Part I: Getting to know

Tecnomatix Plant Simulation 11

Simulation of Production Systems

Contact:
Bastian Schumacher
Bui Minh Duc
Bastian.schumacher@tu-berlin.de
Duc.bm@mail.tu-berlin.de

Version:
15.10.2019

Course schedule

Date	Topics	Content
17.10.2019	Introduction into Simulation	Types of simulations and examples, Examples for material flow simulation, Verification & Validation, Short overview about simulation techniques and Organization of the course Chapter 1: Introduction, Overview & Basic Modeling
24.10.2019	Self-guided script	Chapter 2: Further modeling, programming
31.10.2019	Self-guided script	Chapter 3: Further programming & information flow
	Homework	Task 1: Flow Shop
07.11.2019	Self-guided script	Chapter 4: Shift, Trigger, Generator, Track, Transporter, Conveyors, Tables and Attributes for MUs
	Homework	Task 2: Packaging Task 3: AssemblySystem
14.11.2019	Self-guided script	Chapter 5: Random events and seed values
	Homework	Task 4: ParallelRunningProcesses Task 5: TransportationSystem
21.11.2019	Self-guided script	Chapter 6: Visualization
	Homework	Task 6: FlexibleProductionLine
28.11.2019	Self-guided script	Chapter 7: Simulation-based improvements methods
	Homework	Task 7: Workshop Task 8: Branching
05.12.2019	Advanced Exercises	Intensive programming Introduction into Case Study
12.12.2019	Advanced Exercises + Intro Case study	Intensive programming / Introduction into Case Study
19.12.2019	**Examination**	**Theory and practical exam**
09.01.2020	Case Study	Case Study
16.01.2020	Case Study	Case Study
23.01.2020	Case Study	Case Study
30.01.2020	Case Study	Case Study
06.02.2020	Case Study	Case Study
13.02.2019	**Final presentation**	**Final presentation Case Studies** **Due Date for submission of documentation: 13.02.2019, 23:55** **Due Date for submission of presentation: 13.02.2019, 14:00**

As a preparation for the exam and case study, eight sample tasks are handed out. There is no submission required for these tasks. Nevertheless, it is strongly recommended to work on them as a preparation for the exam and case study. The solutions are then discussed in class, according to the schedule above.

Table of Content

Table of Figure

1 Introduction, Overview & Basic Modeling

1.1 Introduction

This script serves as an introduction in the object-oriented simulation software Plant Simulation and supports the preparation for the seminar. This script leads to the confident control of the program tools. All topics of the seminar should be completed before the advanced exercise part.

Note:

The script is chronologically ordered and includes the entire material of the seminar with all exercises and tasks.

Plant simulation falls into the category of discrete simulation which means that events are considered over time. Among the special features of Plant Simulation is also the object-oriented approach. So objects for example can send messages to other objects to trigger reactions.

1.2 Overview

The overview describes the main tools operations: How to treat the models and objects, how to manage the hierarchy structure and how to configure program settings.

How to start Plant Simulation

There are two different possibilities to start Plant Simulation; following the path Windows Start (Menu)/programs/Tecnomatix or double clicking on the desktop icon of Plant Simulation. The opening screen of Plant Simulation appears.

Note: In the PC pool Plant Simulation can only be started with the desktop icon.

Exiting Plant Simulation

To exit Plant Simulation, go to the file menu and select the option “Exit”. Then Plant Simulation will be closed together with all open models.

Note: Remember to save the model!!

1.2.1 *Working with Models*

Models can be created, saved, closed and opened whenever needed and the handling is described by backup files. When saving, at the path of the existing model, a backup file is created or overwritten.

A model can have more than one Frame and all the Frames can be saved under the same model name.

Creating a Model File 

A new model can be created by pressing on the “**New model**” button or going to the **Data** menu and then clicking **New Model**. The quick key-path to **New Model** is “Strg-N”. A Class library and an empty Frame will appear on the screen. A name must be assigned to the new model in order to save the file for the first time.

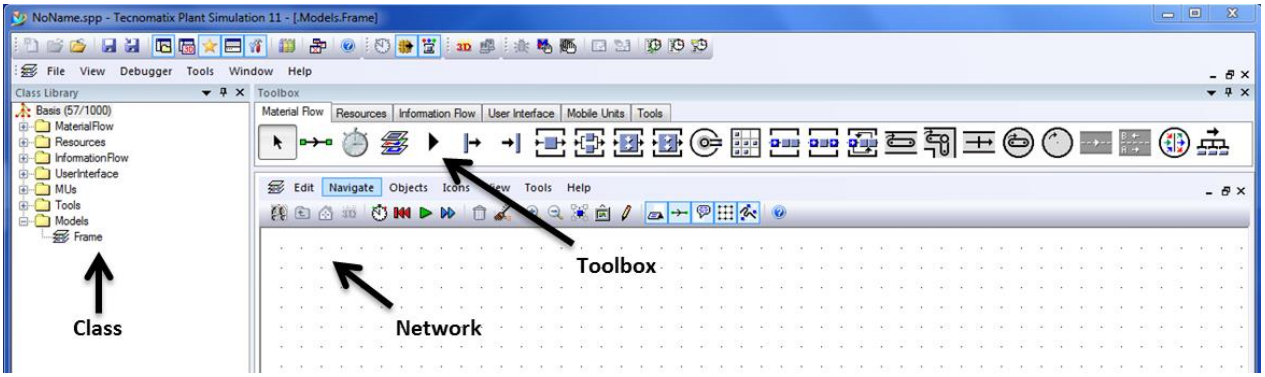

Figure 2-1: Overview Plant Simulation

Work Environment

After opening a new model, you come across the work environment where you can create and edit a simulation model. You see the **toolbox** (on the top), the **class library** (left) and also the empty **network**.

Saving a Model File

To store a model you can either click on "Save" or follow the drop-down menu path "Data/Save". The file is stored in the .spp file format. The extension .spp comes from the old name of Plant Simulation: *Simple Plus Plus*. Every time that you save the model, a backup file (with the extension .bak) will be automatically saved.

The old .spp file is saved as the .bak file and the current model is written into the .spp format. If you want to save a model file under a new name, in another list or backup file (.bak), then you have to save it using the option "Save as". The function "Save as" can be executed by selecting the button or under the menu "Data/Save as".

Note:

In some situations the model may not run successfully due to modeling errors, and could be no longer executable. Sometimes these errors are not recognized immediately. In case of a complex saving and recovering error, it might be better to roll-back working with the last executable model. In the automatically created backup file, you can always retrieve a previous version of your model. However, problems arise when you frequently save a model. In this case there is the possibility that the .bak file will also be not executable. If you save the model seldom, the ".bak" file might be executable but you could lose your modeling work if something unexpectable occurs. You should organize your files and have your own file management system of the models.

Name Convention for Models

In order to organize the model files quickly and clearly, you should assign a name to the model according to the following name convention:

<File_name (Short description)>_<Date (JJJJ_MM_TT)>_<Time (HHMM)>

It is usually meaningful (after changes over time or in a method) to save the model again under a different name.

If you use only the normal "Save", an error can pass through updated versions.

Backup Files (.bak)

The backup files can also be defined manually using the command "Save as" and selecting the file format ".bak". The current model will be saved in the backup file.

Note:
Backup files are saved only under new name.

Closing a Model
To close a model you can either click on the Button “Close Model” or use “File/Close Model” over the menu option.

Opening a Model
You can open a model either by pressing the Button “Open” or through the menu “File/Open Model”.

1.2.2 *Components Hierarchy Structure in Object Levels*

The class library groups and manages the objects of a model. The class library consists of the following sub-sections:

- Material Flow,
- Resources,
- Information Flow,
- User Interface,
- Mobile Unit (MUs),
- Tools and
- Models.

The class library is comparable to the tree structure of Windows Explorer in its structure. New folders and Frames can be created and renamed.

Material flow
Material flow components are fixed to a defined place in the model. They serve to simulate the material flow in a model. For example: Frames or Single Process units.

Mobile Units (MUs)
MUs are not fixed to a defined place in the model. They move throughout the model during the simulation. Examples: Entity, container.

Information Flow
Information Flow components provide information to the simulation, e.g. activate Triggers and process scheduling; or they collect specific information, for example the Time Sequence table.

User Interface
User Interface components graphically display current simulation data. For example, the Chart displays graphical data that Plant Simulation records during a simulation run.

Tools
Tools help to analyze the flow of material in a model. For example: Sankey Diagram and Bottleneck Analyzer.

Models
Simulation models are created in Plant Simulation in the form of a Frame. These Frames also contain objects, which can be linked with one another.

1.2.3 *Working with Objects*

Every object in the class library can be edited. Usually the model to be simulated has specifications that are more complex than the standard settings of the program.

Add

To add an object in the class library, you should select a folder or the basis-object, open the window with the right mouse button and then select in the context menu “New”. You can choose the type of object that you want to add.

Duplicating and Deriving

In the case of duplicating or deriving, a new object is created on the basis of an already existed object. The object must be selected and then in the context menu the action “Duplicate” or “Derive” can be chosen.

By duplicating an equal duplicate of the object is created on the same class level. The duplicated object will not have any other relation with the original object from which it was duplicated.
By deriving an object, a child is produced in the subclass. Father and child are further linked by heritable transmission, meaning, if the father object changes, this change will be inherited by the child automatically.

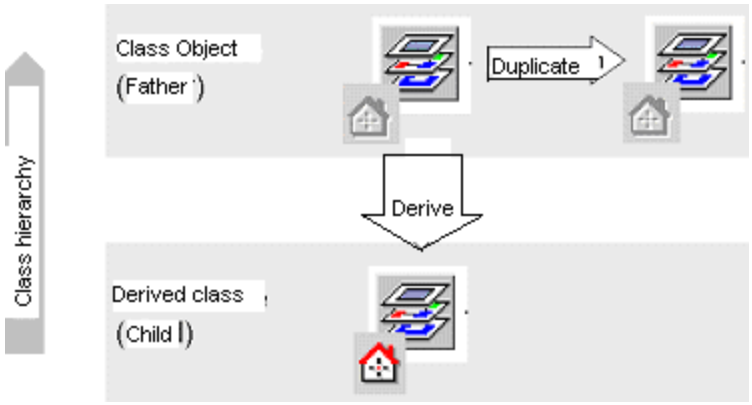

Figure 2-2: Hierarchy

Note:
In case of deleting the upper class (father), the subclass identified with a red house (child) is also deleted.

Rename
With the instruction “Rename” you can assign a new name to an object. The object has to be selected and then by pressing the key “F2” or choosing the option “Rename” in the context menu you are able to assign a new name.

Move or shifting
By shifting an object it is important to consider the hierarchy levels. Shifting in the same hierarchy level can be done by “drag and drop”. Shifting to another hierarchy level is possible by “drag and drop” and holding the “Shift key”. Shifting and copying into another hierarchy can be done by “drag and drop” and holding the “Crtl key” (Strg).

Delete
First select the object and then open the context menu and choose “Delete” or press the key “Del” (Entf). Now you should confirm the operation by pressing, “Enter” and the object will be deleted.

Note: See duplicating and deriving.

1.2.4 *Naming Conventions*

Special conditions must be considered when choosing the name of objects in Plant Simulation.

- Names must begin with a **letter**
- Letters or **numbers** may follow
- **Underscore** is considered a character, other special characters are not accepted
- A name must contain **at least 1 characters**
- Spaces are not allowed
- **Keywords** are not allowed (if, then, else, from, until, loop, result...)
- In the same frame it is not possible to use the **same name**
- There is no difference between **capital and small letters**.

Example:
Right: Station_1, drilling, Conveyor, FTS
Wrong: Station 1, while, 35-Band

Name Scope
All objects on same hierarchic level form a name scope. Within a name scope every object has a defined name. Out of the name scope is it possible to use the same name.

1.2.5 *Pre-setting*

Pre-settings are possible in Plant Simulation and facilitate the work with simulation models. The pre-setting menu is located in the main menu under „Tools/ Options“.

General
For the language you can select among German, English, Japanese and Chinese. The inscription of the menus is not changed over automatically. This conversion has to take place manually. In this menu, the time format change is taken automatically. To change the language click on “Tools”, “Preferences” and select on the index card „General“ the desired language.

Modeling
This menu contains many settings of how Frames can look like, for example the display of object names, predecessors and successors as well as the display of edges.
Simulation
This part defines how often a method can be called by another method, number of parallel active methods and number of methods that are suspended (methods waiting until an event occurs).
For example, a method with the instruction "waituntil" is the entire time active until it is released. If the number of such method is excessively large, then this usually leads to the fact that the computer (PC) is overloaded and Plant Simulation is stopped. On the other hand you need also a sufficient number of active methods within a very large simulation. Therefore, large simulation models require powerful hardware.
Note:
The number of active methods provided by the program is sufficient for the applications of this course.

Units
In this menu you can choose between the metric and imperial measuring system. Here it is also possible to set summer time options.

Editor
This menu controls the display of the methods text in the method window.
Note:
It is recommended activating the control "Display line numbers". It helps trace errors when debugging a method.

1.2.6 *Exercise*

01 Exercise Creating a Model

This exercise will provide a practice to create a model and saving it.

Task:
1. Start Plant Simulation through the Start Windows menu (our version is the educational one!).
2. Click on the new model.
3. Click "OK" and choose the predefined objects
4. Save your model in your own account. Give the name according to the nomenclature. (<File_Name (Short description)>_<Date (YYYY MM DD)>_<Time (HHMM)>). The file name should be "Exercise_Model".
5. Close your model.
6. Close Plant Simulation completely.
7. Start Plant Simulation again.
8. Open your saved model.

02 Exercise Class Library

This exercise serves handling objects in the class library.

Task:

1. Create a folder under Basis.
2. Name this as “Exercise”.
3. Provide a Frame in the folder “**Exercise**”.
4. Duplicate the Frame.
5. Derive the Frame.
6. Shift the folder “Exercise” into the folder “Models”.
7. Delete the father of the derived Frame.

Note:
Into the folder “Exercise” please save all your further exercise Frames.

1.3 Basic Modeling

In this chapter you will be introduced to the basic modeling principles in Plant Simulation. This chapter includes the introduction of components in a Frame as well as the basic construction of a model.

1.3.1 *Basic Components*

To create simple models, only a few elements are required, for example an Entity, a Source, a Drain, a Single Workstation, an EventController and Connectors.
There are lots of possibilities and configurations that can be used to set a model in Plant Simulation. The number of components in a model depends on the desired complexity of the model to be built.

Entity 

The *Entity* is a Moving Unit (MUs). It symbolizes goods being produced or transported. Their capacity is 0, i.e. they cannot take up other MUs. They are characterized by only one dimension, their length. An Entity has attributes. Attributes can describe the conditions of each object and these conditions can be called up and changed according to the simulation requirements.

Source 

The *Source* is an active material flow component. It produces MUs according to its settings. A Source is the easiest way to create MUs in a model. You can define the time of creation and the type of MUs to be created.
The frequency of MUs creation can be controlled by:

- Interval Adjustable,
- Number Adjustable,
- Delivery Table or

- Trigger

The following types of product generation exist:

- Constant i.e. of the same type,
- Sequence Cyclical (table is processed consequently),
- Sequence (table is uniquely processed) or
- Random (the objects are selected randomly from the table)
- Percentage (MUs are produced according to the values which you entered into the Percentage table)

Drain 

Drain is an active material flow component with a capacity of one. The *Drain* destroys MUs after processing them, it means, after the set-up and working time has passed.
Drains represent the opposite of the source and are the simplest way of removing MUs from the model.

SingleProc 

The *Single Processing Unit (SingleProc)* receives and processes a single moving unit (MU). It works actively as a material flow component with a capacity of one. It works on MUs according to a set-up and operating time and sends them to the next station. *Single Processing Units* represent production and processing places in the model.

Connector 

The Connector connects components in one Frame. It has a flow direction indicated by an arrow. However, the connector represents only a logical connection. It does not serve for the illustration of the real physical material flow. The Connector does not have a length; the simulation does not count any time when an entity is moving along it.

EventController 

The event controller coordinates all events during the simulation. Settings and controls of the EventController can be changed on its property window.

EventController Setting

In the EventController's Setting window, initial date and final time of the simulation can be set.

EventController Control

Speed of simulation and processes status can be changed in the EventController's Control window.

Start-Function
It starts the Simulation and processes all *INIT* methods if the button "Reset" was selected in advance. Objects become active unless they are blocked or paused.

Stop-Function
The stop function interrupts the simulation. The simulation can continue by choosing start again.

Step-Function
The simulation will go through step by step; it will execute each event individually.
Note:
The Step-function is suitable to control the simulation during debugging.

Reset-Function
This function resets the simulation time back to zero and executes all the *RESET* methods.

Init-Function
All methods with the name *INIT* are processed. However, *RESET* must be pressed before implementing this function.

1.3.2 *Inserting Components*

In order to create a model, it is necessary to insert components in the Frame. We can do this by inserting either from the component pallet or from the class library.

Toolbox
1. Mark the desired component from the Toolbox.
2. Click on any place on the Frame.
3. To insert several components of the same type, press the Ctrl key simultaneously.

Note: This method is not suitable for connectors. To insert a connector between two objects, you must first choose one and then click on the two objects.

Class Library
Inserting components from the Class Library is done by drag and drop.

1.3.3 *Statistics*

Material flow components and MUs collect different statistical data during the simulation. These can be viewed by right clicking over the component at any time during the simulation. Connectors do not show any statistical data.

Resource-Statistics
By activating the function Resource-statistics, Plant Simulation will collect statistical data at the drain during the simulation for all components.

Product-Statistics
For each object, the statistics for production, transport and storage during each simulation run were collected. Overlapping time is not accounted for.

Vehicle-Statistics
A vehicle collects additional statistical values such as disturbances, tracing, battery consumption as well as number of orders.

Total-Statistics
In the total statistics table, the overall data of all material flow components are collected. Empty cells or entries denote variables that have not been recorded.

1.3.4 *Event debugger*

The event debugger serves as an error-tracing tool. It can be accessed through "List" in the event controller.
See also chapter 2.5.1 Setting Critical Point.

1.3.5 *Exercise*

03a Exercise Basic Model
In this exercise you will learn how to create a simple model and evaluate its output by using the collected statistic data.

Task:

1. Create a Frame with the name "_03a_Basic_Model" under the directory Exercises.
2. Create a simple model by inserting in the Frame a *Source*, a *SingleProc* and a *Drain*. Connect the components with *Connectors*.
3. Insert an EventController.
   Note: To change the Event Controller double click on it, select "Settings" and under "End", you can adjust the running time. Please beware of the time format. The first four zeros are milliseconds. They are separated from seconds with a single dot. The seconds are separated from minutes with a colon as are minutes from hours.
   Example: 01:00:00.0000 (running time one hour)
   20:10.0000 (20 minutes and ten seconds)
4. Set running time for one hour.
5. Determine the number of parts that left the Drain (look at the statistics).
   You will find the statistics by double clicking on the drain and selecting "Statistics".
6. Recalculate by your own if the result is valid.

03b Exercise Basic Model

In this exercise you will learn how to duplicate a frame and how to implement a simple failure-strategy.

Task:

1. Duplicate the frame "_03a_Basic_Model" and rename it to "_03b_Basic_Model". Therefore click with the right mouse button on the current Frame and choose *Duplicate*. Rename the duplicated frame.
2. Double-click on the "SingleProc" and choose the tab "Failures". Click on "New". Change the availability to *85%* and the mean time to repair (*MTTR*) on *10:00*. Click on *Ok* to save your changes.
3. Set the running time for one day.
4. Determine the number of parts that left the Drain (look at the statistics).
5. Interpret the result (also compared to task 03a).
6. How many failures occurred during one day? Discuss about the statistical influence.

03c Exercise Basic Model

In this exercise you will learn to implement a simple exit-strategy.

Task:

1. Duplicate the frame "_03a_Basic_Model" and rename it to "_03c_Basic_Model".
2. Insert a second Drain and name it "Drain1".
3. Connect the "SingleProc" with your new element "Drain1" by a "Connector".
4. Double-click on the "SingleProc" and choose the tab "Exit Strategy". Choose the Strategy Percentage. Apply.
5. Click on "Open List" and fill in line 1 "80" and in line 2 "20".
   Note: Line 1 and 2 are ordered according to the number of the successors. To see this numbers you have to activate the button "Show Successors" (on the model-frame click View → Options → Show Successors).
6. Set the running time for your model on one day.
7. Determine the number of parts that entered the drain (look at the statistics of the drains).
8. Interpret the results.
9. Calculate if your simulation model distributed the entities in a correct way.

2 Further modeling, programming

2.1 Material Flow

Material flow in Plant Simulation occurs along Connectors. Apart from linear material flows, it is also possible to create parallel or circular material flows.
A material flow can be controlled by Methods.

2.1.1 *Standard transferring behavior*

The standard behavior when entities are transferred from station to station is by blocking. Transfer occurs according to the sequence of predecessors. If the station, where the unit should be transferred is occupied, the unit will wait in the predeceasing station until the target station is free. The standard transferring behavior follows the FIFO principle, First-In-First-Out.

2.1.2 *Output behavior*

The output behavior is chosen in the “Control” card of any object.

2.1.3 *Material flow control* 

The Material Flow control offers different control possibilities for splitting the Material Flow in a production line.

The Output Properties of the Flow Control

- To pass the *MUs* cyclically on to all successors, select Cyclic.
- To cyclically pass the *MUs* on to the successor according to the sequence of successors, which you entered into a list, select *Cyclic sequence*.
- To pass the *MUs* on to the successor, which has been waiting the longest for an *MU*, select *Least recently used*.
- To pass the *MUs* on to a successor according to the return value of a *Method*, select *Method*.
- To pass the *MUs* on to the successor, which has waiting the shortest time for an *MU*, select *Most recently used*.
- To pass the *MUs* on to a successor according to the values of attributes of the *MUs*, select *MU Attribute*.
- To pass the *MUs* on to a successor according to their names, select *MU Name*.
- To pass the *MUs* on to a successor according to a percentage distribution, select *Percentage*.
- To pass the *MUs* on to a randomly selected successor, select *Random*.
- To pass the *MUs* on to the successor meeting a certain Property, select Selection from the drop-down list. Then, select a *Property* from the drop-down list Property.
- To pass the *MUs* on to the first available successor, select *Start at successor 1*.

- To copy the *MUs* that enter, and pass a copy each to each of the successors, select *To all successors*.

2.1.4 *State of material flow objects*

The material flow objects provide an LED along the top boarder of the icon that displays the state of the object. Note that the LED can display several states at once, while state icons can only display single state.
The following colors represent these states:

- **Red** the object is **failed**.
- **Blue** the object is **paused**.
- **Green** the object is **working**.
- **Yellow** the object is **blocked**.
- **Brown** the object is **setting-up** (setup).
- **Light blue** the object is **recovering** (no_entry).

2.1.5 *Excursion: Successor and Predecessor*

The predecessors or successors of a component are numbered on the Connectors. With this numbering, a clear identification of predecessors and successors is possible. In manufacturing processes, the sequences and predecessors and successors are vital for determining the right product mixture.

Successor
After activating the function "Show Successors" the successors are arranged in the predetermined order. This property is located in the Frame under: View/Options/Show Successors.

Predecessor
After activating the function "Show Predecessors" the predecessors are arranged in their predetermined order. This property is located in the Frame under: View/Options/Show Predecessors.

2.1.6 *Exercise*

04 Exercise: A production line

A production line in a Frame is modeled. The line consists of a Source, two SingleProc stations (M1 and M8) and one Drain. Everything is interconnected with Connectors, one machine after another.
Insert an EventController with an operating time of eight hours. The bottleneck machine M8 has an operating time of eight minutes

Task:
1. Create a Frame with the name "_04_Production_Line" in the folder exercises.
2. How many units were produced after 8 hours?

05 Exercise: Auxiliary Capacity

To increase the capacity we add a second machine M4 parallel to the machine M8. M4 has an operating time of four minutes.

Task:

1. Duplicate the frame "_04_Production_Line" and rename it to "_05_Auxiliary_Capacity".
2. Apply the changes in the production mentioned above.
3. How many units were produced in 8 hours?
4. Did the production increase as it was expected? Why? Let the model run slowly.

Note:
See standard transferring behavior.

~~**06 Exercise: Material Flow Control**~~

~~Here you can practice the correct way to control the material flow in Plant Simulation.~~

~~Task:~~

1. ~~Duplicate the frame "_05_Auxiliary_Capacity" and rename it to "_06_Material_Flow_Control".~~
2. ~~Find the best configuration for increasing the utilization of the production line by changing the output behavior of machine M1.~~
3. ~~See statistic results on the drain component.~~

2.2 Components with Multiple Capacity

Till now we have considered only components with one unit capacity. However, Plant Simulation offers components with higher capacity. These are called multiple capacity components.

2.2.1 *Material components flow and Objects with multiple capacities*

Components like a Buffer, IOBuffer, Store, ParallelProc, Line, Container, Assembly and DismantleStations give more flexibility to work with "MUs" and other objects.

ConveyorLine 

Conveyors transport MUs over its entire length with constant speed. MU's cannot overtake others on the Line.
Conveyors are used as transport and distribution system (i.e. Conveyors at the post office). Conveyors can have an accumulation characteristic.

Container 

The *Container* is a moving object able to hold other MUs. It can be defined its size and thus its storage capacity by specifying the dimension along the x and the y-axes. The *Container* is used for modeling items that transport entities, e.g. palettes or boxes.

Components of "Store"

In Plant Simulation there are three different kinds of storage components. All three serve to keep the MUs in storage or in buffers.

- **PlaceBuffer:** 

  The PlaceBuffer is a place-oriented component. It lines up several processing units of the same kind one after the other. The processing units are connected and the MUs have to be processed at each station, thus they cannot pass each other. An MU may only leave the station after passing through all the buffer places. The operating time is distributed evenly among all places.

- **Buffer:** 

  The Buffer is not a place-oriented structure; therefore, the operating time is not also uniformly divided.

- **Store:** 

  The Store keeps the MUs in stock and is a matrix-oriented object. You can define the size of a store by specifying its x and y dimensions. Taking in-stock units can be done by standard distribution methods.
  Important: Taking out-of-stock units should be done by a method.

ParallelProc 

The Parallel Processing Unit (ParallelProc) consists of several processing units operating in a parallel form. It can process more than one MU at one time. Its characteristics correspond to those of the SingleProc component.
Note: Capacity and time specifications of a ParallelProc should be defined in its setting window.

Assembly - Dismantle Station

Assembly and dismantle stations serve to illustrate the handling of assembly and disassembly processes.

Assembly station 

The assembly station adds units to a main part and illustrates assembly procedures. It relocates the incoming units, according to the entries during assembly of the main unit.

Note:

Main part must be a Container. When the main part is an Entity, then modus "Delete MUs" must be selected in the property window.

Assembly properties

Assembly table selection:

If no assembly table is selected, then the component waits for an entity from each one of the predecessors.

If a predecessor's table is selected, then the components receive the entity from a predecessors list in which the predecessors and the quantity of the entities are declared.

If MU type is selected, then the component selects the MUs from a list in which the attribute "type" and quantity of entities are declared.

Main MU from predecessor (Number):

Here we fix the predecessor from which the main part will be supplied.

Assembly mode:

The assembly mode determines whether attachments are added to the main part or are destroyed after the assembly. For "Attach MUs" the parts remain added i.e. their statistics. "Delete MUs" deletes the register of the loads.

Exiting MUs:

Here we decide whether the main part, after assembly, will be transferred or there will be a creation of a new part and then transferred. To create a new object, the complete path has to be defined and can be selected from the "MU" box.

Sequence:

Here we make the choice of the sequence of the main part and needed parts or services during the assembly. There are three possible choices: "MUs then Services", "Services then MUs" and "MUs and Services".

- MUs before services: Here the goods are requested before the main part.
- Services before MUs: Here the goods are requested after the main part.
- MUs and services: Here the goods are requested at the same time with the main part.

Dismantle 

The dismantle station models disassembly processes. Attachments are unloaded off the main part or new parts are produced.

Dismantling Settings
Sequence:
There are three kinds of unloading sequence:

- *MUs to all successors:* it dependents on the dismantle mode.
- *Detach MUs:* Transfer the units sequentially to the predecessor. This will not relocate the unloaded main part of.
- *Create MUs:* MUs are created in a way that every predecessor receives one.

MUs existing independent of other MUs
The unloaded MUs leave the dismantle station independently and not related to the condition of the other MUs.

Main MU after other MUs
The main part leaves the station after the attached parts.

Dismantle mode:
The Dismantle mode is different when “Create MUs” or “Detach MUs” is selected. With the selection “Detach MUs” the main part is separated from attachment parts. With the alternative “Create MUs” attachments are the result.

Main MU to successor:
Here we select the connection in which the main part from the Dismantle Station will be transferred in the following sequences.

Exiting MU:
In Dismantling exists the possibility to transfer either the main part or a new one. When the property “Main MU” is selected, then the main part is transferred to the successor. When the property “New MU” is selected, the main part is deleted and a new MU is transferred to the successor.

2.2.2 *Exercise*

07 Exercise: Buffers
Here you will practice handling Buffers in a production line.

Task:

1. Create a Frame with the name “_07_Buffer” in the exercises folder.
2. Provide a production line with a Source, a Buffer, a SingleProc, a Drain and connect them with Connectors. Add also an EventController and a Time Sequence. Assign ten minutes operating time to the SingleProc.
3. Run the model.
4. How many parts are processed in 8 hours? What is the difference between the type “Queue” and “Stack”?

Note: TimeSequence is an ongoing list that can trace key factors along time. In the index card “Record” you should write the appropriate command. For this particular exercise, you should write “buffer.numMu”, mode: “Watch” and mark the box “Active”.

