

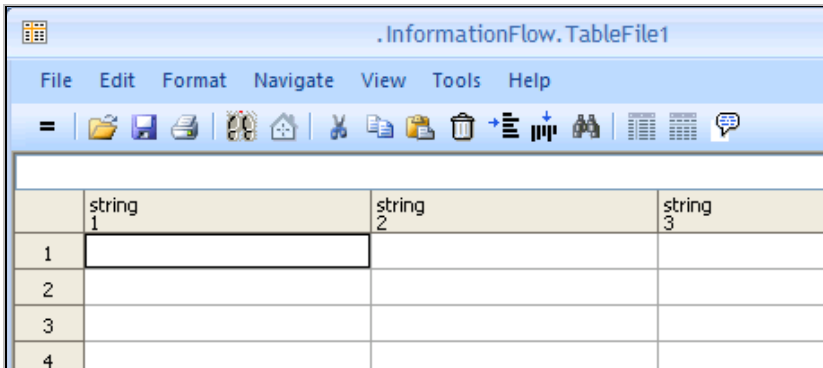
8 Information Flow Objects

The information objects are used for managing information and data. In addition to the global variable and the method, the following objects are information flow objects:

- Lists and tables
- Trigger and Generator
- AttributeExplorer
- Objects for data exchange

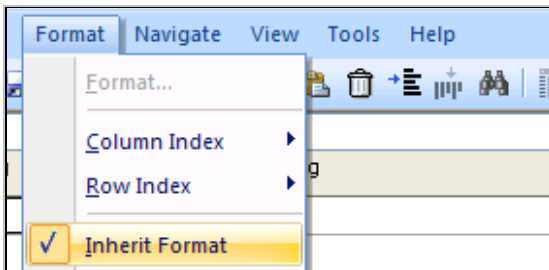
8.1 The List Editor

Enter and select settings and entries in lists and tables in the list editor. Duplicate the object TableFile in the class library, and open the duplicate by double-clicking it. Each column has a data type; each cell has a unique address.

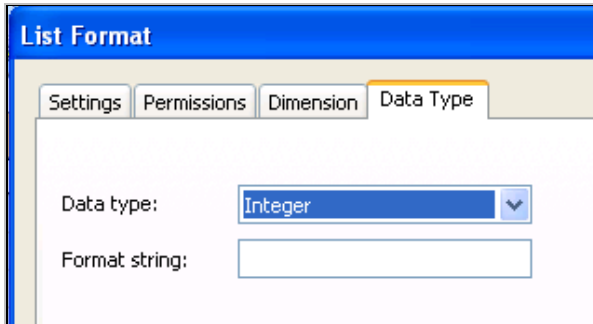


	string 1	string 2	string 3
1			
2			
3			
4			

If you want to type data into a table or list in a frame, you have to turn off inheritance: **FORMAT – INHERIT FORMAT**



For setting the data types of each column, it is best to use the context menu (right mouse button on the column header) **FORMAT ...**



Select the data type of the column here. The format string restricts interactive input (validity of entries). -9, for example, means that numbers with a maximum nine digits can be entered, negative numbers are allowed.

Permissions: On the tab permissions you can endow cells with write protection.

Column/ Row Index: You can define your own column and row indices. This is very helpful especially when you are working with tables. By default, the row and column indices are hidden.

You can view the indices by selecting **FORMAT – COLUMN INDEX – ACTIVE** or **FORMAT – ROW INDEX – ACTIVE**

Example Supply List:

	string 0	object 1	integer 2	string 3	table 4
string		MU	Number	Name	Attribute
2					
3					

The row and column index each have the index 0.

8.2 The CardFile

The CardFile is a one-dimensional list with random access to the content via an index. You can use the CardFile like an array, so that you can store and read many values under one name. This object can easily store data with the same data type. When you insert entries, Plant Simulation moves the following entries back one position. You can remove entries (with “[]”). However, you can also read entries without removing the entry from the CardFile (with the command read).

