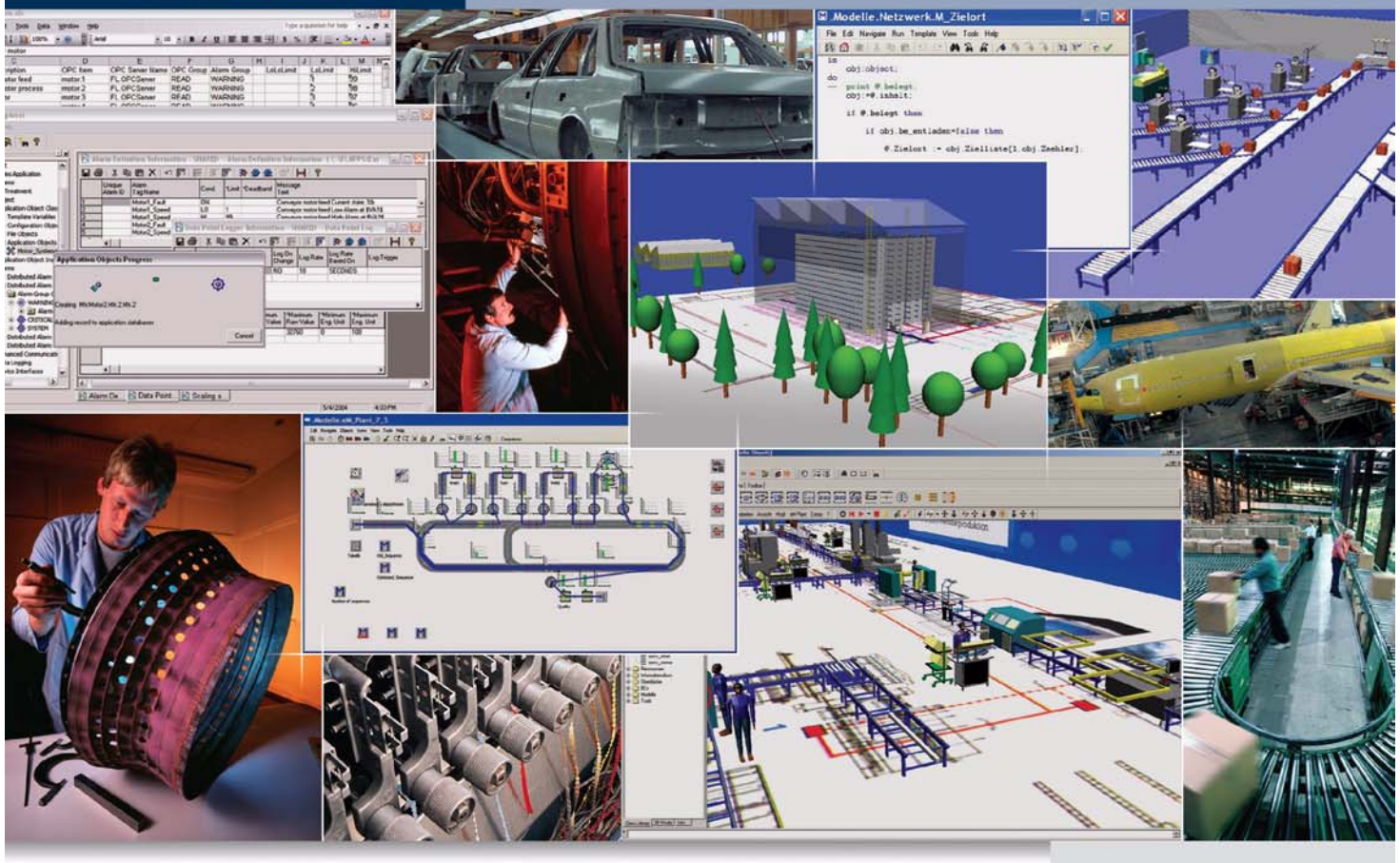
Plant Simulation Transport library

Reference manual

Siemens PLM Software

www.siemens.com/plm



TECNOMATIX

SIEMENS

Proprietary and Restricted Rights Notice

© 2007 Siemens Product Lifecycle Management Software II (DE) GmbH. All Rights Reserved.

This documentation is proprietary to Siemens Product Lifecycle Management Software II (DE) GmbH.

This document contains proprietary information and is protected by copyright. No part of this document may be reproduced, stored in a retrieval system, translated, transcribed, or transmitted, in any form or by any means, without the prior explicit written consent of Siemens Product Lifecycle Management Software II (DE) GmbH.

Siemens and the Siemens logo are registered trademarks of Siemens AG.

Tecnomatix and the Tecnomatix logo are registered trademarks of Siemens Product Lifecycle Management Software Inc.

All other product names or brand names are trademarks or registered trademarks of their respective owners.

Information in this document is subject to change without notice.

TECNOMATIX
Plant Simulation

SIEMENS

Libraries of material flow equipment

December 2008

Table of Contents

Overview about the libraries of material flow equipment 1

AGVS 3

Sample Model 3

A model with local routing tables 3

An example with a production system 4

Rules for Assigning Vehicles 6

Longest Waiting Time 7

Shortest Waiting Time 7

Shortest Distance 7

Randomized Choice 7

Free Dispatch 7

Rules for Assigning Orders 7

Longest Waiting Time 7

Shortest Distance (not yet implemented) 7

Random Choice 7

Description of the Application Objects 8

Tracks 8

Building blocks AGVS_track 8

Building blocks AGVS_straight and AGVS_bend 8

Branches AGVS_br2 and AGVS_br3 9

Junctions AGVS_junction2 and AGVS_junction3 9

The object AGVS_crossing and AGVS_cross 10

The U-turn AGVS_U_turn 10

Interface to other transport systems 11

AGVS_interface 11

Other Stations 12

AGVS_station 12

Blockline Control AGVS_blocklineCtrl 12

Terminal Line Branch AGVS_tl_br 13

Docking station AGVS_dock_station 14

Object AGVS_PutIn 15

Object AGVS_PutOut 15

Transport Manager 16

AGVS_manager 16

Statistical Evaluation 16

The object VehUtil 16

Movable Units 17

Vehicle AGV 17

Part 17

Conveyor 19
Introduction 19
Examples 19
Simple Examples. 19
A model with local routing tables 20
An example with a production system. 21
Description of the Application Objects 22
Conveyor Objects 22
Object cv_Track 23
Objects cv_Straight and cv_Bend 23
Conveyor Elements with Lifting or Turning Facilities 24
Branches cv_br2 and cv_br3 24
Objects cv_junction2 and cv_junction3 24
Object cv_crossing 25
Object cv_batchline 25
Object cv_dockingStation 26
Object cv_PutIn 27
Object cv_PutOut 28
Interface to other Transport Systems 28
Object cv_interface. 28
The Conveyor Control 29
Object cv_Manager. 29
The object Part. 29
EOM 31
Introduction 31
Sample Model. 31
EOM Objects. 32
Inserting the Object Protocol 36
Basic Course of Events 36
Disposition of Vehicles. 36
Block Sections 39
Track Clearing Strategy 39
Buffers. 39
Switches and Confluences 40
Pile-ups in Front of Confluences 40
Switches 41
Lifting Stations 41
Feeding of Vehicles. 42
Loading of Orders. 42
The Transport System Electric Overhead Monorail 43
Description of Objects Vehicles Travel On 43

Straight - EOMstraight 43
Curve - EOMcurve 43
Branch - EOMbranch2 / EOMbranch3 44
Confluence - EOMjoin2 / EOMjoin3 44
Pile - EOMjoinG 44
Lifting station - EOMlift 45
Switches - EOMswitch1 ... EOMswitch5 46
Buffer - VEHbuffer 46
Source Vehicle - VEHinput 46
Loading station - EOMload 47
Unloading station - EOMunload 47
Docking station - EOMdock 48
Work Bench Operation - EOMwork 48
Managing Vehicles 48
EOM Control - EOMcontrol 49
Vehicle - veh 49
Course of Events Across Objects 50
Reset and Init 50
Assigning Destinations 51
HBW 53
Introduction 53
Alternative 1: 53
WarehouseUT 53
Alternative 2: 53
Warehouse Manager 53
Aisle 53
Alternative 3: 54
Warehouse Management for ABC articles 54
Aisle for ABC article 54
Example 54
Description of the Application Objects 56
WarehouseUT 56
The Warehouse Manager WhMgr 57
The Aisle 58
The Warehouse Manager WhMgr_ABC 60
Aisle_ABC 61
Applications and Testmodels 63
Extensions and Restrictions 63
Test Models 63
Demonstration Model 63
Production control 65
Object ProductionManager 65

Workplace Station 66
Set-up and Processing Time 67
Simulation Disruptions 67
The object Part 67

Support for Modeling 69

The Assistant ASSI 69
Application of Transport System Interfaces 69
Application of Transport System Interfaces 70

1

Overview about the libraries of material flow equipment

The libraries Plant Simulation AGVS (**A**utomated **G**uided **V**ehicle **S**ystem), Plant Simulation Conveyor, Plant Simulation EOM (**E**lectric **O**verhead **M**onorail) and Plant Simulation HBW (**H**igh **B**ay **W**arehouse) are a set of libraires to model material flow systems. The purpose of these transport and storage systems is to support a production system, which can model by the Plant Simulation Shop Light.The material flow in a production system is described by order lists and production plans. Of course, you can also use your own modeling of a production system supposed the MUs are equipped with the needed attributes. If you want to consider the perfromance of the material flow system without the connection to a production system, you can use local routing tables. Routing tables determines the next destination of a MU at a station of the material flow. It is also possible to model the connection between different transport systems

Any parameters or controlling rules necessary may be defined in dialog boxes. It was developed with the simulation program Plant Simulation and requires an Plant Simulation Professional License, Plant Simulation Standard License or Plant Simulation Application License. Using the application object template requires knowledge of Plant Simulation..

Note: This application object template should fully cover most modeling tasks requiring AGV systems. However, because of the template's flexibility and general applicability, the application objects and their interrelations often are fairly complex. Therefore, any adaptation of the application object template will require an in-depth knowledge of its internal functionality. Prior to using it for a project, you should check whether the existing functionality meets the requirements of the project. If adaptations are necessary, we recommend checking the cost benefit ratio carefully. In any case, the application object template will be helpful for quickly creating prototypes and as a reference for your own application object templates.

For the automatic registration of all building blocks there is the object *Assistant*.

Every application object offers a user interface providing dialog boxes, pop up menus, and data input lists. All application objects are open for individual modifications. Examples for the application of the libraries are contained in the category *Free Libraries* of the example collection.

2

AGVS

Plant Simulation AGVS is a set of application objects to model automated guided vehicle systems (AGVS).

Any parts to be transported will be moved by the shortest way to the destination. The Plant Simulation AGVS application object template may be combined with other application object templates, such as Plant Simulation conveyor for the modeling of conveyor belts systems.

A track system is assigned to a transport manager *AGVS Manager*. In a simulation model there can be some track systems. The manager provides automatic destination determination and some vehicle and order disposition rules.

For the modeling of the track system the following application objects are provided

- *AGVS docking station* (transfer to a processing station of a produktion system)
- *AGVS track* (modelling of the track system by polygons, consisting of stright line segments and circle segments.)
- *AGVS straight* (standardized straight line segment)
- *AGVS curve* (standardized 90°-curve segment)
- *AGVS branch* (branch in in 2 or 3 directions)
- *AGVS junction* (junction from 2 or 3 directions)
- *AGVS crossing* (crosiing of 2 tracks)
- *AGVS Put in* (transfer of parts into the conveyor system)
- *AGVS Put out* (take out of parts)
- *AGVS Interface* (transfer to other transport systems)
- *AGVS blockline* (definition of regions, in which only one vehicle is allowed.)
- *AGVS terminal line* (entry of a track subsystem, in which the vehicle can move backwards.)

For the automatic registration of all building blocks there is the object *Assistant*.

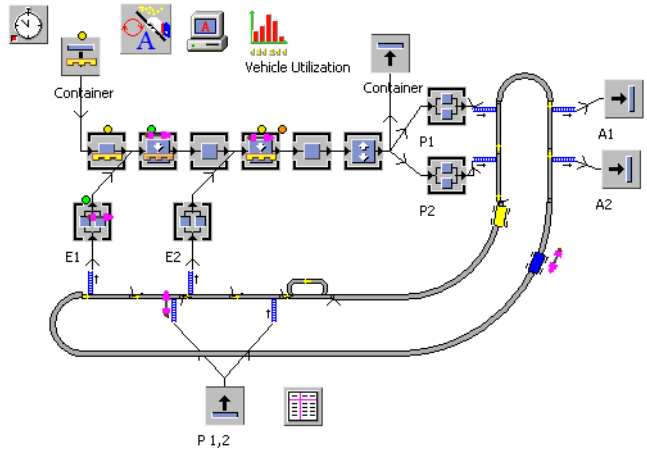
Every application object offers a user interface providing dialog boxes, pop up menus, and data input lists. All application objects are open for individual modifications. Examples for the application of the libraries are contained in the category *Free Libraries* of the example collection.

Sample Model

Let us suppose we have to design an AGVS.

A model with local routing tables

The model *AssemblyLineWithAGVS* shows an application of the routing tables in the objects *agvs_dock_Station* and *agvs_PutIn*. With the help of the routing tables you are able the model the material flow without the usage of a production control and without programming methods.



The model *AssemblyLineWithConv*

Whenever a part is entered into the transport system the next destination point of the part is determined by the part type or by the value of a customized attribute.

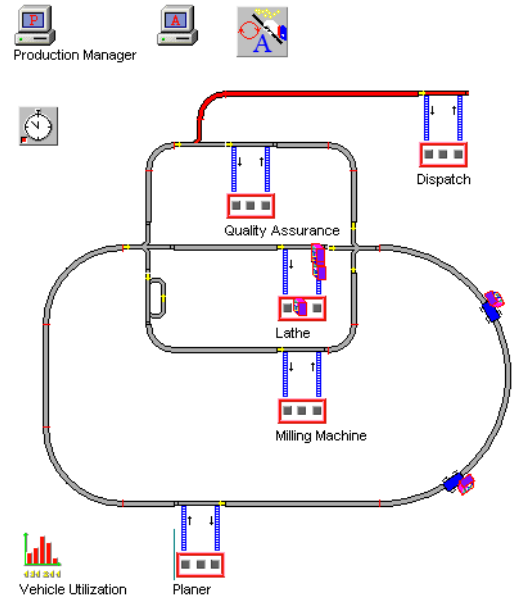
	string 0	object 1
string	Typ der BEs	Nächste Bearbeitungsstation
1	T1	root.E1
2	T2	root.E2
3		

The routing table determines the next destination.

Open the routing table on the tab Destination of the objects *cv_PutIn*.

An example with a production system

The company's production line consists of a lathe, a milling machine, a planer, quality control, and shipping, called dispatch in our model. For this task, a layout has be created similar to that shown below:



Production layout with an AGVS

First, create a new frame and insert the objects *ProductionManager* and *AGVS_manager*. Insert the objects of the transportation system according to the screenshot above. If you turn on the frame's grid then application objects are automatically connected if the distance between them is less than 3 pixels. For the modelling of track, consisting of straight line segments and Circular arces, you use the object *agvs_track*.

The station type (such as lathe, milling machine, etc.) must be entered for each work station. Then, the operations the station are able to execute are defined (double click on the station, then click on the button Open next to available operations).

After creating the layout, check if all application objects are connected. If that is the case, the operations plan and the customer orders for the parts to be produced have to be created.

To accomplish this, open the application object *ProductionManager*. After clicking on the button Master, the following table will appear:

	string 0	table 1
string 0	entitytype	operationsplan
1	part	Plan
2		

TableFile masterData *of the ProductionManager*

Define the part type (in our example *part*). Selecting the cell in the column *OperationsPlan_L* and pressing the key *F2* opens the operations plan for the corresponding entity type. Define the sequence of production steps by entering the operations the part should pass through.

string 0	string 1	time 2	time 3	real 4
	operation	setup time	processing time	factor
1	prepare	3.0000	2.0000	1.00
2	mill	3.0000	2.0000	1.00
3	turn	3.0000	2.0000	1.00
4	check	3.0000	2.0000	1.00
5	dispatch	3.0000	2.0000	1.00

TableFile *masterDataL[*1,1*]*

Then, enter the number of parts. Use the button Orders in the dialog of the ProductionManager. A table will be opened where you enter the type of the part and the desired quantity.

Finally, define the number of AGVs in the application object *AGVS_manager*.

To automatically register all tracks and stations please insert the object *ASSI*. Before you start the assistant *ASSI* press the button Reset. All actions of the assistant are registered in the consol. After using the assistant you can remove this object from your model.

Clicking on the buttons Reset and Start in the EventController, will move the AGVs and execute the orders.

Use the object *VehUtil* for evaluating the statistics of the vehicles (for instance utilization). Activate the object chart in *VehUtil* to display the current utilization.

The example described above is automatically constructed by method *AGVScurves* in the frame *.Models.Transport.build-DEMO* which are contained in the category *Free Libraries* of the example collection.

Rules for Assigning Vehicles

When a part needs to be transported, the following will happen: First a new destination is determined. Then, it is passed on to the docking station. When the part enters the docking station, it tells the *AGVS_manager* that it wants to be transported and therefore needs an AGV. The part is registered in the demand list *TransportOrderL* in the *AGVS_manager* and then the *AGVS_manager* tries to start a transport order (method *tryToStartTransOrder*).

If an AGV is available, an order is assigned and this AGV is marked as *occupied*.

If no AGV is available, the order remains in a demand list until an AGV is available. As soon as an AGV notifies the *AGVS_manager* that it is available, the *AGVS_manager* checks if there are still orders to be fulfilled. If so, an order is chosen depending on the user-defined disposition rules for orders.

If there is no order to be fulfilled, the AGV registers that it is available and is ready to fulfill new orders. The AGV drives to the next station with an available place. This place is reserved for that AGV. When the vehicle gets an order, the reservation will be canceled.