It is recommended to program a Reset-Method. So it is secured that the TimeSequence gets deleted before every simulation run. Therefore create a Method and name this object "Reset". Open the Method and type in the following text:

```
is
do
        timeSequence.delete;
end;
```

08 Exercise: Multiple capacities

Here you will practice with parallel stations in the application of an industrial washing machine.

Task:

1. Create a Frame with the name "_08_MultipleCapacity" in your exercise folder.
2. Insert in the Frame a production line with only one ParallelProc (Capacity 4) and another production line with a flow control before four SingleProc stations. The operation time of each station is 10 min.
3. Run your model.
4. How many parts are washed in one hour?

09 Exercise: Paint Shop

Exercise for assembly and dismantle in a Paint Shop.

Description of the Paint Shop:

In the Paint Shop boxes (Entities) are painted. The casings are moved on hangers (Containers) through the paint cell. It is possible to place boxes on a hanger. The material flow starts from a source (Source_parts) and proceeds through a buffer (Partsbuffer), a hanging station (Hanging), preprocessing (Pretreatment), painting (Painting), drying (Drying), uncoupling (Take_down), and ends up in drain (Drain). The flow of the hangers occurs along the following path: Source (Source_hanging), buffer (Hangingbuffer), hanging (Hanging), preprocessing (Pretreatment), painting (Painting), drying (Drying), uncoupling (Take_down) and back to the buffer (Hangingbuffer).

At "Take_down" the material flow gets splitted: The boxes are moved to "Drain", the hangers next station is the "Hangingbuffer".

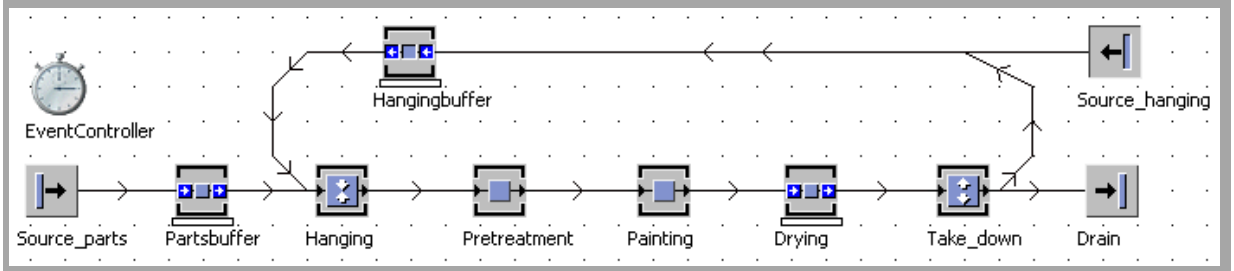

Figure 2-1: Exercise Paint Shop

Data for the Paint Shop:

Table 2-1: Data for exercise Paint Shop

Component	Time	Quantity
Source_parts	1 min	
Source_hanging	1 sec	10
Partsbuffer	1 min	10
Hanging_buffer	1 min	10
Pretreatment	1 min	1
Painting	1 min	1
Drying	1 min	4

Task:
Create a Frame with the name "_09_PaintShop" in the folder exercises.
Build the model as shown in the picture.

1. Run the model.
2. How many boxes are painted in one hour?

Note: To adjust the amount of Source_hanging double click on Source_hanging. In the window „Time of creation" select „Number adjustable" and select the desired amount.
To rotate the icons displayed, click on them once and then select in the menu „Icon" and „rotate". (or with Strg+T).

2.3 Resources

Besides basic components, material flow components and multiple components, Plant Simulation also offers components for representing resources. These resource objects are intended for modeling how and when "Workers" move from the "WorkerPool" to the "Workplace" of a station.

The Resource objects are: The Workplace, the FootPath, the WorkerPool, the Worker, the Exporter and the Broker. When modeling with resource objects, there are two modeling concepts to choose from:

- Broker-Importer-Exporter-concept.
- Worker-WorkerPool-Workplace-FootPath-concept

The second concept is a more elaborate version of the first one.

Workplace 

The *Workplace* is the actual place at the station where the worker performs his job. You can assign a *Workplace* to the material flow objects that also support an *Importer*, i.e. to the SingleProc, the ParallelProc, the Assembly station and the DismantleStation.

FootPath 

You can link several *FootPaths* with *Connectors* to create a network of *FootPaths*. A network of *FootPaths* consists of all *FootPaths* that are joined with *Connectors*, as well as the *WorkerPools* and the *WorkPlaces* which are joined with one of these *FootPaths* with a *Connector*.

The *Worker* moves on this network from his *WorkerPool* to the *WorkPlace*. Provided the *WorkerPool* and the *WorkPlace* are connected to the same network of *FootPaths*, the *Worker* moves from the *WorkerPool* to the *WorkPlace* and back on the shortest possible route.

WorkerPool 

The *WorkerPool* is a place where *Workers* are created and stationed when they do not work or when waiting for an order. You can define teams of workers in the worker pool and assign them different skills, efficiencies, walking speeds, etc. (in the worker Creation Table).

Worker 

The *Worker* represents a working person that performs a job on a Workplace. The *Workers* are created in the WorkerPool.

Exporter 

The *Exporter* exports services. It works in cooperation with the Broker and the *Importers* of the *SingleProc*, the *ParallelProc*, the *Assembly* station and the *DismantleStation*. The *Exporter* offers services and provides them for *Importers*. Think of the *Exporter* as a group of people whose individual members you cannot distinguish and who you cannot address as individuals. If you would like to distinguish individual staff members, you have to model them with individual *Exporters*.

Broker 

The *Broker* works in cooperation with the Exporter and the *Importers* of the *SingleProc*, the *ParallelProc*, the *Assembly* station, and the *DismantleStation*.

The *Broker* is the go-between for services offered and services required. Each *Broker* can manage several *Exporters*, that tender services, and it may receive requests from several *Importers* that require services. A request consists of a list of services and the amount of services necessary. A service name is a *string*.

10 Exercise: Basic Resources

Here you will practice handling Resources in a production line.

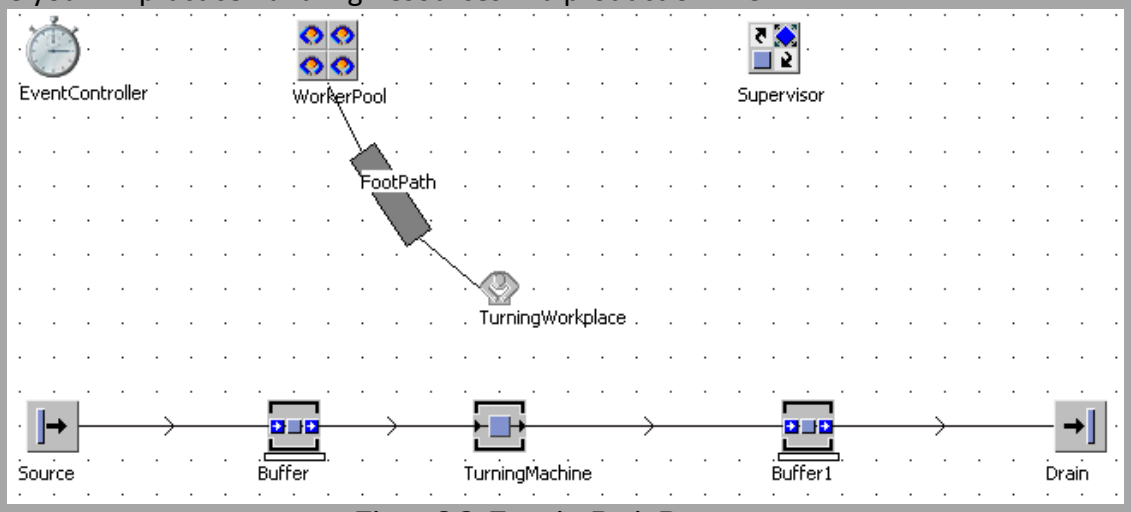

Figure 2-2: Exercise Basic Resources

Task:

1. Create a Frame with the name "_10_BasicResources" in the file exercises.
2. Provide a production line with a Source, a Buffer, a SingleProc, a Place Buffer and a Drain and connect them with Connectors. Add also an EventController.
3. Assign ten minutes operating time for the SingleProc "TurningMachine".
4. Insert Workplace "TurningWorkplace", Workerpool and Broker "Supervisor" in the model. Create a FootPath between WorkerPool and TurningWorkplace by clicking at the FootPath icon at the toolbar and then click at WorkerPool and TurningWorkplace. Open your FootPath and define a length of 3m.
5. Open the SingleProc "TurningMachine" and select in the Importer tab, Broker space "Supervisor" and click Active.
6. Open TurningWorkplace and assign SingleProc "TurningMachine" in the station column.
7. Open WorkerPool and assign at the Attributes tab in the Broker column "Supervisor". Create a table with the following attributes: Amount of worker 1, speed 2 and efficiency 90.

Note: In order to be able to modify the WorkerPool table, first press the small button besides the Creation Table (deactivate).

8. Run the model and understand the behavior of the production line and the worker.

11 Exercise: Human Resource Problems

Exercise for a Human Resource Problem in a small company.

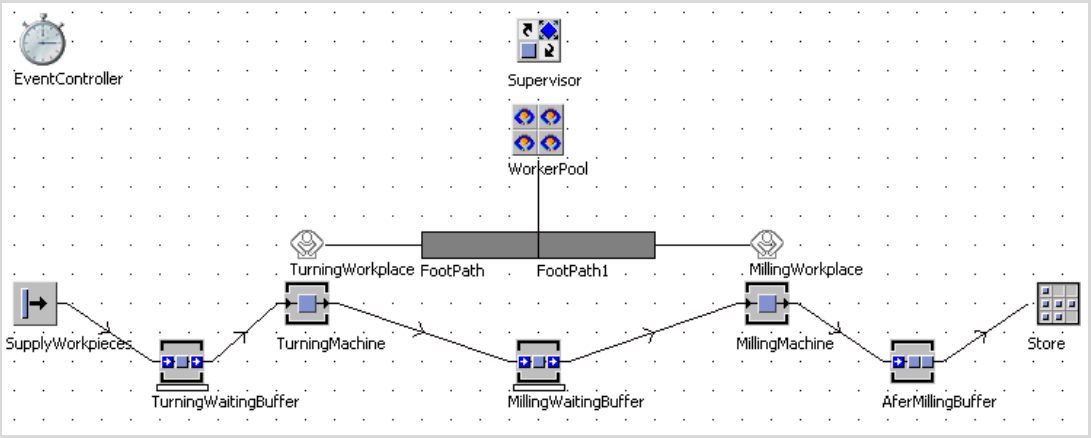

Figure 2-3: Exercise Human Resource Problems

Description of the Human Resource Problem:
In a new small company, the production manager decided to involve two new operations: turning and finish-milling. The process can be described in the following way: Workpieces (Entities named Workpiece) are supplied by the customer (Source) and wait in the TurningWaitingBuffer (Buffer) before being process by TurningMachine (SingleProc) if the machine is free. After turning, it waits in the MillingWaitingBuffer (Buffer) before being processed by the MillingMachine (SingleProc). The Workpiece will wait in the AfterMillingBuffer (Buffer) before being stored (Store). Now the production manager has to decide whether to assign 1 or 2 workers for one process line that operates 8 hours per day.

Production Line Data:

Table 2-2: Data for production line

Component	*Time*	*Capacity*	*Other*
SupplyWorkpiece	10 min		
TurningWaitingBuffer	1 min	10	
TurningMachine	5 min	1	
MillingWaitingBuffer	1 min	4	
MillingMachine	15 min	1	
AfterMillingBuffer	1 hour	10	
Store		100	
Length of the track			4m
Speed of worker			2 m/s
Efficiency of worker			90%

Task:
Let's say you are the production manager of the company and you have to decide how many workers you will assign to each single production line. For solving the task you decide to use simulation, based on the knowledge you got during your master studies at TUB.

1. Create a Frame with the name "_11_HumanResourceProblem" in the folder exercises.
2. Build the model as shown in the figure.
3. Run the model.
4. How many work pieces can be process in one day?
5. For each work piece, the company earns 10€ and for each worker the labour cost is 10€/hour. Decide how many workers you will assign to this production line. (Just consider, Profit = Earning-Labour cost).

2.4 Sim-Talk Basis

Plant Simulation software allows to build models with predefined components without having to write any programming code. This makes Plant Simulation a fast and simple modeling tool. However, the use of a programming language is necessary in order to build complex models. Plant Simulation uses the Sim-Talk programming language. The code of Sim-Talk is implemented in the form of Methods. The Method object is located in the toolbox under "Information Flow".

2.4.1 *Method component*

To use Sim-Talk, first at all, it is needed to insert a "Method" into the Frame. The editing window of a Method appears by double-clicking on the method icon. The source text is divided into two sections: Variable declaration that is located between the "is" and "do" command; and Sim-Talk programming code that should be written between the "do" and "end" command.

Note:

Before closing the editor window, it is important to examine whether the method is properly saved in the Class Library. To do this, click on the green Check icon "Apply Changes (F7)".

Tool Bar

The Tool Bar is present at the method window.

Here you can find many functions directly. Some examples: Do (Strg+Z) and Undo (Strg+Y), Find (Strg+Y), Toggle Bookmark (Strg+F2), Run and Break (F11), Run (F5), Inherit Source Code (F6) and Apply Changes (F7).

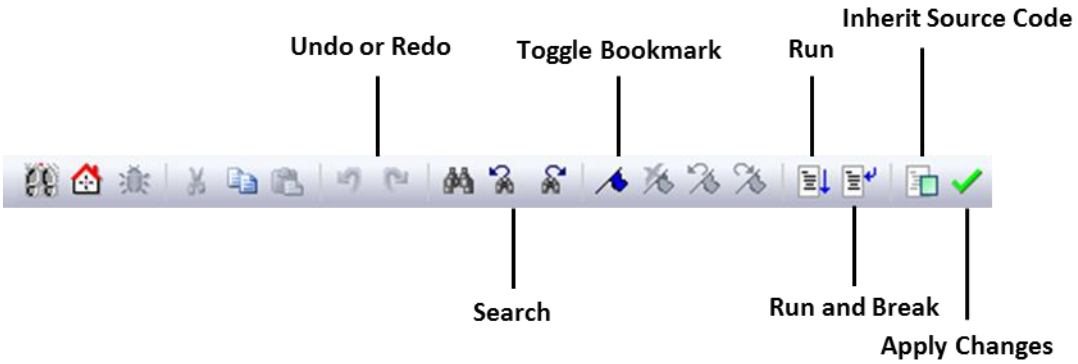

Figure 2-4: Tool Bar of Plant Simulation

Function:
Undo (Ctrl+Z) and Redo (Ctrl+Y), the last changes in the source code can be repaired and/or retrieved.
Find (Ctrl+F), here one can look for a word or indication in the source code of the method.
Placing *Toggle Bookmark (Ctrl+F2)*, in long source codes we can set bookmarks.
Run and Break (F11), this tool serves as line-to-line execution of the program.
Run (F5), all instructions in the method are implemented.
Inherit Source Code (F6), you must deactivate this Button to work on the source code of a method.
Apply Changes (F7), through this button you can save changes in the source code. Changes are saved in the object "Method".

Comments

Comments serve to provide the method source code with notes in order to make the formulated instructions more understandable. It is also recommended to note here which object has called the method.

"--one line Comments"

"/* more than one line
Comments */"

2.4.2 *The Console and Working with Methods*

Information in the Console is a useful and easy way of getting data from a simulation run. The command "print" writes the information in the Console.
Note: In order to see the Console, you should activate it in the tool menu under "View/Viewers/Console".

2.4.3 *Special Methods and Method Names*

callEvery

The command "callEvery(<path>,<string>[,<arguments>]);".calls all methods with the name passed by the argument <string> in the *Frame* denoted by <path> and all of its sub-frames.
The program stops executing the method, where the *callEvery* is located, until all methods called are executed. The optional arguments are passed as arguments to the calling methods.
The sequence, in which the program searches the *Frames,* is determined by the depth-first strategy. Methods found will always be executed immediately.
Below is an example for a depth-first strategy:

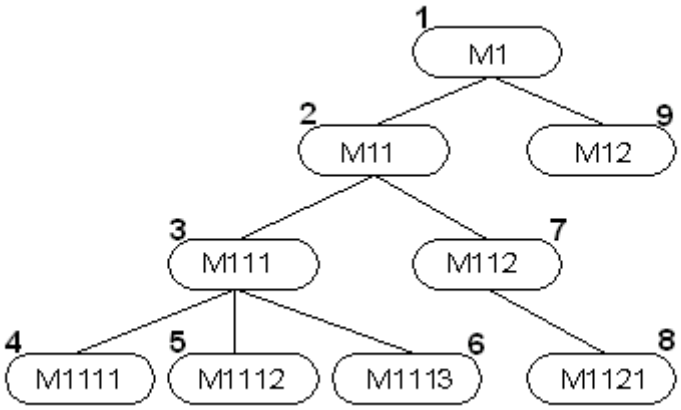

Figure 2-5: Depth-first strategy

We placed several *Frames* within *Frames* in the model. The numbering M x indicates the sequence in which the *Frames* were inserted, i.e. *Frame* M 11 was placed before *Frame* M 12, M 111 before M 112, etc. The numbers 1 to 9 indicate the sequence the program searches for methods with the name specified in *callEvery*, i.e. first in 1, then in 2 etc.
The depth-first strategy starts with the *Frame* entered in <path>, and then searches the hierarchy.

Ref
The "Ref" can be called by using, "ref(<object>);". It prevents accessing the contents or calling the method and returns the reference as an absolute path of type *object*. This reference or the path may then be entered in lists or tables and be passed as argument to other methods.
Wait
The keyword „Wait(<Time>)" interrupts the execution of a method for the duration specified in the field time. This event will be taken into account for the *EventController.*

2.4.4 *Exercise*

12 Exercise: Practice in Console
Here we will practice on how to run a method.
Task:

1. Create a Frame with the name "_12_Practice_in_Console" in your exercise folder.
2. Write "*print "Hello World";* " in the text field of the method.
3. Start the method by pressing the "F5" key; or close the method after saving and press the right mouse button on the method and choose "Run".

2.5 Sim Talk Value Assignment

It is possible to assign values to variables in a method or by calling another method.

Example of adding numbers
By adding, the sum of two or more numbers is assigned to a (local) variable. The assignment is made by the code “C := A+B” the value is assigned from right (A,B) to left C.

Assignment and Return Values
With the help of assignment parameters, it is possible to define a value within a method. The assignment is made as follows: The type of the variable has to be defined before the word “is” in brackets, e.g. „(value1 : real)“.
The return parameter type should be declared after the brackets, two points and the type name, e.g. “:real”. The return parameter can be delivered from one method to another.

Format Types
There are different format types in Plant Simulation. These types are important when assigning a value to a variable. The following data types exist:

Table 2-3: Data types and values

Data Type	Value
boolean	TRUE or FALSE
integer	Integer value
real, length, weight, speed, money	Floating point number
String	Characters, numbers and special characters
Date	Date statement (dd.MM.yyyy)
Time	Date statement, including the time (hh:mm:ss.ss)
datetime	Date statement, including the time (dd.MM.yyyy HH:mm:ss)
list, stack, queue	List with one column
Table	List with two or more columns
object	Reference to a simulation model or an object

2.5.1 *Setting critical points*

Setting critical points gives the users an easy way to monitor the execution of a method. After setting the critical points the method can be executed step by step. The critical points are set and reset in the chosen place with the key “F9”. When running the method, the program will execute up to the critical point. Clicking on the button in the menu bar the simulation can be continued or stopped.

Note: 
This is the icon for debugging.

Tool Bar of the debugger

It helps the debugging activity of a complex simulation model. When the program executes a *Method*, you may press the key combination Ctrl+Shift+Alt, to open the *Debugger*.

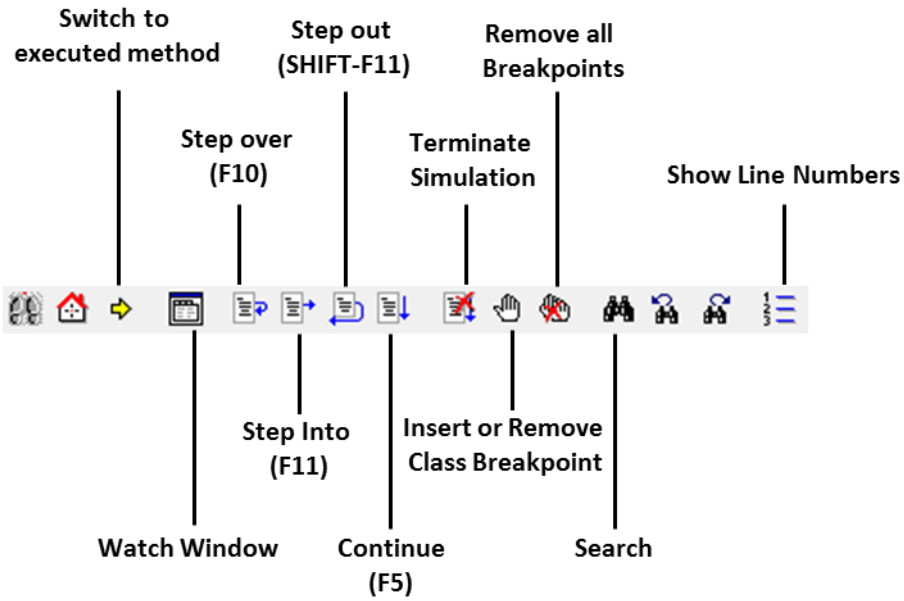

Figure 2-6: Tool Bar of debugger

Monitoring window

In the debugging mode we can open a monitoring window i.e. the help window with the “F12” key. The program will check all local variables in the method and their values and displays how these variables change step by step. For each method in the debugging mode a monitoring window is opened.

2.5.2 *Exercise*

13 Exercise Assignment

Here we will practice on how to assign values to variables.

Task:

1. Make a new Frame with the name “_13_Assignment” in the folder Exercises.
2. Enter a Method M1.
   Write down the source code as shown below in the Method (M1):

```
is
        sum : real;
do
        sum := 10 + 5;
        print “Result”, “: ”, sum;
end;
```

3. Run the method.

14 Exercise Return parameter
This exercise deals with the delivery and return of parameters.

Task:
1. Create a frame with the name "_14_Delivery_Return_Parameter".
2. Insert two Methods M1 and M2.
3. Provide the source code in M1, as shown below:

```
is
        sum : real;
do
        sum := M2 (10, 7.5);
        print "Result :", sum;
end;
```

4. Source code for M2:

```
(number1, number2 : real) : real
is
do
        result := number1 + number2;
end;
```

4. Start the method M1 and explain the result.

15 Exercise Monitoring Window
Handling the debugger-monitoring window.

Task:
1. Produce a Frame with the name "_15_Monitoring_Window" in the folder exercises.
2. Copy the two methods from the exercise "_14_Delivery_Return_Parameter".
3. Set a critical point in line 4 in method M1.
4. Start now the method M1 and open then the methods monitoring windows.
5. Run the methods step by step and explain the steps of the execution.

2.6 Sim Talk Name, Designator, Path

The name and the path identify objects and local variables in a model. Certain limitations apply when choosing a new name. Self-explanatory names, such as .building1 for a *Frame* or forklift for a *Transporter*, make the model's re-work and maintenance easier.
Names, information about the object's path and so-called "Anonymous designators" are used by Plant Simulation in order to facilitate the work with objects.

2.6.1 *Path*

In the previous section we talked about the various name conventions for the objects. Basically, a clear identification of an object is of great importance. The name conventions provide a clear identification in a name area (e.g. in different hierarchy levels of the model). Also objects in other hierarchy levels have to be clearly identified. To make it effective, objects are described via their hierarchy. This description is called "The Path". Thereby, Plant Simulation differentiates between absolute paths and relative paths. The **absolute path**

describes the way from the highest level to the desired object. The **Relative path** describes the way from the current point to the object.
Paths lead to the location of an object within a model. The descriptions of paths are subject to special characteristic.
Each object carries an individual address. We must know these, in order to be able to address the object clearly. We can differentiate according to whether the object is on the same hierarchic level, in the same name area, or in another name area.
With the *relative path* we address the object only if it is in the *same name area.*
We need the absolute path, in order to address directly the object in other hierarchic level.

Name area
All objects on same hierarchic level form a name area. Objects cannot have the same name within a name area. Outside of the name area, identical designations are possible.

Absolute Path
The absolute path describes the location of an object on the basis of the highest hierarchy level. This indication is independent of the location of the calling object. Absolute paths begin themselves always with the class library, following then along respective files and Frames. They consist of names and points. The points serve as separators. The first point represents the origin of the absolute path.
.<Name>.<Name>.<Name>...
The last element of the path consists of the names of the objects addressed in each stage. Following absolute paths consumes significant computing time running the simulation. When the hierarchic levels are changed, the absolute paths must also be renamed.

Relative Path
Relative paths of an object represent the current position of an object. They ***never begin with a point***.
<Name>.<Name>.<Name>...
Relative paths require less computing time than absolute paths. Moreover, methods and components are thereby flexible and universally applicable. The path continues to point to the object even if the component in the hierarchy has been shifted.

2.6.2 *Anonymous identifier*

Anonymous designators, in addition, aid the way to designate objects as universally as possible. They are used in order to add flexibility. They have the advantage that you can insert methods or components by the use of anonymous names more flexibly and more universally into your model.
Normally, the name of a control you defined in a *Method* object does not change. But, the path to the control changes from model to model. If a control requires the current path, query it using anonymous identifiers. Anonymous identifiers make a control more flexible and independent of their context. You may insert them into different models without having to modify them. The following examples refer to the model shown below. F is the top level Frame in the hierarchy. Frames L and W contain additional basic objects.

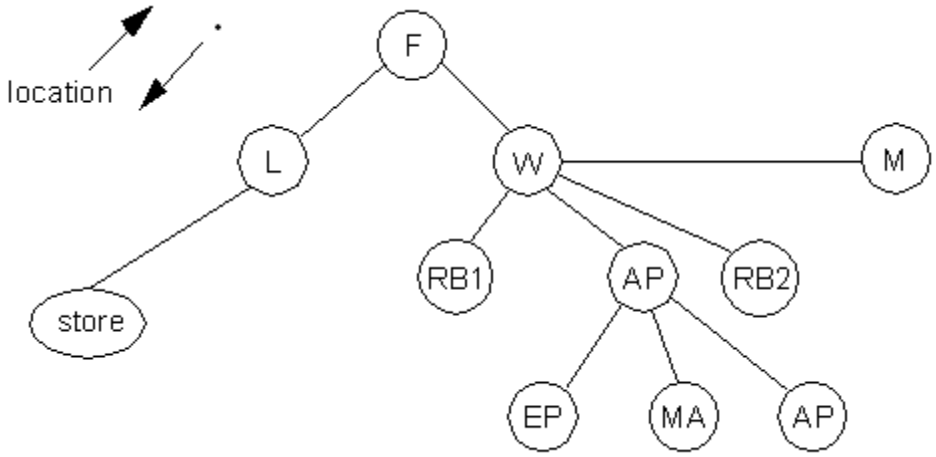

Figure 2-7: Anonymous identifier

MU - @ (at)

The anonymous identifier *@* is used to represent the MU that has triggered the control. When you set an entry or exit control into a material flow object, the anonymous identifier @ allows you to access the MU that entered or is ready to exit.
The assignment is unique even if several controls are triggered at the same simulation time.

Example:

@.move(parallelproc.succ(3));

Current Method – self

The anonymous designator *self* returns the current path and name of the method. "self.Name" returns only the name of the method. As the anonymous identifier *self* returns the currently executed method (*object),* if in the diagram above the method *M,* the *Frame* .F.W is located. Then if we use the anonymous identifier *self* the result will be the path .F.W.M.

Example:

print self.name;

Calling Component - ? (question mark)

The anonymous identifier *?* represents the material flow object or the control that called the *Method*. This component can be in another method.

Example:

?.cont.move(E2); -- moves the contents of the object that
-- called the method to E2.

Current Frame – current

The calling through "current" refers to the path under the Frame (including the frame), in which the method is used. The location of a Frame can be registered for example in lists and tables. In this way you may easily enter the location of a *Frame* into lists and tables or pass it as an argument to methods in other *Frame*s. In the example above *current* returns the location .F.W in the method *M*.

You may also use *current* to distinguish between a local and a global variable of the same name. The argument <name> identifies a local variable, while *current*.<name> stands for the global variable.

Position of the current Frame: "Location" or "~"

Location or ~ refers to the component in the structure hierarchy directly over the object. This is helpful particularly with deeply interlocked Frames. The location of an MU is the object where it is currently positioned. For all other objects, location returns the *Frame* where they are placed. The return value is of type *object*.

Example:

print "shaft is on:", @.location;
print "shaft is on:", @.~;

Highest Frame hierarchy - root

The calling root refers to the highest Frame in the hierarchy. The anonymous identifier *root* returns the top level *Frame* in the hierarchy. This identifier is especially helpful when you do not know the name of the root *Frame*, i.e. the *Frame* highest up in the hierarchy.

Transfer an Object:

@.transfer

Usage:
<mu_path>.transfer;
<mu_path>.transfer(<mu_location>);
<mu_path>.transfer(<integer>);
<mu_path>.transfer(<mu_location>,<length>);
<mu_path>.transfer(<integer>,<length>);

The method *transfer* transfers the MU to the next object. Contrary to *move*, the method *transfer* completely removes the MU from the source object. On location-oriented objects both methods display identical behavior. On length-oriented objects the method removes the entire MU from the source object. The argument <integer> denotes the number of the successor; the argument <length> is the position on a length-oriented object. The method returns FALSE when the *transfer* failed (*boolean*).

