically determined when compiling a simulation project from interconnected conveyor technology components. Each component must therefore provide relevant topological information about itself, i.e. 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 conveyor technology system are defined for a component and how data exchange between the component and the solver is set up.

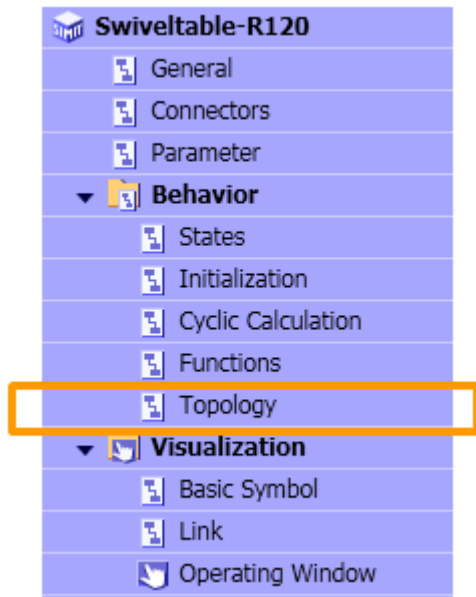
Elements of the conveyor technology system with topological information are

- segments of conveyor sections,
- branches, and
- boundaries.

Relationships with the topological connectors of the conveyor technology component must be defined for each of these elements. To do this, open the topology editor by double clicking on the *Topology* element in the component type navigation menu (Figure 5-1).

---

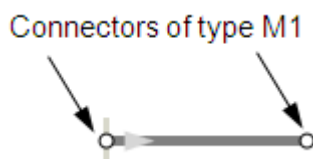
<sup>1</sup> The CTE is included in SIMIT ULTIMATE.



**Figure 5-1:** Topology in the component type navigation menu

### 5.1.1 Topological connectors (connection type *MT1*)

The *MT1* connection type indicates the connectors that are used to connect conveyor technology 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 will be simply referred to as **topological connectors** in the following text. Topological connectors are represented by circles (Figure 5-2).



**Figure 5-2:** Connectors of type *MT1*

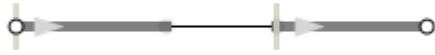
They can be joined by superimposing the connectors. Once joined, both connectors are hidden (Figure 5-3).



**Figure 5-3:** Connecting connectors of type *MT1* by superposition

If a connecting line is used to join topological connectors to one another in the diagram editor, then the connections are hidden and only visible as faint shadows (Figure 5-4).

Connectors of the MT1 type cannot be joined more than once; they can only ever establish a 1:1 connection.



**Figure 5-4:** Connecting connectors of type *MT1* with a connecting line

### 5.1.2 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 solver to determine the calculation order.

### 5.1.3 Topology of a branch

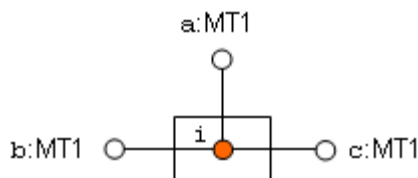
A branch as an internal node in a component type is defined as follows in the topology description:

```
INTERNAL_NODE    i;
```

The component's topological connectors must also be assigned to this branch. Three connectors can be assigned in the topological description as per the following example:

```
FROM    i    TO    a;
FROM    i    TO    b;
FROM    i    TO    c;
```

The three connectors *a*, *b* and *c* must be defined as *MT1* connectors in the component type. Figure 5-5 shows a diagram of the resulting topological structure of the component type.



**Figure 5-5:** Topology of a branch

A branch can also be defined as multiple segments having a common starting or end point that is defined as an *MT1* topological connector.

### 5.1.4 Topology of a boundary

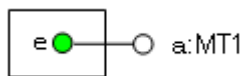
In the topological description of a component type a boundary point as an external node  $e$  is defined as follows:

```
EXTERNAL_NODE e;
```

This external node must also be connected to at least one topological connector of a component. The topological description should be supplemented as follows, for example:

```
FROM e TO a;
```

The topological connector  $a$  must be defined in the component type as a connector of type *MT1*. Figure 5-6 shows a diagram of the resulting topological structure of the component type.