When a part needs to be transported from one station to another, it asks the *AGVS_manager* for an AGV. The *AGVS_manager* then chooses an AGV.

Choices for choosing an AGV are:

- Longest Waiting Time
- Shortest Waiting Time
- Shortest Distance
- Random Choice
- Available Dispatch

Longest Waiting Time

Using this rule, the AGV that registered first at the *AGVS_manager*, and therefore has been waiting the longest, is selected. The AGVs are assigned according to the First-In-First-Out (FIFO) rule.

Shortest Waiting Time

AGVs are assigned according to the Last-In-First-Out (LIFO) rule. That means the AGV that registered last with the *AGVS_manager* is assigned first.

Shortest Distance

The AGV that has to cover the shortest distance to the destination and is available is chosen.

Randomized Choice

An available AGV is randomly chosen by a uniform distribution.

Free Dispatch

Three choices are offered:

1. Minimum Utilization

The AGV with the lowest utilization is assigned.

2. Maximum Velocity

The AGV capable of the highest velocity is assigned.

3. Maximum Battery Charge

The AGV with the highest battery charge is assigned.

Rules for Assigning Orders

When an AGV registers as available and there are still orders to be fulfilled, the assignment of the AGVs to the orders follows different rules:

Longest Waiting Time

The order waiting the longest is assigned to the AGV (FIFO principle).

Shortest Distance (not yet implemented)

The order with the shortest distance to the location of the AGV is assigned.

Random Choice

An order is assigned to the AGV based on a uniform distribution.

Description of the Application Objects

In this chapter, the objects of the application object template are described in detail. First, the functionality, followed by the parameters that may be changed by the user. Application objects having the same or a similar functionality are described only once.

For each object, a dialog for changing parameters is provided. In addition, all dialogs of length-oriented objects contain an input field for setting the length of the application object. Double-clicking the application object opens its dialog.

Tracks

The objects described here are tracks on which AGVs move in one or in two directions. For each connected track system an *AGVS_manager* must be assigned. Therefore, the dialog of the tracks contains an input element for the responsible manager. A manager handles all vehicles which can move on a connected track subsystem.

Building blocks AGVS_track

Icons:

Functionality:

You are able to model the track for the vehicles consisting of straight line segments and circular. arces.

User interface:

Dialog of a track object *AGVS_track*

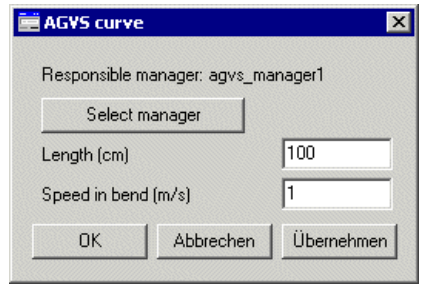
Building blocks AGVS_straight and AGVS_bend

Icons:

Functionality:

Use these objects, to model straights or tracks with 90 degree angles. An entering AGV will move according to the default behavior, i.e. it will move to each of successors in turn.

User interface:



Dialog of the object *AGVS_bend*

Branches AGVS_br2 and AGVS_br3

Icons:

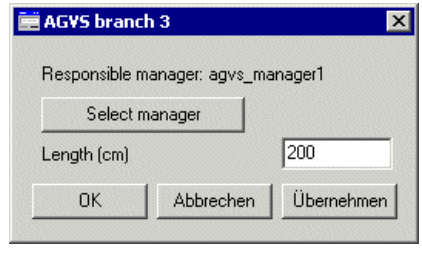
Functionality:

These application objects are used to model branches into two or three directions. When branching, an AGV has to decide in which direction it should move. For this reason, the AGV searches in the sign lists *rgSign** the destination and moves to the corresponding direction.

The sign lists *rgSign** must be filled either manually or by using the assistant *ASSI*.

An entering AGV searches the sign lists for its destination. As soon as the destination is found, the AGV is moved in the corresponding direction. If the destination is not found, the AGV moves straight on.

User interface:



Dialog of the objects *AGVS_br**

Junctions AGVS_junction2 and AGVS_junction3

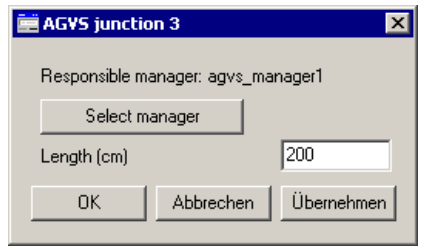
Icons:

Functionality:

These application objects are used for modeling junctions from two (*AGVS_junction2*) or three (*AGVS_junction3*) directions.

Entering AGVs are moved by using the system's default behavior. The AGV entering first has priority.

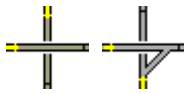
User Interface:



Dialog of the objects *AGVS_junction**

The object AGVS_crossing and AGVS_cross

Icons:

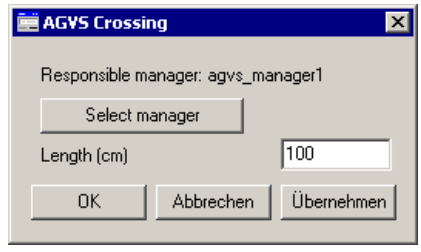


Functionality:

An entering AGV can only be moved straight on in the object *agvs_crossing*. Therefore, do not use this application object to model curves. In addition a right bend is possible in the object *agvs_cross* if the vehicle enters in one direction.

An AGVs blocks the object, such that the AGVcannot crash.

User interface:



Dialog of the object *AGVS_crossing*

The length you entered in the dialog is both: Length and width of the crossing.

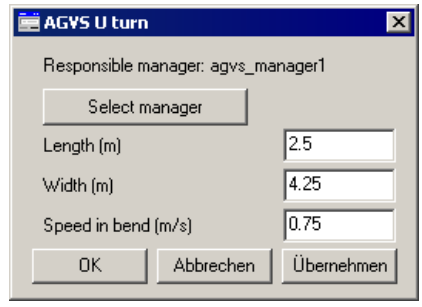
The U-turn AGVS_U_turn

Icon:

Functionality:

You can model U-truns in your track system. The object is not allowed in a terminal line, that is, the AGVs can only drive in directions of movement shown in the icon. Be aware that only one AGV can turn around on the object *AGVS_U_turn*. The AGVs cannot touch each other on the upper and lower tracks, so that several AGVs can be located on these tracks.

User interface:



Dialog of the object *AGVS_U_turn*

Interface to other transport systems

AGVS_interface

Icon:

Functionality:

The application object *AGVS_interface* is used to connect the AGVS with other material flow systems, such as a conveyor or an electric overhead monorail. The connection of two AGV systems is demonstrated in the frame *DEMO_IF*.

Note: An *AGVS_interface* cannot serve as a docking station!

When the *AGVS_interface* is the destination of an AGV, it is loaded or unloaded and moved on to its successor. An unloaded part is moved to the corresponding interface of the other material flow system and generates a request for a vehicle in that AGVS.

User interface:

Dialog of the object *AGVS_interface*

Other Stations

AGVS_station

Icon:

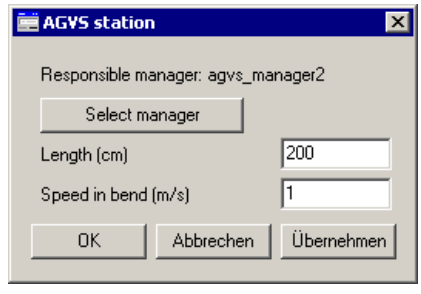
Functionality:

The application object *AGVS_station* is used to model stations in the system. You may define more than one *AGVS_station* within a system. Note that the number of AGVs inserted into the system (See the dialog of the object *AGVS_manager*) must be smaller than the entire capacity of all stations.

If the destination of the AGV is the *AGVS_station*, the AGV is moved from the course to a waiting track and registered as available. Blocked blocklines are set to available.

If an AGV in the waiting track receives an order, it is moved from the waiting track to the course without checking if there is another AGV on the track.

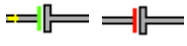
User interface:



Dialog of the object *AGVS_station*

Enter the length of the application object here and the maximum number of AGVs allowed to enter the *AGVS_station* here.

Blockline Control AGVS_blocklineCtrl

Icons: 

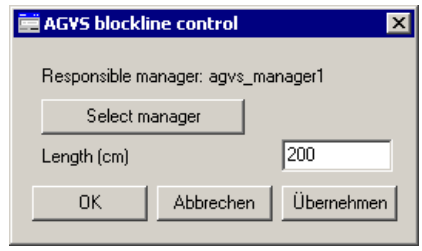
Functionality:

If blocklines are used in a subsystem, a subsystem is partitioned into regions. In each region at most one AGV is allowed.

Note: Using blocklines within terminal lines is not allowed!

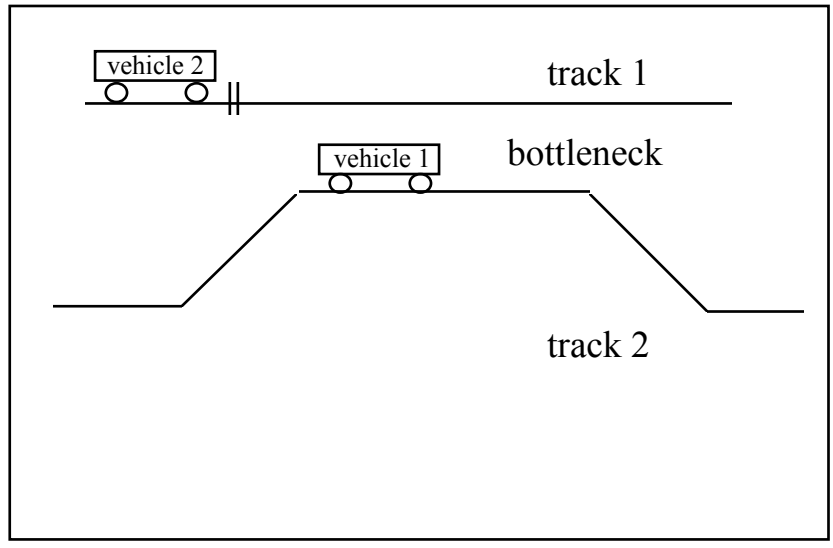
An AGV can move into a region by passing more then one blockline element. If such a building block is entered all entrances of the region are closed (shown with a red icon) and all entrances of the previous region are opened (shown green). The recognition of the regions can be done by the assistant *ASSI*.

User interface:



Dialog of the object *AGVS_blocklineCtrl*

Use blocklines to introduce bottlenecks into the system. This is only possible if the number of AGVs is greater or equal to the number of inserted blocklines.



Definition of a bottleneck

A bottleneck is a place where the distance between two tracks is so small that two AGVs cannot pass each other at the same time. Therefore, one of the two tracks has to be blocked.

Terminal Line Branch AGVS_tl_br

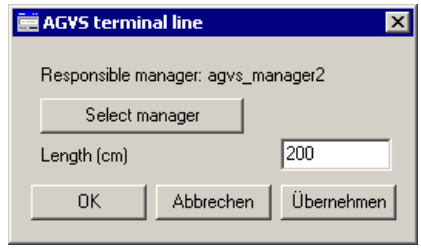
Icon:

Functionality:

The application object *AGVS_tl_br* was created as an entrance to a terminal line (the abbreviation stands for AGVS terminal line branch). When terminal lines are used, you have to place them here, otherwise the terminal line will not be detected.

The direction of an entering AGV is checked. If the destination is not on the terminal line, the AGV will be sent straight ahead. Otherwise, the terminal line will be checked to see if another AGV is already in the terminal line. If the destination is on the terminal line and there is no other AGV in the terminal line, the AGV will be sent onto the terminal line.

User interface:



Dialog of the object *AGVS_tl_br*

Docking station AGVS_dock_station

Icon:

Functionality:

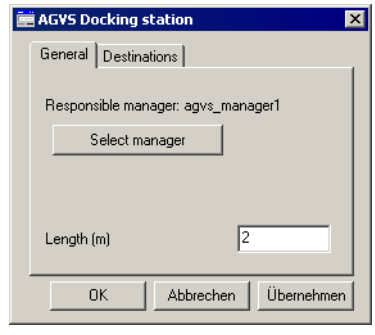
A docking station is a destination for an AGV. The destination station of the part is attached to the docking station, that is the destination of the AGV. Therefore it has to be known which docking station is connected with which station. The assignment docking station - station is managed by the application object *AGVS_manager*.

The modeled docking station may be used in normal track paths and terminal lines.

The destination of the AGV entered is checked. If the destination is the docking station, the AGV is loaded or unloaded and moved on to its successor. The unloaded part is passed to the station and is processed.

When a part is unloaded, a check is performed whether another part needs to be transported. If the AGV is available, this new part is loaded.

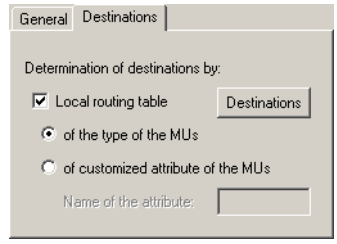
User interface:



The tab General of the Dialog of the object *AGVS_dock_station*

Enter the length of the object and the responsible manager here.

If the next destination is not determined by a production system, then you can define the destination by local routing tables according to the type or the value of a customized attribute of the part. On the tab Destination you can define whether the destination is determined by the type or by an attribute. The type is given by the customized attribute *entity-Type* (string). If there is no such attribute then the type is determined by the name of the part.



The tab Destination of the Dialog *cv_dockingStation*

The button Destination opens the routing table, in which you can assign the destinations.

Object AGVS_PutIn

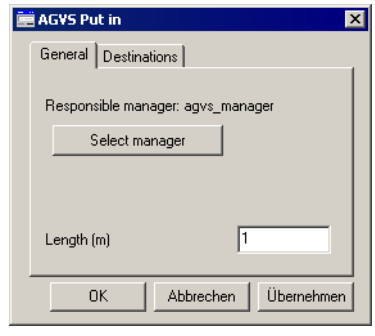
Icon:

Functionality:

The part to be processed (object *part*) is passed to the transport system. The part lead to the transport order. For instance, this object can connect with a source.

For an arriving AGV the object checks whether the transport order is equal to the part, waiting for transportation. If yes then the part is loaded. The functionality of the local routing table is described in the chapter about the docking station.

User interface:



The tab General of the dialog of *AGVS_PutIn*

Object AGVS_PutOut

Symbol:

Functionality:

As the docking station the Put out object is a destination for an AGV. The object can connect with a drain.

For an arriving AGV the object checks whether the destination of the loaded part is equal to the object. If yes the AGV is unloaded. The Put out object is connected with an object which is a destination of the transported part. The assignment of the destination of parts and AGVs is done in the *AGVS_manager*.

User Interface:

The dialog of *agvs_PutOut* is similar to the dialog of *agvs_straight*.

Transport Manager

The *AGVS_manager* is used to manage the AGVs and the track sections of a subsystem, on which the AGVs can move.

AGVS_manager

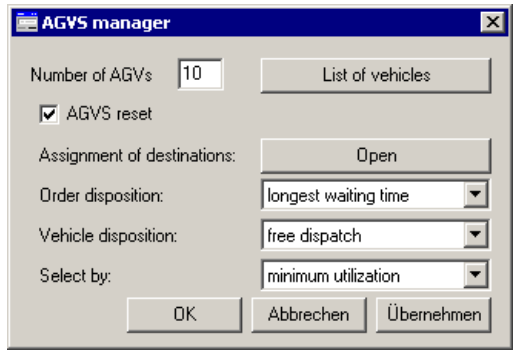
Icon:

Functionality:

The CardFile *rgAnnounce* contains all tracks of the subsystem. The TableFile *rgAssignL* contains the assignments of the vehicle destination (such as docking stations, interfaces) and the workstations. This table file can be opened by a corresponding button. Furthermore all station are registered. These registrations are performed by the assistant *ASSI*.

First, the number of AGVs to be moved in the subsystem may be set. Another possibility is to define what should happen after a reset. If the pop-up menu is set to TRUE, all movable units on all objects are removed after a reset. This is the default setting. If a restart is required after an error, the pop-up menu has to be set to FALSE.

User interface:



Dialog of the object *AGVS_manager*

All settings will take effect after a Reset and an Init of the system.

Statistical Evaluation

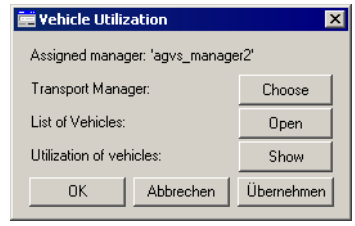
The object VehUtil

Icon:

Functionality:

The statistics of the vehicles, such as the utilization, are evaluated by this object. It has to be placed into the frame containing the *AGVS_manager*.

User Interface:



Dialog of the object *VehUtil*

Movable Units

Vehicle AGV

Icon:

Double-clicking on an AGV opens a dialog where you may adjust the velocity of the AGV.

- The following attributes are available:
  - *origSpeed* is used in the object *agvs_bend*. After passing the bend the speed of the vehicle is set to the original value, datatype *speed*
  - *transportOrder*: The *part* to be transported is stored. This ensures that the AGV load the right part, datatype *object*.
  - *vd_property*. This string attribute is necessary for the control of the AGV at stations. Possible entries are "" and "inQueue". The entry "inQueue" means that the AGV has entered the station but is not in first position. Therefore, it is not possible to assign an order to this AGV, datatype *string*

Part

Icons:

The application object *Part* is the part transported and processed in the system.

- The following attributes are required:
  - *globDestination*. This attribute contains the destination, like a working station.
  - *locDestination*. The path of the destination in a subsystem of transportation, which is reachable by the used vehicle
  - *EntityType*. This optional attribute stores the type of part. The routing plan depends on this type.

3

Conveyor

Introduction

Conveyor systems, such as conveyor belts may be modeled using the application object template Plant Simulation Conveyor. Therefore, the real layout has to be transformed to a model. Any parts to be conveyed will be moved by the shortest way to the destination. The Plant SimulationConveyor application object template may be combined with other application object templates, such as Plant Simulation AGVS for the modeling of automated guided vehicle systems.