Example:
@.transfer(line);
@.transfer(line,3.3);

@.move

Usage:
<mu_path>.move;
<mu_path>.move(<mu_location>);
<mu_path>.move(<integer>);
<mu_path>.move(<mu_location>,<length>);
<mu_path>.move(<integer>,<length>);

The method *move* moves the front of the MU. When you do not enter an argument, the program moves the MU to the different successors of the location. If <mu_location> is present, the MU is moved there. The term <integer> is short for mu_path.location.succ(<integer>), i.e. the MU is moved to the successor with the index <integer>. Note that this notation is processed faster than the full form. It is not permitted if an MU is located on another MU. The argument <integer> denotes the number of the successor; the argument <length> is the position on a length-oriented object.
The program returns TRUE when it moves the MU successfully, FALSE when it fails (*boolean)*.

Example:

```
.MUs.entity:1.move(line);
.MUs.entity:2.move(parallelProc[2,5]);
.MUs.transporter:3.move(track,3.3);
```

Location-oriented objects, such as *SingleProc* or *ParallelProc*, always take in the complete MU while moving. A MU exits the first object entirely and enters the second in its entirety, independent of its length.
Length-oriented objects, such as *Line* or *Track*, on the other hand, only take over the front of the MU immediately and the remainder continually according to the speed you set. So, long MUs may be located on more than one objects at the same time. It is only going to be displayed on the object where its booking point is located!

Example:

```
line.cont.move;
station.cont.move(station.succ(2));
```

Note: You cannot transfer MUs to Source *objects*

2.6.3 *Exercise*

16 Exercise: Designator_self

Task:

1. Create a Frame with the name "_16_Designator_self" in the "Exercises" folder.
2. Insert a method with **Your Name** into the Frame.
3. Insert a method with the source code "print self".
4. Run the method.
5. Additionally, insert the source code "print self.name;".
6. Run your method again.

17 Exercise: Designator_QuestionMark (?)

Task:

1. Create a Frame with the name "_17_Designator_QuestionMark" in the "Exercises" folder.
2. Insert two Methods with the names M1 and M2.
3. Insert the source code "*M2;*"into the method "M1".
4. And in M2 "*print "M2 is called from: ", ?;*".
5. Run the method M1.

~~**18 Exercise: Designator_current**~~

~~Task:~~

~~1. Create a Frame with the name "_18_Designator_current" in the "Exercises" folder.~~
~~2. Insert a method into the Frame.~~
~~3. Provide the source code "*print current;*".~~
~~4. Run the method.~~

19 Exercise: Designator_Location_or_Tilde

Task:

1. Create a Frame with the name "_19_Designator_Location_or_Tilde" in the "Exercises" folder.
2. Insert a further Frame into the Frame with the name "Inter_Network" (from the menu MaterialFlow).
3. Activate the option "Modify Structure" in "Inter_Network". Mark the selection in the drop down menu following the path Tools/ Modify Structure.
4. Insert a Method in "Inter_Network".
5. Insert the source code in the method "*print ~, " and", Location ;*".
6. Run the method.

20 Exercise: Designator_ root

Task:

1. Create a Frame with the name "_20_Designator_root" in the "Exercises" folder.
2. Add into the Frame a further Frame with the name "Inter_Network".
3. Activate the option "Modify structure" in the internal Frame.
4. Insert into the internal Frame a method.
5. Insert the source code in the method "print root;".
6. Run the method.

2.7 *Conditional Instruction*

Basic conditional loop – if

With the conditioned bypass you can make processes depend on the evaluation of a condition. If the condition is valid (i.e. the condition supplies a TRUE value), then the instruction 1 is implemented. If the condition is not valid (i.e. the condition supplies a FALSE value), then the instruction 2 is implemented, and vice versa.

Repetitive conditional structure - if, elseif

With the repeated condition, it is possible to react to different conditions. Plant Simulation evaluates the conditions until it meets the appropriate condition. The implementation of the instruction is sequential. If no condition is applicable, then the last optional "else" is executed.

Overview – inspect

The control structure "inspect" does a simple selection from different cases. It is possible to choose between different conditions and replaces longer "if-then-else-if" loops.

2.7.1 *Exercise*

21 Exercise: Conditional Structure_if

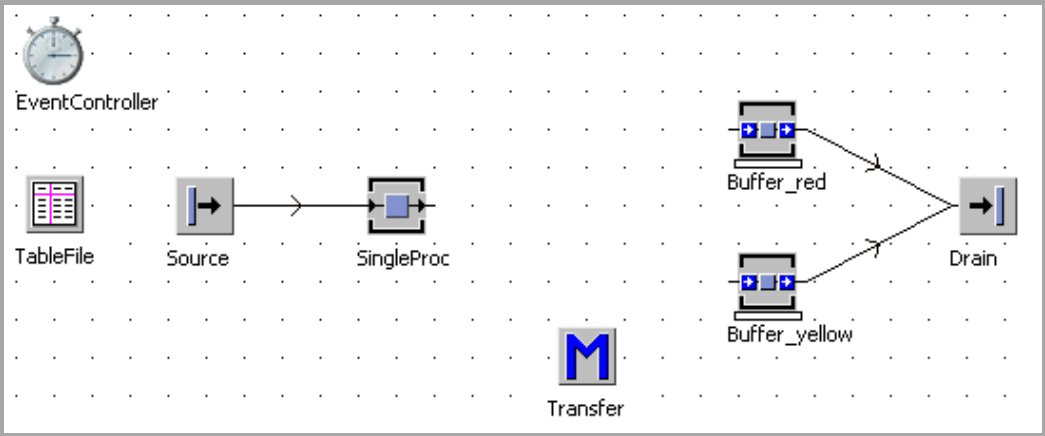

Figure 2-8: Exercise Structure_if

	object 1	integer 2	string 3	table 4
string	MU	Number	Name	Attribute
1	.MUs.Red	1	red	
2	.MUs.Yellow	2	yellow	

Figure 2-9: Exercise Structure_if

Task:

1. Create a Frame with the name "_21_ConditionalStructure_if" in the "Exercises" folder.
2. Create the production line shown above in the Frame.
3. Now open the source and select "Sequence Cyclical" in the "MU Selection" option.
4. Select the Table File with the button "Table" and then select "OK".
5. In the class library open the folder with the name "MUs" and "Duplicate" the "Entity" two times and name the new objects with "Red", and "Yellow".
6. Then, open the table and register your contents as shown above.
8. Insert the source code as shown below into the method "Transfer":

```
is
do
        if @.name = "red"
                then @.transfer (Buffer_Red);
        else
                @.transfer (Buffer_Yellow);
        end;
end;
```

9. Open the component SingleProc and by double clicking on the icon. Then open the register map "Controls" and in "Exit" select the Method "Transfer".
10. Start the simulation and observe the material flow.

22 Exercise: Conditional Structure_if_ elseif

Task:

1. "Duplicate" the Frame "_21_ConditionalStructure_if" and rename the new Frame as "_22_ConditionalStructure_if_elseif".
2. Insert another more Buffer "Buffer_blue" in the Frame and connect it with the Drain through a Connector.
3. Insert the third row in the Table File for the third entry named "Blue". Also create a corresponding MU for Blue; the number is 1 and the name is "blue".
4. Change the source code of the method "Transfer" with the following one:

```
is
do
        if @.name = "red"
                then @.transfer (Buffer_Red);
        elseif @.name = "yellow"
                then @.transfer (Buffer_Yellow);
        else
                @.transfer (Buffer_Blue);
        end;
end;
```

5. Start the simulation and observe the material flow.

23 Exercise Monitoring_inspect

Task:

1. "Duplicate" the Frame "_22_ConditionalStructure_if_elseif" and rename it with "_23_Monitoring_inspect".
2. Change the source code with the following text in the "Transfer" method:

```
is
do
        inspect @.name
        when "red" then @.transfer (Buffer_Red);
        when "yellow"  then @.transfer (Buffer_Yellow);
        else @.transfer (Buffer_Blue);
        end;
end;
```

3. Start the simulation and observe the material flow.

3 Further programming & information flow

3.1 Calling of Material Flow Methods

You need to call a method in order to direct materials in a desired direction through the simulation. To control material flow, the methods are triggered by the Material Flow component at predefined points. The methods have to be handled at exact time and sequence in order to start and maintain a smooth production system.

Entry and Exit Control

This is the equivalent of a light sensor, which checks the entrance and/or the exit of the Material Flow component. The entry and exit control can be connected with methods that are executed as soon as a good enters and/or leaves a component.

Connection of a method to a component

In order to tie up a method to a component, one has to register the names of the method into the appropriate field in the component's property window.

Place-oriented Components

With place-oriented components the standard relocating behavior is present in the front of the mobile unit. If the exit control is set only at "Rear" there is no any relocating behavior. With the selection of "Rear" and "Front" the standard relocation behavior is retained.

Length-oriented Component

With length-oriented components the standard relocating behavior is present in the "Front" and the "Rear" control.

Entry and Exit control for not length oriented objects

Figure 3-1: Entry and Exit control

The main sensors are entrance and exit control. The setup control is triggered when a setup process starts and ends. You can activate the sensor with the front or the rear. The choice depends on the practical case. The rear exit control is triggered when the part has already left the object. For counting, this is the right choice. If you would trigger the control with the front, the MU will still be on the object. If the subsequent object is faulty or is still occupied, the front sensor is triggered twice (once when trying to transfer, the second time when transfer is done). If you select "front" in the exit control, then you need to trigger the transfer by SimTalk, even if a connector leads to the next object (e.g., @.move or @.move(successor)). If you press the F2 key in a field in which a method is registered, Plant Simulation will open the method editor and will load the relevant method.

Entry and Exit control for length-oriented objects

With the backwards entry and exit control are important to note that the orientation of the MUs does not change. The same front and back control remains active.

Note:
Every material flow element has an entrance- and exit-control. You find the controls by double clicking on the material flow element and choosing the index card „controls“. These entrance- and exit-controls can be (but must not) connected with a method. This method tells the station or buffer what to do. For example wait for 10 MUs and pass them to the next station.

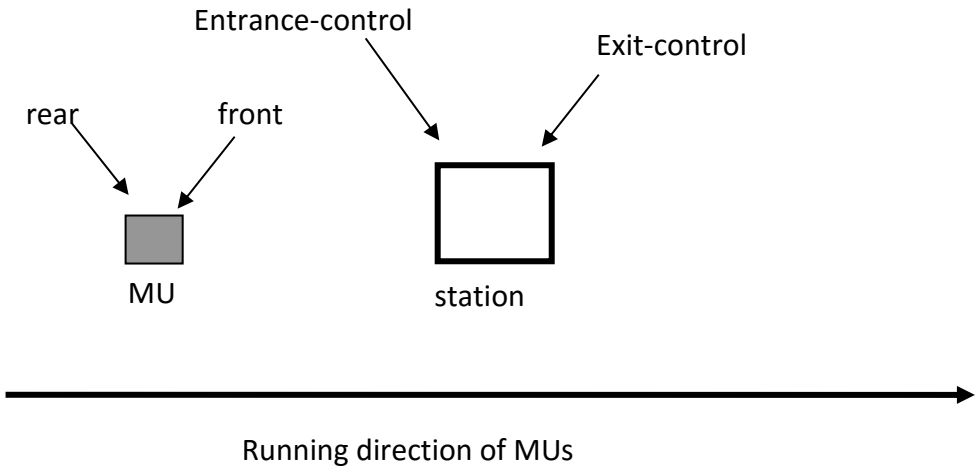

Figure 3-2: Control I

If the entrance-control of the station is on „rear“, then the rear of MU will activate a sensor and the sensor activates the method

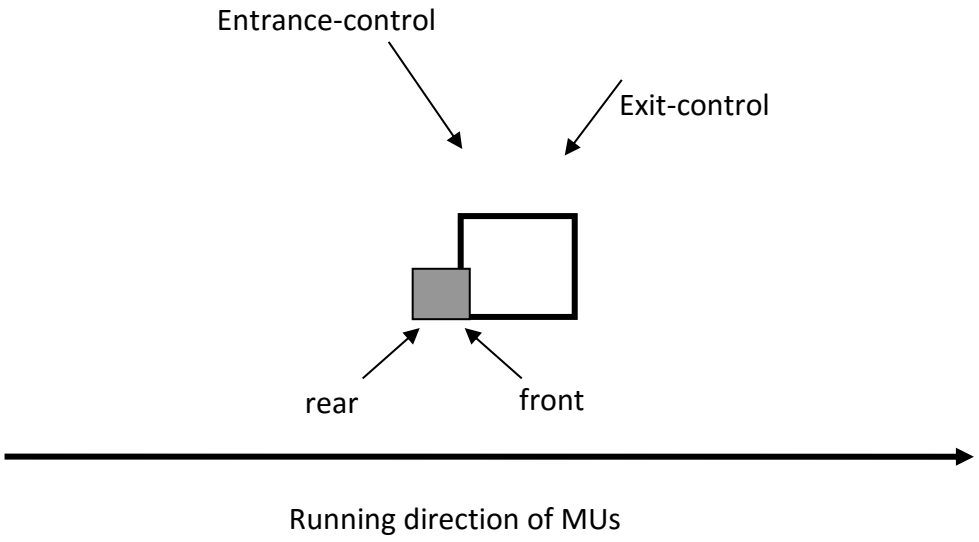

Figure 3-3: Control II

The MU (in Figure 3-3: Control II.) is not yet in the station! If the entrance-control of the station is adjusted on “rear”, then only the rear of the MU will activate the sensor. Only then is the MU in the station (see Figure 3-4: Control III).

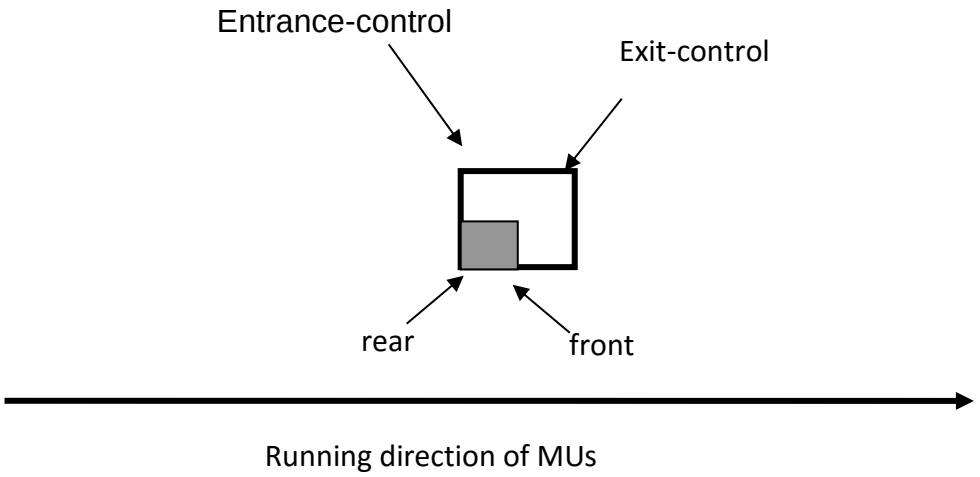

Figure 3-4: Control III

If the exit-control of the station is adjusted on “front”, the method is executed before the unit leaves the station (see Figure 3-5: Control IV). It can happen that MUs cannot leave the station because the method does not specify where it has to move. If this occurs, the station will be blocked (the MUs cannot leave the station).

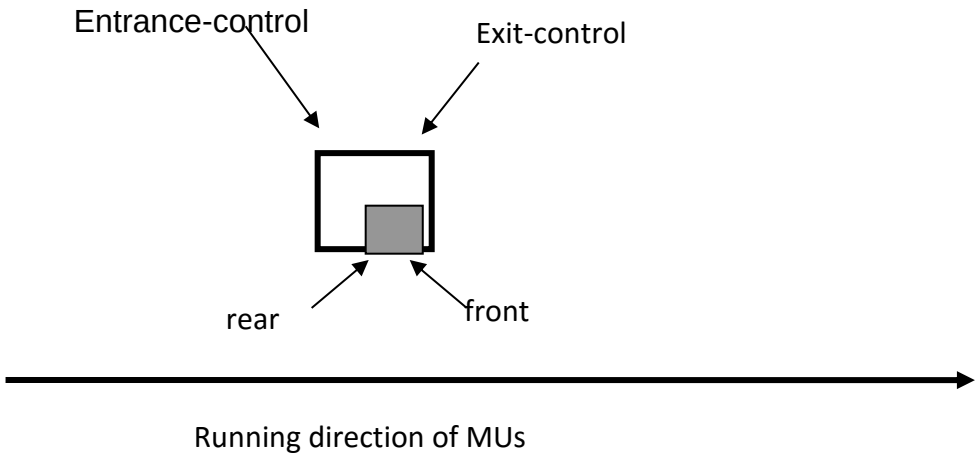

Figure 3-5: Control IV

If the exit-control of the station is adjusted on “rear“, the back of the MU activates the method. The MU has left the station already and is on the way to the next station (see Figure 3-6: Control V).

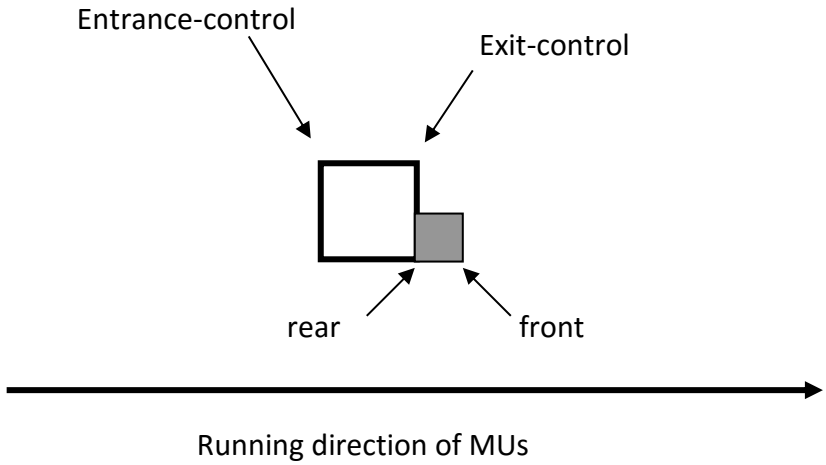

Figure 3-6: Control V

3.1.1 *Exercise*

~~**24 Exercise: Entry- and Exit Control at Stations**~~

~~Task:~~

1. ~~Create a Frame with the name “_24_Entry_and_ExitControl_at_Stations” in the „Exercises” folder.~~
2. ~~Create a production line with a Source, three SingleProc (E1, E2, and E3) and one Drain connected each other by Connectors. Also insert four methods (E1_out, E2_in, E2_out, E3_in).~~
3. ~~Insert the instruction “*print self.name;*” into all methods.~~
4. ~~Insert the methods in the controls from the stations; i.e. call the methods from the station, as follows:~~
   ~~E1_out: Rear - controlled Exit Control E1.~~
   ~~E2_in: Entrance Control E2~~
   ~~E2_out: Rear - controlled Exit Control E2~~
   ~~E3_in: Entrance Control E3~~
   ~~***Note:*** *To enter methods double click on the station and choose index card „controls“. You can choose the method selecting the button “...“.*~~
5. ~~Run the method. Consider thereby the behavior of the Front-controlled Exit Control.~~

~~25 Exercise: Entry- and Exit Control at Lines~~

~~Task:~~

~~1. Duplicate the Frame "_24_Entry_and_ExitControl_at_Stations" and rename it to "_25_Entry_and_ExitControl_at_Lines".~~
~~2. Replace the SingleProc devices by Conveyor Lines.~~
~~3. Attach again the methods as follows:~~
~~E1_out: Rear - controlled Exit Control E1.~~
~~E2_in: Rear - controlled Entrance Control E2~~
~~E2_out: Rear - controlled Exit Control E2~~
~~E3_in: Front - controlled Entrance Control E3~~
~~4. Observe the performance on the console.~~

26 Exercise: MUs transfer

Task:

1. Create a Frame with the name "_26_MUs_transfer" in the „Exercises" folder.
2. Make a production line with a Source, two SingleProc (E1, E2) and one Drain. Connect all these with Connectors except between the two SingleProc.
3. Add a method "Transfer" with the instruction "*@.transfer(E2);*".
4. Start the Simulation.

27 Exercise: MUs Insert

Task:

1. Duplicate the Frame "_26_MUs_transfer" and rename the new Frame as "_27_MUs_Insert".
3. Change the Method "Transfer" to "Insert" with the new source code "*@.insert(E2);*".
4. Start the Simulation.

28 Exercise: MUs Production

Task:

1. Create a Frame with the name "_28_MUs_Production" in the „Exercises" folder.
2. Build a production line with a SingleProc (E1) and one Drain connected by Connectors.
3. Produce an additional method "Init" which includes the source code "*.MUs.Entity.create(E1);*".
4. Start the Simulation.

3.2 Repetitive instructions—Loops

By using repetitive instructions it is possible to realize loops in Plant Simulation. Four different kinds of loops are provided.

for—Loop

The for-loop is used, in order to go through the values within the control variables. The control variable must be of integer type, greater than "0". At the end of the loop the control variable increases and/or decreases automatically by 1 unit. The loop is finished when it reaches the last value. At this point Plant Simulation stops the execution of the loop.

repeat—Loop

The instruction sequence "repeat… until" would wait until the condition is fulfilled, i.e. it results in a "TRUE" value. The loop will go through at least once, before the condition is evaluated for the first time. If the condition is not fulfilled at all, then it will produce an endless loop. If there is an endless loop you can stop the simulation calling the debugger method (combination of keys Strg-Alt--SHIFT).

Example:

```
is
  x : length;
do
  repeat
    x := methodEnterLength;
  until x >= 0;
end;
```

while—Loop

The *while* loop replaces the from-loop. It will execute as long as the condition is fulfilled, i.e. the result of the logical condition is "True". After processing the instruction sequence it checks whether the condition is still fulfilled or not.

From-until—Loop

A from-until loop is defined as follows:

```
from <statementSequence1>
until <condition> loop
  <statementSequence2>
end;
```

After executing <statement sequence 1>, where you may execute initializations, Plant Simulation analyzes the stop condition. If the condition returns FALSE (= do not stop), the program executes <statement sequence 2> and analyzes the stop condition again. If it is TRUE (= stop), Plant Simulation finishes the loop. The keywords loop and end denote the beginning and the end of the loop. The loops are executed until the condition returns TRUE.

3.2.1 *Exercises*

29 Exercise: For Loop

Task:

1. Create a Frame with the name "_29_for_Loop" in the "Exercises" folder.
2. Insert a Method with the name "for_Loop".
3. Insert the source code as shown below:

```
is
        i : integer;
do
        for i := 1 to 10
        loop
                print i;
        next;
        for i := 10 downto 1
        loop
                print i;
        next;
end;
```

4. Run the model.

30 Exercise: repeat_Loop

Task:

1. Create a Frame with the name "_30_repeat_Loop" in the "Exercises" folder.
2. Create a method with the name "repeat_Loop".
3. Write down the following code in the edit window of the Method:

```
is
        i : integer;
do
        i := 0;
        repeat
                print i;
                i := i +1;
        until i > 10;
end;
```

4. Run the model.

31 Exercise: while_Loop

Task:

1. Create a Frame with the name "_31_while_Loop" in the "Exercises" folder.
2. Place a method named as "while_Loop".
3. Fill up the source code in the edit box of the method:

```
is
        i : integer;
do
        i := 1;
        while i < 10 loop
        print i;
        i := i +1;
        end;
end;
```

4. Run the model.

3.3 Paused Suspension

3.3.1 *Wait-until instructions*

The Wait-until instruction "waituntil <condition> prio <integer expression>;" permits conditioned waiting time while processing a Method. While the condition is not fulfilled, the processing of the method will be interrupted. The method is suspended and the interpreter stores the whole calling chain of inclusive parameters and local variables. Then it supervises the condition, while the simulation runs normally. As soon as the condition is fulfilled, it interrupts the interpreter, to process the current method. There must be a condition that returns a Boolean expression. This condition must be once satisfied.

The instruction consists of:

- the keyword waituntil or stopuntil,
- a condition (a *boolean* expression),
- the key word prio, and
- an *integer* expression used to analyze the priority.

Note:
If several suspended methods wait for the fulfillment of the same condition, they will be activated at the same time. Processing takes place upon a priority basis.

3.3.2 *Exercise*

32 Exercise : waituntil_Instruction

Task:

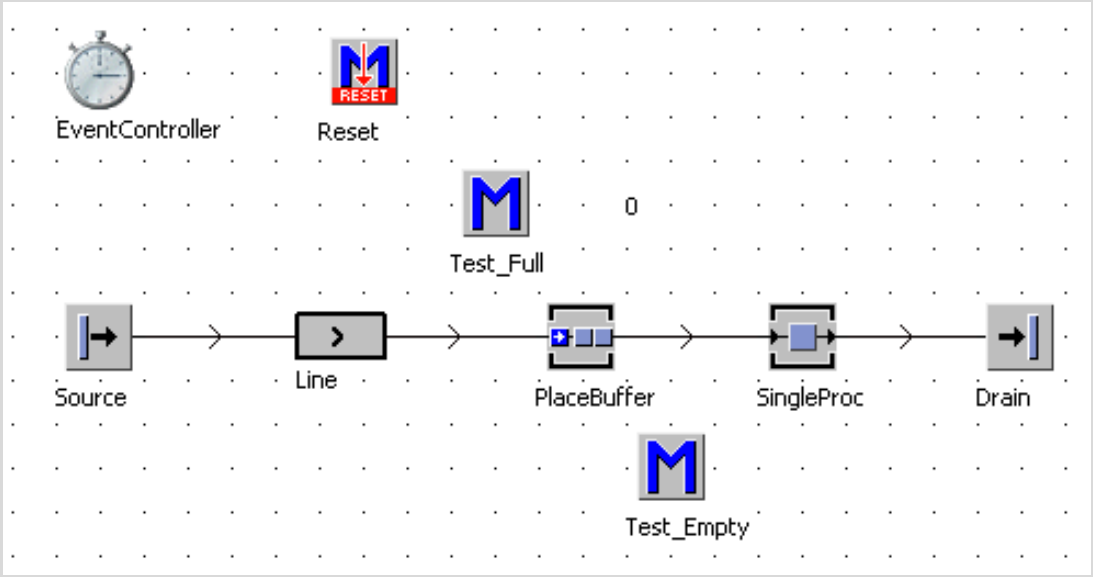

Figure 3-7: Exercise waituntil

1. Create a Frame with the name "_32_waituntil_Instruction" in the "Exercises" folder.
2. Build the production line and all other components as shown in the figure.
3. In the reset-method insert the following source code:
   „Line.pause := false;"
4. Insert a "Display". In the register "Path:" Write "PlaceBuffer.numMu" and then select "Active" and also "Watch" in the "Mode:" register.
5. In the method "Test_Full" insert the following source code:

```
waituntil PlaceBuffer.numMu > 8 prio 1;
    Line.pause := true;
```

And in "Test_Empty" write the following source code—

```
waituntil PlaceBuffer.numMu < 2 prio 1;
    Line.pause := false;
```

6. Set the PlaceBuffer capacity to 10.
7. Call the method "Test_Full" from Entrance and "Test_Empty" from the Exit control of the PlaceBuffer.
8. Run the model and observe the material flow.

3.4 Objects loading and unloading

Container

The Container can transport moving elements of any type. The container is comparable with a pallet or box. The overall capacity does not depend on the adjusted places on the matrix; it

is determined after checking the length of the uploaded commodity. Mobile Units of the type "entity" can be loaded on mobile units of the type "containers" or "transporters".

3.4.1 *Load and unloaded, access to contents of the transporters*

With the instruction ".cont" in a method it is possible to address the content of objects (e.g. MaterialFlow components and Containers), e.g. "<Path>.cont".
Example:

@.move (E2);
E1.cont.move (E2);

Contents are always the first MUs ready to be transported to another position or object.

To unload the content of a container, it is necessary to refer to the content as shown below:
Example:

@.cont.move (E2);
E1.cont.cont.move (E2);

In order to relocate the content of an object (e.g. loading Transporter), the method "cont" is also used:
Example:

@.transfer (E2.cont);
E1.cont.transfer (E2.cont);

@ is used, if the method is calling the current MUs.

3.4.2 *Material flow conditions*

Before accessing, transferring, or performing any activity on an object, it could be important to check the condition of the object itself. The checking process can be performed by querying the content of the MUs with any of the following instructions.

Note: The material flow can take place in different conditions; these conditions can be queried through attributes. Asking for a specific attribute can return a True or False value depending on the network conditions.

<mu_path>.empty; Returns a TRUE if the station is empty, otherwise returns FALSE.

<mu_path>.full; Returns TRUE if all places of the station (e.g. ParallelProc or Buffer) are occupied, otherwise it returns FALSE.

<path>.occupied; Returns TRUE, if it finds at least one MUs on the station, otherwise returns FALSE.

<path>[<integer1>,<integer2>].occupied;

Returns TRUE, if a MUs is found at the place mentioned (in matrix-oriented objects)

<mu_path>.finished; Returns TRUE if the object is neither failed nor paused and the MU either attempted to move or already have been moved (*boolean*). This represents a combination of two conditions: occupied and cont. This

combination is needed very frequently in practical application. (*Note: "finished" can't be used with "waituntil"*)

<path>.ready; Returns the value TRUE, if the operating time for the MUs is over and the MUs is liked to leave the component.

3.4.3 *Access to the contents of a place-oriented component*

The contents of place-oriented components and lines can be accessed through different functions for the individual MUs:

<path>.cont Give access to the next workable MUs.

<path>.mu(<integer>); Supplies the access to a specific workable MUs.

<path>[x,y].cont Supplies the MUs at the place [x,y] on the matrix.

Example:

<Objekt>.mu(1).move

Relocates the first transferable MUs in following station.

@.transfer(<Object>[x,y])

Relocates the current MUs over a place [x,y] on the matrix.

Note:

Before accessing contents of the component it must be guaranteed that the object is occupied!