Example 85: Materials List

Insert a CardFile with the name “material” into an empty frame. Turn off inheritance. Enter the following values:

	string 1
1	wood
2	stone
3	steel
4	reed

A method now is to read line 2 (define a method in the same frame). The following commands are required to do so.

```
is
do
    print material.read(2);
end;
```

Run the method (run-run or F5). The console should display "stone". "Gold" should be inserted in line 3. For inserting entries the CardFile provides the method *insert* (position, value). Change the method as follows:

```
is
do
    material.insert(3, "gold");
end;
```

Run the method with F5!

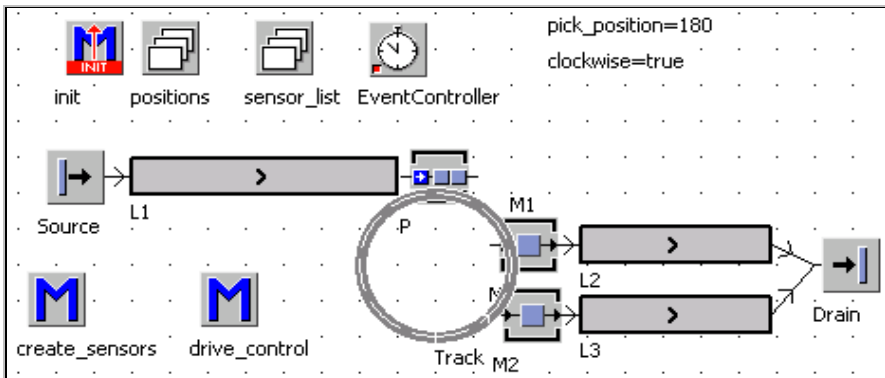
The most important attributes and methods of the CardFile are

Method/Attribute	Description
<code><path>.insert(<integer>, <value>);</code>	Inserts the value <value> at the position <integer>. Entries with the same or a higher index will be moved. In QueueFile and StackFile you only pass the value; in the QueueFile, insertion takes place at the last position, and in the StackFile at the first position
<code><path>.cutRow(<integer>);</code>	Removes the entry with the index <integer>, all other entries move up. The method returns the removed entry. For QueueFile and StackFile, you pass no index. In the QueueFile, the method deletes the first position, in the StackFile the last.
<code><path>.read(<integer>);</code>	Reads the entry with the index <integer> without removing it.
<code><path>.append(<value>);</code>	Appends the passed value to the end of the list

<path>[<integer>]	Returns the value at the position <integer> and deletes the entry
<path>.dim	Returns the number of entries
<path>.empty	Returns true if the list contains no entries
<path>.delete	Deletes the whole list

Example 86: Handling by a Robot

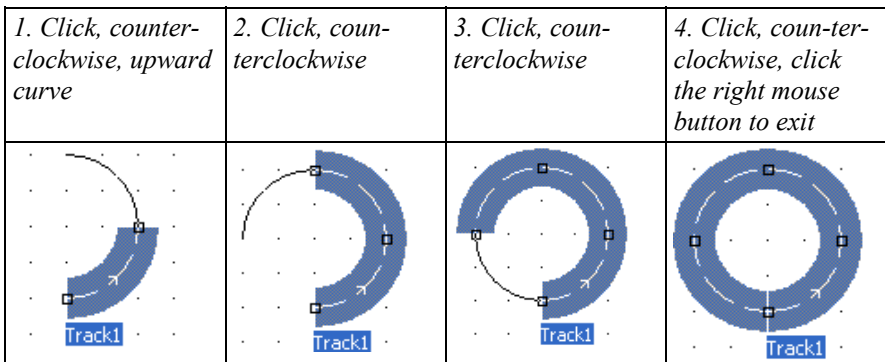
You are to simulate a robot, which loads several machines. The robot takes the parts from a buffer and loads them into two machines. The robot has a swivel range (diameter) of 4 meters. The positions of the machines are located relative to the robot at 90° and 135° clockwise. Create the following Frame:



Settings: Source interval 30 seconds, Buffer P processing time: 0 seconds, capacity one part; M1 and M2 each 55 seconds processing time; L1, L2, and L3 each 1 m/s speed.