A closed track system is assigned to a transport manager *Conveyor Manager*. In a simulation modell there can be some track systems.

For the modeling of the track system the following application objects are provided

- *Conveyor docking station* (transfer to a processing station of a produktion system)
- *Conveyor track* (modelling of the track system by polygons, consisting of stright line segments and circle segments)
- *Conveyor straight* (standardized straight line segment)
- *Conveyor curve* (standardized 90°-curve segment)
- *Conveyor branch* (branch in in 2 or 3 directions)
- *Conveyor junction* (junction from 2 or 3 directions)
- *Conveyor crossing* (crosiing of 2 tracks)
- *Conveyor Batchline* (generation of groups of parts)
- *Conveyor put in* (transfer of parts into the conveyor system)
- *Conveyor put out* (take out of parts)
- *Conveyor Interface* (transfer to other transport systems)

For the automatic registration of all building blocks there is the object *Assistant*.

Every application object offers a user interface providing dialog boxes, pop up menus, and data input lists. All application objects are open for individual modifications. Examples for the application are contained in the category *Free Libraries* of the example collection.

Examples

Examples for the application are contained in the category *Free Libraries* of the example collection.

Simple Examples

The model *ConvSys1* represents a shop floor where the sequence of all processing steps is defined in the processing station *ProcStation*. The next destination point of the part is specified in the customized attribute *globDestination*.The construction of the model is described briefly.

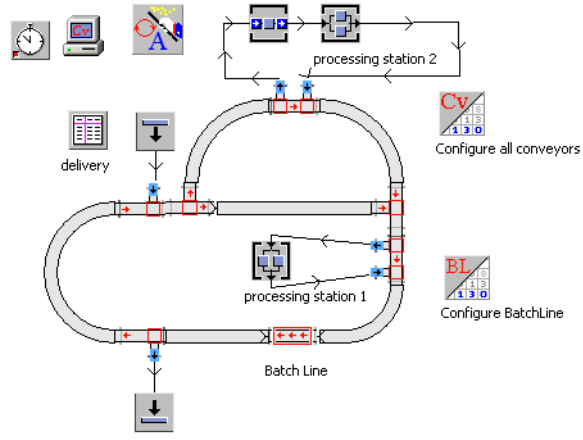


Figure 1:First application of Plant Simulation Conveyor

First of all, you have to create an empty frame. Any kind of conveyor system always consists of a central control *cv_Manager* and several transport elements. The transport elements are placed according to the layout of your system. To ensure an optimized simulation runtime you should try to construct the model with as few transport elements as possible. For this purpose you should use the objects *cv_track*.

For this model we will need processing stations that produce and destroy parts. To build them we use the basic objects *Source* or *Drain* (Processing Station with a Source inserted). The pop-up menu Time of Generation is set to *deliveryTable,* which is shown in Processing Station with a Source inserted. The attribute table (column five) contains the setting of the attribute *globDestination* of the MU (movable unit).

delivery time	MU	number	name	attributes
1:00.0000	.MUs.part	3	wave	attr
1:10.0000	.MUs.part	4	screw	attr

Delivery table of the source

In the attribute list of the delivery table (column 5) the attribute *globDestination* is generated and set.

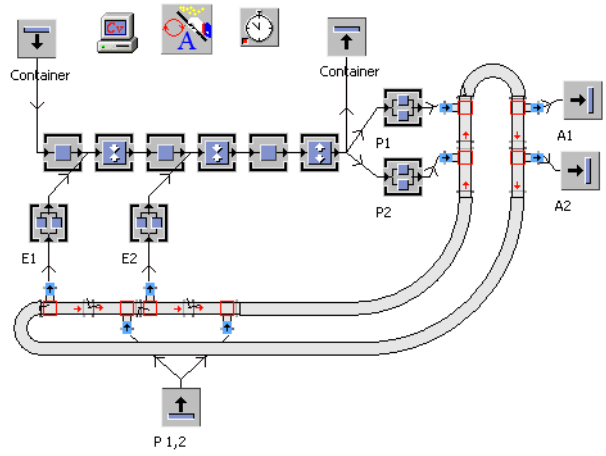
The processing is done in the processing stations. The next destination has to be assigned in the exit control *exControl* of the processing station, which are customized attributes of the processing stations.

```
is
do
      @.globDestination := drain;
      @.move;
end;
```

The simulation run may be started. Then they are transported to the first processing station. Next, the parts are transported to the *cv_batchline* where three parts may be gathered. Afterwards the parts are transported to the second processing station and then to the drain, where they are destroyed.

A model with local routing tables

The model *AssemblyLineWithConv* shows an application of the routing tables in the objects *cv_dockingStation* and *cv_PutIn*. With the help of the routing tables you are able the model the material flow without the usage of a production control and without programming methods.



Das Modell *AssemblyLineWithConv*

Whenever a part is entered into the conveyor system the next destination point of the part is determined by the part type or by the value of a customized attribute.

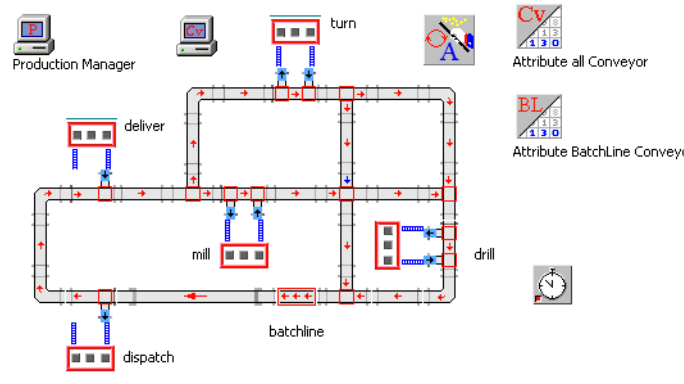
	string 0	object 1
string	Typ der BEs	Nächste Bearbeitungsstation
1	T1	root.E1
2	T2	root.E2
3		

Figure 1:The routing table determines the next destination.

Open the routing table on the tab Destination of the objects *cv_PutIn*.

An example with a production system

When various parts with several operations plans and order lists exist, the manufacturing process must be organized in a flexible way. In other words, the determination of the part´s destination cannot be realized anymore as demonstrated in the examples above. In this chapter we are going to develop a model of a production site step by step. This production site manufactures several different parts. It consists of a conveyor system, a receiving section, a lathe, a drilling machine and a dispatch station (see Layout of a Production with a Conveyor System).



Layout of a Production with a Conveyor System

First, the central control *ProductionManager* and *cv_Manager* are inserted in an empty frame. In the dialog of the *Station* you will enter the type of the station (e.g. lathe, drilling machine). Now you have to define the list of operations for each type of station by double-clicking on the icon of the *station* and clicking the button Open. After the model is constructed, you

should test for any unconnected objects. Finally, the parts to be manufactured and the routing plans are defined: Open the application object *ProductionManager* by double-clicking on its icon. A dialog box will open. Click on the button Master data to open the following table:

	string 0	table 1
string	entitytype	operationsplan
1	screw	Plan
2	wave	Plan

Table *masterdataL* of the *ProductionManager*

Here the type of the part is defined (in our example: *part* and *part1*). Select the cell in the column *operationsplan* and press *F2* to open another subtable containing the operation plan. The operation plan defines the sequence of operations the corresponding part has to pass through.

	string 1	time 2	time 3	real 4
string	operation	setup time	processing time	factor
1	deliver	5:00.0000	1:00.0000	1.00
2	drill	5:00.0000	1:00.0000	1.00
3	turn	5:00.0000	1:00.0000	1.00
4	mill	5:00.0000	1:00.0000	1.00
5	dispatch	5:00.0000	1:00.0000	1.00

The operations plan *masterdataL[1,1]*

Click on the button Orders in the dialog of the production control to display another table.

	string 1	string 2	integer 3	datetime 4	datetime 5
string	orderNo	entitytype	number	release date	due date
1	q1	wave	3	1997/04/12 12:00:00.0000	1997/04/12 14:00:00.0000
2	q2	screw	5	1997/04/12 12:30:00.0000	1997/04/12 14:00:00.0000

Costumer orders

Enter order number, type of part (here *part* and *part1*) and number of parts to be manufactured here. Now, the functionality and fitness for use may be tested separately. The stations are connected with the application objects *cv_PutIn*, *cv_PutOut* and *cv_dockingStation*.

The modeling of the described production without a transport system can be done by executing the method *workshop* in the folder *buildDEMOS*. This method will built a corresponding production model, in which you can check the functionality separately.

Description of the Application Objects

This chapter describes the functionality and the user interfaces in detail. Each application object offers a dialog box for modifying parameters. Modifications are stored by clicking on the buttons OK or Apply. When the length of the track objects is changed a new registration by the assistant is necessary. By the context menu entry Open the frame of the application object is opened.

Conveyor Objects

The following application objects are used to model conveyor elements not equipped with a lifting or turning facility.

Object cv_Track

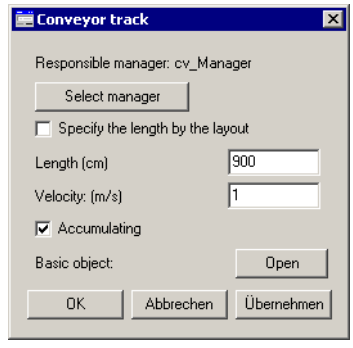
Icon:

Functionality:

The objects described here are tracks of the conveyor. For each connected track system a *cv_manager* must be assigned. Therefore, the dialog of the tracks contains an input element for the responsible manager. A manager handles all vehicles which can move on a connected track subsystem.

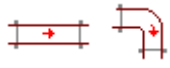
User Interface:

You are able to model the track consisting of straight line segments and circular. arces.



Dialog *cv_track*

Objects cv_Straight and cv_Bend

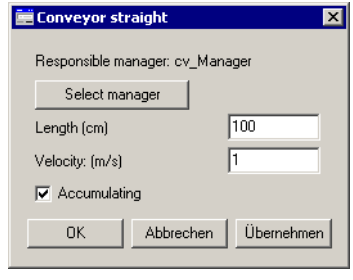
Icons: 

Functionality:

These two application objects represent a straight and a curved element of a material flow system. To minimize run-time we recommend to use as few as possible application objects *cv_Straight*. For this reason, the size of transport elements may be varied. To accomplish this, press the Shift key and the left mouse button, keep them depressed and drag the mouse.

User Interface:

You may define length, transport velocity and operation mode.



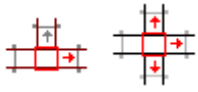
Dialog *cv_Straight*

Conveyor Elements with Lifting or Turning Facilities

Lifting or turning facilities are integrated into various conveyor elements, e.g. *cv_DockingStation* and c*v_Interface*. To model these elements, you have to define the positioning time and the reset time for lifting or turning facilities. The positioning time is needed for processing the current part and the reset time specifies the time that has to pass before a new part may be loaded. Lifting or turning facilities are fully operational if the length of the facilities is at least of the length of a single part and if the succeeding objects are conveyor elements.

Branches cv_br2 and cv_br3

Icons:

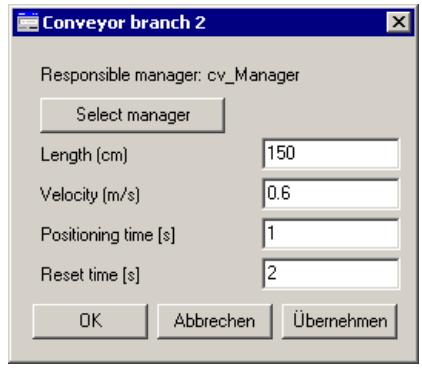


Functionality:

Lifting or turning tables may be modeled by a branching station. A single part may be moved either straight forward or sideways. It is not possible to connect a station with a branch. For this purpose the objects *cv_PutOut* and *cv_PutIn* are provided.

User Interface:

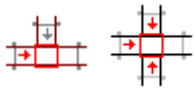
You may modify: length, velocity, positioning time and reset time.



Dialog *cv_br2*

Objects cv_junction2 and cv_junction3

Icons:

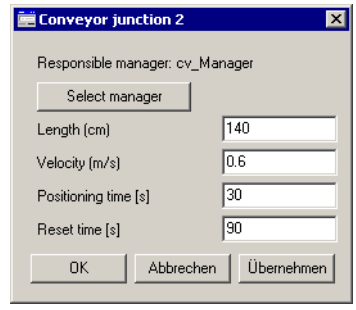


Functionality:

This object represents a lifting or turning table able to load parts from two different directions.

User Interface:

The following parameters may be adjusted: table length, velocity, positioning time and reset time. The dialog box of *cv_junction2* is similar to that of *cv_br2* (see Dialog cv_junction2). It is not possible to connect a station with a junction. For this purpose the objects *cv_PutOut* and *cv_PutIn* are provided.



Dialog *cv_junction2*

Object cv_crossing

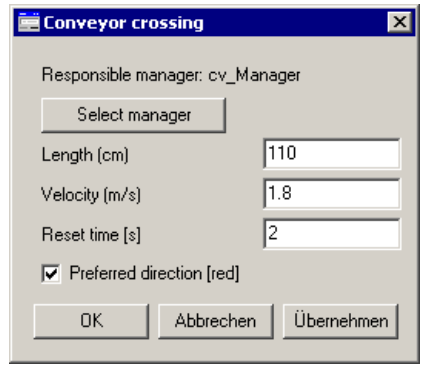
Icon:

Functionality:

This object represents a lifting or turning table which may be used as a crossing conveyor element. However, a part cannot change its direction with this element.

User Interface:

You may modify: length, entrance velocity, positioning time and reset time. You may also set the preferred direction of movement.



Dialog *cv_crossing*

The time needed for passing the crossing element is determined by the length and the conveying velocity of the following conveyor element.

Object cv_batchline

Icon:

Functionality:

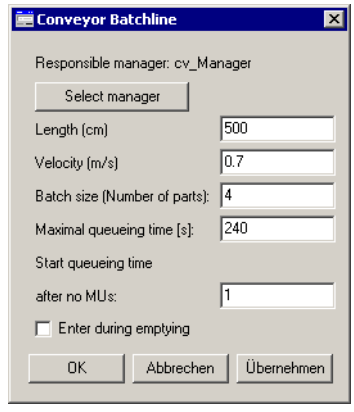
This application object organizes a batch of parts. A user-defined number of parts is collected and then this batch is sent to the next conveyor element. It is also possible to define a queuing duration. After this period of time the parts are sent to the next conveyor element. The beginning of the queuing duration may be specified separately. For example, the user

defines a batch size of five parts and activates the queuing duration after the third part. If there are no further parts entering the *cv_batchline* element within the specified duration time, only three parts will be sent to the next conveyor element. If there is a part arriving the duration time will be started again. However, if there are not more than two parts in this element they will never be sent away, and consequently, they will never reach their destination.

User Interface:

You may define the length, batch size, maximum queuing time, queuing time start and the transport velocity of this element.

The length of the batch-line and the length of the individual parts determine the number of parts to be placed on the batchline.



Dialog *cv_batchline*

Object cv_dockingStation

Icon:

Functionality:

This object represents a lifting or turning table which must be connected with a Station. A Station can be a frame or a sequence of basic objects. In the Station there must be basic object, which are able to perform the processing, like SingleProc, ParallelProc, drain, assembly and disassembly. The docking station can only connected with objects, which can take movable elements.

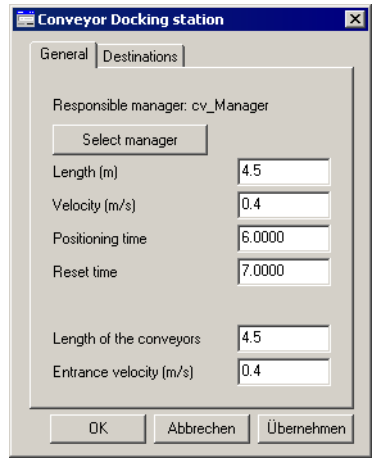
cv_dockingstation contains a loading and an unloading table. Both may operate independently. Even if the part is not being processed by the connected station it will be loaded onto the lifting table of *cv_dockingstation*. The time needed to do this is determined by the velocity and length of the conveyor element *LiftTableLeft* and *LiftTableRight* contained in the application object *cv_dockingstation*.

If one of the turning tables is turned then the *position time* is necessary. The time to put in or put out a part is determined by the length and the velocity of the objects *ConveyorBelt_PartOIn* and *ConveyorBelt_PartOut*. After such an action the *reset time* is needed. After this the next part can be transported. If a part is transported straight on the docking station then no *position time* and *reset time* is necessary. The time needed to do this is determined by the length and velocity of *LiftTableRigh, LiftTableLeft* and *conveyorBelt_Center*.

User Interface:

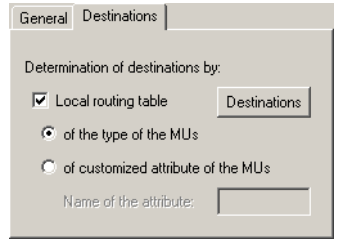
You may modify: length, entrance velocity, positioning time, reset time.

A conveyor element is inserted between the unloading and loading table. The user may adjust length and velocity of this conveyor element.



The tab General of the dialog *cv_dockingStation*

If the next destination is not determined by a production system, then you can define the destination by local routing tables according to the type or the value of a customized attribute of the part. On the tab Destination you can define whether the destination is determined by the type or by a attribute. The type is given by the customized attribute *entityType* (string). If there is no such attribute then the type is determined by the name of the part.



The tab Destination of the Dialog *cv_dockingStation*

The button Destination opens the routing table, in which you can assign the destinations.

Object cv_PutIn

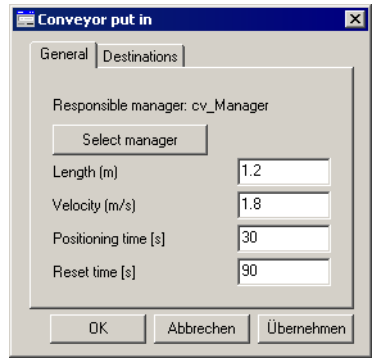
Icon:

Functionality:

This object represents a lifting or turning table that has to be connected with a station that produces parts, e.g. *source*.