3.4.4 *Exercise*

33 Exercise: Content_of_Object_MU_ and_Unload

Task:

1. Create a Frame with the name "_33_Content_of_Object_MU_and_Unload" in the "Exercises" folder.
2. Insert the following elements in the production line:
   One Source with the name "Source_Pallet" (cycle time: 1 minute), one Buffer (cycle time: 1 minute), one SingleProc (cycle time: 1 minute) and one Drain with the name "Drain_Pallet". Link them with a connector. The Source will create MUs "Containers". Insert an "EventController".

Note: The source creates containers when you double click on the source and choose „…" in the index card "attributes". A new window will appear. Click on MUs and then on „containers".

3. Add another Source (cycle time: 1 minute) and a Drain with the names "Source_Parts" and "Drain_Parts". The Source is connected with the Buffer, and the Drain is connected with the SingleProc by the methods "Load" and "Unload" (without any connector).
4. Add two Methods "Load" and "Unload".
5. Fill the Method "Load" with the following source code:

```
is
do
        waituntil Buffer.occupied prio 1; @.transfer(Buffer.cont);
end;
```

and the Method "Unload" with the following source code:

```
is
do
        @.cont.move(Drain_Parts);
end;
```

6. Call the Method "Load" from the "Exit" control of the "Source_Parts" and the Method "Unload" from the "Exit" control of the "SingleProc".
   Important: The standard output behavior of the "Exit" control in the SingleProc must be changed from "Front" to "Rare".

7. Start the simulation and observe the flow of the MUs.

4 Shift, Trigger, Generator, Track, Transporter, Conveyors, Tables and Attributes for MUs

4.1 Shift Calendar, Trigger and the Generator

In order to start or stop an activity within a simulation at a certain time, it is possible to use the Shift Calendar, the Trigger or the Generator.

4.1.1 *Shift calendar* 

The Shift Calendar can pause one or more components according to a Shift Model. The attribute "pause" is set to true for the addressed Material Flow components. To activate the resource, drag down the desired component from the Menu Bar on the icon of the Shift Calendar or select "Control" in the registration map of the specific material flow object and set a reference to the desired Shift calendar.

Shift Times

Several Shifts per day can be noted in one week, inclusive weekend. One can define these settings by opening the "Shift Times" window.

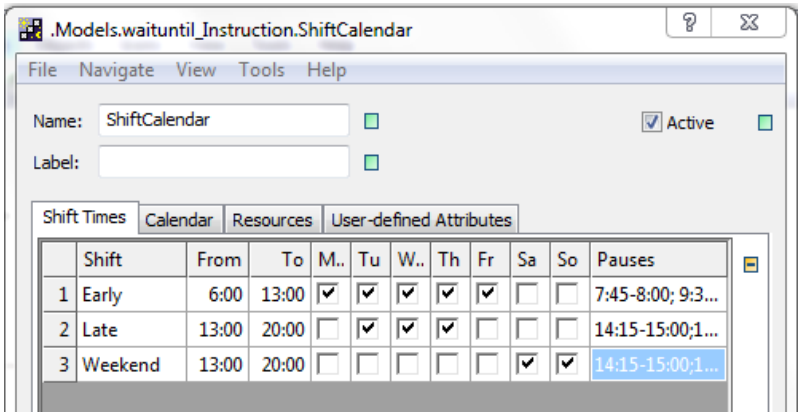

Figure 4-1: Shift Calendar

Holidays

By selecting “Calendar” one can define holidays and their implications on the schedule.

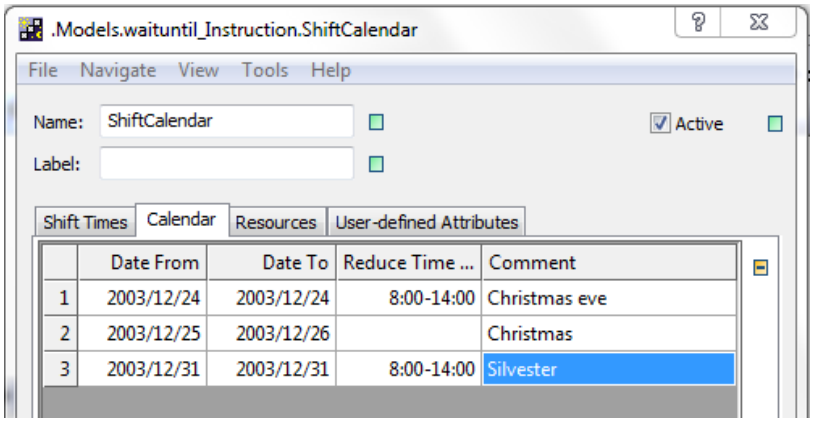

Figure 4-2: Shift Calendar: Holidays

4.1.2 *Trigger*

The Trigger is used to control temporarily dependent processes. It can be used to control attributes or to call methods. When calling a method, the trigger sends two initial parameters to the method: The last value sent by the trigger and the current value at the actual time of the activation of the trigger. The specific data type and the associated value are defined in the window “Values”.

Calling a method by a trigger

Double click the Trigger and select the “Actions” window. First select the Combo box and click “Apply”. Now you can select “Methods” and define the method or methods that should be called by the Trigger in the method table. At the times, defined in the “Values” table, the method is called and the command is processed. The method must receive both values of the initial parameters, the old and the new value.

4.1.3 *Generator*

The generator activates the defined method, at regular or statistic times according to a fixed starting point in a defined interval or during a defined duration.

Call of a method by a Generator

The methods can be embodied in the register map “Controls”. They are activated either by interval or by duration.

Note:
Interval and duration occur together, always.

4.1.4 *Exercise*

34 Exercise: ShiftCalendar

Task:

1. Create a Frame with the name “_34_ShiftCalendar” in the “Exercises” folder.
2. Create the following Frame:

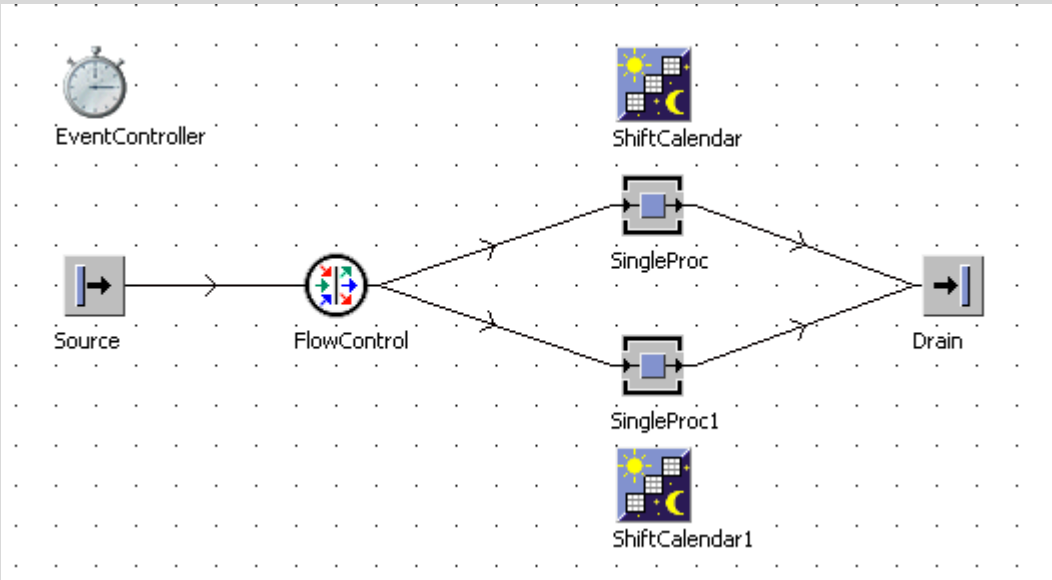

Figure 4-3: Exercise Shift Calendar I

3. Enter the following data in the Shift Calendar:

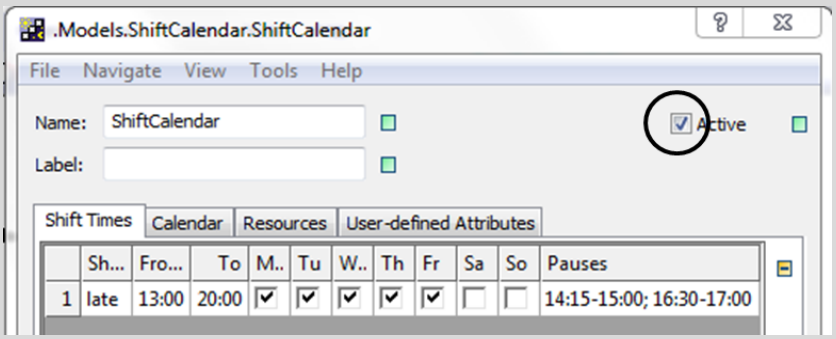

Figure 4-4: Exercise Shift Calendar II

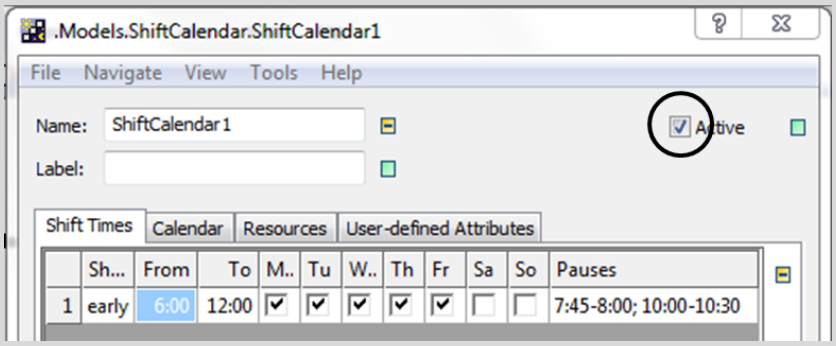

Figure 4-5: Exercise Shift Calendar III

Note:
To be able to enter data in the Shift Calendar you have to deactivate the small green box.

4. To link the ShiftCalendar to the SingleProc and to link ShiftCalendar1 to SingleProc1, one must choose in the register map "Controls" and the option "Shift Calendar". Alternatively the Shift Calendar can be connected by drag & drop.
5. Start the simulation.

35 Exercise: Trigger_ValueTable

Simulate a delivery cycle for the entrance of a store. The delivery time should be daily between 6.00 and 10.00 and between 16.00 and 20.00 o'clock.

Task:

1. Create a Frame with the name "_35_Trigger_ValueTable" in the "Exercises" folder.
2. Create a production line with one Source, two Buffers ("Transfer", "Store_entry"; each 1 minute cycle time) and one Drain.
3. Add a Trigger, a Time Sequence and an Init and a Reset-method.
4. Enter the following command into the init method: "*Trigger.insertTriggeredAttr(Store_entry, "pause");*".
   Enter the following command into the reset-method: "timeSequence.delete;".
   Note: You can create an Init-Method, selecting a method, choosing "rename" and enter "init" as a new name.
5. Enter the path: "*Transfer.statNumOut*" into the register map "Record" of the Time Sequence and select "Active".
6. Select the register map "Period" of the Trigger: Switch to "Repeat periodically" and set a "Period length" of one day.
7. On the map "Values" of the Trigger, open the "Value table". Deselect "Format → Inherit Format" under the menu-option.
8. In the column "Value" the data type needs to be changed from String, to Boolean, in order to insert Boolean type data. Select the column "Value", open with right-click the context menu and open the option "Format". Change now the data type from string to Boolean.
9. Enter the values shown below:

Contents | Start Values

0.0000

	time 1	boolean 2
string	Point in Time	Value
1	0.0000	true
2	6:00:00.0000	false
3	10:00:00.0000	true
4	16:00:00.0000	false
5	20:00:00.0000	true

Figure 4-6: Exercise Trigger Value-Table

10. Start the simulation and observe your results in the TimeSequence.

36 Exercise: Generator

Task:

1. Create a Frame with the name "_36_Generator" in the "Exercises" folder.
2. Create a production line with a Source, two Buffers (cycle time: each 1 minute) with the names "Delivery" and "Store_Entry" which have a capacity of 100 and one Drain. Connect them with the connector. Also add a Reset method, one Init method, one Event Controller and a Time Sequence.
3. The Reset method has the following source code:

```
is
do
        TimeSequence.delete;
end;
```

And the Init Method:

```
is
do
        Store_Entry.pause := true;
end;
```

4. Now add two methods and two Generators. Name the two Generators "Generator_Delivery" and "Generator_Stop" and the methods "Delivery_Meth" and "Stop_Meth".
6. Now set the starting time of the generator "Generator_Delivery" on 6:00 o'clock and the Interval of one day (1:00:00:00). Now select control/interval and connect the method "Delivery_Meth". In the method "Delivery_Meth" write the following code:

```
is
do
        store_Entry.pause := false;
end;
```

7. Set the starting time of the generator “Generator_Stop” to 15:00 o'clock and the interval to one day. Select control/interval and connect the method “Stop_Meth”. In the method “Stop_Meth” write the following code:

```
is
do
        store_Entry.pause := true;
end;
```

8. In the TimeSequence “Record” place the Path: “Delivery.statNumOut” and activate the TimeSequence.
9. Start the Simulation.

4.2 Track and Transporter

Modern manufacturing plants are increasingly using automatic transport systems. These systems move independently throughout the factory and transport parts.

4.2.1 *Components of Track* 

The *Track* allows to model transport lines. It also supports automatic routing. The Transporter is the only movable object, to use the *Track*. You might, for example, utilize both to model an AGV (Automated Guided Vehicle) system. You can also change the dimension of the *Track.*

A track is a length-oriented object used to control the flow of material components with arbitrary capacity. Its length determines the capacity of the way. In the TimeSequence registry we can register the way to perform the actions.
Note: The options for the entry of methods correspond to the Lines.

4.2.2 *Conveying devices for the Transporter* 

The *Transporter* is a moving object with a propulsion system of its own. It can be defined its loading capacity. The *Transporter* can hold MUs and move them freely. It represents forklifts, AGVs, etc. In the basic adjustment the “Transporter” possess a place-oriented matrix loading area. Through deselecting one can activate a Length oriented loading area. In the register “Battery” you can register battery values for “Basic consumption”, “Capacity” and “Charge current”. In the registration map “Controls” different methods for the control are deposited over the “Transporter” and the distance.

Driving control:	On which “Track” the “Transporter” at the exit is transferred
Destination:	If a destination is registered with the “TimeSequence” of the successors, then the appropriate successor value is increased.
Destination control:	It calls, as soon as the “Transporter” reaches its destination and passes through rear-end.
Collision control:	The predecessor driving “Transporter” is called first, if the distance between both results to zero
Speed control:	Defines the terminal velocity after acceleration when stopping and/or after delay.

Methods for the "Transporter"

The Transporter can be produced - like other material (i.e. "Entity" or "Container") - by a source. Like other MUs "Transporter" can be transferred by Connector or information can be transferred on following components. Beyond that the Transporter has its own driving power and can independently move on a Track. The Track, on which the Transporter moves, is connected with Connectors to the following components and itself. At the very beginning or in some point of simulation, to create a Transporter on a certain object and / or at a certain position we need to write the following method syntax:

Example: .MUs.Transporter.create(Track)

.MUs.Transporter.create(Track, 3.1) [position in meters]

4.2.3 *Exercise*

37 Exercise: Track_and_Transporter01

Task:

1. Create a network with the name "_37_Track_and_Transporter01" in the "Exercises" folder.
2. Create a production line with a Source, one Track and one Drain. Connect them with the connector. Also add an EventController.
3. The Source will create the Transporter.
4. Run the simulation.

38 Exercise: Track_and_Transporter02

Task:

1. Create a network with the name "_38_Track_and_Transporter02" in the "Exercises" folder.
2. Create a production line with two Tracks and one Drain (sequential). Connect them with the connector. Also, add an EventController.
3. The Init has the following source code:

```
is
do
        .MUs.Transporter. create(Track);
end;
```

4. Run the Simulation step by step.

39 Exercise: Track_and_Transporter03

Task:

1. Create a network with the name "_39_Track_and_Transporter03" in the "Exercises" folder.
2. Create the Network as shown below:

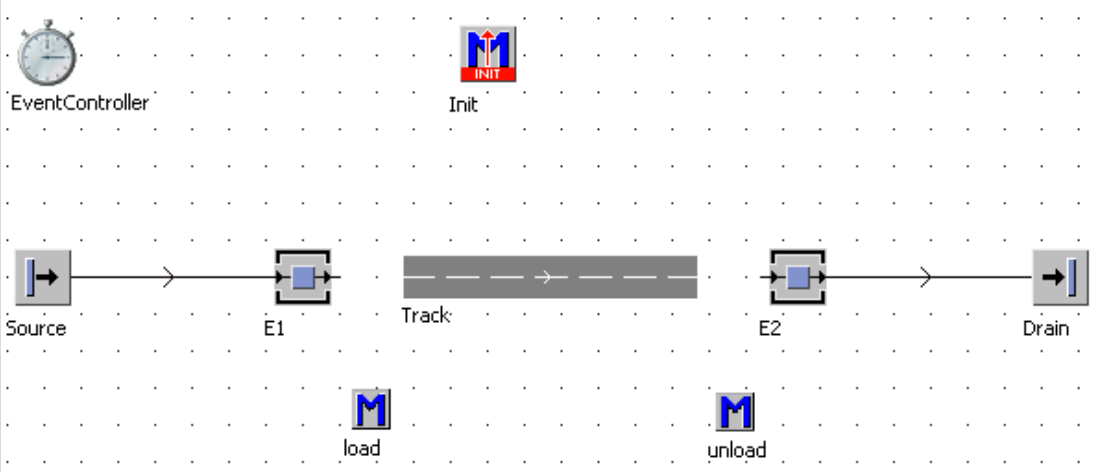

Figure 4-7: Exercise Track and Transporter 03

Note:

In order to enlarge a component, hold on "Strg" and "Schift" Key together and clicks on the one end of the component and pull it towards the desired direction.

3. Create a Transporter on the Track through the Init method.
4. Fill up the method "Load" with the following source code:

```
is
do
        waituntil E1.occupied prio 1;
        waituntil E1.cont.finished prio 1;
        E1.cont.transfer(@);
        @.Backwards:= false;
end;
```

And in the "Unload" Method insert the following source code:

```
is
do
        if @.occupied
        then
                waituntil E2.empty prio 1;
                @.cont.move(E2);
        end;
        @. Backwards:= true;
end;
```

5. Connect the method "load" to the backward exit of the track and the method "unload" its front exit.
6. Start the Simulation.

40 Exercise: Track_and_Transporter04

Task:

1. Create a network with the name "_40_Track_and_Transporter04" in the "Exercises" folder.
2. Create the Network as shown below:

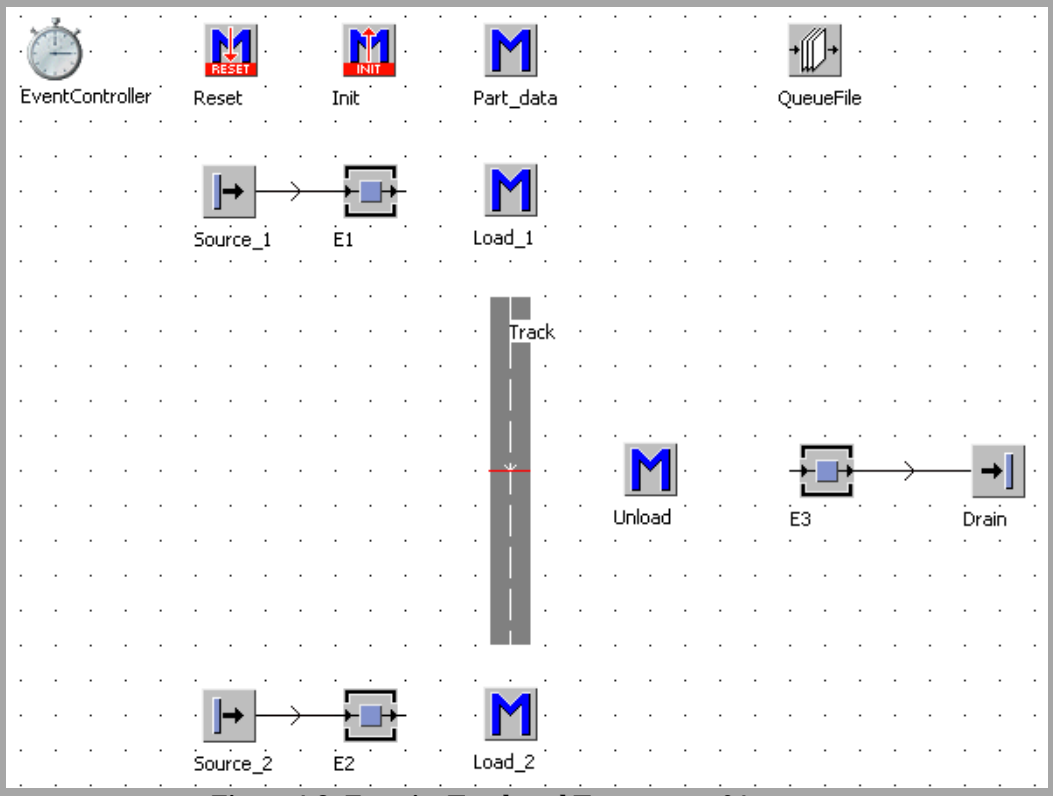

Figure 4-8: Exercise Track and Transporter 04

Note:
The Track is 10 meter long.

3. Fill in the methods with the following source codes:

Reset-Method:

```
is
do
        QueueFile.delete;
end;
```

Init-Method:

```
is
do
        .MUs.Transporter.create(Track);
end;
```

Part_Data:

```
is
do
        QueueFile.push(?.name);
end;
```

Load_1:

```
is
do
        @.speed := 0;
        waituntil E1.occupied prio 1;
        waituntil E1.cont.finished prio 1;
        E1.cont.transfer(@);
        @. Backwards := false;
        @.speed := 1;
end;
```

Load_2:

```
is
do
        @.speed := 0;
        waituntil E2.occupied prio 1;
        waituntil E2.cont.finished prio 1;
        E2.cont.transfer(@);
        @. Backwards := true;
        @.speed := 1;
end;
```

Unload:

```
is
do
        @.speed := 0;
        if @.occupied then
                waituntil E3.empty prio 1;
                @.cont.transfer(E3);
        end;
        waituntil QueueFile.dim > 0 prio 1;
        if QueueFile.pop = "E1" then
                @. Backwards := true;
        else
                @. Backwards := false;
        end;
        @.speed := 1;
end;
```

4. Connecting the Methods:
   Part_Data will be called from the Entrance Control of both of E1 and E2.
   Load_1 should be called through the backwards Exit control of the Track with Rear control.
   Load_2 connect it with the Exit Control of the Track (Front).
   Unload has to be called from the sensor, put in the middle way of the Track i.e. 5 meters away.
   Note: To install the sensor: Double click on Track. Choose register index "controls". Click on button "sensors". A new window will appear. Click on "insert" and then on "…"_button. Choose "unload" and adjust in row "position" 5 meter.

5. Start the Simulation.

4.3 Conveyor (belt / chain / rolls)

Conveyors component represent conveyor systems or lines. Examples can be found at logistic providers like DHL or UPS. Conveyors in the logistic centers deliver the different parcels from station to station. Another example is an assembly line in a factory. In this case the parts have the same distance in between them.

4.3.1 *Characteristics of the Conveyor* 

The conveyor is a length-oriented element with constant speed. The MUs cannot pass over others MUs in a conveyor component. The standard relocating behavior of a conveyor distributes the MUs to the different successors in turn. Changing the control setting of the conveyor component one can change the conveyor behavior. Roll-conveyor or simple belt conveyor is a characteristic that can be controlled in the property window of the component. Here the MUs always have the same distance in between them.

Sensor

Sensors can be installed anywhere at the conveyor and are useful to have "access" to the MUs. The sensor can be combined with a Method. If the MU passes the sensor the Method is activated.

4.3.2 *Exercise*

41 Exercise Conveyor01

Task:

1. Create a Frame with the name "_41_Conveyor01" in your folder "Exercises".
2. Create a production line with a source, a single-station, a conveyor, another single-station and a drain. All these elements are connected with connectors. Don't forget to add the Event-Controller.
3. Rename the single-stations as "E1" and "E2".
4. The conveyor has got a length of 10m. Please use the attributes of the conveyor to change the length.
5. Run the simulation.

42 Exercise Conveyor02

Task:

1. Duplicate the Frame "_41_Conveyor01" and rename the new Frame as "_42_Conveyor02".
2. Remove the connector between "E1" and the conveyor and change the processing time of "E1" to 0:30 minutes.
3. Insert the Method "Loading" with the content:

```
is
do
        while conveyor.occupiedlength > 9.6 loop
                wait 240;
                print conveyor.occupiedlength;
        end;
        @.transfer(conveyor);
end;
```

4. Connect the method "Loading" with the Exit control of the single-station "E1".
5. Run the simulation.

43 Exercise Conveyor03

Task:

1. Duplicate the Frame "_42_Conveyor02" and rename the new Frame as "_43_Conveyor03".
2. Remove the connector between the conveyor and the single-station "E2".
3. Insert the Method "OffLoading" with the content:

```
is
do
        waituntil E2.empty prio 1;
        @.transfer (E2);
end;
```

4. Connect the method "Off-Loading" with the exit control of the conveyor.
5. Run the simulation.

44 Exercise: Line04

Task:

1. Duplicate the Frame "_43_Conveyor03" and rename the new Frame as "_44_Conveyor04".
2. Rename the single-station "E2" as "EndStation" and the method "Off-load" as "Offload_End".
3. Create a further single-station "Middle", a method "OffLoading_Middle" and a further drain.
4. The method "Off-loading middle" contains:

```
is
do
        if @.name = "Middle"
        then
                @.transfer (Middle);
        end;
end;
```

Please change the text of the method "Offload_End" into:

```
is
do
        waituntil EndStation.empty prio 1;
        @.transfer (EndStation);
end;
```

5. Install a sensor after a length of 5 meters on the conveyor. Open the property registry window "Controls" then open "Sensor". Click on the button "Insert". Now define 5 meters in the position option and insert the method "OffLoading_Middle". Press the button "OK" and close the window.
6. The source now should deliver to types of MUs in a cyclical way. One MU is named "Middle", the other is named "Finish"
7. Run the simulation and observe the function of the sensor.

45 Exercise: Conveyor

Task:

1. Create a Frame with the name "_45_Conveyor05" in your folder "Exercises".
2. Create the Frame as shown below.

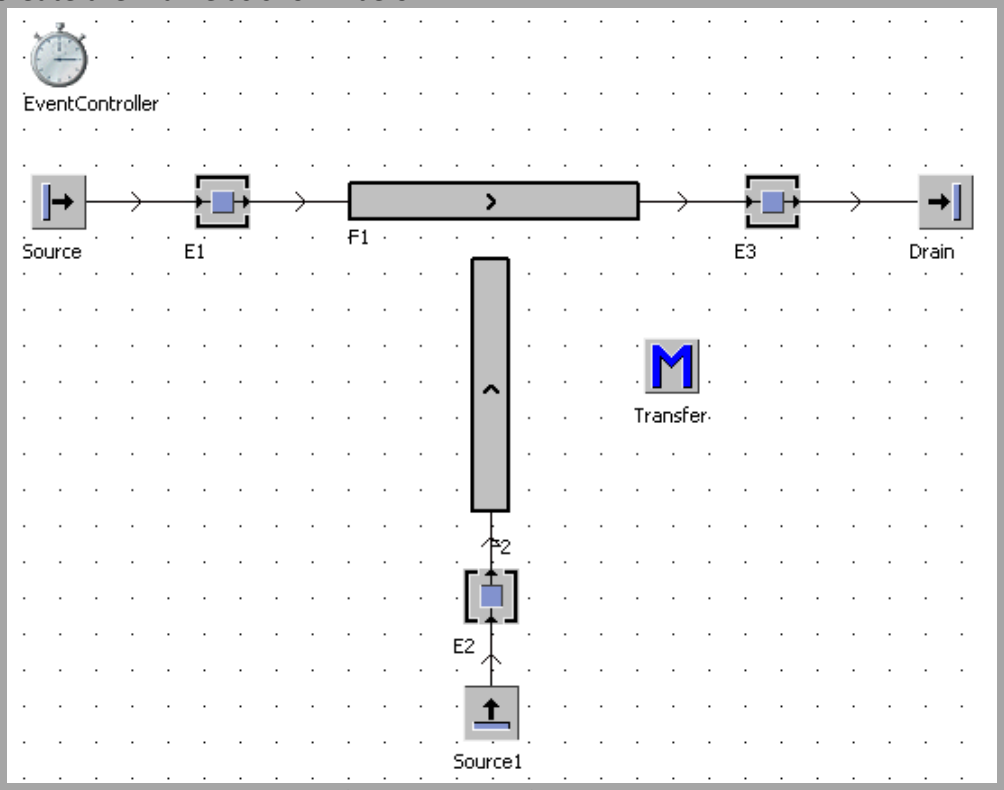

Figure 4-9: Exercise Conveyor

3. Insert the following text into the source code of the method of "Transfer":

```
is
do
        @.transfer (F1, 5);
end;
```

Note:

Make sure that conveyor F1 must be longer than 5 meter length.

4. Run the simulation.

4.4 Tables

4.4.1 *Characteristics of a Table*

The element "Table" is comparable to an Excel table. The sheet is divided in rows and columns. Each row / column can be assigned to different values to address a cell. The format of a column /row is indicated in the column /row heading.

Rows-/ Column Index

To activate the row- and column index, you have to deactivate the option "Inherit Format". With the activation of the index a column / row "zero" appears, in which you can add the user defined index of the row or column.

Note:

The index can be reached from "outside" the table. Thus you can address a desired place in the table.