Creating the Circular Track

For the robot to show the correct behavior, you must create the track as follows. The easiest way is to insert the track from the toolbox. First, click the track button in the toolbox. Then click in the frame. Press **Ctrl + Shift**. This activates curve mode.



You must insert a connector from the end of the track to the beginning of the track. If you need a different radius, enter these settings into the dialog **EDIT PARAMETER OF CURVE**:

Edit Parameters of Curve

Line segments

Tangential angle: ° fixed

Line length: m fixed

Arc segments

Tangential angle: ° fixed

Arc length: m fixed

Center angle: ° fixed

Radius: m fixed

If the option **FIXED** is not selected, you can set the radius of the arc segment by dragging the mouse. Then, proceed like this:

Design the robot: The robot consists of two parts the robot propper, which moves on the track (in reality Plant Simulation rotates the icon), and a gripper, which moves on the robot (forward and backward). Both are transporters. Duplicate two transporters, and rename them to robot and gripper. Select the following settings in the transporter robot: The robot has a length-oriented load bay with a length of 0.5m:

Attributes Failures Controls Battery Product Statistics Statistics User-defined

Length: m

Speed: m/s Backwards Is tractor

Acceleration Acceleration: m/s²

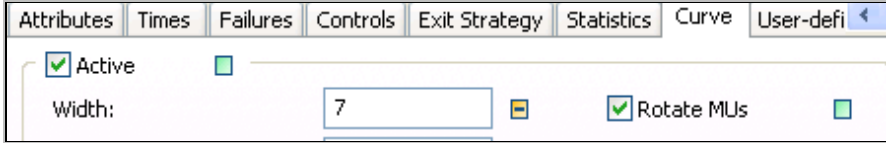
Automatic routing Deceleration: m/s²

Destination: ...

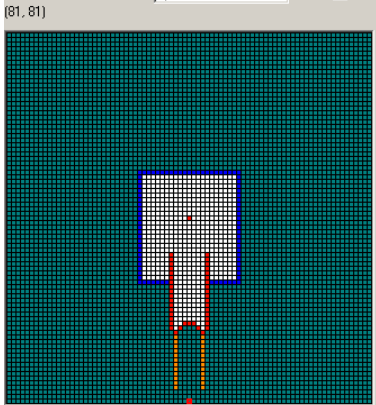
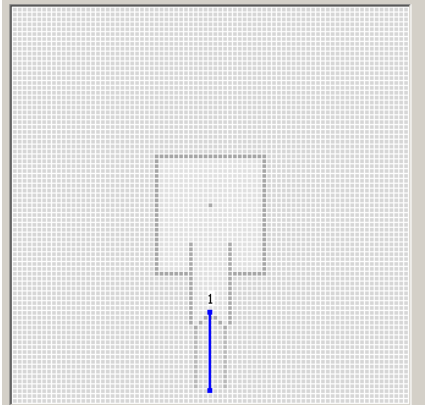
Matrix load bay

Load bay length: m Capacity:

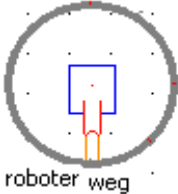
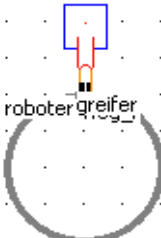
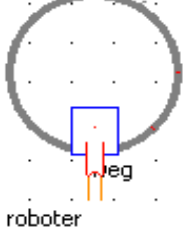
Icon operational, pause, failed, waiting: You have to create an icon that can be rotated by Plant Simulation. Therefore, you must set the reference point to the edge of the icon (in this example at the bottom, middle). The track has the setting **CURVE ROTATE MUS** on the tab curve.