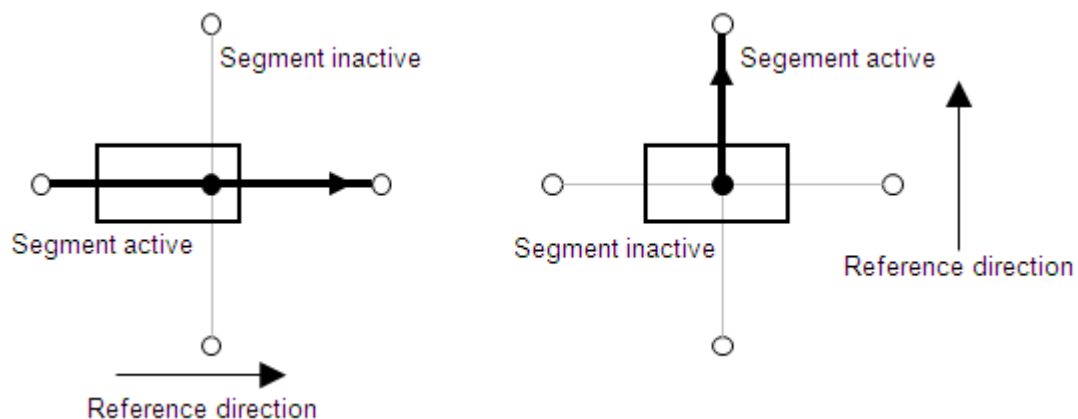


**Figure 5-6:** Topology of a boundary

### 5.1.5 Determining the next section

An object is only ever assigned to one segment in the network. Transport can only take place if the transition to the next segment is clearly defined. Therefore at branching points only one possible subsequent segment can be identified as active; all other alternative segments must be inactive.

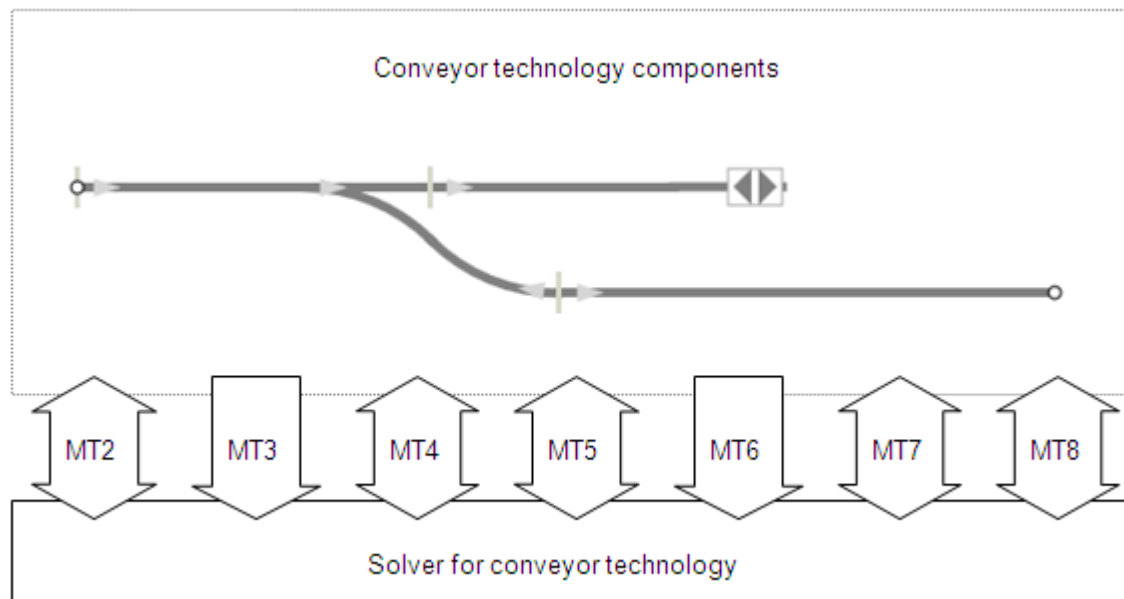
An exception occurs at a branch as an internal node of a component, if the outline of an object covers the branch and only one of the segments connected to the branch is active. In this case the object is assigned to this active segment.