Changing Data Type
Every cell needs to get a data-typ. Initially the type is set on string. To change the type it is needed to deactivate "Inherit format", then mark the sheet, row or column → right-click to "Format" and changing the Data type.

So it is possible to address tables from outside via two ways:

- Employing their index: By their position designated according their row- and column-index
- Employing relative: By their position designated by the caption in line / row 0

Methods for tables
For working with a table, the following methods are available:

Table.setCursor (<integer>,<integer>);
It is used to address a desired table field, e.g. (1,2).- column 1, row 2

Table.CursorX := <integer>
The cursor addresses the desired column.

Table.CursorY := <integer>
The cursor addresses the desired row.

Table.xDim;
Determines the last occupied column

Table.yDim;
Determines last occupied row

Table.sort (3,"ab")
Sorts within a column
Table.find (`[1,1]..`[.*], <value>)*
The table is searched for the given <value> in the specified area.

Table.meanValue(`[1]..`[*]);
Delivers the average value from a table

Table.delete (`[1,1]..`[.*]);*
Deletes the indicated range (or the entire table)

Note: The convention to address a table: "TableName[Column, Row]".

A big advantage of Plant Simulation is the characteristic of multidimensional tables. So it is possible to choose the data-type "table" within the table. Such multidimensional tables can be called from outside via programming. Therefore the syntax is the following: "TableName[Column, Row] [Column_subtable, Row_subtable]…". Equally it is possible to call the cells direct or indirect via self-chosen references.

46 Exercise: Table01

Task:

1. Create the Frame "_46_Table01" and name the new Frame "_46_Table01".
2. Create a table ("TableFile") and two methods ("Method1", "Method2").
3. Deactivate "Inherit Format" in the TableFile, add the column- and row-index and fill the table with the following content:

	string 0	string 1	string 2
string		column_one	column_two
1	line_one	column_one, line_one	column_two, line_one
2	line_two	column_one, line_two	column_two, line_two
3	line_three	column_one, line_three	column_two, line_three
4			

Figure 4-10: Exercise Table 01

4. The methods contains:

Method1:

```
is
do
        print tableFile[2, 2];
end;
```

Method2:

```
is
do
        print tableFile["column_two", "line_two"];
end;
```

5. Execute one method at a time and see the results.

47 Exercise: Table02

Task:

1. Duplicate the Frame "_46_Table01" and rename the new Frame as "_47_Table02".
2. Change the data-types of the columns as "table" and double-click then on "column_one, line_one". The Sub-table appears. Write in the sub-table the following:

	string 1	string 2	string 3	string 4	string 5
1	sub1, 1	sub2, 1			
2	sub1, 2	sub2, 2			
3					
4					
5					

Figure 4-11: Exercise Table 02

3. Delete "method2" and write in the method "method1" the following text and execute this method:

```
is
do
        print tableFile["column_one", "line_one"] [2, 1];
end;
```

48 Exercise: Table03

Task:

1. Create a new Frame "_48_Table03" in the exercise-folder.
2. Create a Table ("TableFile") and two methods: "Method01" and "Method02".
3. Change the first and second column of the table into the data-type "integer".
4. Write the following code in "Method01":

```
is
        i, j : integer;
do
        j := 1;
        for i := 1 to 1000 loop
                tableFile[1,i] := j;
                j := j + 1;
        next;
end;
```

5. Write the following code in "Method02":

```
is
        i, j : integer;
do
        j := 1;
        for i := 1 to tableFile.ydim loop
                tableFile[2,i] := z_uniform(1, 1, 100);
                j := j + 1;
        next;
end;
```

6. First execute "Method01", see the result and then execute "Method02" and interpret what happened.

49 Exercise: Table04

Task:

1. Create a new Frame "_49_Table04" in the exercise-folder.
2. Create the layout like the following figure:

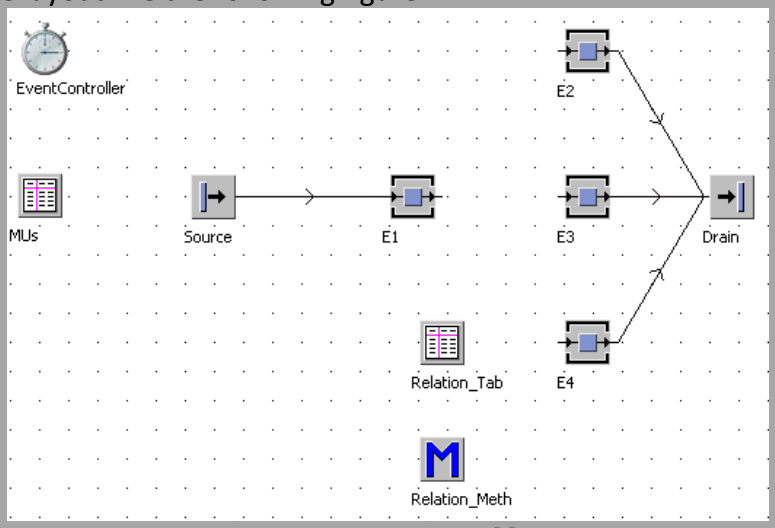

Figure 4-12: Exercise Table 04

3. The following “MUs” get produced:

	object 1	integer 2	string 3	table 4
string	MU	Number	Name	Attribute
1	.MUs.Entity	1	small	
2	.MUs.Entity	1	medium	
3	.MUs.Entity	1	big	

Figure 4-13: Exercise Table 04 MUs

This is the content of the „Relation_Tab”:

	string 0	object 1
string		Destination
1	small	E2
2	medium	E3
3	big	E4
4		

Figure 4-14: Exercise 04 Relations

And this is the content of the „Relation_Meth“:

```
is
do
        @.move(Relation_Tab["Destination", @.name]);
end;
```

4. Interprete the events.

4.5 Attributes for MUs

Each object in Plant Simulation has a number of built-in attributes (such as length, speed, name, …) and methods. An Attribute defines the behavior or state of an object. A method queries information of an object and returns a value. It calculates a value and/or starts one or several actions that control the behavior of the object. The window “Show Attributes and Methods” that can be called by a right-mouse click on the object.
Moreover it is possible to define Custom Attributes.
Such custom Attributes add properties and features to an object to meet the individual modeling needs. Plant Simulation is able to change the attributes or the values of the attributes while simulation run.

50 Exercise: Individual attributes

Task:

1. Duplicate the Frame “_23_Monitoring_inspect” and rename it to “_50_Attribute”
2. Open the “TableFile” and give every element in the column “Attributes” the name “color” in the way it is pictured subsequent:

	object 1	integer 2	string 3	table 4
string	MU	Number	name	Attribute
1	.MUs.Red	1	red	color
2	.MUs.Yellow	2	yellow	color
3	.MUs.blue	1	blue	color

Figure 4-15: Exercise Individual attributes I

3. Open the sub-table for the red MU and write the following into:

	string 1	integer 2	boolean 3	st 4
string	Name of Attribute			
1	red		true	
2	yellow		false	
3	blue		false	

Figure 4-16: Exercise Individual attribues II

Do this again with the remaining colors (please change the true and false in the particular way!

4. Replace the content of the method “Transfer”:

```
is
do
        if @.red then @.transfer(Buffer_red);
        elseif @.yellow then @.transfer(Buffer_yellow);
        else @.transfer(Buffer_blue);
        end;
end;
```

5. Open one product and switch to the tab User-defined Attributes. Here the status of the defined attribute can be seen.

5 Random events and seed values

5.1 Random events

5.1.1 *Random numbers*

In Plant Simulation, a random number generator creates random numbers, using the MRG63k3a random number generator. These random numbers are merged into a fraction located in the interval between 0 and 1.
The random number generator is initialized with a certain start value, the so called seed value. Seed values are stored in a table.
By selecting “Tools → Random Number Seed Values” the table which contains the seed values appears. If required, the values can be changed here manually.
In the new Version of Tecnomatix Plant Simulation (Version 11) these seed values only apply to distribution functions such as z_uniform, z_normal, etc, which will be explained in the following chapter.
When simulating random processes, Plant Simulation automatically uses a dedicated random number stream for each and every material flow object. So when you insert an object it is guaranteed that all random components of this object are different and stochastically independent from all other objects that use random numbers. However you can also set the random number stream of an object with the attribute RandomSeed.
Example:

```
SingleProc.myAttribte.RandomSeed := 10;
```

This will make sense if you want to compare two production lines and you insert two identical Source objects, which are going to create identical parts within identical intervals.

5.1.2 *Distribution functions*

Plant Simulation can create random numbers with the functions described below, which return random numbers according to the desired distribution.
The argument *s* (below-mentioned table) designates the random number stream (compare Seed Values in the previous chapter) and has the data type integer. All other arguments are the arguments of the corresponding distribution function. They all are either of data type real or integer.
The following table shows all in Plant Simulation possible distribution functions including the needed arguments:

Table 5-1: Distributions in Plant Simulation

Function	Results in the
z_beta(Stream:<integer>,Alpha1,Alpha2)	Beta distribution
z_binominal(Stream:<integer>,n,p)	Binomial distribution
z_Cauchy(Stream:<integer>, Mu:<real>, <theta:real>[, LowerBound:<real>, UpperBound<:real>]);	—
z_cemp(Stream:<integer>,Table)M	Continuous Empirical (cEmp) distribution
z_demp(Stream:<integer>,Table);	Discrete Empirical (dEmp) distribution
z_emp(Stream:<integer>,Table,column);	Primitive Empirical (Emp) distribution
z_erlang(Stream:<integer>, Mu:<real>, Sigma[,LowerBound:<real>,UpperBound:<real>]);	Erlang distribution
z_Frechet(Stream:<integer>, Mu:<real>, <theta:real>[, LowerBound:<real>, UpperBound<:real>]);	—
z_gamma(Stream:<integer>,Alpha,Beta[,LowerBound:<real>,UpperBound:<real>]);	Gamma distribution
z_geom(Stream:<integer>,p[,LowerBound:<real>,UpperBound:<real>]);	Geometric distribution
z_Gumbel(Stream:<integer>, Mu:<real>, <theta:real>[, LowerBound:<real>, UpperBound<:real>]);	—
z_hypgeom(Stream:<integer>,m,n,p);	Hypergeometric distribution
z_Laplace(Stream:<integer>, Mu:<real>, <theta:real>[, LowerBound:<real>, UpperBound<:real>]);	—
z_Logistic(Stream:<integer>, Mu:<real>, <theta:real>[, LowerBound:<real>, UpperBound<:real>]);	—
z_LogLogistic(Stream:<integer>, Mu:<real>, <theta:real>[, LowerBound:<real>, UpperBound<:real>]);	—
z_lognorm(Stream:<integer>, Mu:<real>, Sigma[, LowerBound:<real>, UpperBound<:real>]);	Lognormal distribution
z_negexp(Stream:<integer>,Beta[, LowerBound:<real>, UpperBound<:real>]);	Negative exponential distribution
z_normal(Stream:<integer>, Mu:<real>, Sigma[, LowerBound:<real>, UpperBound<:real>]);	Normal distribution
z_ParaLogistic(Stream:<integer>, Mu:<real>, <theta:real>[, LowerBound:<real>, UpperBound<:real>]);	—
z_Pareto(Stream:<integer>, Mu:<real>, <theta:real>[, LowerBound:<real>, UpperBound<:real>]);	—
z_poisson(Stream:<integer>,Lambda);	Poisson distribution
z_triangle(Stream:<integer>,c,a,b);	Triangular distribution
z_uniform(Stream:<integer>,Start,Stop);	Uniform distribution
z_weibull(Stream:<integer>,Alpha,Beta);	Weibull distribution

The most important distributions are explained in detail subsequent.

Negexp distribution

The negative exponential distribution is called negative because of the negative prefix of the exponent. The realizations are non-negative real numbers. The parameter Beta (β) designates the mean time in seconds between two events.

The function can be used to visualize times between independent events and to model in-between arrival times of customers in a service system, the duration of a repair job or the absence of employees from their job site.

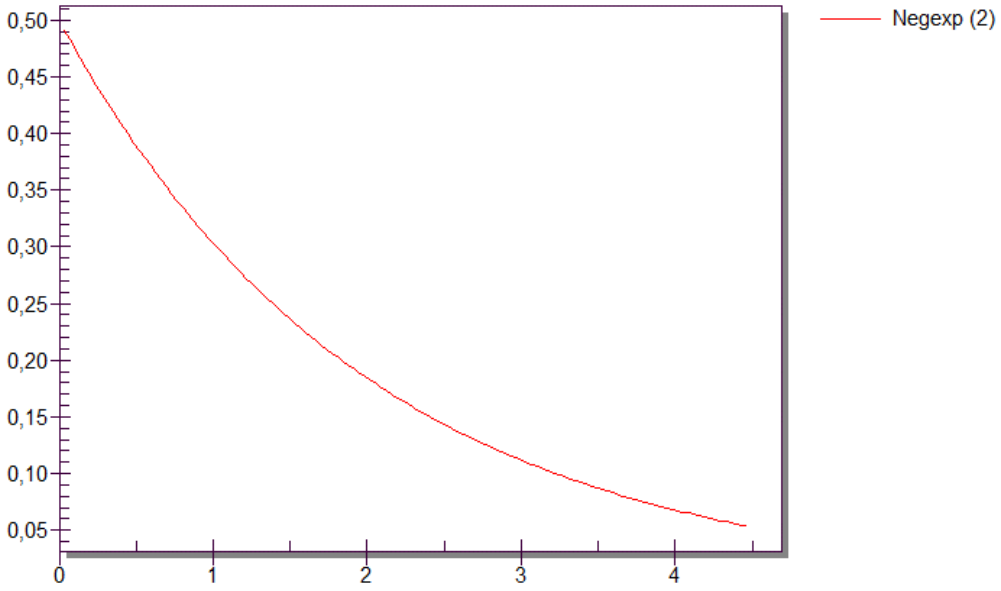

Figure 5-1: Negative exponential distribution

Normal distribution

The normal distribution is a continuous distribution. Normal means that for a time query Plant Simulation rolls a number that corresponds to a normal distribution with the expected value Mu (μ) and the standard deviation Sigma (σ). As a rule of thumb 2/3 of all possible values are located within the interval [μ-σ, μ+ σ].

This distribution can be used to model random numbers that can be realized as the sum of a large number of random numbers, for example, to describe measuring errors.

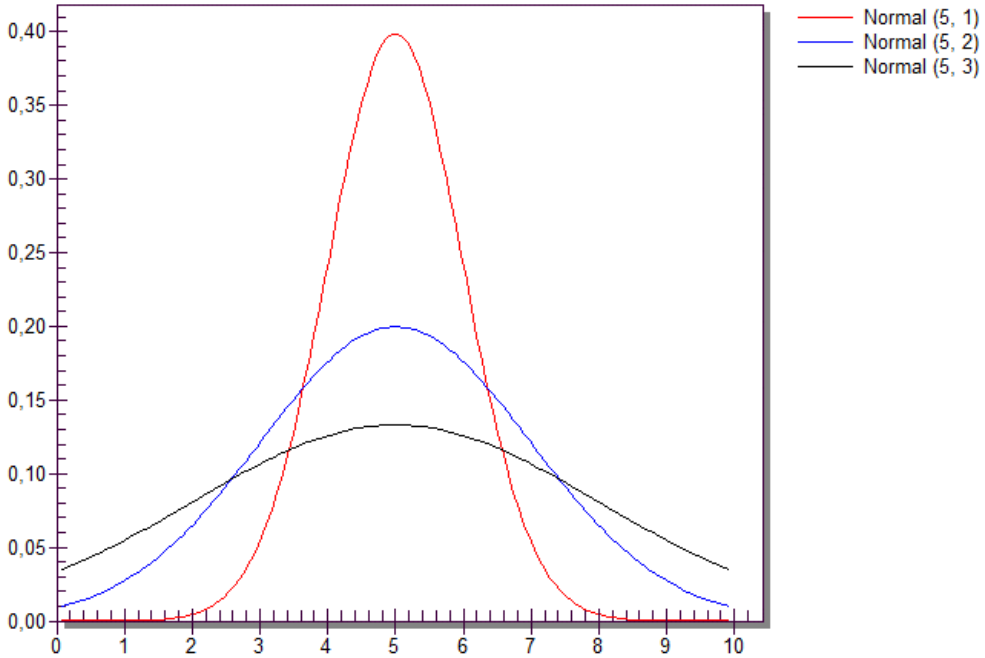

Figure 5-2: Normal distribution

Triangular distribution

The *triangular distribution* is defined by these parameters: The mode c, the minimum value a and the maximum value b of the interval within which the random numbers are created.

The distribution can be used if not much is known about the distribution of a value.

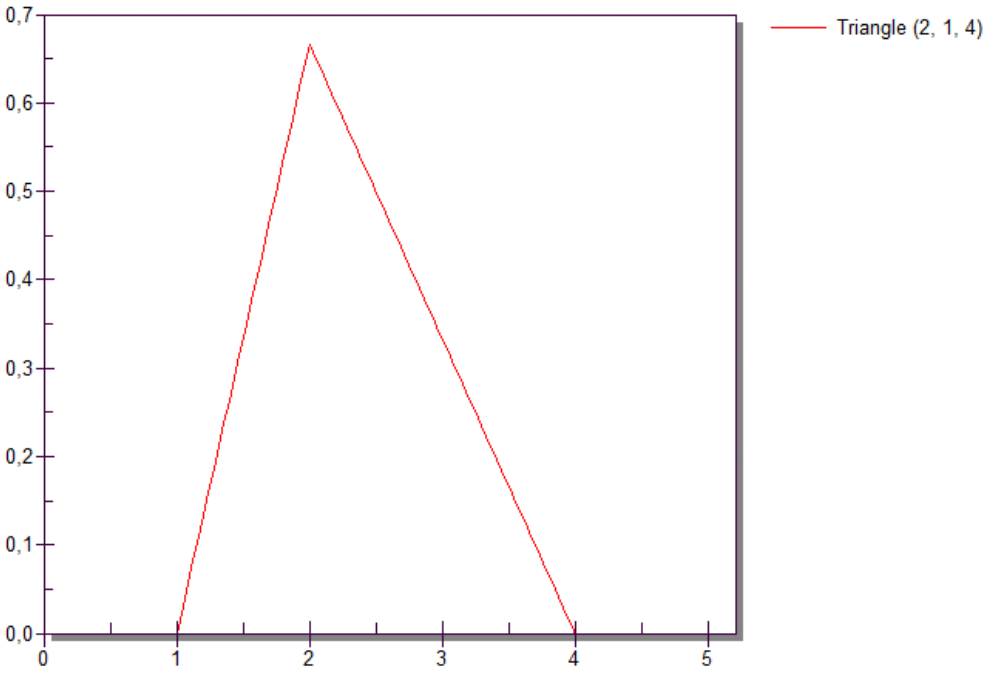

Figure 5-3: Triangular distribution

Uniform distribution

The uniform distribution is used to model a random number that is located between the interval bounds Start and Stop. This distribution is used when not much is known about the random number.

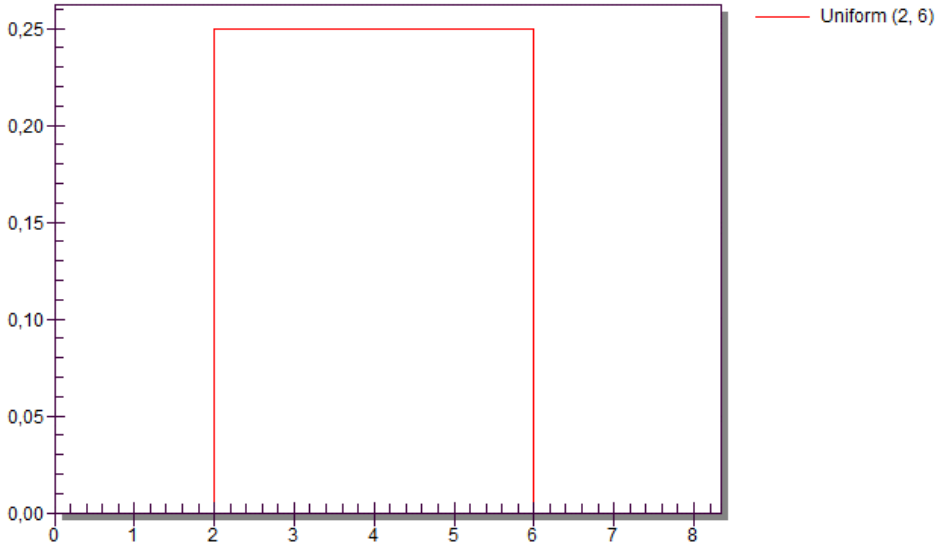

Figure 5-4: Uniform distribution

51 Exercise: random number stream

The following task shows that a dedicated random number stream will be automatically used for each and every material flow object. This stream can be set with the attribute *RandomSeed*.

Task:

1. Create a new Frame "_51_ random number stream" in the folder Exercises
2. Build the model as shown in the following figure:

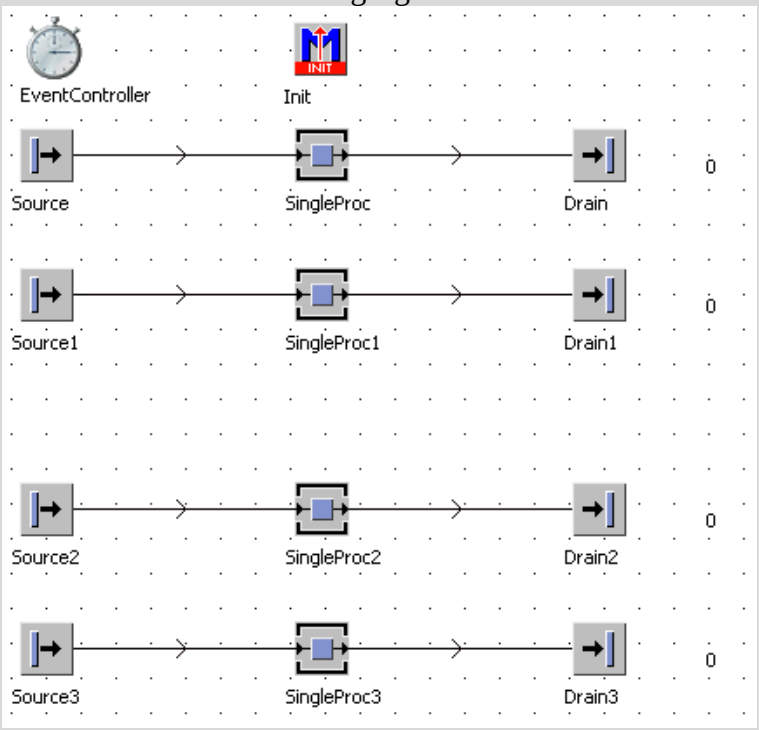

Figure 5-5: Exercise random number stream model

3. Set the attributes of the four sources like the figure bellow:

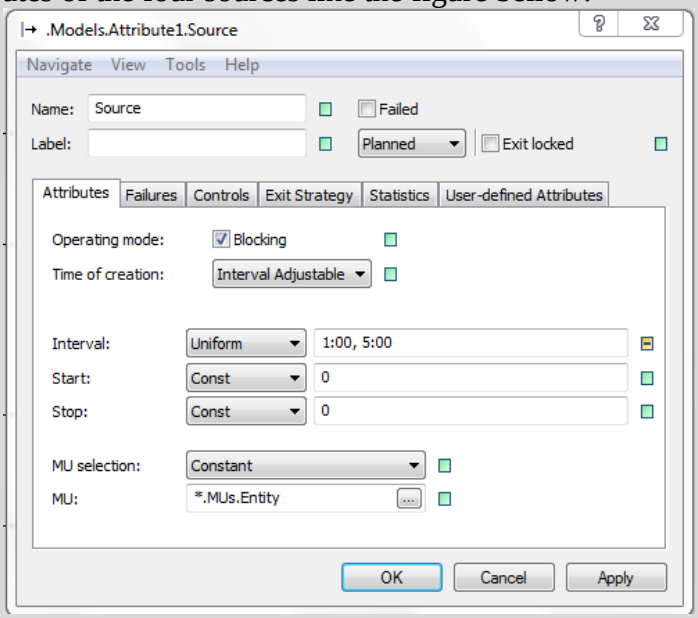

Figure 5-6: Exercise random number stream setting of sources

4. Fill up the source code in the edit box of the Init-method:

```
is
do
        Source2.RandomSeed := 20;
        Source3.RandomSeed := 20;
end;
```

5. Run the simulation for 1 hour and compare the number of MUs that go into the four drains.

Note: *You can run the simulation more times and then compare the result with each other.*

Experimentmanager 

The *ExperimentManager* supports in executing simulation studies. A simulation study consists of several experiments, which the *ExperimentManager* automatically executes. The manager will carry out simulation studies

- to investigate different variants of the model parameters and
- to arrive at statistically safe results.

A simulation study contains several experiments. Each experiment executes several simulation runs, each of the run leads to an observation of specified indicators.

52 Exercise: ExperimentManager01

Task:

1. Create a new Frame "_52_ExpereimentManager01" in the folder Exercises
2. Put in the frame a Source, a SingleProc and a Drain and connect these elements with each other.
3. The SingleProc has a uniform distribution for the processing time between 1 minute and 1 day.
4. Create a Display for the Drain that shows the number of MUs that go in.
5. Insert the ExperimentManager from the toolbox.
6. Open the dialog of the ExperimentManger and click on "Output Values". A table opens. Write the following in the field 1: "*Drain.statNumIn*".
7. Change the number of observations per experiment from the predefined 10 to 5000 (in the tab Define). Apply. Switch to tab Run and click on Reset. Now start the Experiment by clicking on the button Start.
8. After the experiment has finished the statistics opens automatically.
9. Check the results and find out the average output of MUs within one month.

53 Exercise: ExperimentManager02

Task:

1. Duplicate the existing Frame "_09_PaintShop" and rename it to "_53_ExperimentManager02"
2. Now change the following data:
   - Hanging: Processing Time: Triangle with "*1:30, 1:00, 2:00*"
   - Pretreatment: Processing Time: Uniform with "*1:00, 2:00*"
   - Painting: Processing Time: NegExp with "*1:20*"
3. The goal of this task is to determine the optimal number of hangings with respect to the uncertainness of Hanging, Pretreatment and Painting. So input the ExperimentManager into the frame. Define the Output Value of the drain ("Drain.statNumIn"). The number of observations per experiment should be 500 to ensure statistical safety.
4. Because we like to achieve that with every experiment the number of hangings gets counted up we have to define rules. Go to the tab Rules, activate the checkbox Use rules and then click on Create. Type in the name "MyRule" and click on Ok. Now click on Method action and type in the following text.

```
is
do
        .Models.Exercise._53_ExperimentManager02.Source_hanging.number :=
        .Models.Exercise._53_ExperimentManager02.Source_hanging.number + 1;
end;
```

5. Close the method-window and define 50 numbers of experiments. Then it is very important to set the number of hangings that should be created in the Source_hanging on *1*.
6. Switch to the Run-tab in the ExperimentManager, Reset the ExperimentManager and start the Simulation.
7. After running, try to read out of the statistics the optimal number of hangings.
8. Change the capacity of the Hangingbuffer to 100. Run the Experiment again.
9. Determine the new result.
10. Discuss the differences of the experiment with 10 and 100 Hangings of the Hangingbuffer.

5.2 Analysis

Plant Simulation offers many modules to analyze the flow of material. To this tools count

- Display
- Chart
- Report
- BottleneckAnalyzer
- SankeyDiagramm

In the following chapter some of this tool gets presented.

5.2.1 *Display* 

The *Display* shows a value, for example of an attribute, at all times throughout the entire simulation run. When the *Display* is not active, *Plant Simulation* shows its icon in the *Frame*, where is inserted. When it is active, the *Display* shows a value. Depending on the Mode it updates its display either periodically (Sample mode) or when the displayed value (Watch mode) changes.
The *Display* shows the value either as text (string), as a bar or as a slice of a pie. Bar and pie display are only supported for numerical values. They are easy to comprehend, but not very well suited if accuracy is important.
For bar or pie representation the *Display* shows the value in relation to a specified range. An empty bar/pie indicates values smaller than or equal to the lower bound of the range. A full bar/pie represents values beyond or equal to the upper bound of the range. Modify the overall size of the bar/pie representation by dragging its outline with the left mouse button while holding both the Shift and the Ctrl keys down.
The *Display* shows the actual minimal and maximal values as dashed lines, which is not very accurate. In Bar and Pie mode it shows the current values in the text boxes Minimum and Maximum the tab Data.
The Display can be connected per drag & drop.

5.2.2 *Chart* 

The *Chart* graphically displays the data sets that Plant Simulation recorded during a simulation run. Set data to be plotted by:

- Using a table containing the data, such as simulation results.
- Defining input channels that record values of attributes of interest of the objects.

A *Chart* object, which is inserted into a *Frame*, provides a *Statistics Wizard*. Select the menu command Statistics Wizard on the context menu of the *Frame* to open it. Select the class(es) of object(s) for which you would like to show statistics values in the *Chart* under **Class**. The *Chart* only adds material flow objects to the display for which you selected the check box Resource statistics on the Tab Statistics.

54 Exercise: Chart

Task:

1. Duplicate the Frame "_09_PaintShop" and rename it to "_54_Chart"
2. Create a Display that shows the current number of MUs that are received by the drain (Remark: The exact command therefore is "Drain.statNumIn" in the Display-menu; don't forget to activate the display)
3. Now we like to have for every SingleProc the statistic values in form of an graph. Therefore insert a "Chart" into the frame. By a right-click on the Chart the select "Statistic Wizard…". Under Class only activate SingleProc. Click on Ok.
4. Interpret the Graph.

5.2.3 *SankeyDiagramm* 

Sankey diagram are a specific type of flow diagram, in which the width of the arrows is shown proportionally to the flow quantity. The SankeyDiagram watches and displays instantiated MUs of MU classes.