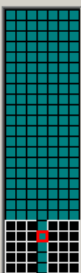
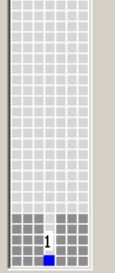
If the transporter now drives on the track (the reference point is located on the track), Plant Simulation rotates the icon to the position of the transporter on the track. This only works if you insert the track counterclockwise (see above). You must set the length of the transporter on a very small value (e.g., 1 mm), so that the movement matches the position of the transporter on the circle otherwise, e.g., the transporter triggers a control with the front and stops a few angular degrees before the actual position. Try to follow these guidelines for creating an icon for the robot (81 x 81 pixels). The robot is created at the end of the track. The end of the track is located “below”, so the icon of the robot must also point downward.

Draw mode	Animation mode
 <p data-bbox="138 1284 560 1345">The reference must be downward in the middle!</p>	 <p data-bbox="569 1284 1014 1416">The animation line must be drawn from top to bottom. Set the number of animation events to 25 (Icon Editor – Icon – Number of Animation Events).</p>

To test your robot, you can insert it by dragging it onto the track. The position should look as follows:

<p><i>Correct: (track is drawn counterclockwise), the reference point is located at the edge of the icon</i></p>	<p><i>Wrong: Path drawn clockwise</i></p>	<p><i>Wrong: The reference point is not located at the edge of the icon</i></p>
		

The gripper may be designed like this:

Draw mode	Animation mode
	

The reference point is located at the lower end of the icon. There also is an animation point for the MU which is to be transported. The orientation and size must be coordinated with the robot.

2. Create sensors on the track: The angles relative to the loading position are known. The loading position in this example is located at 180° (0° is down). The positions of the machines are 90° and 135° clockwise from the loading position. Handling robots especially calculating the sensor positions can be very time-consuming. The sensors are created by a method. The method will delete all old sensors from the track and then create new sensors from a list and attach a predetermined method (drive_control).

Working with Sensors in SimTalk

There are two ways to access the sensors on a track, a conveyor or a transporter via SimTalk:

```
<path>.sensorID(<integer>),
```

via the sensorID (if known) or

```
<path>.sensorNo (<integer>),
```

via an enumeration. You can query the number of sensors with the method

```
<path>.numSensors
```

You create new sensors using the method:

```
<path>.createSensor (<integer>, <string>, <object>,
<boolean1>, <boolean2>)
```

You must indicate the position of the sensor, the type of position (“length” or “relative”), the method to be executed (so that the method is not called by its name, use the method ref ()), and a boolean value for the front and rear control. Sensors will be destroyed using the method

```
<path>.deleteSensor (<integer>)
```

You must pass the ID of the sensor to be deleted. The easiest way to delete all sensors is to delete the first sensor repeatedly, until no sensor is left.

The sensor itself has the following attributes and methods:

<i>Attribute/ method</i>	<i>Description</i>
<code><sensor>.position</code>	returns the position of the sensor, if the position type is "relative" the returned value is a percentage value based on the length of the object, if the position type is length, the returned value is a length position on the object
<code><sensor>.front</code>	sets and gets whether the front control is enabled
<code><sensor>.rear</code>	sets and gets whether the rear control is enabled
<code><sensor>.positionType</code>	gets and sets the position type, possible values are: length or relative

Example: The frame includes two global variables:

```
pick_position=180
clockwise=true
```

Enter the following values in the table positions (positions of the sensors from the pick position):

	real 1
1	90.00
2	135.00

The method `create_sensors` is to first delete all sensors, then create the pick position as sensorID 1, then create all sensors from the list positions (at the relevant positions).