**Figure 5-7:** Implementation at a branch as an internal node


## 5.2 Connection to the solver

The segments, branches and boundary points of a component and the solver exchange data via special conveyor technology connectors. There are seven different connection types *MT2* to *MT8* available for this purpose (Figure 5-8). Their use in conveyor technology components is explained in sections 5.2.1 to 5.2.6 below.



**Figure 5-8:** Data exchange between conveyor technology components and solver

Connectors of connection types *MT2* to *MT8* only establish a connection to the solver and must therefore be set as **hidden** on the component symbol. In the connector properties set the usage "*Property view only*" (see Figure 5-9) or "*CTE view only*".

Property	Value
Usage	Property view only
Visibility Default	

**Figure 5-9:** Visibility in the Properties dialog of a connector of type *MT2* to *MT8*

Connectors of connection type *MT2* to *MT7* should be defined in the **OUT** direction, a connector of connection type *MT8* in the **IN** direction.

### 5.2.1 Connection type *MT2* for segments

A segment can exchange variables with the solver via a connector of type *MT2*. For a connector with the name *MT* the topological description of the segment is supplemented as follows:

```
FROM a TO b: MT;
```

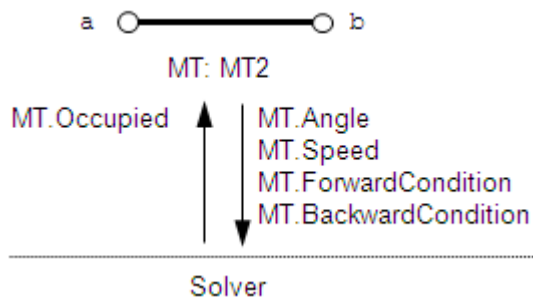
The connector must always be defined in the **OUT** direction (Figure 5-10). It connects the component to the solver in order to exchange variables that are relevant to the segment between the solver and the component.

Name	Connection type	Direction	Dimension
MT	MT2	OUT	1

**Figure 5-10:** Definition of a connector of type *MT2*



Figure 5-11 shows the input and output signals with the direction of data flow between the component and the solver.



**Figure 5-11:** Signals of a connector of type *MT2*

The output signals for the segment are set or calculated in the component and sent to the solver:

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 an anticlockwise direction. For a straight connection the angle should be set to zero.  
This signal is only evaluated when the simulation is initialised; 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 solver 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.

### 5.2.2 Connection type *MT3* for stoppers

A segment can exchange variables with the solver via a connector of type *MT3*. For a connector with the name *MT* the topological description of the segment is supplemented as follows:

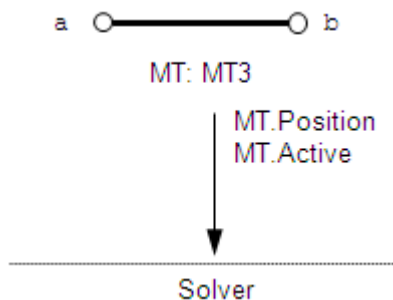
```
FROM a TO b: MT;
```

The connector must always be defined in the **OUT direction** (Figure 5-13). It connects the component to the solver in order to exchange variables between the solver and the component.

Name	Connection type	Direction	Dimension
MT	MT3	OUT	1

**Figure 5-12:** Definition of a connector of type *MT3*

Figure 5-13 shows the output signals with the direction of data flow between the component and the solver.