In order to put this element into operation the *reset time* is needed. The time used for loading a part is determined by the velocity and length of the conveyor object. The table together with the loaded part is now moved to unload the part. This is part of the positioning time.

User Interface:



The tab General of the Dialog *cv_PutIn*

As in the object *cv_dockingStation* you can determined the next destinations for the parts by local routing tables on the tab Destination.

Object cv_PutOut

Icon:

Functionality:

This object represents a lifting or turning table which has to be connected with a station that receives parts, e.g. *drain*.

A part to be dispatched is moved on the table. Then the table is turned and the part is leaving the table. Finally, the table returns to its starting position.

This application object is similar to the object *cv_br2* except that *cv_PutOut* may be defined as destination for any part. If a part has registered *cv_PutOut* as destination, it will automatically take the shortest possible way to reach this application object.

User Interface:

You may modify: length, entrance velocity, positioning time and reset time of the table. The dialog of *cv_PutOut* is similar to the dialog of *cv_straight*.

Interface to other Transport Systems

Object cv_interface

Icon:

This application object represents an interface to another material flow system. The material flow system to be connected may be a model, either from the application object templates eM-Plant Conveyor or from eM-Plant_AGVS (Automated Guided Vehicle System).

Functionality:

The functionality is similar to the docking station *cv_dockingStation*.

User Interface:

The following parameters of the tables may be adjusted: length, entrance velocity, positioning time and reset time. Adjustable parameters of the conveyor belt are: length, velocity and operation mode.

The dialog of *cv_interface* is similar to the dialog of *cv_dockingStation*.

The Conveyor Control

Object cv_Manager

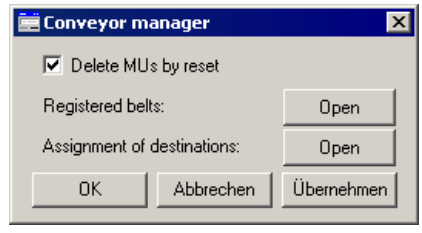
Icon:

Functionality:

The application object *cv_Manager* provides lists with current data for all available objects, interfaces and destinations possible. Any changes made by the user are recorded and managed by *cv_Manager*. Let us say that *cv_dockingStation* is connected with *Station* and that both objects are registered in a destination list.

User Interface:

You may define the reset behavior of the model: Either all parts are deleted or not. You may also set whether a new initialization takes place after modifications.



Dialog *cv_Manager*

The object Part

Icon:

The application object *part* represents the items to be manufactured. Each part to be transported must have the attributes *globDestination* and *locDestination*. Die Assignment of the local and global destination can be checked in the transport manager *cv_Manager*.

globDestination:

This attribute contains the path to the next working station. Datatype: *object*.

LocDestination:

It determines a local destination in the transport system of the part, like a docking station. Datatype: *object*.

EntityType:

This optional attribute determines the kind of part. It is evaluated by the local routing tables.Datatype: *string*.

4

EOM

Introduction

The application object template Plant Simulation EOM (Electric Overhead Monorail) has been developed to reproduce a discontinuous feeding system by using an electric overhead monorail (EOM) system. To model a transport system and the real-life behavior of an *Electric Overhead Monorail* you to will have to create the layout in Plant Simulation using the objects provided: straight, curve, loading and unloading station and switch. Enter parameters specific to your needs and needed to control your system, into tables and global variables of the objects inserted into your models. You also have to enter the number of vehicles required for filling the transport orders as well as their placement within the system. The transport matrix provides another convenient way of simulating throughput. It allows you to define the material flow by any number of loading stations to any number of unloading stations.

The application object template *Electric Overhead Monorail* has been developed with Plant Simulation and requires an Plant Simulation development license and knowledge in Plant Simulation.

Sample Model

The model *EOMsystem* provides a sample of a manufacturing chain where parts that are produced are passed through a number of processing stations according to a pre-defined schedule. The material flow system is realized by the elements of the application object template Electric Overhead Monorail. The production control determines the next target for the vehicles. The time it takes to move from station to station is realistically duplicated by traveling speed and by blockages.

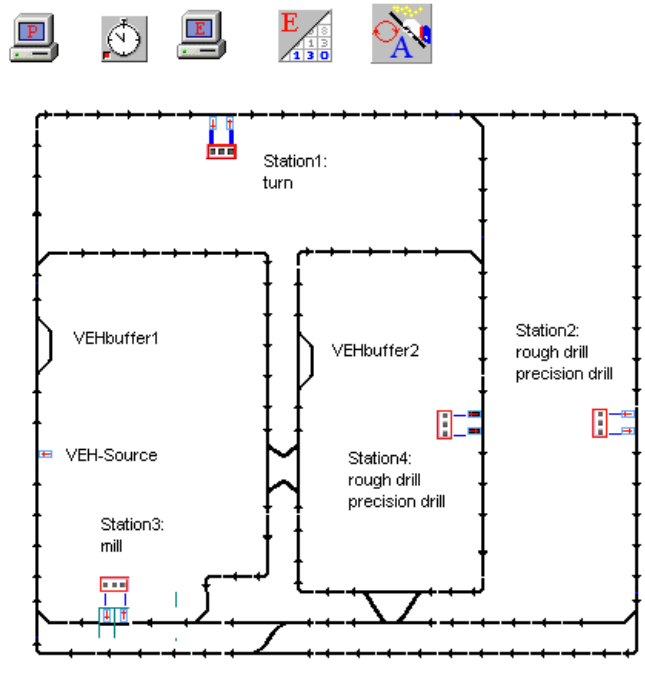


Figure 1:Layout of a production chain using the Electric Overhead Monorail

The model consists of the following components:

- The production control *ProdMgr* (customer orders, production schedules of the parts, order registration).
- The processing station *Station* for processing parts.
- Objects of the *Electric Overhead Monorail* for modeling the transport route.
- The EOM dispatch station *EOMcontrol* for managing vehicles and parts.

The following paragraphs will give a short description of the course of proceedings:

In the production control *ProdMgr* you use a schedule and an order list to define which part will be loaded when and where and will be processed in which way. All processing stations are controlled by the *ProdMgr*.

The empty vehicles are fed into the system by the vehicle source. The source produces vehicles circulating in the system as well as vehicles waiting in buffers, and in loading and unloading stations.

The loading and unloading station represent the interfaces between the processing stations and the feeding system. An allocation of the processing station to the docking station is only necessary for the unloading stations, as the processed parts are assigned their next processing station according to the production plan by the *ProdMgr*. You have to set the allocation of the stations to the docking station in a table of the *EOMcontrol*.

How to create a model and which parameters have to be set for the individual objects is described below. You find a detailed description of all objects in chapters 4 and 5.

The following chapter describes the steps for creating the model depicted in Layout of a production chain using the Electric Overhead Monorail. The completed model *EOMexample* is located in the object library.

EOM Objects

Before inserting objects often used routing elements are parametrized in the object library: the length was set to 5 m, the capacity of all objects, except the buffer route in empty vehicle buffers, is 1. The length of the objects is identical with the blocking route. You, the user have to enter all additional values into lists, tables and global variables (green in the objects).

Loading Stations

The entrance buffers of loading stations are connected with the exit buffers of the respective processing station via a connector. The global variable *loadtime* holds the time it takes to load a vehicle. The list *StationsL* holds the entries of preceding frames where available empty vehicles are requested by processed parts. If available vehicles without a transport order are to be stopped when passing the station and be put in a waiting status, the global variable *veh_wait* has to be set to TRUE.

	object 1
1	.EOMsystem.EOMunload1
2	.EOMsystem.VEHbuffer1

List *StationsL* of loading station 1

Unloading Stations

The exit buffers of unloading stations are connected with the entrance buffers of the processing stations via a connector. The *EOMcontrol* performs the logic allocation of unloading and processing stations. The variable *unloadtime* holds the time it takes to unload a vehicle. To make the vehicle stop at the station after unloading, the variable *veh_wait* in the unloading station is set to TRUE. The list *StationsL* contains the succeeding loading stations where continuing vehicles after unloading search for workpieces to be picked up. The buffer list *BufferL* holds the nearest buffer.

	object 1
1	.EOMsystem.EOMload2
2	.EOMsystem.EOMload3
3	.EOMsystem.EOMdock4

	object 1
1	.EOMsystem.VEHbuffer2

Lists *StationsL* and *BufferL* of unloading station 2

Docking station

The combined loading and unloading station *EOMdock4* has been connected to processing station 4. It has the same functionality as loading and unloading stations. You have to set the variables *loadtime* and *unloadtime*. Additionally you have to fill out the lists *VEHStationsL* (for requesting available empty vehicles) and *CARGOStationsL* (for searching for waiting parts). The variable *veh_wait* has been set to FALSE in this object, so that vehicles unloaded here continue on their way. Arriving empty vehicles will pass this station without stopping. Before continuing, unloaded vehicles search for an empty vehicle buffer to call on in the buffer list *BufferL*.

	object 1
1	.EOMsystem.VEHbuffer2
2	.EOMsystem.VEHbuffer1
3	.EOMsystem.EOMunload1

	object 1
1	.EOMsystem.EOMload3
2	.EOMsystem.EOMload1
3	.EOMsystem.EOMload2

Lists *VEHStationsL* and *CARGOStationsL* of *Dockstation4*

EOMcontrol

The *EOMcontrol* contains the table *STATattachL*, that allocates processing stations to unloading stations. This is necessary because the *ProdMgr* assigns the parts to be processed the name *GlobDestName* of the station as next target while the vehicles of the application object template use the names of the unloading stations *LocDestNameUnload*. The *EOMcontrol* also contains the frame *CARGOinput* for feeding the parts via a transport matrix. This frame is not discussed here, as it is not used in the sample model.

string	object 0	object 1
0	GlobDestName	LocDestNameUnload
1	.EOMsystem.Station1	.EOMsystem.EOMunload1
2	.EOMsystem.Station2	.EOMsystem.EOMunload2
3	.EOMsystem.Station3	.EOMsystem.EOMunload3
4	.EOMsystem.Station4	.EOMsystem.EOMdock4

Table *STATattachL* of the *EOMcontrol*

Straights and Curves

As all parameters of the objects *EOMcurve* and *EOMstraight* have already been set in the object library, you do not have to enter anything when inserting them into a frame. Differing lengths, also representing blockages, may naturally set anew any time in the model. As vehicles move slower in curves than on straights you may enter the curve speed via the customized attribute *vCurve*.

Branching

For branching operations fill out the destination list *DestL* and the route table *WayDataL*. The destination list contains all local destinations that may be reached via the corresponding fork. Enter the length of the routes and the blocking times for prematurely unblocking the route for the corresponding routes. Once the blocking time has passed, the entrance ob-

ject of the fork is unblocked and vehicles may enter before this vehicle has reached the end of its blocking route, i.e. the end of the exit object. The blocking route control is discussed in Chapter 3.

	object 1
1	.EOMsystem.EOMdock4
2	.EOMsystem.EOMload3
3	.EOMsystem.EOMunload3
4	.EOMsystem.VEHbuffer2
5	.EOMsystem.EOMload1
6	.EOMsystem.EOMunload1

Destination list *DestL* of the fork *EOMbranch2*

	string 0	real 1	time 2
string 0	Route	distance	blockingtime
1	way_in-way_out1	5	0.0000
2	way_in-way_out2	6	0.0000
3			

Routing table *WayDataL* of the fork *EOMbranch2*

Junctions

For junctions you also have to fill out the table *WayDataL*. In addition to the values known from branching, junctions have the additional parameter *priority* to set right of way rules. By default priorities for both routes are set to 1, the highest priority available, to realize the FIFO rule (first-in-first-out). To assign the right of way to a certain vehicle when several vehicles want to enter, you have to set the priority of the other route to a value greater than 1, i.e. lower the priority for that route.

	string 0	real 1	time 2	integer 3
string 0	Route	distance	blockingtime	priority
1	way_in1-way_out	5	0.0000	1
2	way_in2-way_out	6	0.0000	1

Routing table *WayDataL* of the junction *EOMjoin23*

Switches

As switches are a combination of forks and junctions, you have to fill out the destination lists *DestL* and the routing tables *WayDataL*.

	object 1
1	.EOMsystem.VEHbuffer2
2	.EOMsystem.EOMdock4

Destination list *DestL* of the switch *EOMswitch2*

	string 0	real 1	time 2	integer 3
string 0	Route	distance	blockingtime	priority
1	way_in1-way_out1	8	0.0000	1
2	way_in1-way_out2	10	0.0000	1
3	way_in2-way_out1	10	0.0000	1
4	way_in2-way_straight	5	0.0000	0
5	way_straight-way_out2	5	0.0000	1

Destination list *WayDataL* of the switch *EOMswitch2*

Buffer

The buffer *way_buff* differs from all other routing elements by its unlimited capacity, described by the entry -1. The number of vehicles staying on a buffer depends on the length of the buffer and the stack length of the vehicles, defined by the customized attribute *stackLength*. As the length of the buffer in our sample model is 10 m, we insert the routing element *way* being 5 m long besides *way_straight* to compensate for the length.

You always have to fill out two lists in buffer objects: the list *BufferL* holds an entry for the buffer that is going to be the target when the active buffer is occupied. Enter the succeeding loading stations into the list *StationsL*. As soon as a vehicle reaches the frontmost buffer position, the program checks if a part is ready to be moved on is located in one of these stations. In the object *way_straight* the attribute *prio2way_out* has been set to 2, so that a vehicle may always leave the buffer as it has been assigned a transport order.

	object 1
1	.EOMsystem.VEHbuffer2

	object 1
1	.EOMsystem.EOMload1
2	.EOMsystem.EOMload2
3	.EOMsystem.EOMdock4

BufferL and *StationsL* of the empty vehicle buffer *VEHbuffer1*

Vehicle Source

The table *VEHtableL* in the *Vehicle Source* has been filled in with the vehicles to be fed into the system. Besides creating a varying number of vehicles in both buffers, the source feeds 6 circulating vehicles into the system every 15 seconds. Note that in front of loading and unloading stations only a single vehicle may be waiting.

string	object 1	object 2	integer 3
0	v_source	v_typ	v_num
1	.EOMsystem.VEHinput	.veh	6
2	.EOMsystem.VEHbuffer1	.veh	2
3	.EOMsystem.VEHbuffer2	.veh	3
4	.EOMsystem.EOMload1	.veh	1
5	.EOMsystem.EOMdock4	.veh	1
6			

Vehicle source *VEHtableL* in the object *VEHinput*

Vehicle Parameters

The application object template contains the vehicle that is derived in any number by the source and fed into the system. Before simulation start the following parameters were set:

- length of vehicle: 1 m
- speed of vehicle on straights: 1 m/s
- stack length *stackLength* of the vehicles in buffers: 3 m
- speed of vehicle *vCurve* in curves: 0.5 m/s

Inserting the Object Protocol

Errors that occurred during the simulation run are written to message, warning and error lists. These lists are kept by the object *Protocol*.

Now that we chose all necessary settings, let us start the model via the *EventController*.

Note: To limit the complexity of the model not all objects provided were used.

Basic Course of Events

The following chapters describe the functioning of the *Electric Overhead Monorail*.

Disposition of Vehicles

The interaction between available EOM vehicles and freights to be transported is the main subject of the material flow system *Electric Overhead Monorail*. It is of secondary importance if transport orders to be processed are assigned by the production control *ProdMgr* or by a transport matrix. The only difference is that the *ProdMgr* may cause the parts to be transported to several processing stations in sequence, as the transport matrix just moves the parts from a source station to a destination station and are then destroyed. The request of vehicles and the search for waiting parts is executed by lists that are created in the corresponding object and are to be filled in by the user.

An *EOMvehicle* may have the following states of loading and disposition, marked by different colors:

a)loaded

- having transport order, on its way to the destination (unloading station) gray

b)not loaded

- circulating in the system without destination - green
- circulating in the system with the destination "next available buffer" - green
- waiting in a loading or unloading station - blue
- finished, ready to leave buffer - red
- assigned location in buffer - red
- having transport order, on its way to the destination (loading station) - yellow

To fulfill the existing transport order, the application object template *Electric Overhead Monorail* provides the following strategies:

- A part ready to be picked up is going to be moved to the entrance buffer of a loading station and requests an available empty vehicle. All available empty vehicles are registered in a vehicle list in the *EOMcontrol*. When no empty vehicle is found, the processed part is registered in the *EOMcontrol* in a parts list. To limit the space where empty vehicles are searched, the loading stations have and additional list where you may enter the neighboring stations. Initially only these stations are searched. Only no empty vehicle was detected there, the remaining stations are searched.
- Empty vehicles that are waiting search for processed parts that are registered in a parts list of the *EOMcontrol*. Once a part is located, its entry is removed from the list and the vehicle move to the corresponding loading station. If the vehicle's search was in vain, it will be registered in the vehicle list of the *EOMcontrol*. To limit the search for parts to be picked up, these objects also have a stations list, where you enter the loading stations that have priority during a search.
- A part ready to be picked up is loaded onto a circulating empty vehicle without a transport order that happens to pass the loading station. The part is loaded and transported to the next destination of the vehicle. Any existing transport order for a vehicle issued by the part will be deleted.
- Loaded vehicles transport the parts to the corresponding unloading stations and pass them on to the exit buffer of the *station*. Depending on the chosen settings, the vehicle either stops after unloading or looks for a part to be picked up. If it did not find a transport order, it will move into an empty vehicle buffer (provided the buffer list was filled in) or circulates in the system and moves into the next available buffer. Circulating vehicles may also be put in waiting mode by setting a variable in front of the loading station.

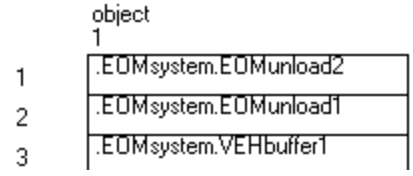
You may define different loading and unloading times for all loading processes.