Method `create_sensors`:

```

is
  i:integer;
  id_sens:integer;
  posi:real;
  posi_further:real;
  number:integer;
do
  -- delete all sensors on the track
  number:=track.numSensors;
  for i:=1 to number loop
    id_sens:=track.sensorNo(1).id;
    track.deleteSensor(id_sens);
  next;
  -- pick_position as first sensor
  posi:=track.length/360 * pick_position;
  --insert the sensor in the track
  id_sens:=track.createSensor(posi,"Length",
    ref(drive_control),true,false);
  for i:=1 to positions.dim loop
    if clockwise then
      -- track is counterclockwise, subtract positions
      -- less than 0 --> subtract the rest from the
      --end of the track
      posi_further:=posi-positions.read(i)*
        track.length/360;
      if posi_further<0 then
        posi_further:=track.length+posi_further;
      end;
      --create the sensor
      id_sens:=track.createSensor(posi_further,
        "Length", ref(drive_control),true,false);
    end;
  next;
end;

```

3. Create sensors on a transporter, program the control of the gripper: The gripper should have (here simplified) two positions. One position is at the end of the robot for loading and unloading and the other at the beginning of the robot at which the movement of the robot (here its rotation) is triggered. The control of the gripper can then be quite simple. The robot sets the destination of the gripper and starts the forward movement of the gripper. At the end of the track, the gripper loads the loaded part onto the destination object. If the gripper is empty, then the part is loaded from the destination object. The robot controls the correct timing and the technological sequence. When the gripper has finished its work, it moves back and its rear triggers the movement of the robot as such.

Note:

We recommend to program the control for the gripper in the class library. The method for the robot has to be addressed with its absolute path; otherwise, problems with the instantiation might arise.

Create the method `gripper_control` in the class library. Then open the transporter robot (0.5 m long) and click the button `Sensors`:

Insert two sensors, and assign the method `gripper_control` to the sensors (important, as an absolute path!):

ID	Position	Front	Rear	Path
1	0m		x	.Models.gripper_control
2	0.5m	x		.Models.gripper_control

In its most basic form, the method `gripper_control` could look like this:

```
(sensorID:integer)
is
  target:object;
do
  if sensorID=2 then
  -- first front
  @.stopped:=true;
  if @.numMU >0 then
    target:=@.destination;
    waituntil target.empty and target.operational
    prio 1;
    @.cont.move(target);
    @.backwards:=true;
    -- pause ?
    @.stopped:=false;
```

```

else
  -- empty
  -- wait for parts
  target:=@.destination;
  waituntil target.occupied prio 1;
  target.cont.move(@);
  -- start gripper
  @.backwards:=true;
  @.stopped:=false;
end;
elseif sensorID = 1 then
  -- turn the robot
  @.stopped:=true;
  ?.stopped:=false;
end;
end;

```

This gripper control also works very well for controlling grippers in machine portals.

4. Insert the robot and the gripper on the track: The control of the robot on the track is to be triggered by SensorIDs. Add the user-defined attribute *targetSensorID* (integer) to the robot in the class library, and set the start value to 1:

Name	Value	Type	C
targetSensorID	1	integer	*

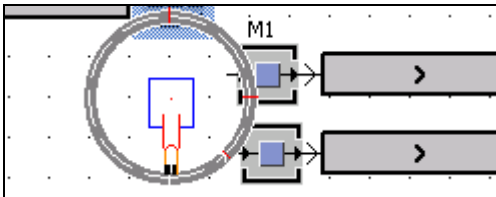
The robot first moves to the pick position and waits there until the first part arrives. Robot and gripper are created in the Init method:

```

is
do
  deleteMovables;
  .MUs.robot.create(track);
  .MUs.gripper.create(track.cont,0.4);
  -- stop the gripper
  track.cont.cont.stopped:=true;
end;

```

If everything worked as intended, the robot should be located on its track and the gripper on the robot:



5. Program the method `drive_control`: When the robot arrives at its target sensor, it stops. The robot waits at sensor 1 until the place `p` is occupied. Then the robot sets the destination of the gripper to `P` and starts the gripper (forward). Thereafter, the robot sets the next `targetSensorID` and the correct value for the moving backwards. The machine should be identified via the `SensorID`. This is most easily accomplished with a Cardfile (`sensor_list`). Enter the following data into the `sensor_list` (data type object):

	object 1	
1	p	
2	M1	
3	M2	

Method `drive_control`:

```
(sensorID:integer)
is
  gripper:object;
  target:object;
do
  gripper:=?.cont.cont;
  if @.targetSensorID=sensorID then
    @.stopped:=true;
    gripper.destination:=sensor_list.read(sensorID);
    -- next target ??
    if gripper.destination= P then
      waituntil (M1.empty or M2.empty) and
      P.occupied prio 1;
      gripper.backwards:=false;-- drive gripper
      gripper.stopped:=false;
      -- to M1 or M2
      if M1.empty then
        @.targetSensorId:=2;
      elseif M2.empty then
        @.targetSensorId:=3;
      end;
      @.backwards:=true;
    else
      gripper.backwards:=false;
      gripper.stopped:=false;
      @.targetSensorID:=1;-- to pick position
      @.backwards:=false;
    end;
  end;
end;
```

The robot now loads the two machines.

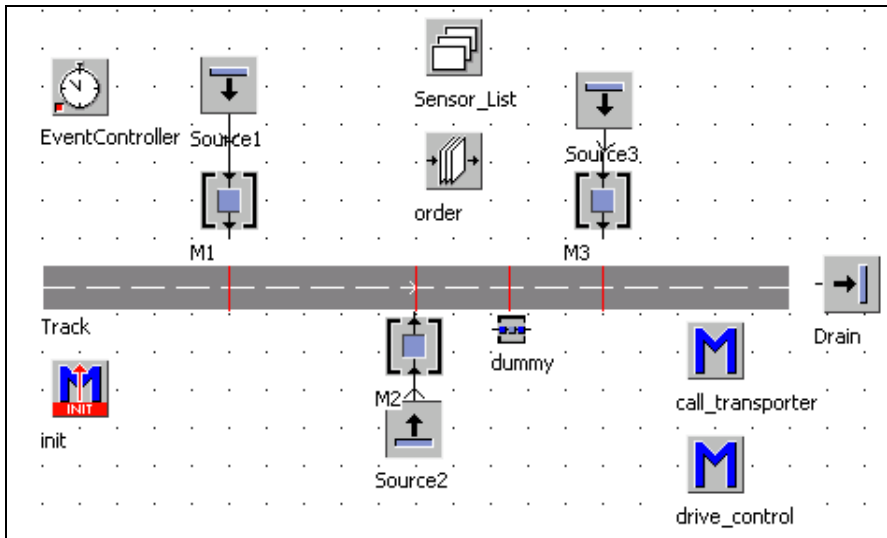
8.3 StackFile and QueueFile

The StackFile and QueueFile are one-dimensional lists which are accessed according to the FIFO (First in first out, queue) or LIFO (Last in first out, stack) principle. New entries will be inserted into the StackFile at the top of the list; the last element inserted is the first entry which will be removed. New entries will be added to the QueueFile at the bottom of the list, the first element will be removed. The main methods for working with stacks and queues are push (element) and pop. With delete, the entire content of a list will be deleted.

Example 87: Queuing

A transporter unloads three machines. It waits in its waiting position (here, 12.5 m), until a driving order arrives. The transporter then drives to the machine and unloads the part. The transporter drives with the part to the end of the track and unloads the part there.

Create the following Frame:



Sensors on the track (length 20 meters):

ID	Position	Front	Rear	Path
1	5m	x		drive_control
2	10m	x		drive_control
3	12.5m	x		drive_control
4	15m	x		drive_control

Settings: Source1, Source2, and Source3 non-blocking, interval 1 minute; M1, M2, and M3 1 minute processing time, availability 50%, 2 minutes MTTR, select

different random number streams for the different machines; transporter speed 1 m/s, capacity one part; drain 0 seconds processing time. The transporter has a user-defined attribute `targetSensorID` (integer). The start value is 3.