**Figure 5-13:** Signals of a connector of type *MT3*

The output signals for the segment are set or calculated in the component and sent to the solver:

1. **Position** (analog)

The position of the stopper on the segment as a percentage of the total length of the segment.

This signal is only evaluated when the simulation is initialised; changing the value during the cyclical calculation has no effect.

2. **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.

### 5.2.3 Connection type *MT4* for sensors

A segment can exchange variables with the solver via a connector of type *MT4*. For a connector with the name *MT* the topological description of the segment is supplemented as follows:

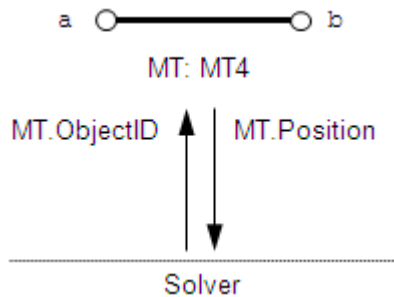
```
FROM a TO b: MT;
```

The connector must always be defined in the **OUT direction** (Figure 5-14). It connects the component to the solver in order to exchange variables between the solver and the component.

Name	Connection type	Direction	Dimension
MT	MT4	OUT	1

**Figure 5-14:** Definition of a connector of type *MT4*

Figure 5-15 shows the output signals with the direction of data flow between the component and the solver.



**Figure 5-15:** Signals of a connector of type *MT4*

The output signals for the segment are set or calculated in the component and sent to the solver:

1. **Position** (analog)

The position of the sensor on the segment as a percentage of the total length of the segment.

This signal is only evaluated when the simulation is initialised; changing the value during the cyclical calculation has no effect.

The component receives information about the segment from the solver 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.

## 5.2.4 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 supplemented as follows:

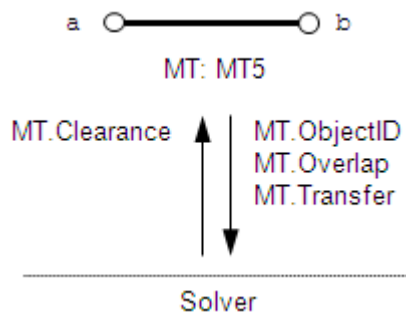
```
FROM a TO b: MT;
```

The connector is always defined in the **OUT direction** (Figure 5-16). It connects the component to the solver in order to exchange variables between the solver and the component.

Name	Connection type	Direction	Dimension
MT	MT5	OUT	1

**Figure 5-16:** Definition of a connector of type *MT5* as an output

Figure 5-17 shows the input and output signals with the direction of data flow between the component and the solver.



**Figure 5-17:** Signals of a connector of type *MT5* as an output

The output signals for the segment are set or calculated in the component and sent to the solver:

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 (see section 5.3.1 and 5.3.2).
2. **Overlap** (analog)  
Overlap in millimetres 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)  
If this signal is set to *True*, the solver starts the placement process. The component can only set the signal for one cycle, after which it must return to *False*.

The component receives information about the segment from the solver via the input signals:

1. **Clearance** (analog)  
The available length of segment in millimetres viewed in the preferred direction.

Connection type *MT5* is used in the *Rail-S4* and *Conveyor-S4* library components, for example.

### 5.2.5 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 solver, for a connector *MT* as follows:

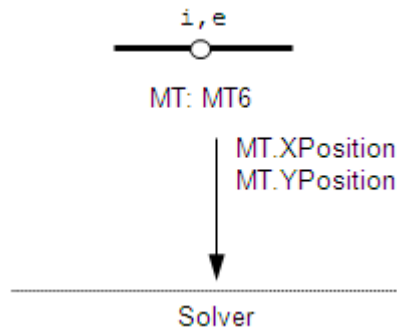
```
EXTERNAL_NODE E: MT;
INTERNAL_NODE I: MT;
```

The connector must always be defined in the **OUT direction** (Figure 5-18). It connects the component to the solver in order to exchange variables that are relevant to the node between the solver and the component.