Event: part is looking for a vehicle

After processing has been finished in the processing station, a part is assigned a new destination according to the production plan and is passed on to the entrance buffer *part_avail* of the associated loading station *EOMloadx/EOMdockx*. Once there, the objects (*VEHbuffer, EOMunload, EOMload, or EOMdock*) entered in the list *StationsL/VEHStationsL* are searched in sequence for available empty vehicles. If no vehicle has been found in the entered station or the station list *StationsL* does not contain an entry, the transport order is passed to the *EOMcontrol* via the list *EOMcontrol.VEHavailableL* and extended to the entire system. After a successful search the vehicle is ordered and its entry is deleted from the list that holds all available vehicles: *EOMcontrol.VEHavailableL*. The variable in the loading object *veh_required* is set to the ordered vehicle for test reasons. If the waiting part has already been picked up by a circulating vehicle before the ordered vehicle arrived, the order is canceled by this variable and the vehicle is set to "free". The variable is also reset when the vehicle called arrives. Before continuing the user-defined loading time *loadtime* is taken into consideration.

As the list *StationsL* is processed in sequence, first the buffers and unloading stations located before the loading station are entered, to keep the distances traveled as short as possible.

If *StationsL* does not contain an entry or if no empty vehicle was found, the waiting part is entered into the list *EOMcontrol.CARGOwaitingL*.



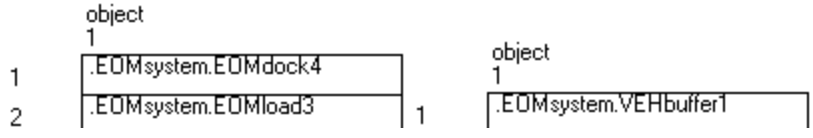
List *StationsL* of a loading station

Event: vehicle is looking for an order

Vehicles located at the first buffer position in *VEHbufferx.way_buff* or those just unloaded in the unloading station *EOMunload/EOMdock* (and which should not stay in a waiting position, defined by the global variable *veh_wait)*, independently search for a new transport order. These object also contain a list *StationsL/CARGOStationsL*, where you enter the neighboring loading stations of type *EOMload* or *EOMdock*. According to the sequence entered the stations are searched for parts to be picked up. After a successful search the global variable *veh_required* in the destination object is set to the searching vehicle that is on its way to pick up the part. If the waiting freight has already been picked up by a circulating vehicle, the order is canceled by this variable and the vehicle is set to "free" again. The variable is also reset after the ordered vehicle has arrived and the part has been loaded.

If the search in the entered stations turned out to be without success, the list *EOMcontrol.CARGOwaitingL* is used to automatically search the entire system for the parts that have been waiting the longest (FIFO rule, i.e. independent of their location). The entry of any part found is deleted. Enter the neighboring loading stations in the list to realize the shortest transport routes possible.

If there is no waiting part in the entire system, a vehicle located in a buffer *VEHbufferx* is entered into the vehicle list *EOMcontrol.VEHavailableL*. A vehicle in an unloading station searches for a buffer it may target in the list *BufferL*. Is the buffer entered here, them it will become the next target of the vehicle. Otherwise the vehicle starts to circulate in the system and is going to look for the next available buffer by itself. When a circulating vehicle passes a loading or docking station where the variable *veh_wait* is set to TRUE, the vehicle remains at that station in waiting status and is going to be registered with the *EOMcontrol* in the list of all available vehicles (*EOMcontrol.VEHavailableL*). It stays at the station until it is unloaded there, is requested by another station or is going to be moved on by a succeeding vehicle.



Lists *StationsL* and *BufferL* of an empty vehicle buffer

Event: circulating vehicle takes on a waiting part

Whenever a circulating empty vehicle passes a loading station or when a vehicle is on its way to a buffer, it checks if a waiting part is located in the entrance buffer of the loading station *part_avail*. It then moves to pick up this part. The variable *veh_required* now checks if the part has already ordered an empty vehicle from a buffer or from another station. If this is the case the vehicle ordered is set to "free" again. If the waiting part did not order a vehicle, it is going to be registered in *EOMcontrol.CARGOwaitingL*.

Event: loaded vehicle passes on freight

At unloading stations *EOMunload/EOMdock* loaded vehicles compare their target with the current location. When they are the same, they pass their freight to the exit buffer *part_unload* of the station. The time it take to unload is set by you in the variable *unloadtime*. If the variable *veh_wait* is set to TRUE, the unloaded vehicle remains in waiting status and is

registered in the vehicle list *EOMcontrol.VEHavailable*. Here, it may be requested by parts ready to be transported or it may be moved on by arriving vehicles. If the vehicle is not supposed to stop after unloading, it searches for a new transport order by itself, moves on to a buffer or circulates in the system.

Block Sections

To prevent vehicles from colliding/piling up, the *Electric Overhead Monorail* uses the railway blocking principle. A *block section* is defined as an area, where a single vehicle may stay. For this reason the track of the entire system is made up of segments, each of which may represent a block section. The length of the basic object *Track* is the length of the block section. For safety reasons, the railway blocking always keeps at least one block section free between moving trains: a vehicle only releases the entrance the preceding block section when it has reached the end of its own block section and is ready to enter the next piece of track. It does not matter if this vehicle may move on immediately or if it is blocked by a vehicle in front of it, as only one vehicle may stay on a block section at any one time. The vehicle that reaches the end of a block section triggers the release of the section it has left and the blocking of the section it entered.

The block section strategy is realized in all objects, except for the stacking of vehicles in buffers and the passing through branches, switches and junctions in the track. Branch, switch and junction represent entrances of block sections, but you may define different track lengths, blocking times and priorities.

Track Clearing Strategy

As mentioned above, vehicles may stay in unloading stations (*EOMunload*), in loading stations (*EOMload*) or in combined stations (*EOMdock*) according to the variable *veh_wait* until they are requested by a part. As a waiting vehicle would block all succeeding vehicles, we have introduced the global track clearing strategy *veh_clear* in the object *EOMcontrol*. For each blockage a blocked vehicle checks the status of the blocking vehicle. If the blocking vehicle itself is blocked, it means that either a blockage occurred or the vehicle is loaded or unloaded. If on the other hand the vehicle is *waiting*, the track clearing strategy takes effect, and sends the waiting vehicle on its way and deletes its entry from the list *EOMcontrol.VEHavailableL*.

Buffers

When reaching the buffer entrance object *way_in* circulating vehicles and vehicles with any buffer (*VEHbuffer)* as target check the occupancy of the buffer track *way_buff,* where the empty vehicles are lined up according to the stacking distance. The stacking distance *stackLength* is a customized attribute of the vehicles that has to be set by the you when using buffers. The occupied buffer length in these objects is checked by the global variable *occuLength*. Set the length of the entire buffer in the track object of *way_buff*.

All vehicles that are located in first position of a buffer, which did not find a transport order when arriving, are registered in the *EOMcontrol.VEHavailableL*. They only leave their position when they are requested by a part to be transported. When a vehicle cannot leave immediately after an order because the exit object *way_out* is occupied at the moment by a passing vehicle or has been deactivated, the vehicle waits for this object to be released. If the transport order is deleted during this waiting time because a circulating vehicle loaded the part that issued the order, the vehicle remains in the buffer and is going to be registered again as being available.

When a vehicle cannot enter a buffer because it is occupied already, the list *BufferL* in the objects *VEHbuffer* is searched for a new buffer. When a buffer is found there, it is going to be the new target of the vehicle. If no buffer is found, the vehicle will circulate in the system and searches for an empty buffer by itself.

The straight buffer passage consists of the objects *way* and *way_straight*, where blocking applies. If you are going to lengthen the straight piece of the buffer track proper *way_buff* by inserting block sections, you have to insert them between the existing objects *way* and *way_straight*.

Switches and Confluences

Switches and confluences share the characteristic that vehicles coming from different directions intend to enter the same object at the same time or need that object as a point of intersection until the end of the entire block section. The FIFO rule prevents two vehicles from entering at the same time. The vehicle arriving first reserves all objects it need for continuing on its way. When the vehicle cannot enter immediately because of a reservation or a blocking, the exit control *way_inx_oCtrl* of the entrance object *way_in*x the vehicle passed through is entered into the table *WaitL* of the targeted exit object *way_out*. Whenever an exit object is activated anew (which is triggered by an arriving object at the end of the succeeding block section) the program executes the control waiting the longest in the waiting list *WaitL*. This causes all waiting vehicles to be channeled through the switches one after the other.

To realize right of way rules, you may set a priority attribute in the track table *WayDataL* for switches and confluences. By default the attribute is set to 1. When arriving vehicles have identical priorities, the FIFO rule applies. When they have different priorities, the controls to be executed are entered into the table *WaitL* according to their priority and are going to be executed after activating the exit object one by one. The value1 stands for the highest priority.

For switches and confluences you have to set the attributes for the priority and the track length of all pieces of track to pass through. During the simulation run the track length of the targeted exit *way_out* you entered is used. In curves the program automatically switches to curve speed. Enter a time for each track after which the object is released in the column *blockingtime* of the table *WayDataL*. This time starts when the vehicle exits its entrance object. Any succeeding vehicle may enter the entrance object at an earlier point in time. A vehicle arriving from a different direction may pass the crossing point on its way to another exit. Setting the blocking time to the default value 0 means that normal blocking behavior applies. The entrance object is only released, when the vehicle reaches the end of its block section. Enter the track length of the entrances directly into the objects.

	string 0	real 1	time 2	integer 3
string 0	Route	distance	blockingtime	priority
1	way_in1-way_out1	8	6.0000	1
2	way_in1-way_out2	10	8.0000	1
3	way_in2-way_out1	10	8.0000	2
4	way_in2-way_out2	12	10.0000	2

Table *WayDataL* of a switch object

Pile-ups in Front of Confluences

To increase throughput while passing confluences (which is especially important for jobs where time is of the essence) vehicles coming from the same direction are grouped. This prevents switches from having to change directions continually for vehicles coming from several directions. Use the object *EOMjoinG* to model this kind of behavior: it allows vehicles piled up to pass in a pile until there is a gap or the number of vehicles you entered is reached. A pile run is triggered by a blocked vehicle on one of the entrance tracks (*way_in1* or *way_in2)* whose controls are executed after activating the exit object *way_out*. The program checks if another vehicle arrived after the blocked vehicle. Then the entrance objects these vehicles passed through are released before reaching the block sections according to the release time *freeifgroup* taken from the table *FreeifgroupL* as is the case for normal confluences. Opposed to that the predecessor is checked for a vehicle ready to leave as soon as the previous one is released, and the vehicle ready to leave is assigned to the vehicle pile. Now we have the problem that new vehicles may advance continually because the driving speed across the switch may be reduced considerably in the curve or the track length across the switch may be considerably longer than in the entrance object. To prevent this, you may limit the size of the pile by using the variable *group_max*. You may set the release time for pile driving for each track object separately. When crossing the exit objects, the pile state is going to be canceled.

If pile driving is not set, this object works as a normal confluence with two additional block sections in front of the entrances and after the exit. You set the pieces of track across the switch (the length of the track *way_out*), the blocking times

for the normal blocking behavior and the priorities in the table *WayDataL*. Set the track lengths of all tracks directly in the objects.

string	string 0	time 1
0	Building block	freeifgroup
1	way_in1	3.0000
2	way_in1_gp1	3.0000
3	way_in1_gp2	3.0000
4	way_in2	3.0000
5	way_in2_gp1	3.0000
6	way_in2_gp2	3.0000
7	way_out	3.0000
8	way_out_gp1	3.0000
9	way_out_gp2	3.0000

Table *FreeifgroupL* in a pile object

Switches

The application object template *Electric Overhead Monorail* realizes branching tracks in two or three different directions. Enter the targets to be reached by the pieces of branching track in the target list *DestL*. Enter the length of the exits and the blocking times in the table *WayDataL*, enter the length of the entrance in the object itself. The entrance object *way* is deacivated after the exit of the vehicle until it reaches the targeted exit object *way_outx* or until the blocking time elapses. Then it is activated again.

Lifting Stations

Lifting stations are branching stations in a vertical direction after the vehicles have to share the lifter. Arriving vehicles use the lifting bridge either to pass a block section or to be lifted to a different level. During lifting the corresponding other entrance tracks are blocked. Enter the exit control of the blocked entrance track into the waiting list *WaitL* accordingly. As soon as the lift is available again, it is going to be requested by the blocked vehicle and reserved. When vehicles arrive at both entrances during lifting, a user-defined right of way regulation from the table *RightofWayL* determines who is going to be served first. The program compares pairs of targeted tracks of the waiting vehicles in this table and then decides who may order preferred depending on the current position of the lifting bridge and the priority. If two or more vehicles have the same priority, the FIFO rule is used. Otherwise the value 1 stands for the highest priority. Set the length

of all tracks directly in the objects. The lifting time *lift_time*, and any waiting position of the lifter that may have to be targeted are set by the global variable *lift_waitPos*.

string	string 0	integer 1	integer 2
0	Route	lift_in_bottomPos	lift_in_topPos
1	way_inB-way_outB	1	1
2	way_inT-way_outT	1	1
3	----------		
4	way_inB-way_outB	1	1
5	way_inT-way_outB	1	1
6	----------		
7	way_inB-way_outT	1	1
8	way_inT-way_outT	1	1
9	----------		
10	way_inB-way_outT	1	1
11	way_inT-way_outB	1	1

RightofWayL in a lifting station

Feeding of Vehicles

The application object *VEHinput* is used for feeding empty vehicles into the transporter system. Empty vehicles may either be created in the buffer objects, in the loading and unloading stations, in front of the combined docking station in waiting position, or may be fed into the system as circulating vehicles without a target definition by the source *VEHinput*. The vehicles in the buffer are lined up according to the stack distance set in *stackLength*, while a single waiting vehicle is created in front of the passing stations.

Loading of Orders

Instead of using a combination of production control and processing stations, you may also use the object *CARGOinput* to define the transports to be made. In a transport matrix the loading stations represent the sources where the orders are loaded, while the unloading stations represent the drains, where the parts are destroyed after unloading. The sources form the row index, while the drains form the column index. If you do not intend to use the matrix you created, because you want to use the production control, to have to set the global variable *matrix_flag* to FALSE. The part to be loaded is defined by the variable *cargo*. Enter the number of transports per hour.

In the loading station the transports are passed to *part_avail* via the object *cargo_source* and *cargo_enter* taking the processing times in *ProcTimeL* into account. In *part_avail* they are taken over by the vehicles. The object *CARGOinput* may either be inserted in the *EOMcontrol* or directly in the model.

obje	object 0	integer 1	integer 2
0		.EOMtest.EOMdock1	.EOMtest.EOMdock2
1	.EOMtest.EOMdock1		6
2	.EOMtest.EOMdock2	5	
3	.EOMtest.EOMdock3	12	
4	.EOMtest.EOMdock4		6

Transport matrix *TransportM* in *CARGOinput* for loading orders

The Transport System Electric Overhead Monorail

Use the application object Plant Simulation EOM to comfortably reproduce any conveyor system in Plant Simulation.

When developing the objects we aimed at creating as flexible a basis for modeling a transport system as possible.. Regarding the functionality proper, we attempted to keep the objects as basic and simple as possible. Basic processes that are identical for many tasks are already implemented, so that you only have to model details specific to your system.

The objects *ProdMgr* and *station* enable you to model a production system using order lists and working plans. The work pieces are represented by the object *part*. This object is part of the transport system and of the production. It is described in chapter 5.

Description of Objects Vehicles Travel On

Vehicles travel on the objects described here. For all of them the features described above, such as blocking and clearing strategies, etc., apply, so that they will not be described here again. We will describe though actions the user has to take, such as entering the length for all track elements. For some objects (switch, confluence and branching) you set the length of the exits employing the track table *WayDataL*, as these require additional data. The capacity of the track objects is always 1, except for the buffer track *way_buff,* and may not be changed.

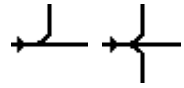
Straight - EOMstraight

Use this object to model straight track elements. The length entered here always corresponds to a single block section, as it only accepts a single vehicle You may model four different directions with it. You recognize the original object by a dot in the top left corner.

Curve - EOMcurve

Use this object to reproduce curves in a track system. As curves are normally passed with a slower speed, the curve speed is reduced by the customized attribute *vCurve* in the entrance control *way_iCtrl*. In the exit control *way_oCtrl* the speed is then reset to its original value *vOri*. You may model four different directions with it. You recognize the original object by a dot in the top left corner.

Branch - EOMbranch2 / EOMbranch3



Use this object to model branching operations in two or three directions. For the program to be able to recognize all destinations to be targeted by the vehicles using the corresponding exits, you have to enter them with their absolute path into the destination lists *DestL*. Enter the lengths of the block sections that have to be traversed, into the track table *WayDataL*, and the blocking times for prematurely releasing the entrance object *way_in*.