55 Exercise: Sankey

Task:

1. Duplicate the Frame "_23_Monitoring_inspect" and rename it to "_55_Sankey"
2. Insert the module "SankeyDiagram" from the toolbar; open it and click on the "Open"-button. Here the MUs, that should be watched must be written in:
   1) "*.MUs.red*"
   2) "*.MUs.yellow*"
   3) "*.MUs.blue*"
3. Start the simulation and then click on "Display" in the Sankey-dialog.
4. Interpret the results.

6 Visualization

6.1 Modeling in 3D Viewer

Plant Simulation supports the modeling and animation of models in a virtual space. The 2D model is assigned a 3D model, which is controlled by the corresponding 2D model. The Plant Simulation 3D Viewer makes the modeling in a 3D Environment possible. It is an object-oriented modeling and visualization tool for displaying and animating an existing Plant Simulation 2D model in three-dimensional space or directly creating a 3D model from scratch. This 3D viewer is however not a program of its own, but fully integrated into Plant Simulation. Once you start the 3D Viewer, saving the Plant Simulation 2D model also saves the 3D part of the model into the same *.spp file.

6.1.1 *Advantages of 3D Modeling*

A 3D model basically serves for illustrating production sequences. Modeling in the 3D Viewer especially makes sense, when you want to present the 3D model to stakeholders of a simulation study. In addition, 3D models can impress visitors at trade fairs or at in-house exhibitions.

Based on a visual 3D simulation the discussion of the simulation results, such as the effects of storing or control strategies onto the material flow, can be focused better compared to schematic 2D models. Employees, who have no simulation expertise and who are not familiar with the abstract display of data, find it easier to follow the 3D visualization. A suitable 3D visualization supports the integration of employees from the operative field into the discussion and evaluation of the simulation results.

Be aware that a Plant Simulation 2D model does not create any additional analytical information, such as about blockages or accumulations caused by geometry collisions. You cannot use the 3D model for an exact analysis of geometry collisions. The same is true for defining robot movements or for animating complex insert operations or processing operations.

6.1.2 *Getting to know 3D Objects*

In the 3D Viewer, there are two important concepts to be distinguished: plain graphic object and viewer object. A plain graphic object is just that, namely a graphic, while a viewer object has a number of attributes that describe its state. The graphic of the viewer object is one of these attributes. A viewer object is, as rule, connected to its corresponding object in Plant Simulation 2D via its name. To show the graphics of the viewer object, right-click anywhere in the background in the 3D Library and select Display Graphics on the context menu.

Plant Simulation 2D and the 3D Viewer establish if an object in 2D corresponds to an object in 3D by checking its name and its path in the model. The online connection between objects exists as long as:

- They are located at the same position in the structure in the 2D Class Library and in the 3D Library.
- They have the same name.
- You selected the check box Create in 3D in Plant Simulation 2D. (Select Tools > 3D in the dialog of the objects, then you can select or clear the check box)

To activate the online connection, you should also be sure that the button Connect 3D Viewer  is selected. When you click  again to deactivate the online connection and you then rename an object in either part of the model, Plant Simulation 2D does not consider the

objects to be identical any more. You will typically deactivate the online connection, when you want to insert a graphic object into the 3D part of the model, which you do not want to appear in the 2D part of the model. Be aware that not all items of a 2D model are relevant for the 3D Viewer model, such as the objects in the folders InformationFlow (Methods, tables,) and UserInterface. The 3D Viewer part of the model, on the other hand, may also contain items that are irrelevant for the simulation in Plant Simulation 2D, such as room dividers, safety fences, etc.

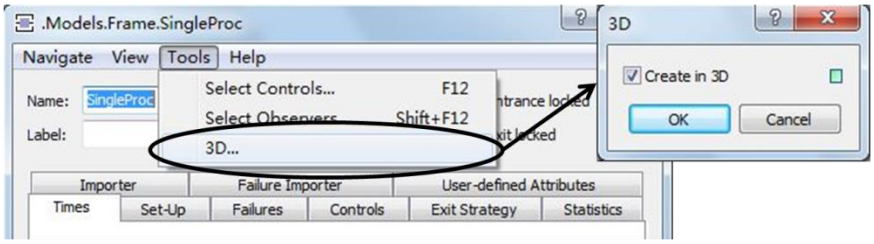

Figure 7-1 Check box Create in 3D

6.1.3 *Create a model in 3D*

To create a model in the 3D Viewer, you will basically proceed as follows: open a new Plant Simulation 2D model and select File > New. Then click 3D on the 3D toolbar 3D, or select View > Activate 3D Viewer to start the 3D Viewer, this shows the 3D Library and a new, empty Frame in the 3D scene window. You are recommended to show the grid before inserting an object into the scene window by clicking Show Grid. This way you can place the object at the exact location, where you want it to be. Then select the object you want to insert in the Toolbox or in its folder in the 3D Library. Drag the mouse pointer to the location of your choice and click the left mouse button once to insert it there. The connecting of two objects with the connector in 3D Viewer is as same as the operation in 2D. Note that by default the 3D Viewer does not show Connectors. To show or hide connections in the scene window, you can click on the 3D Standard toolbar (refer to the following figure).

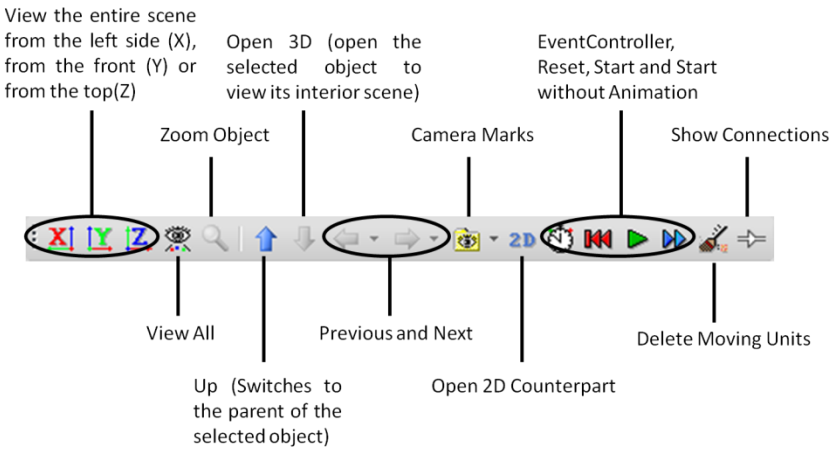

Figure 7-2 3D Standard toolbar

Besides basic objects, the 3D Viewer provides you a number of predefined 3D objects. Turn to Manage Class Library > Libraries > Standard Libraries and select all the items, whose names start with 3D, then click Apply.

The following points will show you, how the scene in 3D Viewer can be manipulated with mouse:

- To you rotate the scene, hold down the left and the right mouse buttons and drag the

mouse. The rotate mouse pointer looks like this 

- To pan the scene, hold down the right mouse button and drag the mouse. The pan mouse pointer looks like this 
- To move the camera forwards and backwards, when you select the setting Modes > Perspective Projection, turn the mouse wheel.
- To zoom the scene, when you select the setting Modes > Orthogonal Projection, turn the mouse wheel.

6.1.4 *Create a 3D model from a 2D model*

In most cases we are going to generate a 3D model based on an existing 2D model, so that our simulation project can be illustrated in the 3D Viewer. In order to complete the generating process, you can:

- First start Plant Simulation and open the simulation model for which you would like the 3D Viewer to automatically create a three-dimensional view.
- Then define which of the objects located inside of a Frame you want to visualize in the 3D Viewer. Open the dialog window of each object, select Tools >3D and decide if the check box should be selected in the window that appears (refer to the previous chapter). Note that the check boxes of built-in objects from the Class Library have been selected as default.
- Start the 3D Viewer: Either click  on the 3D toolbar  , or select the menu command View > Activate 3D Viewer in the Plant Simulation main window. (The 3D Viewer automatically creates the three-dimensional part of the model. You will notice that the structure of the 3D view matches the structure of the simulation model in Plant Simulation 2D. As the 3D Viewer just visualizes the simulation, it does not show objects, which are irrelevant for the animation.)
- Continue to model in 3D with the online connection open or closed.
- Subsequently define three-dimensional graphics for the 3D counterparts of the corresponding Plant Simulation objects. The 3D Viewer will automatically insert default cuboids as placeholders for the objects. In some cases you will want to replace these default graphics with pictures that resemble the actual machines used in your plant. To assign a different graphic to an object, you can select the object, and click  on the 3D Modeling toolbar  , then select the graphic you want to use. (Plant Simulation provides a series of predefined graphics, which are saved in the program package in s3d format. You can find them in the folder where you installed Plant Simulation. The default path of version 11 is “C:\Program Files\Tecnomatix\Plant Simulation 11\3D\s3d-graphics”.
- Edit the animation paths to ensure a good-looking animation. (compare the following chapter)

6.1.5 *Animation Paths*

For the purpose of characterizing a better demonstration for simulation models, the 3D Viewer uses a set of points in space as paths for animating objects. Animation paths describe the route an animated mobile object takes when it moves on objects or through the scene. The cameras, through which you view the scene in a fly-through, also use animation paths. You can create and edit a path in the dialog 3D Properties on the Tab MU Animation, the Tab Self Animation, or on the Tab Camera Animation.

To work with animation paths of parts on an object, Select the object, and click on the 3D Modeling toolbar or select the object and select Edit > 3D Properties. This opens the dialog 3D Properties. Click on the Tab MU Animation, Self Animation or Camera Animation and create a new animation path or edit an existing animation path. The three types of animation paths have different functions in 3D animation scene:

- MU animation paths: The anchor points of the MU animation paths are the vertices of the line, polycurve, or spline on which incoming MUs are animated. When a MU transfers onto a material flow object, it usually moves on the animation path. For numbered 2D animation points you can define corresponding 3D animation paths that use the same name. In addition, 2D capacities (x,y) are mapped to 3D locations as animation paths named #x#y. For the ParallelProc, the 3D Viewer predefined paths from the locations #0#0 to #1#1. For the Store, paths from the locations #0#0 to #2#2 are predefined.

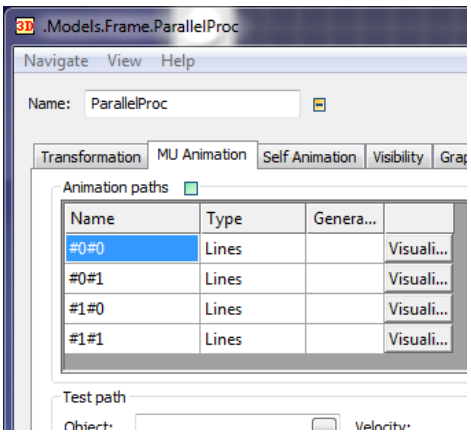

Figure 7-3 Predefined animation paths of ParallelProc

  The polycurve animation anchor points are the vertices of a curved object, i.e., a sequence of curved and straight lines, on which MUs are animated, when they are transferred onto a material flow object. The spline animation anchor points are the vertices of a spline curve.
- Self animation paths: The anchor points are the vertices of the line, polycurve, or spline on which the selected object itself moves. You can, for example, use a self animation path to simulate the movement of a robot. The polycurve animation anchor points are the vertices of a curved path, i.e., a sequence of curved and straight lines, on which the material flows object itself moves. The spline animation anchor points are the vertices of a spline curve.
- Camera animation paths: The anchor points are the vertices of the line, polycurve, or spline on which the camera moves through the scene. The polycurve animation anchor points are the vertices of a curved path, i.e., a sequence of curved and straight lines, on which the camera moves. The spline animation anchor points are the vertices of a spline curve.

The creating or editing of an animation path could be carried out on the corresponding animation tabs of 3D Properties. As an example the figure 6-26 shows, how the MU animation paths can be generated. In a similar way you can also work with self animation and camera animation.

Besides animation paths, the 3D Viewer provides another type of path to extrude shapes. This

path makes sense only by Extrusion objects, which are objects with a shape built from a profile expanded along a path. The length-oriented objects Track, TwolaneTrack, Line and FootPath in the standard library belong to that. You can edit the extruding paths on the tab Extrusion of 3D Properties of corresponding objects.

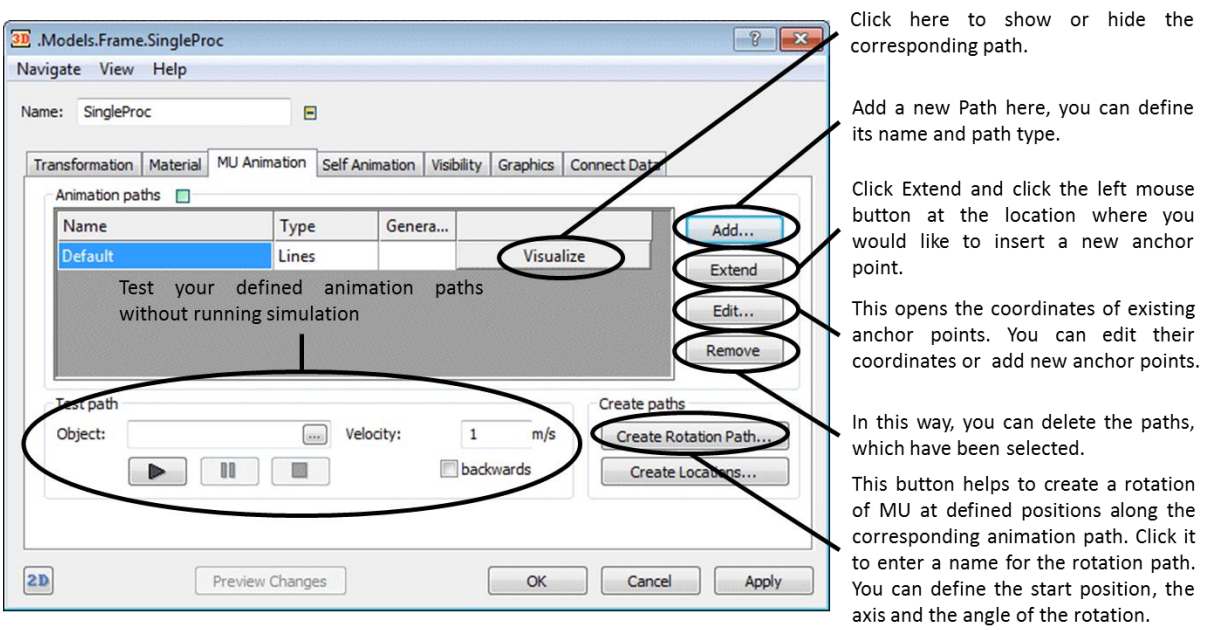

Figure 7-4 Editing MU Animation of a 3D object

6.1.6 *Exercise*

Exercise 56: 3D modeling

In this task you will practice, how a 3D model can be generated based on a 2D model.

Task:

1. Create a Frame with the name "_56_3D_Modeling" in the folder exercises.
2. Build a simple production line as shown in the figure below.

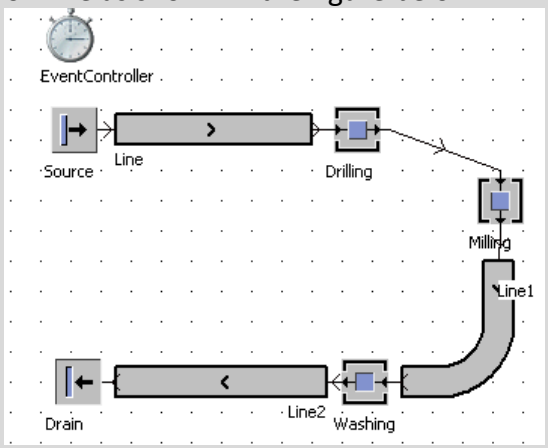

Figure 7-5 Exercise 56 Layout

3. All the objects that we have inserted into the frame should appear in the 3D Viewer. So make sure that the check boxes *Create in 3D* of all the objects are selected (in this situation we have to do nothing, because all the check boxes have been selected as default). Then click the button 3D to activate 3D Viewer.

4. Keep the online connection open and continue to edit the model in the 3D Viewer. Turn to the Toolbox and find the object PickAndPlace on the tab material Flow. Insert it between the drilling machine and milling machine. Then click the button on the 3D standard toolbar to show the connections between objects. Delete the connection between drilling and milling machines and integrate the PickAndPlace object into the production line.(connect them as shown below)

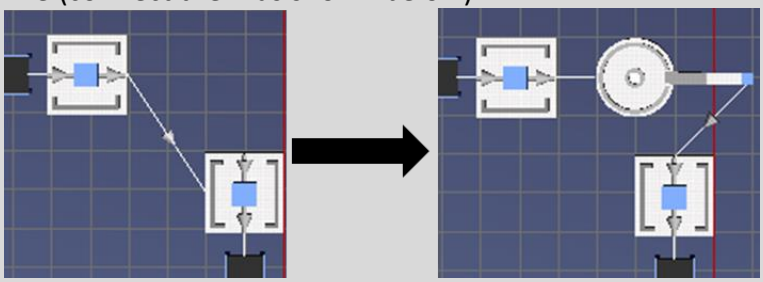

Figure 7-6 Exercise 56 Insert a robot

5. Now we are going to assign different graphics to the newly created objects to be able to distinguish them. Select an object and click the button , then find a suitable graphic for it. In this way, the following four graphics: HorizontalDrillingMachine.s3d, MillingMachine.s3d, WashingMachine.s3d and sm_nh1_130_30.s3d should be assigned to the objects: Drilling, Milling, Washing and PickAndPlace one after another.
6. Next you have to define the MU animation paths. In this model we are going to edit the default animation path of the drilling machine. Select the object Drilling and click the button and turn to the tab MU Animation, then select the path named with Default in the table and click the button Edit on the right side. Define the anchor points as shown below and then click OK.

	Position			Rotation							Time
				Angle	Axis			Center			
	X	Y	Z		X	Y	Z	X	Y	Z	
	-1.70	-1.00	1.50	0.00	0.00	0.00	1.00	0.00	0.00	0.00	undefi...
	0.48	-1.00	1.50	0.00	0.00	0.00	1.00	0.00	0.00	0.00	undefi...
	0.48	-0.40	1.10	0.00	0.00	0.00	1.00	0.00	0.00	0.00	undefi...
	1.30	-0.40	1.10	0.00	0.00	0.00	1.00	0.00	0.00	0.00	undefi...
	1.30	-0.40	1.10	0.00	0.00	0.00	1.00	0.00	0.00	0.00	undefi...

Figure 7-7 Exercise 56 Edit the anchor points of an animation path

You can compare the default animation path and the new defined animation path in the following figure.

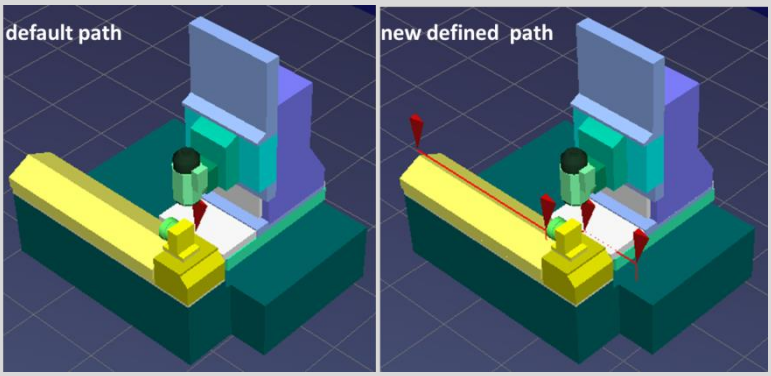

Figure 7-8 Exercise 56 Comparison of two animation paths

7. Run the simulation and enjoy your 3D journey!

6.2 Dialog

By help of the element “Dialog” your own dialog boxes can be created. These dialog boxes are used by the user to facilitate the operation of the objects in a simulation. Complex settings in frames and sub frames can be set using a dialog. A dialog serves as an interface between the simulation and the user, so that a simulation can be operated by users who have less or no knowledge of Plant Simulation.

6.2.1 *Dialog Object*

Each dialog object manages a single dialog box. A dialog may consist of the following basic elements:

- Comments/labels
- Text fields
- Buttons, menus
- List-Boxes
- Radio buttons and checkboxes
- Tabs
- Images
- Tables

When opening the Dialog object you will see something similar to Figure 7-8. The button *Show Dialog* shows the dialog you are creating (can be used to test your dialog). The Button *Edit Dialog* can be used to open the dialog in edit mode, so that you can insert object or change their position in the dialog. Alternatively you can insert and edit object to the dialog using the tab *Elements* (right click and chose the object to be inserted from the list). In the tab *Position* you can set the position of the dialog box when it is called.

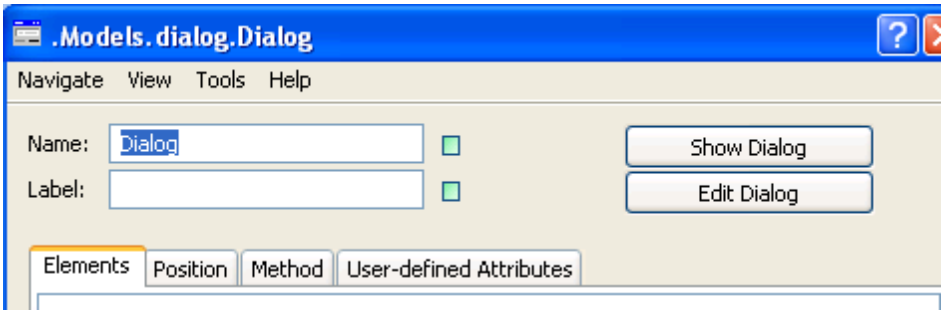

Figure 7-8 Dialog object

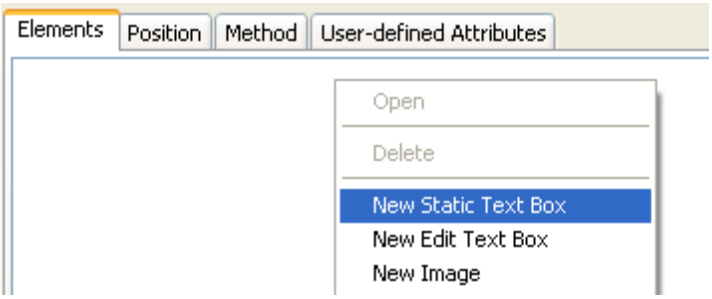

Figure 7-9 inserting an object to the Dialog

In the tab *Method* you can set the call back method (default: self.callback), and callback parameters for the buttons ok, apply and cancel (see 7.2.2).

6.2.2 *Callback method*

When the user clicks or uses the dialog in any other way, the dialog calls a method. This method is called *Callback*. The method Callback has an input parameter of type string, which makes it possible to specify which action/button the user has just performed/clicked. For example when the user clicks on the button *Apply,* the dialog calls the method *Callback* with the input parameter *"Apply"*. The callback method is defined in the tab Method (default: self.callback). It can be opened by pressing *F2. OK* calls the callback function twice: once with action *apply* and again with action *cancel.* Within the callback function you need to program the actions which should be performed when the user clicks on the respective buttons.

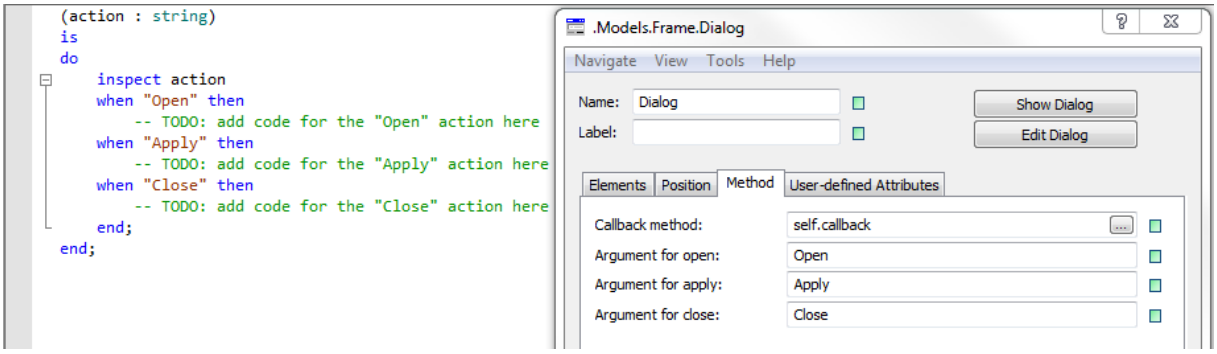

Figure 7-10 Callback method

6.2.3 *The Static Text Box*

A Static text boxes explain the dialog. It is used as labels of the input text boxes and for general operating instructions.

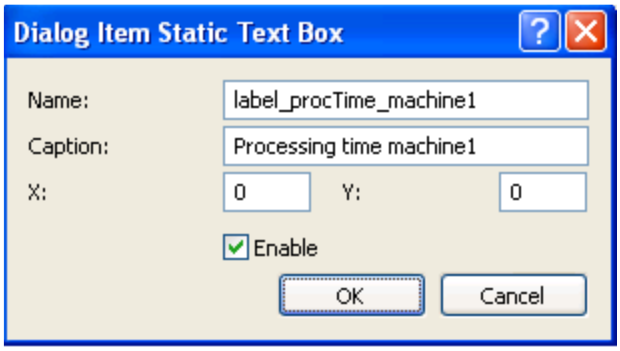

Figure 7-11 Static Text Box settings

The Static Text Box has mainly the following settings:

- A name, which is used when addressing the Static Text Box.
- A Caption: This is the text/content of the Text Box, which will appear in the Dialog box. The Content of a static text box can be changed in run time with the method <path>.setCaption(<string1>, <string2>). The method *setCaption* sets the caption of the dialog element <string1> to the text <string2>.
- Position: Both X and Y coordinates, to determine the position in the dialog box. The positions start at X = 0, Y = 0 (top left). You can set this quite easily afterwards by clicking the button *Edit Dialog.* Then, drag the fields to the correct position.
- Sensitivity: if the Static Text Box is enabled or disabled. This can be changed during runtime with the method <path>. setSensitive(<string1>, <boolean>). The method

setSensitive sets the item with the name string1 to be enabled *boolean= true or disabled Boolean = false.*

6.2.4 *The Edit Text Box*

The user can enter text into text boxes. These texts can be used afterwards by methods to preform specific actions or to be saved as settings for objects in Plant Simulation.

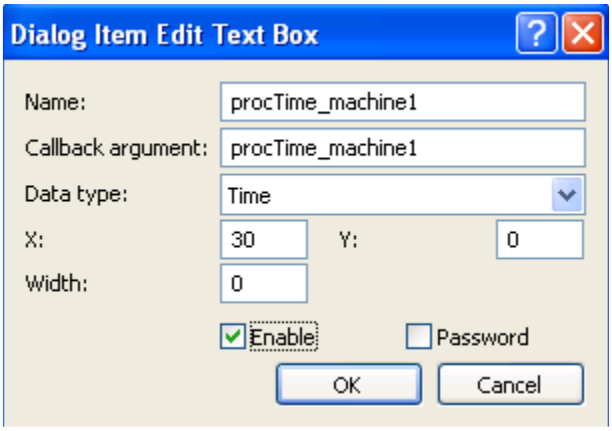

Figure 7-12 Edit Text Box settings

The Edit Text Box has mainly the following settings:

- A name, which is used when addressing the Edit Text Box.
- A Callback argument: This argument will be passed to the callback method when the user changed the content of the Edit Text Box.
- Data type: this setting restricts the possibilities of the user input to specific data types.
- Position: Both X and Y coordinates, to determine the position in the dialog box. The positions start at X = 0, Y = 0 (top left). You can set this quite easily afterwards by clicking the button Edit Dialog. Then, drag the fields to the correct position.
- Width: this specifies the width of the Edit Text Box.
- Sensitivity: if the Static Text Box is enabled or disabled. This can be changed during runtime with the method <path>. setSensitive(<string1>, <boolean>). The method setSensitive sets the item with the name string1 to be enabled boolean= true or disabled Boolean = false.
- Password: specifies if the user entries are masked or not.

Important methods for the Edit Text Box are:

- <path>.setCaption(<string1>, <string2>) : Set the contents of the text box <string1> to the new contents <string2>
- <path>.getValue(<string1>): Returns the contents of the text box <string1> (data type text).

6.2.5 *Buttons*

A dialog object has three default buttons *ok, apply, and cancel.* In the tab *Elements* you can clear the *Show Default Buttons* to let these buttons disappear. A button has similar options as

the Static and Edit Text Boxes. See figure 7-13. Both the methods *setCaption* and *setSensitve* are also applicable on buttons.

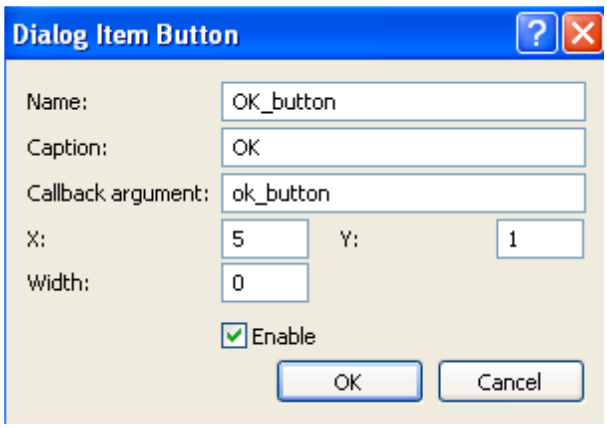

Figure 7-13 Edit Text Box settings

6.2.6 *Radio Buttons*

Radio buttons have only to possible values (true or false). It is possible to Group Multiple Radio buttons together so that only one of them can be selected. A new selection deselects the previous selection. The grouping with each other is set with a GroupID (integer). All Radio buttons with the same GroupID belong together.