Name	Connection type	Direction	Dimension
MT	MT6	OUT	1

**Figure 5-18:** Definition of a connector of type *MT6*

Figure 5-19 shows the output signals with the direction of data flow between the component and the solver.



**Figure 5-19:** Signals of a connector of type *MT6*

The output signals for the node are set or calculated in the component and sent to the solver:

1. **XPosition** (analog)

The X-position of the node in millimetres relative to the top left corner of the component.

This signal is only evaluated when the simulation is initialised; changing the value during the cyclical calculation has no effect.

2. **YPosition** (analog)

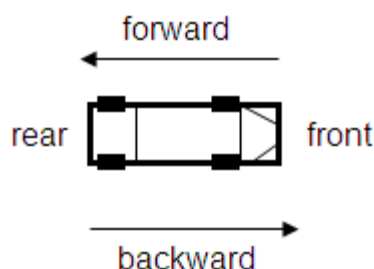
The Y-position of the node in millimetres relative to the top left corner of the component.

This signal is only evaluated when the simulation is initialised; changing the value during the cyclical calculation has no effect.

### 5.2.6 Connector type *MT7* for programming objects

An object can exchange variables with the solver 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 (see Figure 5-20).



**Figure 5-20:** Orientation of objects

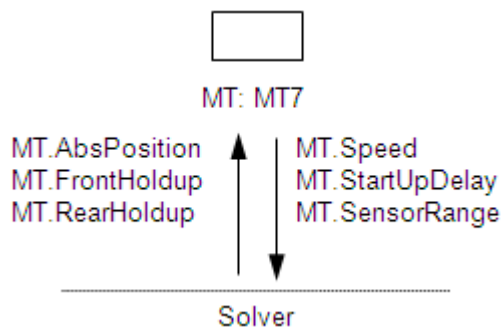
The connector of type *MT7* must always be defined in the **OUT direction** (Figure 5-21). It connects the component to the solver in order to exchange variables that are relevant to the object between the solver and the component.

A component can have no more than one connector of type *MT7*.

Name	Connection type	Direction	Dimension
MT	MT7	OUT	1

**Figure 5-21:** Definition of a connector of type *MT7*

Figure 5-24 shows the output signals with the direction of data flow between the component and the solver.



**Figure 5-22:** Signals of a connector of type *MT7*

The output signals for the object are set or calculated in the component and sent to the solver:

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 solver, 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 solver 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.

### 5.2.7 Connector type *MT8* for transferring objects at boundary points

Objects can be transferred from the solver to a component and vice versa at boundary points as external nodes. Once a component rather than the solver has control over the

object, the position of the object cannot be changed by the solver. The object remains visible on the diagram, however, and can now be positioned by the component by means of corresponding system functions (see section 5.3.8). This allows more complex transport operations, which cannot be simulated with a path-based solver, to be implemented in conveyor components.

A boundary point as an external node can exchange variables with the solver via a connector of type *MT8*. For a connector with the name *MT* the topological description of the external node is supplemented as follows:

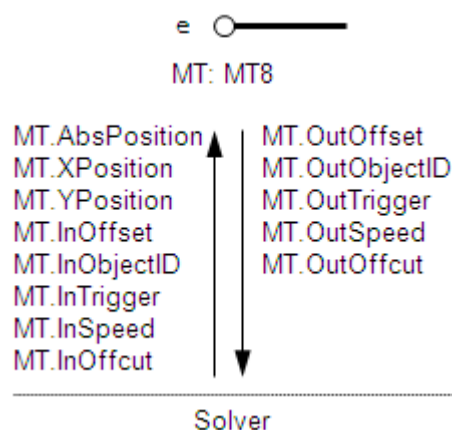
```
EXTERNAL_NODE E: MT;
```

The connection must always be defined in the **IN direction** (Figure 5-14). It connects the component to the solver in order to exchange variables between the solver and the component.

Name	Connection type	Direction	Dimension
MT	MT8	IN	1

**Figure 5-23:** Definition of a connector of type *MT8* as an input

Figure 5-24 shows the output signals with the direction of data flow between the component and the solver.