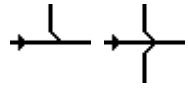
	object 1
1	.EOMsystem.EOMdock4
2	.EOMsystem.EOMload3
3	.EOMsystem.EOMunload3
4	.EOMsystem.VEHbuffer2
5	.EOMsystem.EOMload1
6	.EOMsystem.EOMunload1

Destination list *DestL* for branches

	string 0	real 1	time 2
string 0	Route	distance	blockingtime
1	way_in-way_out1	5	3.0000
2	way_in-way_out2	6	4.0000

Track table *WayDataL* for a branch going in two directions

Confluence - EOMjoin2 / EOMjoin3



Use these objects to model joining operations from two or three directions. When several vehicles arrive at the same time from different directions, the vehicle arriving first takes priority (FIFO rule). When a certain direction should take priority after a blockage, the priority rule applies . Enter the parameters into the track table *WayDataL*.

	string 0	real 1	time 2	integer 3
string 0	Route	distance	blockingtime	priority
1	way_in1-way_out	9	6.0000	1
2	way_in2-way_out	12	6.0000	2

Track table *WayDataL* for a confluence from two directions

Pile - EOMjoinG

In the object *EOMjoinG* groups of vehicles (piles) are formed. This prevents the system from having to switch directions constantly in confluence objects. Fill out the track table *WayDataL* for a normal passage without piles. Enter the release times of the individual objects into the table *FreeifgroupL* for the passage of piles. This enables groups of vehicles to pass through the entire object quickly and shortens the waiting time of the succeeding vehicle. You may enter a different release time for each track object.

string 0	string 0 Building block	time 1 freeifgroup
1	way_in1	3.0000
2	way_in1_gp1	3.0000
3	way_in1_gp2	3.0000
4	way_in2	3.0000
5	way_in2_gp1	3.0000
6	way_in2_gp2	3.0000
7	way_out	3.0000
8	way_out_gp1	3.0000
9	way_out_gp2	3.0000

Table *FreeifgroupL* in a pile object

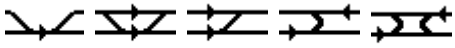
Lifting station - EOMlift

Provided the lifting bridge is free, vehicles may enter a lifting station immediately after arriving. According to their destination, which is compared with the entries in the destination list *DestL*, they will continue straight on or are lifted to a different level. Any vehicles arriving while the lifting bridge is occupied are blocked and entered into the waiting list *WaitL* according to their priority. After the lifting bridge has been released, it is going to be requested by and reserved the vehicle with the highest priority. You will have to fill out the right of way table *RightofWayL* (depending on the current position of the lifting bridge and two possible track combinations). Additional parameters you have to enter are the lifting time *lift_time* and the waiting position of the lifting bridge *lift_waitPos* when it is resting. When you do not set this variable, the lifting bridge remains in the position it held when the last vehicle left it.

string 0	string 0 Route	integer 1 lift_in_bottomPos	integer 2 lift_in_topPos
1	way_inB-way_outB	1	2
2	way_inT-way_outT	2	1
3	----------		
4	way_inB-way_outB	1	2
5	way_inT-way_outB	2	1
6	----------		
7	way_inB-way_outT	1	2
8	way_inT-way_outT	2	1
9	----------		
10	way_inB-way_outT	1	2
11	way_inT-way_outB	2	1

Table *RightofWayL* in a lifting station

Switches - EOMswitch1 ... EOMswitch5



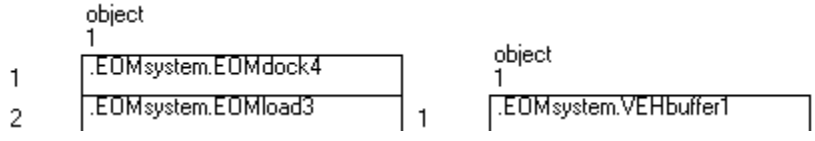
In general, the same rules apply for the switch object as do for confluences, with the difference that the priority for entering in case of two exit objects may be set differently. You may thus set the priority for each entrance object *way_in* in relation to each exit object *way_out* via the table *WayDataL*.

Use the blocking variable *blockingtime* (data type *time*) to release the blocked entrance object before the vehicle arrives at the end of the block section at the exit object. When using a blocking time you of course have to take into account that the time entered is shorter than the required driving time. Enter all destinations being targeted by the vehicles via the corresponding exits into the destination list *DestL*.

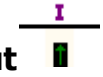
Buffer - VEHbuffer

Unused empty vehicles are kept in readiness in the buffer object. All arriving empty vehicles without destination or with the local buffer as destination, will enter *way_buff*, provided buffer capacity is adequate, and will be lined up there according to their stack distance. Once a vehicle moves up to the first buffer position, it will search the loading stations you entered into the list *StationsL* for freight to be picked up. As the buffer exit may be viewed as a confluence from two directions, you may enter priorities. Set the customized attributes *prio2way_out* in the track object *way_buff* and *way_straight* according to their priority.

A vehicle that may not enter the buffer because it is full, will search for an alternative buffer in the buffer list *BufferL*. Once the vehicle finds a buffer it will become its new destination. If you did not enter anything into the list, the vehicle will circulate in the installation and search for another buffer independently.



Stationslist *StationsL* and buffer list *BufferL*

Source Vehicle - VEHinput 

Within the object *VEHinput* the vehicle table *VEHtableL* is of special interest, as the entries there set how many vehicles are created. The variable *v_typ* stands for the vehicle, *v_num* for the number. Vehicles circulating in the system are created in the source, if the path of *VEHinput* was entered in the column *v_source*. Besides creating circulating vehicles, any number of stacked empty vehicles may be created in any number of buffers of type *VEHbuffer*. Or a single waiting vehicle may be created in front of a loading, unloading or docking station. You always have to enter the absolute path in *v_source*.

string	object 1	object 2	integer 3
0	v_source	v_typ	v_num
1	.EOMsystem.VEHinput	.veh	6
2	.EOMsystem.VEHbuffer1	.veh	2
3	.EOMsystem.VEHbuffer2	.veh	3
4	.EOMsystem.EOMload1	.veh	1
5	.EOMsystem.EOMdock4	.veh	1
6			

Table *VEHtableL* for creating vehicles

Loading station - EOMload

After processing has been completed in a processing station or when using a transport matrix, the parts are moved into the entrance buffer *part_avail* of the loading station. The loading station uses the list *StationsL* to request an available vehicle. Enter the paths of the preceding buffer *VEHbufferx* or of the loading, unloading and docking stations *EOMload, EOMunload, EOMdock*. This ensures that any vehicle requested from there has as short a track to drive as possible.

If this is not important, you may also leave the list empty, as the program will automatically search for an available vehicle in the entire system according to the list *VEHavailable* in *EOMcontrol*. The entries in *StationsL* are searched in sequence by the method *part_avail_oCtrl*.

	object 1
1	.EOMsystem.EOMunload2
2	.EOMsystem.EOMunload1
3	.EOMsystem.VEHbuffer1

list *StationsL* for the search of driving course

Use the global variable *loadtime* in each station to set the time used for loading. Requested vehicles are registered in the variable *veh_required*.

Unloading station - EOMunload

After reaching their destination, loaded vehicles pass their freight to the exit buffer *part_unload* of the station. For this, the program takes the user-defined unloading time *unloadtime* into consideration. Provided the global variable *veh_wait* has been set to TRUE, the vehicle remains in front of the station after unloading and is going to be registered in *EOMcontrol.VEHavailableL*.

A vehicle continuing on its way uses the list *StationsL* to search for waiting parts. Enter the paths of the succeeding loading stations into the list. This ensures that for empty travels as short a track as possible is chosen. When no part to be picked up is located at these stations, the search is automatically extended to all stations within the system vial the *CARGOwaiting* in the *EOMcontrol*.

	object 1
1	.EOMsystem.EOMunload2
2	.EOMsystem.EOMunload1
3	.EOMsystem.VEHbuffer1

List Stations*L* of an unloading station for searching parts

If no part is found, the empty vehicle searches the list *BufferL* for an empty vehicle buffer. Here, you may set the next buffer to be targeted. If this list does not contain any entries, the vehicle circulates with the destination of the highest frame.

	object 1
1	.EOMsystem.VEHbuffer2

List *BufferL* for assigning buffers

U + L

Docking station - EOMdock

The station *EOMdock* combines the features of a loading and an unloading station. For disposition strategies the same rules apply as for the single processing stations. Thus the docking station may be connected to a processing station. You may also use it as a source or a drain for a transport matrix. In *EOMdock* parts ready to be transported search for available vehicles and unloaded vehicles search for waiting parts. The list *StationsL* has been replaced by the lists *VEHStationsL* and *CARGOStationsL*, where you enter the stations to be searched. The list *BufferL* is used for the drive of an unloaded vehicle to the empty vehicles buffer. Set loading and unloading times by the global variable *loadtime* and *unloadtime*.

	object 1
1	.EOMsystem.VEHbuffer2
2	.EOMsystem.VEHbuffer1
3	.EOMsystem.EOMunload1

	object 1
1	.EOMsystem.EOMload3
2	.EOMsystem.EOMload1
3	.EOMsystem.EOMload2

Lists *VEHStationsL* and *CARGOStationsL* of a docking station

WORK

Work Bench Operation - EOMwork

During work bench operation a work piece is either processed while hanging on the vehicle or it is unloaded, processed and reloaded onto the same waiting vehicle. Advancing vehicles are blocked during that time. Set the handling times via the global variables *unloadtime*, *worktime* and *loadtime*. When workpieces are processed while hanging on the vehicle, the loading times have to be set to 0. The work bench cannot destroy work pieces. So, enter the next destination into the variable *nexLocDest*. The work piece is passed on to another workbench until an unloading or docking station is entered as next target, where it will be unloaded and destroyed.

The object *EOMwork* cannot be used in conjunction with a production control.

Managing Vehicles

The object *EOMcontrol* manages available vehicles and waiting parts that cannot be assigned in the individual stations.

EOM Control - EOMcontrol

The *EOMcontrol* keeps list that are processed by the interface methods. Enter the processing, the loading and unloading and the docking stations into the table *STATattachL*.

The method *veh_setParams* is called whenever the status of a vehicle changes.. Parameters, such as destination, disposition status and active icon have to be set anew.

The clearing method *veh_clear* in the *EOMcontrol* is activated by each vehicle as soon as it meets with a blocked object. If the blockage is caused by a vehicle waiting in front of a station, this vehicle is forced to move on.

- *VEHavailableL* List of vehicles available to take on a transport order. These vehicles are either waiting in first position to leave a buffer or are in waiting position in the object *way* in a loading, unloading or docking station. This list is used by the methods *veh_require*, *veh_register* and *veh_unregister*.
- *CARGOwaitingL*List registering all parts waiting to be transported. This list is used by the methods cargo_*require*, cargo_*register* and cargo_*unregister*.
- *STATattachL*In this table the assignment of the processing stations *GlobDestName* and the stations *LocDestNameUnload* via the method *stat_glob2loc* is performed. Each processing station inserted into the system has to be assigned.

string	object 0	object 1
0	GlobDestName	LocDestNameUnload
1	.EOMsystem.Station1	.EOMsystem.EOMunload1
2	.EOMsystem.Station2	.EOMsystem.EOMunload2
3	.EOMsystem.Station3	.EOMsystem.EOMunload3
4	.EOMsystem.Station4	.EOMsystem.EOMdock4

Table *STATattachL* for assigning stations

- *CARGOinput*Alternatively to the production control *EOMcontrol* this frame is used to define the transport orders via a transport matrix. The loading stations represent the sources and the unloading stations the drains where the parts are destroyed after unloading. Use the customized attribute *place* to mark the part to be destroyed. When loading the part via the *EOMcontrol* the value of the attribute is always 0, when loading it via the transport matrix it is not equal 0. The unit of the orders to be loaded is transports per hour.

obje	object 0	integer 1	integer 2
0		.EOMsystem.EOMunload1	.EOMsystem.EOMunload2
1	.EOMsystem.EOMload1		12
2	.EOMsystem.EOMload2	20	
3	.EOMsystem.EOMload3	10	10
4	.EOMsystem.EOMdock4		12

Matrix *TransportM* in *CARGOinput* for loading orders

Vehicle - veh

The vehicle *veh* transports the parts (work pieces) through the system according to the pending transport orders runs on the pieces of track while considering the blocking and clearing algorithms. Length and driving speed are user-defined. The following customized attributes were introduced for fine-tuning the control and management of the vehicles:

vOri:

This attribute holds the original driving speed for straight pieces of track (*speed*).

vCurve: (user-defined)

This attribute holds the speed for passing through curves (*speed*).

stackLength: (user-defined)

This attribute holds the stack length for stacking vehicles in buffers. When entering a buffer, the length of the vehicle is set to this value in the entrance control way_buff_iCtrl and reset to the original value when leaving the buffer in the exit control way_out_iCtrl (length).

fd_dispoStatus:

According to its active driving and loading status, a vehicle is in one of the following disposition times (*string*).

- "free"unloaded circulating in the system
- "hasOrder"on its way to pick up a part
- "loaded"loaded on its way to an unloading station
- "loading"vehicle is being loaded
- "waiting"unloaded waiting in front of an unloading station
- "working"waiting while a part is being processed on a work bench
- "inBuffer"unloaded, waiting in an empty vehicle buffer

lLoc:

This attribute registers the entrance object *way_in* that was passed in switches and confluences. This enables the entrance object to be released after exiting or after blockage times (p *object*; ***clearRef***).

This attribute also holds a reference to the clearing method *veh_clear* in the EOM control *EOMcontrol*. This method recognizes waiting vehicles in front of unloading stations and sends them on when needed (*object*).

group:

This attribute serves to mark a vehicle as part of a group when moving in a pile (*boolean*).

Course of Events Across Objects

The following chapters describe the course of events across objects.

Reset and Init

Clicking on Reset in the *EventController* resets the model into its initial state: all MUs in all objects of the application object template are deleted, and all dynamically entered information in lists, tables and global variables are removed.

In the objects of the *Electric Overhead Monorail* this applies to all entries made during simulation into the lists of the available vehicles and waiting freight. Also reset are all variables needed for controls.

In the production control *ProdMgr* this applies to free capacities and states of stations, to the lists of available stations and to the lists of parts waiting for an operation. The table containing the production orders is also deleted and created anew according to the settings in the table *OrderL*. In the *station* the contents of all *NwData* objects and of the results tables are deleted.

During initialization the model is set into a state allowing it to be run: the attributes of the objects are initialized with the user-defined entries, and the creation of vehicles and transport orders is started. In the *ProdMgr* the method *wc_startOrder*

is initialized, enabling the loading of parts according to the transport order list. During initialization the stations report their free capacity and the state *avail* to the corresponding *ProdMgr*.

Assigning Destinations

As soon as a work piece has been processed completely, it is moved into the exit buffer of the station. There the entrance control *st_PartReady* calls the method *wc_nextDestination*, to determine a new destination for the part. The *ProdMgr* determines the next processing step (operation) to be taken using the working plans of the part type and the its position in the working plan.

Once the processing step is determined, a station has to be picked that is able to execute it and is available. For this the *ProdMgr* uses the table *OperationL* to find all station types able to execute the processing steps. The table *StatTypeL* lists all available stations for all station types. Using this information, a list of all station able to perform the task is generated. This list then calls the method *MD_StatChoose*, that picks a station.

The selected station is then reserved by the *ProdMgr* via the method *ws_occupyStation*: the free capacity of the station is reduced by 1 and the station us removed from the list of available stations in *StatTypeL*. The address of the station is then transmitted to the calling station as the new destination for the part.

If no station was available for the required processing step (operation) the part is registered as waiting for the corresponding processing step in the table *OperationL*. *Non_avail* is returned to the requesting station, so that the part is kept there until a new destination has been determined.

The list of reserved parts is worked of as soon as a station registers with the production control as being available via the method *wc_StationFree*. This is the case when a part has been completely processed in the station. In case parts are waiting for a processing step (operation) the station that registered as free can execute, a list of waiting parts is generated and a part is selected.

5

HBW

Introduction

Plant Simulation HBW is a set of Application Objects used to model High Bay Warehouses (HBW) efficiently. Plant Simulation HBW was developed on the basis of Plant Simulation and requires an Plant Simulation Development License to be used. This manual assumes the reader has a basic familiarity with Plant Simulation.

Please note that this Application Object Template should fully cover most modeling tasks in a high bay warehouse environment. However, because of their flexibility and generality, the Application Objects and their interrelationships with each other are often fairly complex. Therefore, any adaptation of the Application Object Template will require a good knowledge of its internal functionality. Thus, prior to using this Application Object Template for a project, one should check whether the current functionality fulfills the requirements of the project. If adaptations must be made to the template, we recommend checking the ratio of the benefits to the effort required carefully. In any case, an Application Object Template will be helpful for creating prototypes rapidly and as acting as a reference for your own Application Object Templates.

Plant Simulation HBW offers three possible alternatives. They are distinguished by their degree of detail:

Alternative 1:

Modeling of a HBW as a black box without considering single aisles. Storage and retrieval times can be defined: The following Application Objects are provided:

WarehouseUT

- warehouse capacity
- storage locations
- behavior when there is a surplus or lack of components

Alternative 2:

Modeling of a HBW including warehouse management and aisles with storage and retrieval units, but no storage locations. The storage locations needed to determine the cycle time are simulated by a random generator. The following Application Objects are provided:

Warehouse Manager

- list of components
- statistical evaluation
- aisle loading

Aisle

- aisle:
  - number of floors and columns
  - grid of floors and columns
  - storage position in a x/y coordinate system

- rack serving unit:
  - velocity in x- and y-direction
  - acceleration in x- and y-direction
  - positioning and handing-over time

Alternative 3:

Modeling of a HBW, including aisles with storage and retrieval units, as well as storage locations. Storage areas of A, B and C articles can be defined. The following Application Objects are provided:

Warehouse Management for ABC articles

- list of components (A, B, C)
- statistical evaluation
- aisle loading (A, B, C)

Aisle for ABC article

- aisle:
  - number of floors and columns
  - grid of floors and columns
  - storage locations in a x/y coordinate system
  - storage areas of A, B and C articles

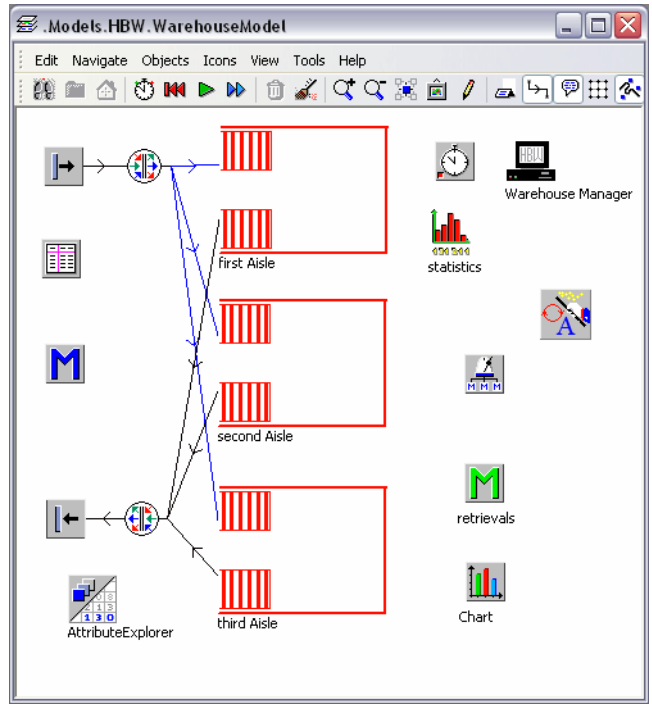
- rack serving unit:
  - velocity in x- and y-direction
  - acceleration in x- and y-direction
  - positioning and handing-over time

Every Application Object offers interfaces providing dialog boxes, Popup menus and input data lists. A description of the functionality and the interactions of all Application Objects can be found. Help functions will assist the user. Every Application Object is ready for user-defined adjustments. Examples are also included.