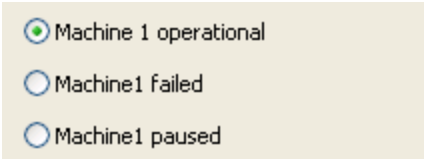

Figure 7-14 Example of a Radio Button Group

Radio buttons provide the following methods:

- <path>.setCaption(<string1>, <string2>)
- <path>.setSensitive(<string1>,<Boolean>)
- <path>.setCheckBox(<string>, <boolean>): Sets the status <boolean> of Radio button <string>.
- <path>.getCheckBox(<string>): Returns the status of the Radio button <string>

6.2.7 *Checkbox*

Like Radio buttons, Checkboxes can be either selected or cleared. A checkbox represent two states (e.g. active or not active). Unlike Radio buttons they can't be grouped together. A Checkbox provides the same methods as the radio button.

6.2.8 *Drop-Down List Box*

The problem with Radio buttons is that if you have many options they will take a large room. A solution for this problem is the use of a Drop-Down List. A Drop-Down List gives the user the possibility to choose an option from a list.

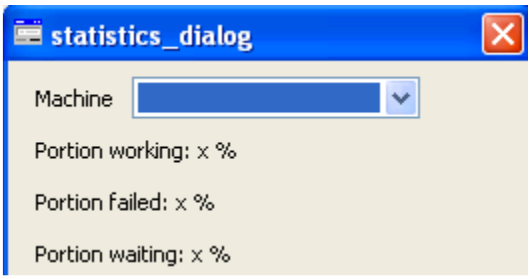

Figure 7-15 Drop-down List

Items can be simply added to the drop-down list be entering them to a table see Figure 7-16

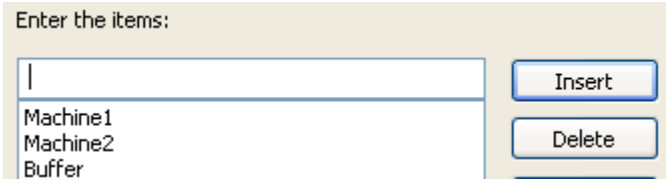

Figure 7-16 Drop-down List items

A drop-down list provides the following methods:

- To identify the selected entry the method <path>.getValue(<string>) (pass the name of the drop-down list box) is used.
- To select the entry with the index <integer> in the dialog element with the name <string1> the method <dialog>.setIndex(<string1>, <integer>) is used.
- To determine the index of the selected list entry the method <dialog>.getIndex(<string>) is used.

6.2.9 *List View*

The List View is used to display the content of a table in a dialog box. The user can select a row from the table by double clicking on it.

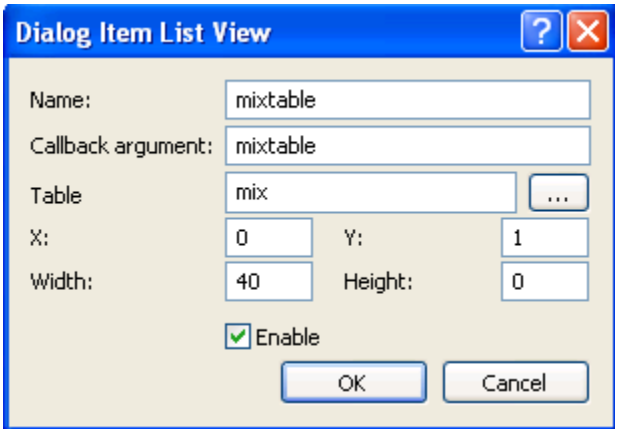

Figure 7-17 List View Settings

A List View provided the following methods:

- <path>.setTable(<string>,<object>): Sets the table of the element <string> to <object>.
- <path>.setSensitive(<string1>,<Boolean>): Activates/deactivates the List View <string1>.

- <path>.setTableRow(<string>,<integer>): Sets the selected row in the List View.
- <path>.getTable(<string>): Returns the table name of the List View <string>
- <path>.getTableRow(<string>): Returns the index of selected row (starting from 1)

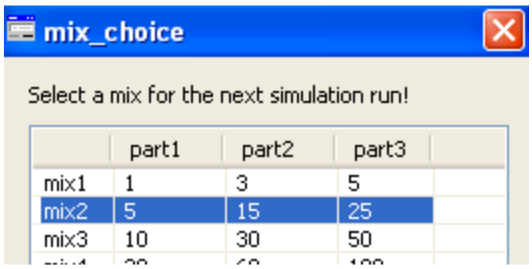

Figure 7-18 List View example

6.2.10 *Exercise*

Exercise 57: First Dialog Box

In this Task you will create a dialog box for the model you created in a previous Exercise. We will use Exercise 50; however feel free to use another Exercise as long as it has Machines, buffers, and a table. The First Dialog box will allow the user to change the processing time of a Machine (SingleProc).

Task:

1. Duplicate the Frame "_50_Attribute" and rename it to "_57_ DialogBox_1".

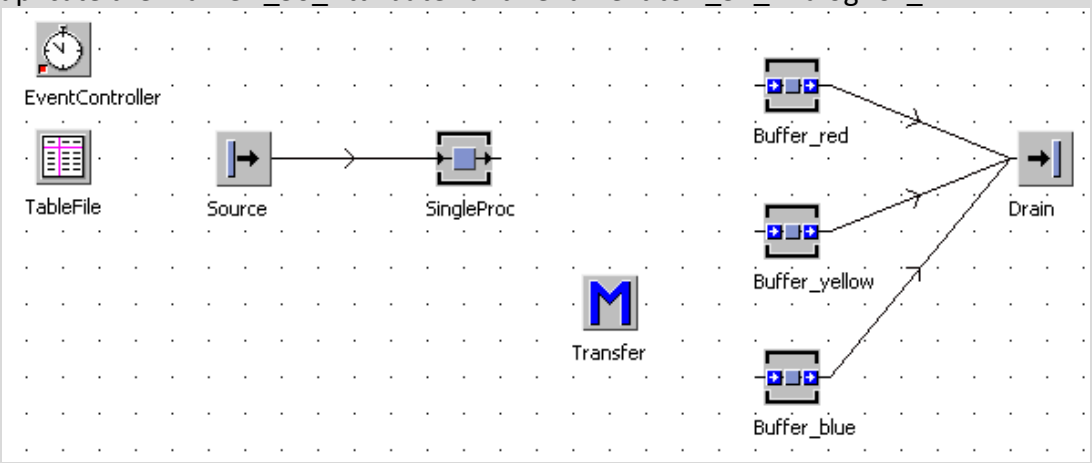

Figure 7-19 Exercise 57 Initial Layout

2. Drag and drop a Dialog object from the tool bar to the frame.
3. Open the object, right click in the tab Elements, and create a Static Text Box, with the caption "Processing Time:"

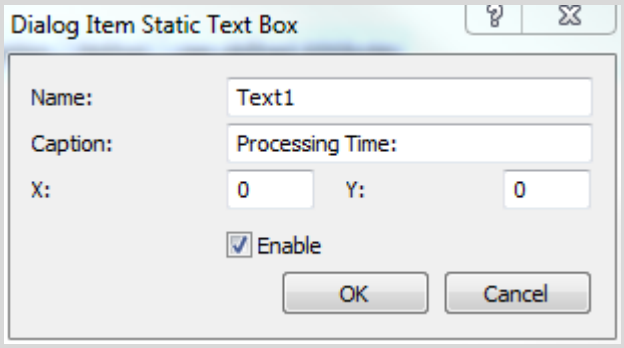

Figure 7-20 Exercise 57 Static Text Box Settings

4. Create a Edit Text Box and set it to be an input field for the processing time

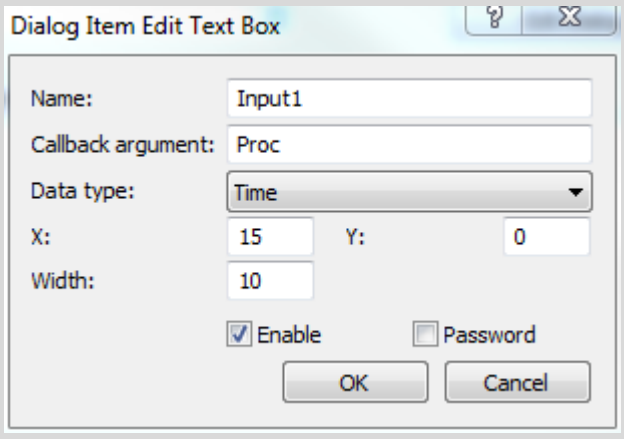

Figure 7-21 Exercise 57 Edit Text Box Settings

5. Click on the tab User Defined Attributes then open the callback method by double clicking on its name. Then Deactivate the Format inheritance.
6. Change the callback method. When the user clicks on apply the processing time of the SingleProc will change to what the user inputted.

```
(action : string)
is
do
    inspect action
    when "Open" then
        -- TODO: add code for the "Open" action here
    when "Apply" then
        SingleProc.procTime:=str_to_time(?.getValue("Input1"));

    when "Close" then
        -- TODO: add code for the "Close" action here
    end;
end;
```

Figure 7-22 Exercise 57 Method Callback

7. Test the dialog and see how the Processing time of the SingleProc changes.

Exercise 58: Dialog Box 2

In this task you will add new functions to the Dialog Box you created in Exercise 57. This Dialog box will allow the user to change the processing time of a Machine (SingleProc), Buffer_red, Buffer_yellow, Buffer_blue, and Drain.

Task:

1. Duplicate the Frame "_57_ DialogBox_1 " and rename it to "_58 _DialogBox_2".

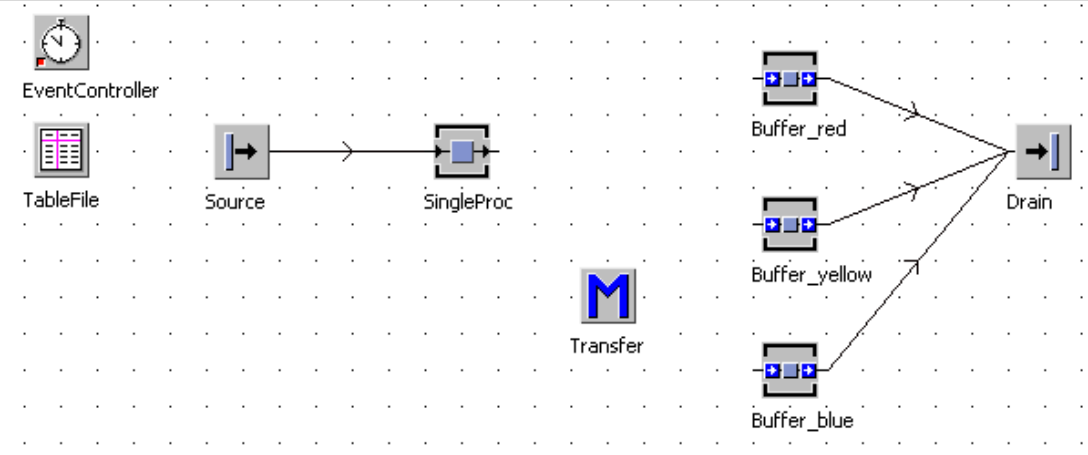

Figure 7-19 Exercise 58 Initial Layout

2. Open the Dialog Box then press on Edit Dialog.
3. Move Input1 and Text1 down so that you leave place for another text and a drop-downList.
4. Insert a static Text Box (caption="Machine:") and a drop-down List

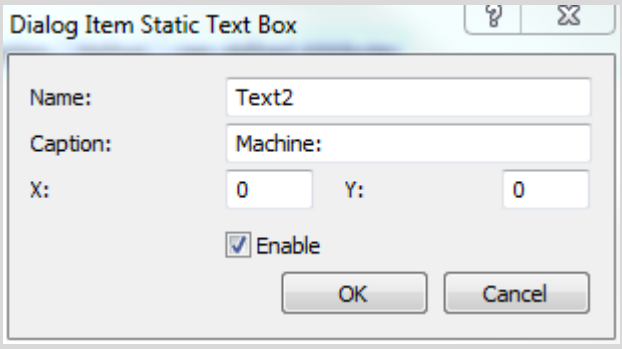

Figure 7-20 Exercise 58 Text2 Setting

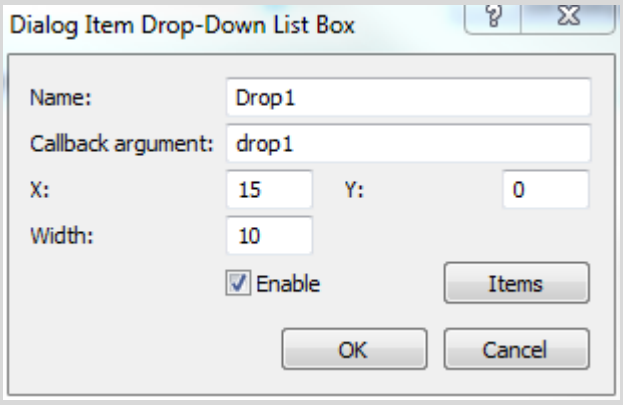

Figure 7-21 Exercise 58 Drop-down List setting

5. The drop-down List should have the items SingleProc, Buffer_red, Buffer_yellow, Buffer_blue, and Drain.
6. Open the callback method and change its code so that it changes the processing time of the chosen object.

```
(action : string)
is
do
    inspect action
    when "Open" then
        -- TODO: add code for the "Open" action here
    when "Apply" then
        extendPath(?.getValue("Drop1")).procTime:=str_to_time(?.getValue("Input1"));

    when "Close" then
        -- TODO: add code for the "Close" action here
    end;
end;
```

Figure 7-22 Exercise 58 Callback method

7. Try the Dialog.

Exercise 59: Dialog Box 3

In this task you will add new functions to the Dialog Box you created in Exercise 61. This Dialog box will allow the user to view and edit the Tablefile, in addition of changing the processing time of a Machine (SingleProc), Buffer_red, Buffer_yellow, Buffer_blue, and Drain.

Task:

1. Duplicate the Frame "_58_ DialogBox_2 " and rename it to "_59 _DialogBox_3".
2. In the dialog add a List View for TableFile.

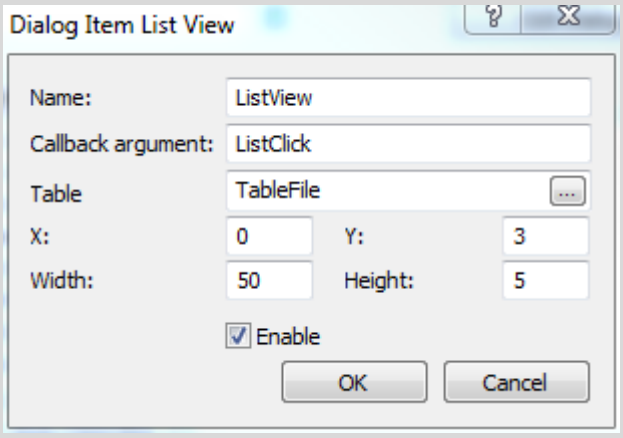

Figure 7-23 Exercise 59 List View settings

3. Add Four Static Text Box as the figures Below

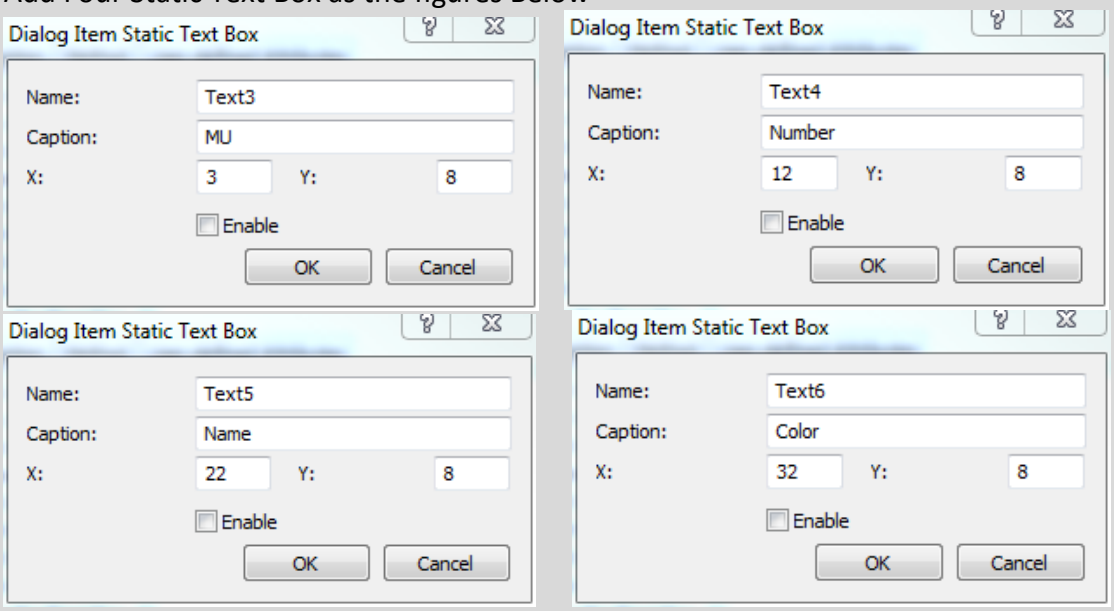

Figure 7-24 Exercise 59 Text Box Settings

4. Add three Edit Text Box and a Drop down list as in the figures below

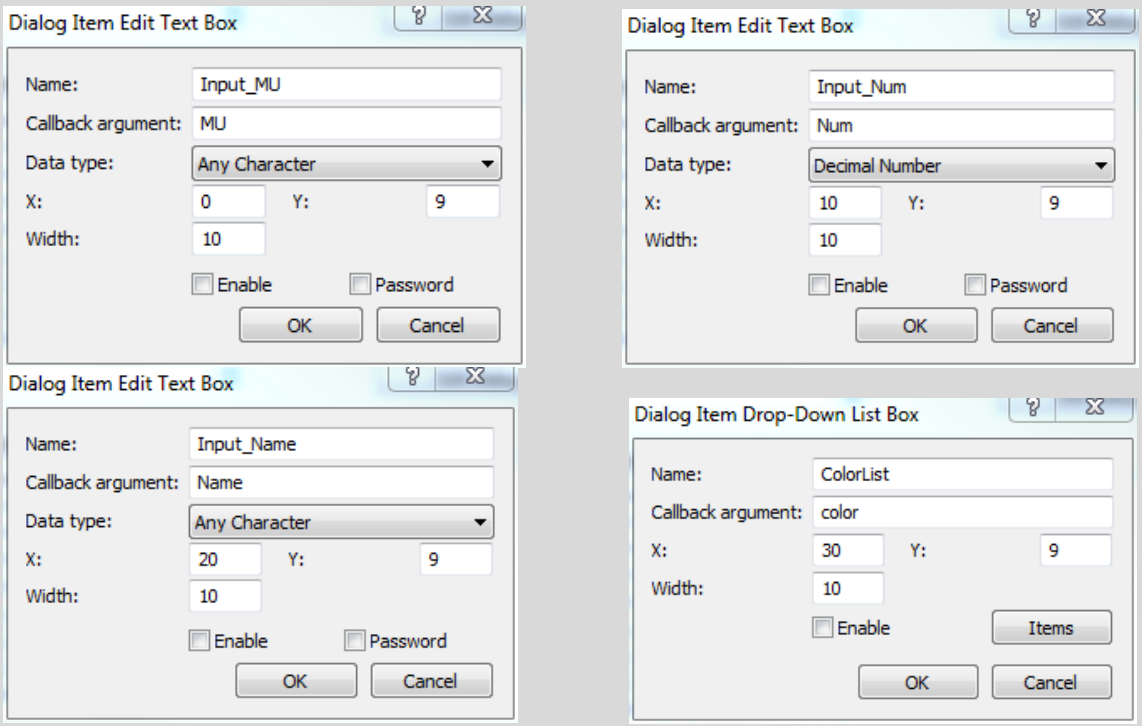

Figure 7-25 Exercise 59 Edit text and Drop-Down List Settings

5. Add a button as follows

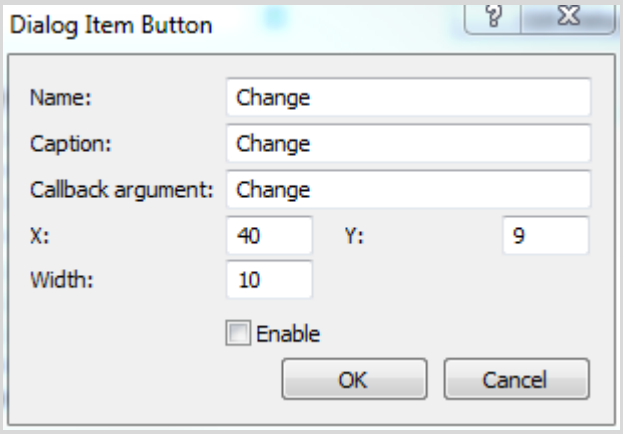

Figure 7-26 Exercise 59 Button Settings

6. Change the callback method so that when a row in the List View is double clicked, the edit/static text boxes, dropdown list, and the button “Change” will be enabled and the data of this row will appear in them and can be changed.
Hint: you have to update the List view so that the changes appear.

```
(action : string)
is
do
    inspect action
    when "Open" then
        -- TODO: add code for the "Open" action here
    when "Apply" then
        extendPath(?.getValue("Drop1")).procTime:=str_to_time(?.getValue("Input1"));

    when "Close" then
        ?.CallBack("reset");

    when "ListClick" then
        local row:=?.getTableRow("ListView");

        -- get the values of the row and enable the items
        ?.setCaption("Input_MU", To_str(TableFile["MU",row]));
        ?.setSensitive("Input_MU",true);

        ?.setCaption("Input_Num", To_str(TableFile["Number",row]));
        ?.setSensitive("Input_Num",true);
        ?.setCaption("Input_Name", To_str(TableFile["Name",row]));
        ?.setSensitive("Input_Name",true);

        --find the color row
        for local color:=1 to TableFile["Attributes",row].ydim loop
            if TableFile["Attributes",row][3,color] then
                ?.setIndex("ColorList", color+1);
                exitloop;
            end;
        next;
        ?.setSensitive("ColorList",true);

        ?.setSensitive("Text3",true);
        ?.setSensitive("Text4",true);
        ?.setSensitive("Text5",true);
        ?.setSensitive("Text6",true);

        ?.setSensitive("Change",true);

    when "reset" then
        -- in this method there will be the reset operations for the dialog

        -- rest the values to default
        ?.setCaption("Input_MU", "");
        ?.setSensitive("Input_MU",false);

        ?.setCaption("Input_Num", "");
        ?.setSensitive("Input_Num",false);
        ?.setCaption("Input_Name", "");
        ?.setSensitive("Input_Name",false);
        ?.setIndex("ColorList", 1);
        ?.setSensitive("ColorList",false);

        ?.setSensitive("Text3",true);
        ?.setSensitive("Text4",true);
        ?.setSensitive("Text5",true);
        ?.setSensitive("Text6",true);

        ?.setSensitive("Change",true);
```

```
61:
62:
63:     when "Change" then
64:         row:=?.getTableRow("ListView");
65:
66:         TableFile["MU",row]:= ?.getValue("Input_MU");
67:         TableFile["Number",row]:= str_to_num(?.getValue("Input_Num"));
68:         TableFile["Name",row]:= ?.getValue("Input_Name");
69:
70:         local color:=?.getIndex("ColorList");
71:         for local counter:=1 to TableFile["Attributes",row].ydim loop
72:             if counter=color-1 then
73:                 TableFile["Attributes",row][3,counter]:= true;
74:             else
75:                 TableFile["Attributes",row][3,counter]:= false;
76:             end;
77:         next;
78:
79:
80:         ?.setTable("ListView",TableFile);
81:         ?.Callback("reset");
82:     end;
83:
84:
85: end;
86:
```

7. Try the Dialog

7 Simulation-based improvement methods

7.1 Kanban System

Kanban is a scheduling system for lean and just-in-time (JIT) production. It was developed by Taiichi Ohno, at Toyota, to find a system to improve and maintain a high level of production. One important determinant of the success of production scheduling based on demand "pushing" is the ability of the demand-forecast to receive such a "push". Kanban, by contrast, is part of an approach where the "pull" comes from the demand. The supply or production is determined according to the actual demand of the customers.

To model production facilities using Kanban objects, you can turn to the Object Libraries. The Kanban objects can be added to your simulation model by selecting File > Manage Class Library > Libraries > Standard >Free > Kanban or clicking  on the Standard toolbar

This then adds the toolbar Kanban to the Toolbox.

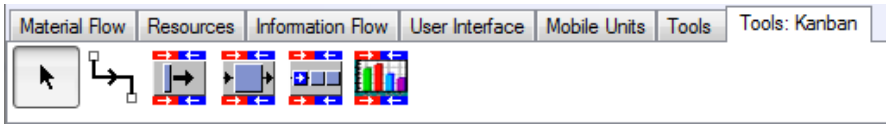

Figure 7-1 Toolbar Kanban

KanbanSingleProc 

The KanbanSingleProc triggers the entire Kanban mechanism. It creates the requests, which are passed to the KanbanSource via the KanbanBuffer.

The Kanban system can create Kanban orders with a random selection or via a sequence table. The first order is created during the initialization phase of the model. The next order is generated after this part has arrived and has been processed. You can set if the KanbanSingleProc waits for a part of the selected type or creates an order as soon as the part has been processed.

KanbanSource 

The KanbanSource employs the Kanban principle. It produces parts, when it receives a request from a KanbanBuffer or KanbanSingleProc.

KanbanBuffer 

The KanbanBuffer follows the Kanban principle. It only moves parts on to its successors, when other objects, usually a KanbanBuffer or a KanbanSingleProc, order these parts.

The KanbanBuffer maintains a threshold value for each part, the minimum inventory. When the inventory falls short of this number, parts are ordered from the supplier of this part. The ordered amount is calculated like this:

Ordered amount = maximum inventory - current inventory - still open, but already ordered amount.

KanbanChart 

You can use the KanbanChart to show the different part types located in the KanbanBuffer. You can select the KanbanBuffer to be observed in the dialog of the KanbanChart, or you can drag the KanbanBuffer to be observed onto the icon of the KanbanChart and drop it there.

7.1.1 *Exercise*

Exercise 60: Kanban System01

Task:

1. Create a new Frame"_60_Kanban_System01" in the exercise-folder.
2. Build a production line with a KanbanSource, a KanbanBuffer, a KanbanSingleProc and a Drain. Connect them as shown in the following figure. Insert one KanbanChart, which is used to observe the KanbanBuffer.

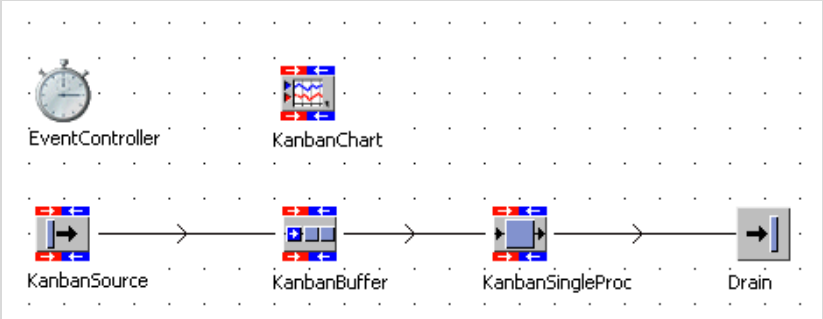

Figure 7-2 Exercise 60 Layout

3. Insert parts in the KanbanSoure by clicking the button Insert New Part on the tab Part information as shown below, so the parts will be produced, when the KanbanBuffer order them.

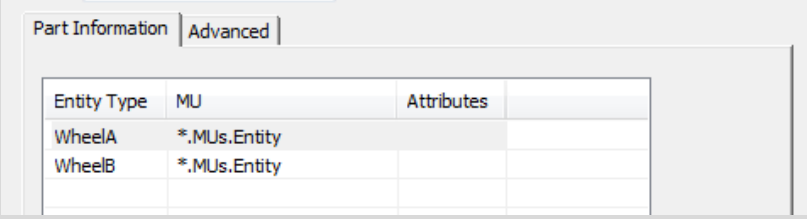

Figure 7-3 Exercise 60 KanbanSource

4. In order to configure the KanbanBuffer, you can insert parts also by clicking the button Insert New Part on the tab Part Information of the KanbanBuffer. Then define their Max.Stock, Min.Stock, Initial Stock and Supplier (see the following figure).

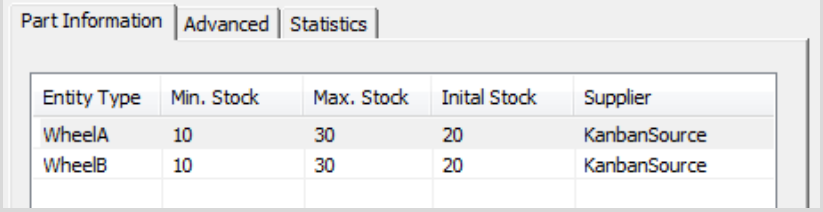

Figure 7-4 Exercise 60 KanbanBuffer

5. The KanbanSingleProc decides which part it intends to process next and orders his part from the KanbanBuffer located in front of it. You can configure the KanbanSingleProc as shown below.

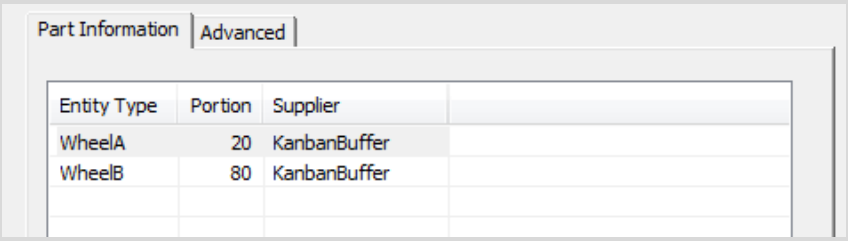

Figure 7-5 Exercise 60 KanbanSingleProc

Two part types in the KanbanSingleProc should be defined, WheelA and WheelB. WheelA is ordered with a probability of 20%, WheelB is ordered with a probability of 80%.