**Figure 5-24:** Signals of a connector of type *MT8* as an input

The output signals for the external node are set or calculated in the component and sent to the solver:

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 solver; in this case the value corresponds to the length of the projecting part of the object. All values are given in millimetres.
2. **OutObjectID** (integer)  
The object ID of the object that viewed from the component is projecting onto the adjacent section controlled by the solver. The value is zero if no object is projecting.
3. **OutTrigger** (binary)  
The component sets this signal to *True* for one cycle to show that the adjacent section controlled by the solver 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 solver has to take this time into account in the next cycle in order to calculate an additional movement.

The component receives information from the solver 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 millimetres.

3. **YPosition** (binary)

The relative Y-position of the connector linked to the node relative to the top left corner of the component, in millimetres.

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 solver, 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 millimetres.

5. **InObjectID** (integer)

The object ID of the object that viewed from the section controlled by the solver is projecting into the component. The value is zero if no object is projecting.

6. **InTrigger** (binary)

The solver sets this signal to *True* for 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 solver.

8. **InOffcut** (analog)

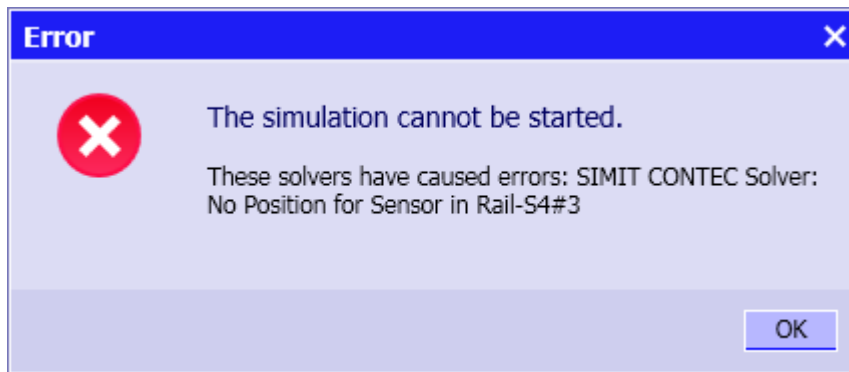
The time that the current cycle could not include because the object had reached the transfer node, measured in seconds. The component has to take this time into account in the next cycle in order to calculate an additional movement.

Connector type *MT8* is used in the *TransferCarriage* library component, for example.

## 5.2.8 Errors in the component code

If interfaces to the solver are created in conveyor technology components but not all necessary signals are made available, the simulation start request is denied and a corresponding error message displayed.





**Figure 5-25:** Missing calculation of an interface signal

### 5.3 System functions

If you create your own component types you can utilise various system functions to access objects in the simulation.

All the functions described below supply an error code as the return value. In order for a function to be able to supply return values in a transmitted variable, this variable must be defined as a field (vector), in this case with the dimension 1. Please refer to Table 5-1 for the meaning of this return value.

Value	Description
0	No error
-2	The object cannot be positioned directly because it is currently under the solver's control.
-3	The object cannot be positioned or rotated because it is not currently assigned to a conveyor section.
-10	There is no object with the specified ID.
-11	The object with the specified ID is not available. <sup>2</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

**Table 5-1:** Return values of conveyor technology system functions

<sup>2</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.

### 5.3.1 **\_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 the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- name (string)  
The instance name of the object.

### 5.3.2 **\_MT.GetObjectByType**

The *\_MT.GetObjectByType* function returns the ID of an object of the component type *type* if it is available as an object, otherwise it returns the value -1. The return value type is *long*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- 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 returned here, all existing lists are searched.

### 5.3.3 **\_MT.Restore**

The *\_MT.Restore* function releases an object. The return value is *void*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- 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.

### 5.3.4 **\_MT.GetWidth**

The *\_MT.GetWidth* function returns the width of an object in millimetres. The return value type is *double*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be queried.