Example

In order to describe the functionality and the working methods of the warehouse Application Object, we will construct an easy simulation model of a warehouse with three aisles that are equipped with a rack serving unit. The task is to consider the influence of the number of different types of parts stored, the frequency of storing and retrieval demands, and the geometric properties of the three aisles.

This chapter describes the construction steps of the desired model and the performance of the corresponding simulation. The entire model is presented in the Object Template, and is found under the name *WarehouseModel*. A possible realization of the described model is given in The developed model.



The developed model

First, a new frame is generated and possibly renamed to (for instance) *WarehouseModel*. After it has been opened, an Event-Controller and a Application Object *WhMgr,* a source, a drain and two flow control building blocks are inserted. To register all aisles please insert the assistant *ASSI* and start it. It is desired that the source produces infinitely many parts of random type in intervals of one minutes.

object 1	real 2	string 3	table 4
MU	Frequencies	Name	Attributes
.node	6.00	Part1	attr
.node	2.00	Part2	attr
.node	7.00	Part3	attr
.node	9.00	Part4	attr
.node	4.00	Part5	attr
.node	5.00	Part6	attr

string 1	integer 2	boolean 3	string 4
Name of Attribute			
type	1		
EntityType			part
currIcon			Typ2

The frequencies table *Objecttypes* of the basic building block *source*

We choose in the Popup menu Time of Generation *Interval Adjustable* and in the Popup menu MU-Selection *Random*. In order to set the frequency of the parts a table file is inserted and formated as shown in The frequencies table Objecttypes of the basic building block source. The source will produce parts with attributes *type* and *EntityType*, initialize these attributes and set the prepared icon for the parts. If all 60 prepared parts will be used it is recommended to fill the TableFile *Objecttypes* by using a method. Please cf. the method *fill* in the frame *WarehouseModel*.

So far, a simulation model with only a 'storing' function is possible. To test the warehouse with all its features, retrievals are initiated using a generator. Therefore, a generator is inserted (from the Object Template) in the model and a method *retrievals* is written. Now the generator is opened and the method *retrieve* is assigned to the interval control. For example, if we want the first retrieval to start 10 minutes after the beginning of the simulation, *Start* must be set to *const* 10:00.00.

The duration between retrievals can be set in the dialog box *interval*. For example, if the time is 10 seconds, the user should set *const* 10.00 in the interval dialog box. Finally, the user should press the button Apply and OK. Finally, we write the method retrievals:

```
is
     type : integer;
do
  type := floor(z_uniform(1,1,61));
  WhMgr.RetrievalDemand("part"+to_str(type));
end;
```

If information about the movement, waiting and handing-over times of the rack serving unit is desired, the Application Object *IntStat* is inserted into the model. Every 10 minutes, the statistical data of the rack serving unit is updated in the *WhMgr* and in each aisle. The corresponding tables can be opened in the aisles' dialog boxes and the *WhMgr*.

The capacity of the aisles can be defined by changing the number of floors and columns. For this purpose, the menu *Aisle Data* in the dialog box of the aisle is opened. The third and the fourth item determine the distances between the floors and the columns, respectively. Finally, the X- and the Y-coordinates of the storing and the retrieval position can be set. In the menu *Rack Serving Unit Data*, the X- and the Y-components of the maximum velocity and acceleration of the *RSU* (rack serving unit) are inserted. The X-direction is the horizontal direction and the Y-direction is the vertical one. Note that all physical data must be inserted in the units shown in brackets.

Description of the Application Objects

All Application Objects have dialog boxes where the user can adjust all the settings necessary for their use. In every dialog box, there is a button Help which provides some help for each Application Object and its parameters. Note that tables with statistical data for the RSU and information about stored parts (in the case of the second and third detailed model versions) are available using the dialog boxes.

WarehouseUT

The Application Object *WarehouseUT* has an entrance and an exit. The capacity of the warehouse, the storing and the retrieval behavior, and the processing times of the storage and the retrieval locations are adjustable. There is no differentiation between aisles or articles. The duration times for storage and retrieval are defined by *procTime* at the *storingPlace* and the *retrievalPlace*.

Icon:

User interface

For a retrieval of an object, the method *RetrievalDemand* can be used. The object type must be defined as a parameter (of data type: *string*).

The parameters for the storage locations, retrieval locations, and warehouse capacity can be set in the dialog box.

You can also adjust the behavior of the Application Object if the warehouse is full and a storing order comes in. The user can choose whether the dimension of the warehouse is increased or an error message is printed. It is also possible to set the behavior for the case of a retrieval where the desired object type is not stored in the warehouse. In this case, the computer can give an error message or create a new object for the desired type.

The dialog box of WarehouseUT

Functionality

An object reaching the storage location will be entered in the TableFile *Objectlist* when it moves into the warehouse. This table contains a QueueFile for every stored type.

If storage is impossible, the dimension of the warehouse increases or an error message is printed depending on the setting in the user dialog box. If *increase dimension* is chosen the capacity of the warehouse will be increased without limit or until the user sets *Error Message* in the dialog box.

On receiving a retrieval demand, a check is made to see if the retrieval location is free. If so, the desired part is added to the *retrieval* list. Otherwise, a check is made to see if the part type is entered in the *Objectlist* and is therefore stored in the warehouse.

If the ordered part is stored, then the first object in the QueueFile for that type is taken out of the *Objectlist* and moved to the retrieval location. That means, the part that has the been stored the longest will be retrieved. If the QueueFile becomes empty, the part type is removed from the *Objectlist*.

If the desired part type is not stored in the warehouse, a new object is created on the retrieval location or an error message is printed. This behavior depends on the settings in the dialog box. If an object leaves the retrieval location and there is still an order in the *retrieval* list, then the first order is taken out and the corresponding object will be retrieved or created.

The Warehouse Manager WhMgr

In the other alternatives, individual aisles are designated. A Warehouse Manager (abbreviated by *WhMgr*) manages the storage and retrieval of the warehouse aisles. Each aisle is equipped with a rack serving unit (abbreviated by *RSU*). The Warehouse Manager provides information about the stored parts, the occupancy of the aisles and the moving, waiting and handing-over time of the rack serving units for each aisle. For this purpose, it is necessary to know the mechanical parameter of the rack serving unit and the geometric data for the aisles. This data must be entered by the user.

The Warehouse Manager manages a warehouse that consists of aisles of the type *Aisle*. A high bay warehouse is mapped by exactly one *WhMgr* and at least one *Aisle*.

Icon:

User interface

For a retrieval of an object, the method *RetrievalDemand* can be used. The object type must be defined as a parameter (of type = string).

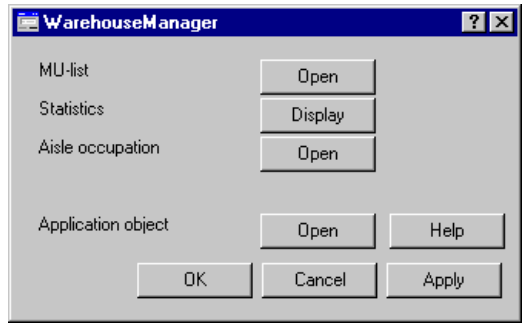
The method *MinUsedAisle* returns the aisle number that contains the fewest objects.

The method *MaxUsedAisle* returns the aisle number that contains the most objects.

The method *MaxEntityInWarehouse* returns the type of object stored most frequently in the warehouse.

These three results are given by pressing the button AisleOccupation in the dialog box.

Furthermore, the user can open the TableFile *Stat_Tab* containing status times of the *RSU* for each aisle and the actual *MU_List*.



The dialog box of the Warehouse Manager *WhMgr*

Functionality

The *WhMgr* manages the aisles of the warehouse and the stored objects. Retrieval orders are taken and sent to the aisles.

The stored objects are entered in the TableFile *MU_list*. An entry exists for each object type stored in the warehouse. This list contains references to other tables in which the objects of a given type are stored with comments about storage time and aisle. The user can order a retrieval by calling the method *RetrievalDemand*. The type of the object is passed as a parameter in string form. If this type of object is stored in the warehouse, it is entered in the *OrderBuffer* in an aisle.

If the desired type is not stored in the warehouse, the object is created (the aisle from which it is retrieved is selected at random) or an error message is printed. This behavior depends on the setting in the dialog box for the aisle.

The Aisle

The Application Object *Aisle* describes an aisle without management of single locations. The geometry of the aisle, movement values for the *RSU* and the storing and the retrieval behavior are adjustable. Note that the button OpenAO opens the network for the Application Object.

Icons:

User interface

The user can determine the behavior for storage and retrieval in the dialog box.

The dialog box of *Aisle*

The statistical data of the *RSU* can be shown. While pressing the button Data of the Aisle, a further dialog box is opened (see Description of the Aisle geometry) in which the geometric data of the aisle can be entered (number of floors and columns, X- and Y-position of the storage and retrieval locations and the distances between the floors and columns denoted by FloorGrid and ColumnGrid, respectively).

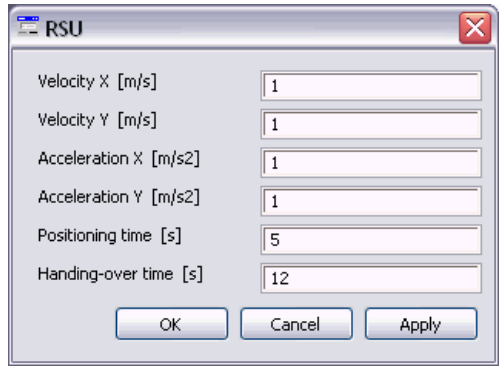
Description of the Aisle *geometry*

To change the data of the rack serving unit (*RSU*), the user must press the corresponding button. In this dialog box, (see The dialog box of the RSU) the velocity, acceleration in X- and Y-direction and positioning time and handing-over times of the *RSU* can be entered.

Functionality

The Application Object *Aisle* takes incoming objects to the storage location. If the *RSU* is free, it gets the object and moves it directly to the location in the warehouse. To calculate the movement time of the *RSU,* the storage location in the warehouse is determined at random.

A retrieval demand is only accepted if the retrieval location is not occupied. If this location is free, the *RSU* moves to the part's storage location and performs the retrieval. Note that the storage location of the part is not known and is also chosen at random.



The dialog box of the RSU

To determine the movement time of the *RSU,* it is assumed that the *RSU* moves diagonally, which means the longer of the two movement times (X- or Y-direction movements), is taken as the overall movement time. The positioning time is added to this value. Note that 'positioning time' is defined as the time taken to 'fine tune and home in' on the final position of a location.

The Warehouse Manager WhMgr_ABC

Icon:

User interface

WhMgr_ABC is an extension of *WhMgr*. This new Application Object has all properties of the Warehouse Manager *WhMgr* as explained earlier.

For a given type of objects, the methods *Art_distribMin* and *Art_distrMax* return the number of the aisle(s) in which the minimum or maximum number of objects are stored, respectively. If there is more than one aisle with the same (minimum or maximum) number of objects, the resulting aisle is chosen at random.

The methods *is_A_article* and *is_B_article* detect if a given part belongs to the article class *A* or *B*, respectively.

The dialog box is the same as the box of *WhMgr* (see The dialog box of WarehouseUT).

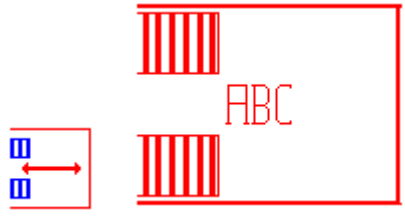
Functionality

Please see the corresponding section of the *WhMgr*.

Aisle_ABC

The Application Object *Aisle_ABC* is an extension of Aisle. The user is able to manage the storage of objects according to the locations they were stored and the kinds of the articles denoted by A, B and C. The so-called ABC distribution is a well known method for articles with distinct properties, such as shelf life. Articles of type A are the most often stored or retrieved, type B the second most and type C the third most. Therefore, the storage locations of type A-articles are chosen nearest to the entrance or exit of the warehouse such that storage and retrieval can be performed quickly (see The storage area for A, B and C-articles).

Icons:

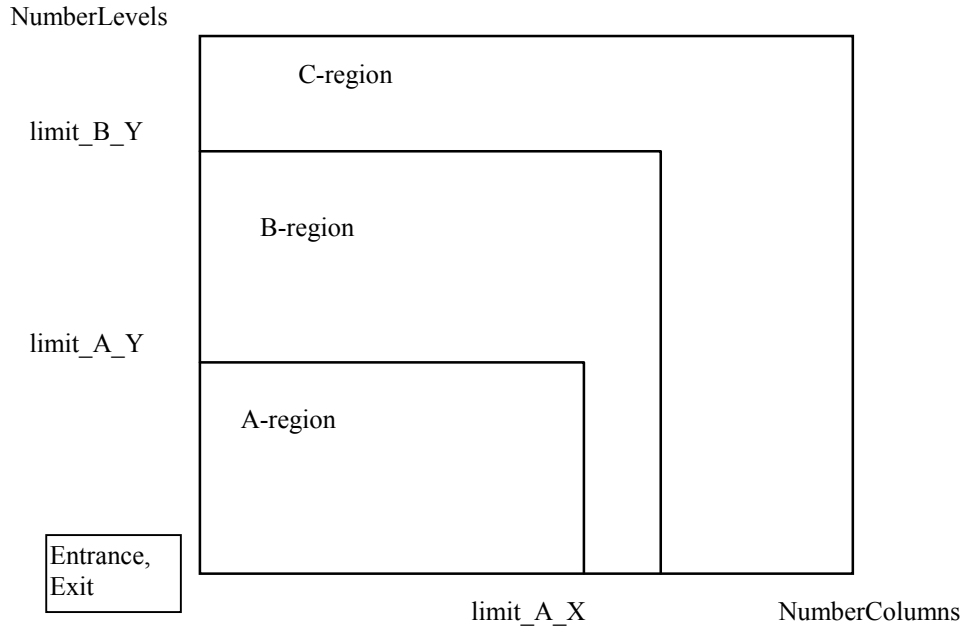


User interface

The data of the aisle and the *RSU* can be adjusted in the dialog box.

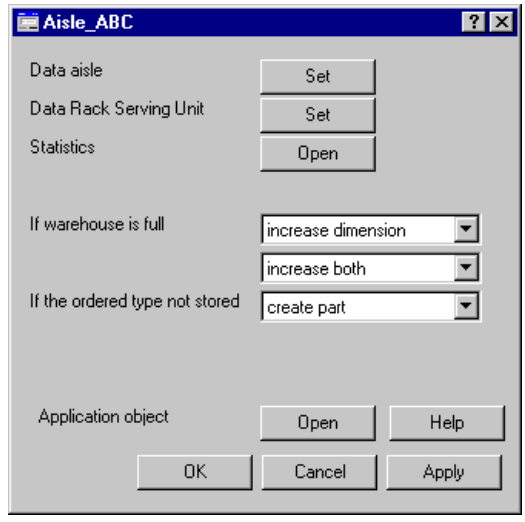
Furthermore, the user can define the limits for storing A and B articles. C articles are stored as a matter of priority in the remaining locations. The user has to check if the following relationships are fulfilled:

0	<	limit_A_X	<	limit_B_X	<	NumberColumns,
0	<	limit_A_Y	<	limit_B_Y	<	NumberFloors



The storage area for A, B and C-articles

Since the dialog boxes of the Application Objects *Aisle* and *Aisle_ABC* are the same, no further explanations are given.

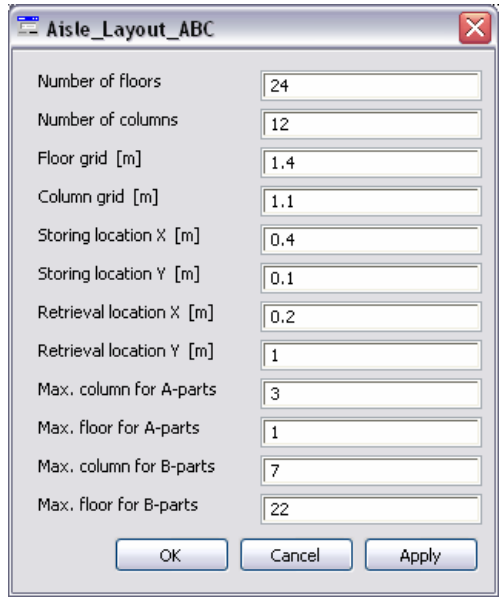


The dialog box of Aisle_ABC

Functionality

The Application Object takes incoming parts to the storage locations. If an unfinished order exists for that type, the *RSU* gets the object and moves it directly to the retrieval location. Otherwise, the *RSU* gets the object (if free) and moves it to the storage location.

The storage location is chosen according to the given ABC-distribution. The search starts in the first column from top to bottom or alternately in both rows in the required region. If an object does not find a storage location in its designated region, it can then be stored in the other regions.



The dialog box of Aisle_ABC *for storing A-B-C-articles*

The search for a storage location is performed as follows:

	for A-article	for B-article	for C-article
first	A-region	B-region	C-region
then	B-region	C-region	B-region
last	C-region	A-region	A-region

Applications and Testmodels

Extensions and Restrictions

It is possible to combine the warehouse Application Object with other Application Objects. Note that the parts which should be stored in the warehouse must have an attribute *EntityType* of the data type *string*.