6. Drag the KanbanBuffer onto the icon of the KanbanChart and drop it there.
7. Start the simulation.

Note: *To visualize the occupation of the buffer, right-click the KanbanChart, select Show and start the simulation. The charts then shows the typical saw tooth progression of the occupation of the buffers.*

Exercise 61: Kanban System02

Task:

1. Create a new Frame"_61_Kanban_System02" in the exercise-folder.
2. Insert the objects und connect them as the figure shows below.

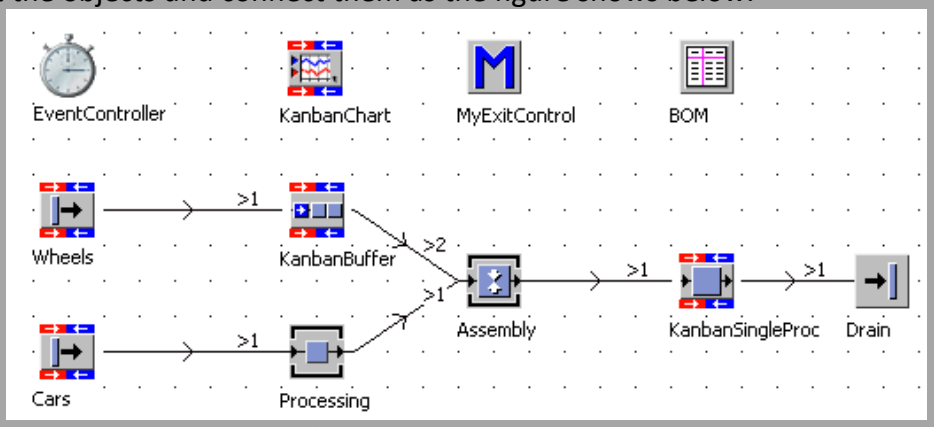

Figure 7-6 Exercise 61 Layout

Note: *To view the order in which you connected the stations, select View > Options > Show Predecessors.*

3. Configure the KanbanSources "Cars" and "Wheels" as shown below, so the sources will produce the parts, when the KanbanSingleProc and the KanbanBuffer orders them.

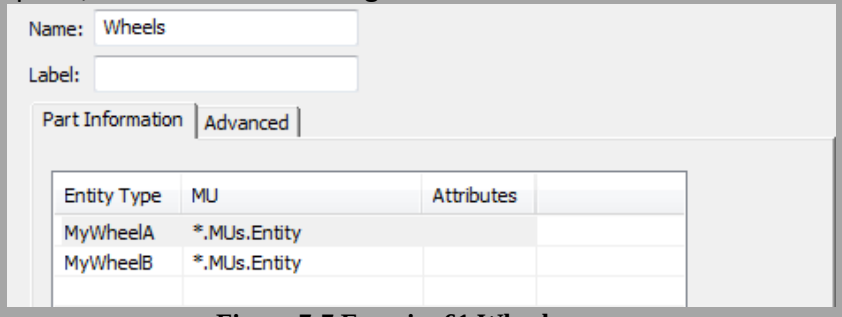

Figure 7-7 Exercise 61 Wheels

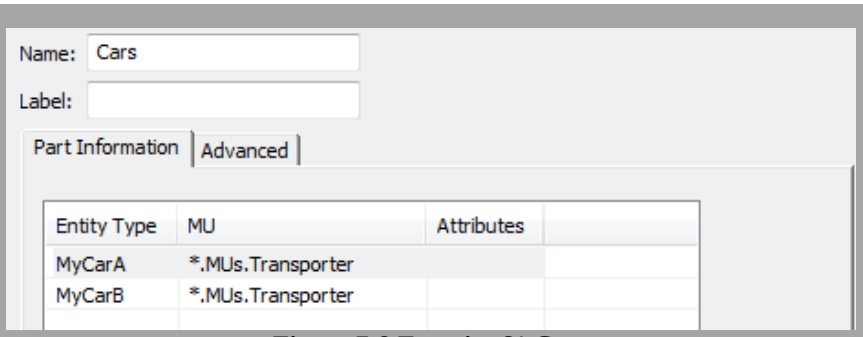

Figure 7-8 Exercise 61 Cars

4. By configuring the KanbanBuffer, you have to enter the type of part you want to order, the amount of parts you want to keep in stock and the station which produces the parts.

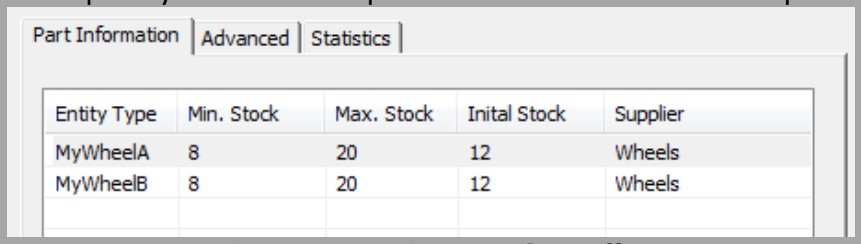

Figure 7-9 Exercise 61 KanbanBuffer

5. Configure the Assembly station as follows:
   Main MU from Predecessor: 1
   Assembly mode: Attach MUs
   Exiting MU: Main MU
   Sequence: MUs then Services
   Then open the Assembly table and enter the values as shown below.

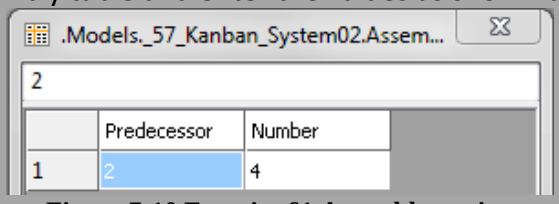

Figure 7-10 Exercise 61 Assembly station

6. To configure the KanbanSingleProc, which is the crucial station of the entire system as it orders the parts to be produced and assembled, when they are needed, click Insert New Part on the tab Part Information. Enter the types of parts you want to order and the suppliers as shown in the following figure.

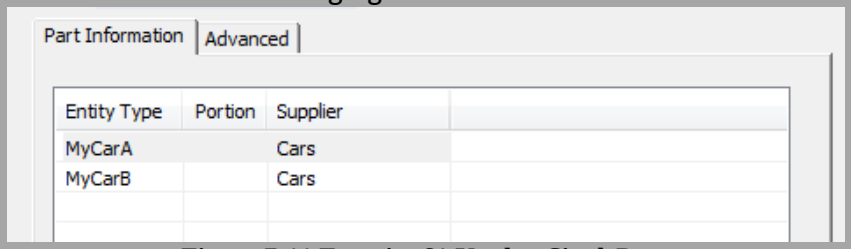

Figure 7-11 Exercise 61 KanbanSingleProc

Note: *To delete an entry from this table, select Tools > Open Kanban Information Table and delete the respective row from that TableFile. Instead, you can also double-click the entry on the tab, delete the EntityType and then close the dialog.*

7. Then turn to the Tab Advanced, click Open Kanban Sequence table and enter the sequence into the table. Make sure that the check box Cyclical is selected, because we want the sequence of orders to be repeated (see figure below).

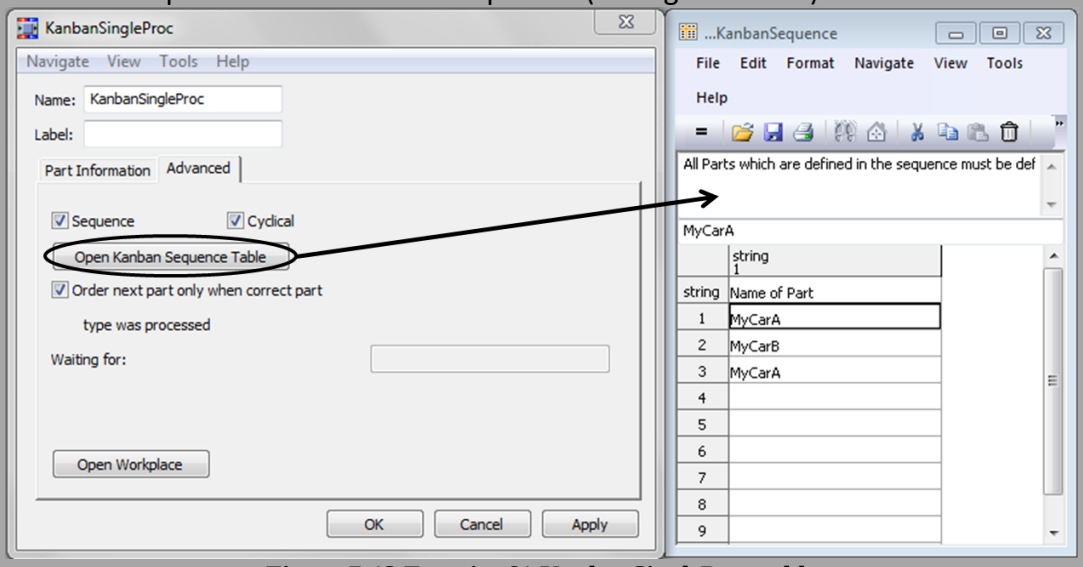

Figure 7-12 Exercise 61 KanbanSingleProc table

8. To make the KanbanBuffer order the wheels MyWheelA or MyWheelB, for the car models MyCarA or MyCarB, we enter this source code into the method MyExitControl.

```
is
        RequiredWheelType: String;
do
        RequiredWheelType := BOM [1, @.EntityType];
        Kanbanbuffer.RequestParts(RequiredWheelType, 4);
        @.move;
end;
```

Connect then the method with the exit control of the singleProc Processing.

Note: *This Method instruct the buffer to get the correct type of wheels according to the TableFile BOM (bill of materials), which you are going to fill in the next step. It then moves the part on to the successor.*

9. In the table BOM, activate the column index and the row index. Then fill it as shown below.

	string 0	string 1	string 2
string	Car Type	Wheel Type	
1	MyCarA	MyWheelA	
2	MyCarB	MyWheelB	
3			

Figure 7-13 Exercise 61 BOM

10. Select the KanbanBuffer in the dialog of the KanbanChart or use drag and drop, so that you can use the Chart to observe the inventory of the KanbanBuffer.
11. Run the Simulation.

7.2 Genetic algorithms

7.2.1 *Introduction into genetic algorithms*

In the computer science field of artificial intelligence, a genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic is routinely used to generate useful solutions to optimization and search problems.
In the biological evolution, once an individual acquires an advantageous characteristic, either by mutation or by cross-breeding, it may use these advantages and pass them on to its offspring. They, and their descendants, will benefit from these selection advantages, so that over the generations all individuals of the corresponding species will acquire these characteristics. Individuals without these advantages will have fewer offspring because of this selection disadvantage, so that the lack of an advantageous characteristic will lead to their extinction.
The GA works along the same line as the biological evolution does. The most successful solutions are selected from a given initial set of proposed solutions and used for creating new solutions. Then, better solutions are created over succeeding generations. If a proposed solution has a selection advantage and if it then is used for creating new solutions depends on the so called fitness value that is produced by a simulation model.
Major advantages of genetic algorithms are that they are task-independent and that they return good results. Tecnomatix Plant Simulation provides the GA-tools, which are ideally suited in effectively supporting in a variety of simulation-based optimization tasks. By selecting File > Manage Class Library > Libraries > Basic objects >GA, you can find the 5 basic GA objects. This then adds the toolbar Genetic Algorithms to the Toolbox. There is a control object (GAOptimization) for performing the optimization in your model. The optimization problem can be defined by the other 4 table objects.

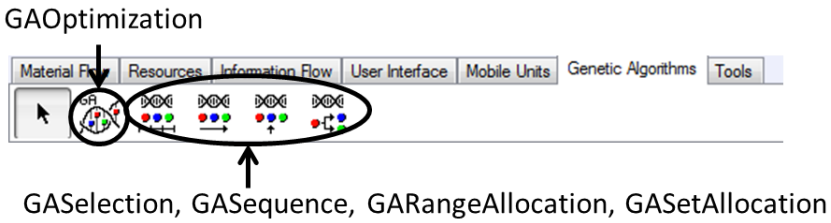

Figure 7-14 Toolbar Genetic Algorithms

Click File > Manage Class Library > Libraries > Wizard for Genetic Algorithms, so that the GAWizard can be added to your simulation model.

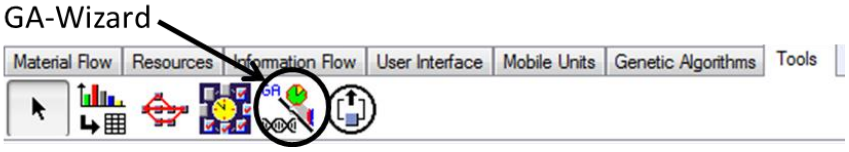

Figure 7-15 GAWizard

You can perform the optimization by either
1. the basic control object GAOptimization and a set of GA tables or
2. the GAWizard together with GA tables.
Note that the GAWizard contains the object GAOptimization.

7.2.2 *Types of optimization tasks*

You might, for example, have to optimize the loading sequence of orders for a production facility. A simulation model will provide you with the entire processing time for each proposed loading sequence of orders. The evolution cycle will then produce successively better sequences that will process the orders in as little time as possible. This is a typical example of sequence task in a production and manufacturing environment. In general the tasks can be divided into three types: sequence task, selection task and allocation task.

Sequence task

When solving sequence tasks, the relative sequence of the order of the item of the definition set is important. Each item may only occur once in each solution. Genetic Algorithms provide the object GASequence to solve these tasks.

Selection task

When solving selection tasks the selection of a defined number of items from a defined set has to be improved. Thus, the solution set is always smaller than the definition set. Each item of the definition set may only occur once in the solution set. Genetic Algorithms provide the object GASelection to solve these tasks.

Allocation task

When solving allocation tasks, items from another set (range) are allocated to the items of the definition set. Each item of the definition set is assigned an item of the range, in which the items out of the range may be used once, several times or not at all. You can also define a range of its own for each item of the definition set. Genetic Algorithms provide the object GASetAllocation to solve these tasks. If the range consists of numerical values within a defined interval, use the object GARangeAllocation.

7.2.3 *The process and genetic operators*

Using the definition of the basic tasks, the object GAOptimization generates solutions for the individual basic tasks. The proposed solution for a basic task is called a chromosome. All chromosomes together form the individual. The fitness value of the individual will be determined by the simulation model or an evaluation method. Each individual may be considered as a solution for a GA task.

Initialization Phase

First, Genetic Algorithms initialize the chromosomes of the start generation with the optional initialization values. You can enter the initial rate in percent, which is the number of items to be initialized, on the tab Attributes of the objects GASelection, GASequence, GARangeAllocation, and GASetAllocation. The number of predefined chromosomes is between 0 and the size of the generation.

The Evolution Cycle

After the initialization, the individuals of the start generation are evaluated by determining a fitness value for each solution. Then, a selection rule, which can be selected under Parent selection in the object GAOptimization, determines, which parents out of the start generation are going to be used. The individuals you picked are going to be the parents. Both parents are copied, so that daughter and son are clones of mother and father. Then, the genetic operators are going to be applied to daughter and son. Whether the operators are used or not depends on the probability of the operators, which you define. The evaluation method then starts calculating the fitness values.

According to the offspring selection the individuals for the next generation are chosen from each family. Once all individuals for the next generation are determined, new pairs of parents

are determined from these individuals that then generate the children of the next generation, etc.
This cycle is terminated, when the maximum number of generations or a defined grade for the fitness value or the average of the fitness values is reached.

Genetic operators

Mutation operator: the operator, which randomly changes individual genes.
Inversion operator: the operator, which inverts the sequence within a defined range of the individual.
Crossover operator: As opposed to the mutation and inversion operators, the crossover operator is applied to two chromosomes. It exchanges items between these two. It initially selects two randomly crossing points and then exchanges the ranges between these two points.
Note that by different basic tasks the genetic operators have different ways of functioning. E.g. for allocation tasks the mutation operator stochastically determines a value from the allocation set or chosen from the defined interval and then allocated to the selected gene. While for sequence tasks the mutation operator exchanges two randomly chosen genes. For further information of function principles of the genetic operators, please turn to Add-Ins reference.

7.2.4 *GAWizard* 

In the previous section we have mentioned that Plant Simulation provides five basic GA objects: GAOptimization  , GASelection  , GASequence  , GARangeAllocation  and GASetAllocation  . You can control the optimization process directly with these basic objects (the configuration of GAOptimization and four GA-tables e.g. definition of contents, attributes and genetic operators will not be elaborated here. You can find the details in the Add-Ins Reference). However it is sometimes complicated to define and run the optimization in this way, especially when you have to carry out a complex task, which consists of a set of different basic tasks. Instead of individually configuring GA objects, you can accomplish this much more comfortably with the GAWizard. In the following section we are going to introduce the object GAWizard and its configuration.
The GAWizard integrates genetic algorithms into an existing simulation model. The evaluation is based on a simulation run as well as for evaluation based on calculations in methods.

Definition of the optimization problem

You can define the optimization problem with the GAWizard by using drag & drop or GA-tables.
Drag & drop: Hold down Shift and drag the tables, the sequence of which you would like to optimize onto the GAWizard and drop it there. When you would like to optimize an attribute of an object, hold down Shift and drag this object onto the GAWizard, drop it and select the attribute. Then, you can change the value range of the parameters to be optimized into the table ProblemDefinition. To open this table, click the button Open next to Definition of the optimization parameter on the tab Define.
GA-tables: When you use GA-tables for describing your optimization problem, hold down Shift and drag these tables onto the Wizard and click Apply. This registers all GA-tables with the GAWizard. Then configure all GA-tables on the tab Define. Select the check box Configuration method, click the button Edit and program the method setParameter. The global variable individual is a table of the individual to be evaluated, which the genetic algorithm created.

Definition of the fitness calculation

You can determine the fitness value with the GAWizard by using the table Fitness or the method calculateFitness.
The table Fitness: Select the radio button Fitness calculation by table on the tab Define and click Open. This opens the table Fitness. Enter the names of the values into the cells of the column Target value. When the fitness value is determined by several target values, enter their weighting into the cells of the column Weighting (you can also drag the object, whose attributes and methods you would like to use for evaluating the created solutions onto the GAWizard and drop it there. Then select the attributes and methods of the object).
The method calculateFitness: Select the radio button Fitness calculation by method on the tab define and click Edit. This opens the method calculateFitness. Take special care of the paths to the objects for which you define parameters, as the method calculateFitness is located in the Frame in which the object GAWizard is located.

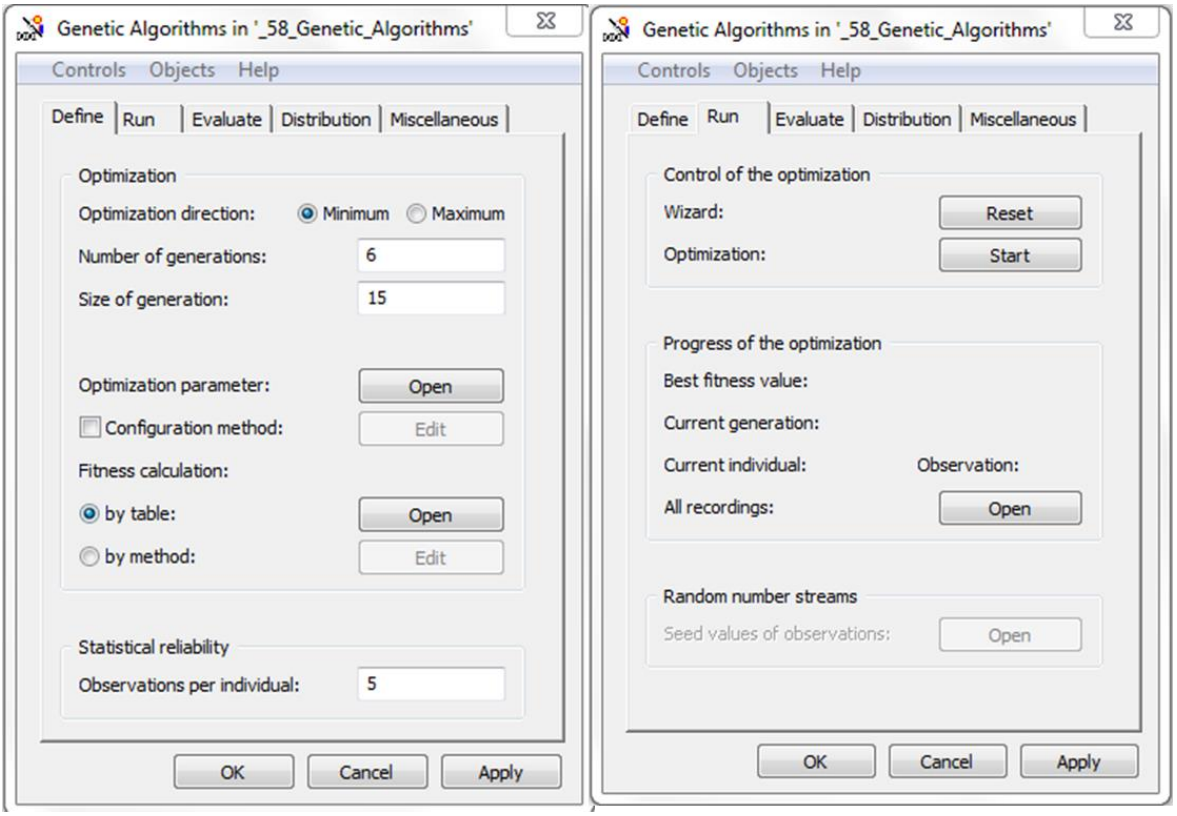

Figure 7-16 Dialog window of GAWizard

In addition you have to select the direction of the optimization: Minimum or Maximum and enter the size of generation, the Number of generations on the tab Define. Note that for complex problems the generation size must be chosen sufficiently large. Enlarge the number of generations step-by-step until the results will not improve further.

7.2.5 *Exercise*

Exercise 62: Genetic Algorithms

In this task we are going to improve a production line with the help of GA. The goal is to achieve a minimum production time. This task is a combination of two basic tasks: determining the sequence of delivery (sequence task) and allocating the capacity of a buffer (allocation task). The GAWizard will be used to build the simulation model and to control the improvement process. With this exercise you can get an insight on the application of the GA.

Task:

1. Create a Frame with the name "_62_Genetic_Algorithms" in the folder exercises.
2. Build the model as shown in the figure.

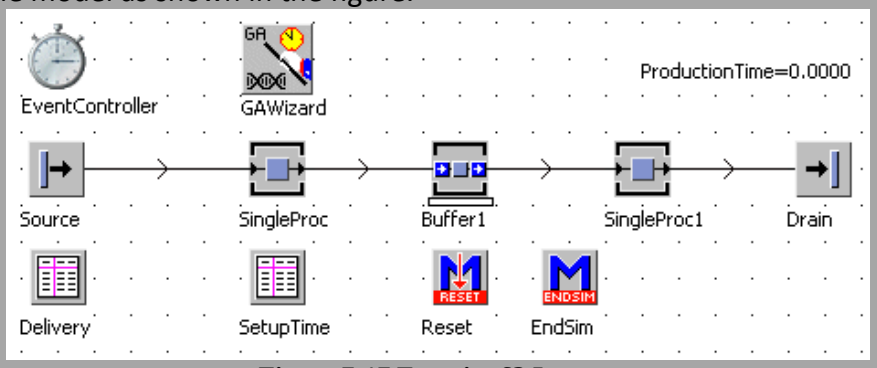

Figure 7-17 Exercise 62 Layout

3. The source is going to produce the products according to the table file Delivery. To configure the source, drag the Delivery onto Source and drop it there. Then select Sequence by the MU Selection on the tab Attribute of the source object. The table file Delivery should be created as shown below.

	object 1	integer 2	string 3	table 4
string	MU	Number	Name	Attribute
1	.MUs.Entity	1	A	
2	.MUs.Entity	1	B	
3	.MUs.Entity	1	C	
4	.MUs.Entity	1	D	
5	.MUs.Entity	1	E	
6	.MUs.Entity	1	F	
7				

Figure 7-18 Exercise 62 Delivery

4. The SingleProc has a constant processing time of 1 minute. The Set-up time is determined by the table file SetupTime. Select Matrix (Type) as the type of Set-up time on the tab Time of SingleProc and enter the table file name "SetupTime" in the following textbox. The following figure shows the content of SetupTime.

	string 0	time 1	time 2	time 3	time 4	time 5	time 6	stri 7
string		A	B	C	D	E	F	
1	A	0.0000	3:20.0000	1:00.0000	2:00.0000	3:20.0000	5:00.0000	
2	B	2:00.0000	0.0000	5:00.0000	3:00.0000	1:00.0000	2:30.0000	
3	C	1:30.0000	2:40.0000	0.0000	4:10.0000	3:20.0000	3:00.0000	
4	D	5:00.0000	3:00.0000	3:00.0000	0.0000	3:00.0000	1:00.0000	
5	E	3:20.0000	1:00.0000	2:40.0000	2:00.0000	0.0000	4:10.0000	
6	F	2:00.0000	2:30.0000	2:00.0000	1:30.0000	3:20.0000	0.0000	
7	-	3:00.0000	3:00.0000	3:00.0000	3:00.0000	3:00.0000	3:00.0000	
8								

Figure 7-19 Exercise 62 Setup Time

5. The processing time of SingleProc1 is determined by a negative exponential Distribution (Negexp). Enter 2 minutes ("2:00") as the value of beta. Its Set-up time is zero.
6. Set the data type of the variable ProductionTime as "time". This variable is used to observe the total production time of completing the 6 products A to F.
7. At the end of each simulation the total production time should be assigned to the variable ProductionTime. Enter the following code in the method EndSim.

```
is
do
        ProductionTime:= EventController.SimTime;
end;
```

At the beginning of each simulation the variable ProductionTime should also be reset. Enter the following code into the method Reset.

```
is
do
      ProductionTime:= 0;
end;
```

8. Configure the GAWizard. First of all we have to define the problem. Hold down Shift and drag the table file Delivery onto GAWizard and drop it there. Similarly hold down Shift and drag the object Buffer1 onto GAWizard and drop it there. Then select the attribute "Capacity". Click the button Open of the item Optimization parameter on the tab Define of GAwizard, a table will appear. Change the upper bound of the parameter "root.Buffer1.capacity" to 10. The table is finally set as shown in the following figure.

	string 1	object 2	string 3	integer 4	string 5	string 6
string	Parameter:	root.Delivery	Parameter:	root.Buffer1.Capacity	Parameter:	
1	Sequence of	root.Delivery	Lower bound	1	Lower bound	
2	6 Elements		Upper bound	10	Upper bound	
3			Increment	1	Increment	
4						
5						

Figure 7-20 Exercise 62 Definition of problem

Then the fitness calculation should be defined. Drag the variable ProductionTime onto the GAWizard and drop it there. This operation sets the ProductionTime as the target value. Click the button Open of the item Fitness Calculation, the table Fitness appears.

	string 1	real 2
string	Target value	Weighting
1	root.ProductionTime	1.00000
2		
3		

Figure 7-21 Exercise 62 Definition of fitness calculation

Finally set the optimization direction, the number of generations, the size of generation and the observations per individual as shown below.

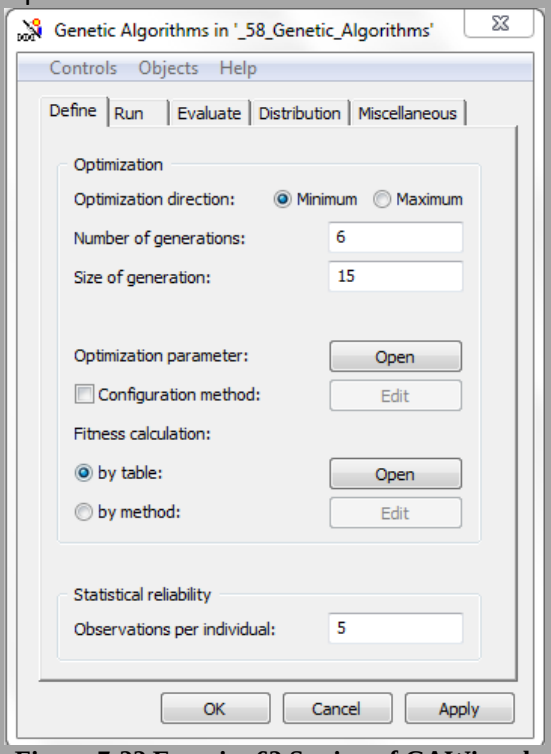

Figure 7-22 Exercise 62 Setting of GAWizard

Note: *If the evaluation is done by a simulation model containing random components (here the processing time of SingleProc1) then you must perform a set of simulations runs for each new generated individual. The resulting fitness values are called Observations. The Genetic Algorithm will use the mean value of all observations of the fitness of an individual.*

9. Run the Simulation once with the EventController and note down the value of ProductionTime. Then click the button Reset and Start one after other on the tab Run of GAWizard. After the program is finished with the optimization run, a report appears, on which the best solution is recorded. You can find the shortest production time and the best settings of delivery sequence and buffer capacity, with which the best production time can be achieved. Then find out how much production time has been saved compared with the previous simulation run.

Note: *Please note that GA is not able to determine an exact solution of an optimization problem. It is a stochastical optimization procedure, which finds a very good approximation solution based on random search in a reasonable time. After the optimization process, the Wizard transfers the input values for the best solution it found into the simulation model. For example the table file Delivery has been changed to the best sequence. To return to the initial settings of the model, you can click the button Reset on the tab Run of GAWizard.*