### 5.3.5 **\_MT.GetHeight**

The *\_MT.GetHeight* function returns the height of an object in millimetres. The return value type is *double*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be queried.

### 5.3.6 **\_MT.GetDepth**

The *\_MT.GetDepth* function returns the depth of an object in millimetres. The return value type is *double*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be queried.

### 5.3.7 **\_MT.GetAngle**

The *\_MT.GetAngle* function returns the angle in degrees (-180° to +180°) of an object relative to the tangent of the segment on which it is located. The return value type is *integer*.

Note that this function can only be called for objects that are assigned to a conveyor section.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be queried.

### 5.3.8 **\_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 millimetres 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 the solver's control.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be changed.
- x (analog)  
X-position in millimetres relative to the conveyor section component.
- y (analog)  
Y-position in millimetres relative to the conveyor section component.

### 5.3.9 **\_MT.SetAngle**

The *\_MT.SetAngle* function sets the angle in degrees ( $-180^{\circ}$  to  $+180^{\circ}$ ) of an object relative to the tangent of the segment on which it is located. The return value type is *void*.

Note that this function can only be called for objects that are assigned to a conveyor section.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be changed.
- angle (integer)  
Angle in degrees relative to the tangent of the segment.

### 5.3.10 **\_MT.AddAngle**

The *\_MT.AddAngle* function increases the angle in degrees ( $-180^{\circ}$  to  $+180^{\circ}$ ) of an object relative to the tangent of the segment on which it is located. The return value type is *void*.

Note that this function can only be called for objects that are assigned to a conveyor section.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be changed.
- angle (integer)  
Angle in degrees relative to the tangent of the segment.

### 5.3.11 **\_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 the solver's control.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- 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.

### 5.3.12 **\_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 the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the variable to be queried.

### 5.3.13 **\_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 the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the variable to be queried.

### 5.3.14 **\_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 the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the variable to be queried.

### 5.3.15 **\_MT.GetByteValue**

The *\_MT.GetByteValue* function is used to query a byte (state) of an object. The return value type is *byte*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- id (long)  
The ID of the object to be queried.
- property (string)  
The name of the state to be queried.

### 5.3.16 **\_MT.GetByteArray**

The *\_MT.GetByteArray* function is used to query a byte array (state) of an object. The return value type is *void*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- 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, please enter it in the notation for new state values, i.e. 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.

**CAUTION**

Please make sure that all arrays are large enough to execute this operation, otherwise unforeseeable errors may occur in the simulation sequence.

---

### 5.3.17 `_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 the following transfer parameters:

- `error` (long[])  
Error code as shown in Table 5-1.
- `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.

### 5.3.18 `_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 the following transfer parameters:

- `error` (long[])  
Error code as shown in Table 5-1.
- `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.

### 5.3.19 **\_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 the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- 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.

### 5.3.20 **\_MT.SetByteValue**

The *\_MT.SetByteValue* function is used to write a byte (state) of an object. The return value type is *void*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- 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.

### 5.3.21 **\_MT.SetByteArray**

The *\_MT.SetByteArray* function is used to write a byte array (state) of an object. The return value type is *void*.

Specify the following transfer parameters:

- error (long[])  
Error code as shown in Table 5-1.
- 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.

**CAUTION**

Please make sure that all arrays are large enough to execute this operation, otherwise unforeseeable errors may occur in the simulation sequence.

## 5.4 System variables

The system variables listed in Table 5-2 are available for modelling conveyor technology components. These variables are available in the component's behaviour description with read access only.

Variable name	Description
_WIDTH	Width of the component in pixels
_HEIGHT	Height of the component in pixels
_SCALEX	Horizontal scaling of the component. The factor 1 is the default setting.
_SCALEY	Vertical scaling of the component. The factor 1 is the default setting.
_TECHSCALE	Scale of the diagram on which the component is located. The number of millimetres corresponding to one pixel is specified.

**Table 5-2:** System variables