If a retrieval demand comes in which is not realizable, then a part for the type of object is generated (depending on the settings by the user in the dialog box). Such objects are marked in red during the animation. In order to generate the desired objects for special applications, the method *RetrievalDemand* must be modified.

Test Models

Executing the methods *BuildUT, BuildAisle* and *BuildAisle_ABC* the models *Test_UT, Test_Aisle* und *Test_Aisle_ABC* are built up. An Application Object *WarehouseUT* or an *Aisle* is connected with a *source* and a *drain*. To retrieve a part, the method *Retrievals* must be modified and started by the user. It is interesting to observe the *MU_List* together with the corresponding queues in the Warehouse Manager *WhMgr* and the *OrderBuffer* in the *Aisle*.

The table containing the status times of the rack serving unit (*RSU*) is designed as follows:

	string 0	time 1	time 2	time 3	time 4	time 5
string		last event time	waiting	handing-over	occupied	available
1	total	25:49.0000	9:17.0000	8:24.0000	4:51.0000	3:17.0000
2	transfer	25:41.0000	0.0000	12.0000	12.0000	8.0000
3	provisional	0.0000	0.0000			
4		10:00.0000	3:44.0000	3:12.0000	1:53.0000	1:11.0000
5		20:00.0000	7:06.0000	6:31.0000	3:44.0000	2:39.0000

The statistics table of a aisle

The status times of the *RSU* belonging to an aisle are registered in columns 2 through 5. The time of the last change to the table is inserted in position (1,1). Under it, the time of the previous change is given. For any given row, the *last event time* (column 1) equals the sum of times in the remainder of that row. From the row marked *provisional* and down, the status times of the *RSU* for the aisle are stored in 10 minute intervals. (Figure 10 shows that between 25:32 and 25:44, the *RSU* had a handing-over time of 12 seconds.)

Demonstration Model

In the model *WarehouseModel,* three aisles are connected to build a warehouse. To retrieve parts, the method *retrieve* is written. The *Generator* calls this method in time intervals of 10 seconds. At a given simulation time, the *MU_List* in the control Application Object has the following three items:

entitytype	aisle list
yellow	table21
green	table22
black	table23

The *MU_List* contains all stored parts.

For instance, the entry *table21* shown in The MU_List contains all stored parts., which contains all stored type *yellow*, can be expanded to the table shown in A table of the MU_List entry [2,1] (i .e. table21). Note that the parts are arranged according to their storage time:

aisle	storing time	stored part
.Models.WarehouseModel.Aisle1	2:32.0000	.MUs.part:724
.Models.WarehouseModel.Aisle2	3:08.7000	.MUs.part:728
.Models.WarehouseModel.Aisle1	15:39.0000	.MUs.part:764
.Models.WarehouseModel.Aisle3	17:14.0000	.MUs.part:769
.Models.WarehouseModel.Aisle3	20:18.6000	.MUs.part:781
.Models.WarehouseModel.Aisle3	22:13.1000	.MUs.part:784
.Models.WarehouseModel.Aisle1	25:44.0000	.MUs.part:795
.Models.WarehouseModel.Aisle3	27:11.0000	.MUs.part:801

A table of the *MU_List entry [2,1] (i .e. table21)*

The *Diagram* building block shows the number of stored parts in the aisles dependent on the simulation time. Of course, it is possible to represent other simulation parameters.

All status times of the *RSU* for each aisle are summarized in the statistics table in *WhMgr*. Note that the data is collected every 10 minutes (see The statistics table of the Warehouse Manager).

time	aisle	waiting	handing-over	occupied	empty
10:00.0000	Aisle1	8:02.0000	1:12.0000	27.0000	19.0000
10:00.0000	Aisle2	9:14.0000	0.0000	27.5000	18.5000
10:00.0000	Aisle3	6:42.4000	0.0000	2:12.6000	1:05.0000
20:00.0000	Aisle1	16:05.0000	2:24.0000	49.0000	42.0000
20:00.0000	Aisle2	18:11.6000	0.0000	59.1000	49.3000
20:00.0000	Aisle3	13:09.3000	0.0000	4:34.6000	2:16.1000

The statistics table of the Warehouse Manager

Note that the data of the *RSU* for the first aisle is contained in The statistics table of a aisle.

6

Production control

Object ProductionManager

Icon:

Functionality:

The *ProductionManager* controls the sequence of production steps for each part type. An operations plan for the production of each part is stored. A list containing all operations for the stations should help to create the operations plans. The orders for the parts, together with the quantity and earliest start date, also have to be entered.

Clicking on the button Master will open the following TableFile:

	string 0	table 1
string 0	entitytype	operationsplan
1	part	Plan
2		

TableFile masterDataL

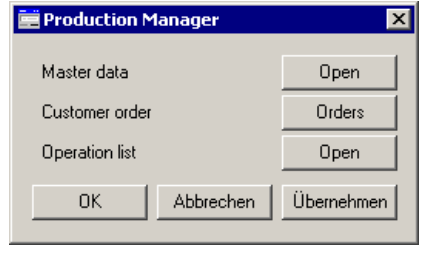
The first column contains the type of part to be created. The second column contains the name of the corresponding operations plan. After opening the table for the operations plan (such as *prodSch*) by double clicking on the entry, the sequence of operations for the workflow has to be determined. It is possible to define set-up time and processing times for each operation. The station (see Figure 18) has to be defined in the dialog to show whether the processing time is part, station, or factor specific. Part specific processing times have to be entered into the operations plan contained in the *ProductionManager*. If the processing time depends on a factor (for instance, the time for milling a piece of wood could depend on its length), this factor can then also be entered.

Clicking on the button Orders in the dialog of the *ProductionManager* will open the following TableFile. Enter part types and quantities here.

	string 1	string 2	integer 3	datetime 4	datetime 5
string 0	orderNo	entitytype	number	release date	due date
1	08/15	part	6	1997/10/01 00:00	1997/10/02 00:00

TableFile *OrderL* with one order

User interface:



Dialog of the *ProductionManager*

Workplace Station

Icons:

The application object *Station* is used to simulate different stations. All stations are registered and managed by the *ProductionManager*. Set-up times and processing times may be chosen to be part-, station-, or factor-specific. In this application object, parts are processed one by one.

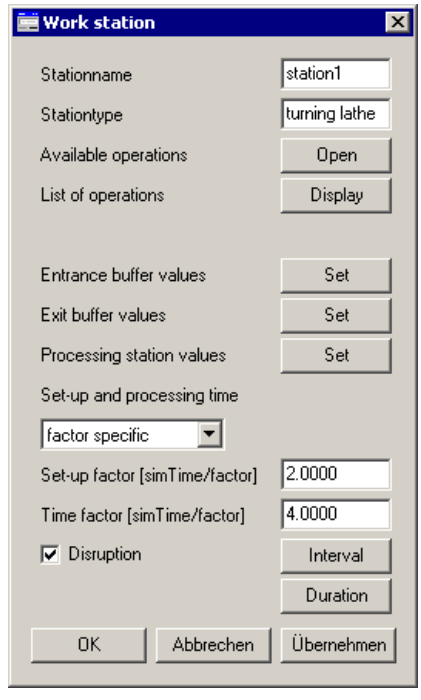
Functionality:

When a part enters the station, it will stay in the single processing station *setup_place* for the set-up time and stay in the single processing station *Work* for the processing time. After these periods of time, a new destination for the part is determined and it is passed to the transport system. If there is no transport system then the part is directly sent to the next processing station.

User interface:

After inserting this application object, an data entry field will be opened where the type of station has to be defined. Then, the operations the station is able to execute have to be defined. You accomplish this by clicking on the button Open. Note that stations of the same type have an equal number of sets of operations. The operations lists are updated automatically. Naturally, the operations entered have to be contained in the operations plan.

The values (capacity, processing time) of the entrance and exit buffers may be adapted to your needs by clicking on the corresponding buttons in the dialog.



Dialog of the object station

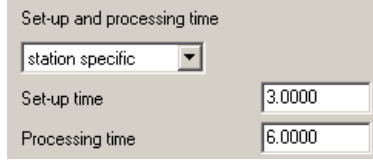
The *Station* also allows the simulation of disruptions. When the Checkbox is chosen, Duration and Interval may be adjusted. Duration and Interval of disruptions may also follow any of a number of available statistical distributions.

Set-up and Processing Time

To meet the needs of set up operations, the application object *Station* contains the object *setup_place*. Set-up times are needed for changing the part type, the order or the operating step. Set-up times are either specified in the routing plan or in the station. There are three possibilities to define set up and processing times:

- input in the operations plan
- input in the dialog of the application object *Station*
- input in the dialog of the basic object.

The dialog of the application object *Station* allows the user to define the set-up and processing time as follows: according to the operations plan (*part specific*), to the station (*station specific*) or depending on factors (*factor specific*). If the user chooses the mode *station specific,* two additional input options for set-up and processing time are opened. In this case the set-up and processing time do not depend on the type of part or the current operation. If the user chooses the mode *factor specific* two additional input options will also be displayed. Here the time factor and the set-up factor must be entered.

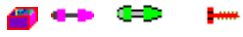


A portion of the Dialog of the station object

Simulation Disruptions

Machine down time may be modeled by a disruption by employing the object work which is part in the application object Station. If a station is disrupted it cannot proceed with processing parts until the disruption is eliminated. To set a disruption the dialog of the station has two buttons, *Interval* and *Duration*.

The object Part

Icons: 

The application object *Part* is the part transported and processed in the system.

- The following attributes are required:
  - *GlobDestination*. When using interfaces, a path to the destination of the other system is entered. Otherwise, a local destination is stored.
  - *Local*. The station served last is stored here.
  - *OperationsPlanPos*. The current position of the operations plan is stored here.
  - *OperationsPlan*. The operations plan for the part is stored here.
  - *EntityType*. This stores the type of part. The routing plan depends on this type.
  - *DestDeterminated*. This stores whether a destination was detected or not.
  - *LocDestination*. The path to the station in the current network is stored. If using an interface, the reachable interface is stored here.
  - *ProdCosts*. The costs of production for the part may be stored here.

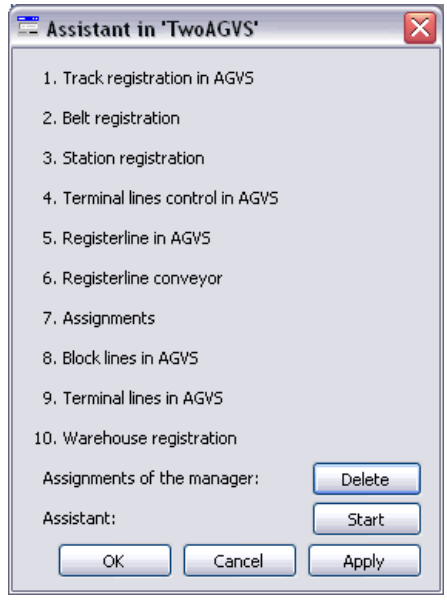
7

Support for Modeling

The Assistant ASSI

Icon:

If a new AGVS is built then the destination lists must be filled. This is important for the determination of the driving direction of the AGV in branches. Furthermore distances between certain building blocks are needed for disposition rules.



The Assistant *ASSI*

The assistant performs the following function automatically:

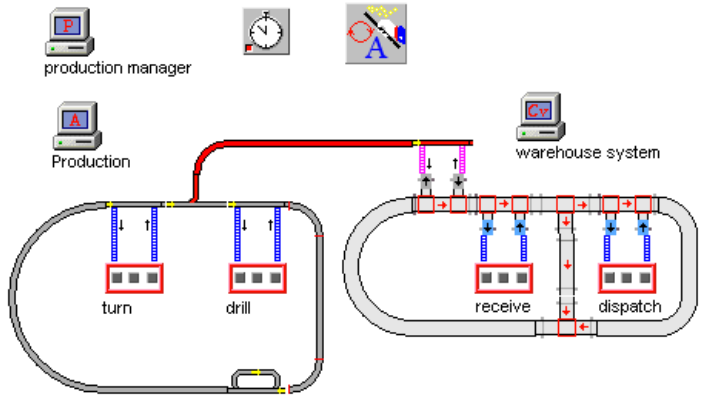
- The registration of tracks and responsible manager
- The registration of all work stations in the ProductionManager
- Recognition of terminal lines
- Generating the sign lists in branches (register line)
- Recognition of the connection between transport- and production system
- Determining the regions defined by blocklines and their entrances
- Generating the sign lists in branches in terminal lines

If the layout or the length of a track are changed then the distances and the destination lists must be newly constructed. Such changes are recognized by the system.

Application of Transport System Interfaces

Using the Application Object Libraries Plant Simulation_AGVS and Plant Simulation_conveyor, complex systems with different kinds of transport methods may be modeled. Material flow between the transport systems is realized by the in-

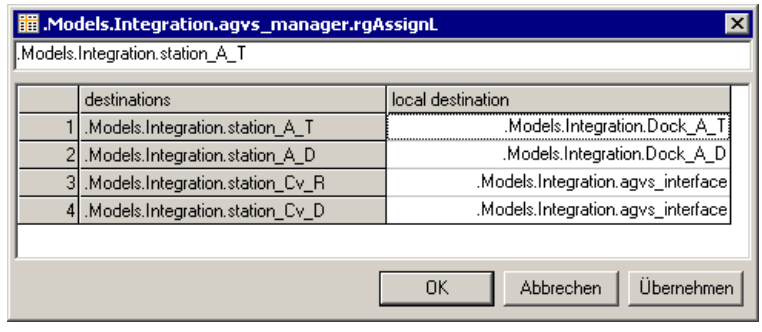
terface application objects such as *Cv_interface* or *AGVS_interface*. In order to model a transport system with two AGVS, two objects *AGVS_manager* are inserted that controls the AGVs in each subsystem. Compare the following screenshot for a design of the model:



Design of a transport system with a AGVS and a conveyor system

To define the assignment of the track subsystem and the responsible manager please press the button Reset in the Assistant *ASSI*. Insert into an arbitrary track the desired manager. The assistant assigns all connected tracks the this manager.

The TableFile *rgAssignL* shown in The TableFile rgAssignL of AGVS_manager illustrates the assignment of workstations to the destination in the track subsystem. The local destinations are track elements which can be reached in the track system of *AGVS_manager*. For instance, a part should be transported to the work station *station_A_T* then the AGV can directly drive to the docking station *Dock_A_T*.



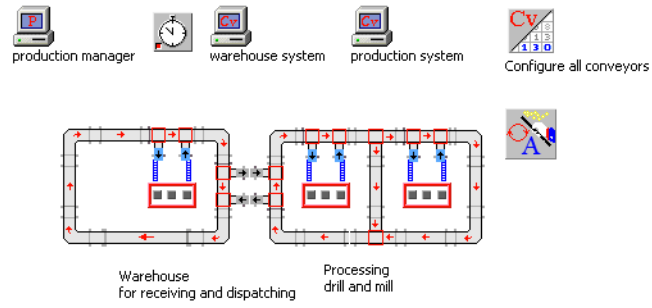
	destinations	local destination
1	.Models.Integration.station_A_T	.Models.Integration.Dock_A_T
2	.Models.Integration.station_A_D	.Models.Integration.Dock_A_D
3	.Models.Integration.station_Cv_R	.Models.Integration.agvs_interface
4	.Models.Integration.station_Cv_D	.Models.Integration.agvs_interface

The TableFile *rgAssignL* of *AGVS_manager*

The part destination *station_Cv_D* can only reach by the interface *agvs_interface*.

Application of Transport System Interfaces

Using the Application Object Templates eM-Plant _AGVS and eM-Plant _conveyor, complex systems with different kinds of transport methods may be modeled. Material flow between the transport systems is realized by the interface application objects such as *cv_interface* or *AGVSinterface*. In order to model a transport system with two conveyor systems, two objects *ConveyorManager* are inserted that controls the Conveyors in each subsystem. Compare the following screenshot for a design of the model:



Model of a Conveyor System containing two Submodels

To define the assignment of the belt subsystem and the responsible manager please press the button Reset in the Assistant *ASSI*. Insert into an arbitrary belt the desired manager. The assistant assigns all connected belts to this manager.

The TableFile *rgAssignL* shown in The TableFile rgAssignL of cv_manager2 illustrates the assignment of work stations to the destination in the belt subsystem. The local destinations are elements which can be reached in the subsystem of *cv_manager2*. For instance, a part should be transported to the work station *station22* then the part can directly go to the docking station *agvs_dock_station22*.

string	object 0	object 1
0	destinations	local destination
1	.interface.station11	.interface.cv_interface1
2	.interface.station21	.interface.cv_dockingStation21
3	.interface.station22	.interface.cv_dockingStation22

The TableFile *rgAssignL* of *cv_manager2*

The part destination *station11* can only reach by the interface *cv_interface1*.

About Siemens PLM Software

Siemens PLM Software, a division of Siemens Automation and Drives (A&D), is a leading global provider of product lifecycle management (PLM) software and services with 4.6 million licensed seats and 51,000 customers worldwide. Headquartered in Plano, Texas, Siemens PLM Software's open enterprise solutions enable a world where organizations and their partners collaborate through Global Innovation Networks to deliver world-class products and services. For more information on Siemens PLM Software products and services, visit www.siemens.com/plm.

SIEMENS

Division headquarters

United States
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
972 987 3000
Fax 972 987 3398

Regions

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
800 498 5351
Fax 972 987 3398

Europe
Norwich House Knoll Road
Camberley, Surrey
GU15 3SY
United Kingdom
44 1276 702000
Fax 44 1276 705150

Asia-Pacific
Suites 6804-8, 68/F., Central Plaza
18 Harbour Road, WanChai
Hong Kong
852 2230 3333
Fax 852 2230 3210

© 2007. Siemens Product Lifecycle Management Software Inc. All rights reserved. Siemens and the Siemens logo are registered trademarks of Siemens AG. Teamcenter, NX, Solid Edge, Tecnomatix, Parasolid, Femap, I-deas, JT, UGS Velocity Series, Geolus and the Signs of Innovation trade dress are trademarks or registered trademarks of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in the United States and in other countries. All other logos, trademarks, registered trademarks or service marks used herein are the property of their respective holders.

10/07