1. The `init`-method creates the transporter on the track. The transporter is to stop at the `sensorID` 3.

The method `init` looks as follows:

```
is
do
    .MUs.Transporter.create(track, 1) ;
end;
```

The transporter should always stop when the `SensorID` of the track matches the `targetSensorID` of the transporter.

Program the method `drive_control`:

```
(sensorID : integer)
is
do
    if sensorID = @.targetSensorID then
        @.stopped:=true;
    end;
end;
```

2. The machines call the transporter after having processed the parts. In this example, the machine should enter its `sensorID` into a `QueueFile`. The assignment of machines to the sensors is entered into a `Cardfile` (`sensor_list`). If a sensor is not to be used, enter the object `dummy` instead (as a wildcard). You cannot leave empty rows in a `Cardfile`. Also add a dummy object to the frame; otherwise, you will receive an error message. In our example, the sensor list looks as follows:

	object 1
1	M1
2	M2
3	dummy
4	M3

If a part is finished on a machine, it triggers the exit sensor. The machine then has to search the `SensorID` in the `sensor_list` and enter the `SensorID` into the `QueueFile` order.

Searching in Lists

Searching in lists in *Plant Simulation* works as follows. *Plant Simulation* uses an internal cursor for searching. This cursor is set to a hit. With help of the cursor, you can determine the position within the list. At the end of the search, the cursor keeps its old position where it found the entry. Therefore, it is necessary to first set the cursor to the position 1.

Syntax:

```
<path>.setCursor(1);
```

Then you can use the method `<list>.find(value)` for searching a value. The method `find` returns `true`, if the value was found, and `false`, if the value was not found. If the list includes the queried value, the cursor is set to the corresponding position. In a third step, you need to read the cursor position:

```
position:= <list>.cursor.
```

Example: Program the method call `_transporter` as the exit control of the machines `M1`, `M2`, and `M3`. Select the data type integer for the `QueueFile` (order). The method call `_transporter` could look as follows:

```
is
  sensorID:integer;
do
  -- search sensorID
  -- set the position of the cursor to the beginning
  sensor_list.setCursor(1);
  -- search for the machine
  sensor_list.find(?);
  -- position the cursor
  sensorID:=sensor_list.cursor;
  -- insert into order
  order.push(sensorID);
end;
```

3. The transporter is waiting at the waiting position until an order arrives. Then it drives to the machine. Using the attribute `dim`, you can determine the number of entries in a list. This attribute is observable; it can be monitored with an observer or a `Waituntil` statement. The direction can be determined using the sensor IDs. In this case, this is easy because the sensors are not in a mixed order. If you insert a new sensor afterwards (e.g., for a new machine), the sensor IDs get mixed up, which means that a greater sensor IDs do not necessarily mean a greater amount of length of the position.

Sensor Position, Sensor ID, Direction

The direction of a transporter is to be identified. These arguments have to be passed:

- *track*
- *current sensorID*
- *target sensorID*

The return value of the function is of type *boolean*. Create the method (*getDirection*) in the current example. Using `<track>.SensorID(id)` you can access all information that is associated with a sensor. The method *SensorID* returns an object of type *sensor*. The attribute *position* returns the position of the sensor.

Example 88: Determining Sensor Positions

Program the method *getDirection*:

```
(track:object;sensorFrom:integer;sensorTo:integer)
:boolean
is
  posFrom:real;
  posTo:real;
  backwards:boolean;
do
  posFrom:=track.sensorID(sensorFrom).position;
  posTo:=track.sensorID(sensorTo).position;
  backwards:=(posFrom>posTo);
  return backwards;
end;
```

Extend the method *drive_control* as follows.

```
(sensorID : integer)
is
do
  if sensorID = @.targetSensorID then
    @.stopped:=true;
    if sensorID= 3 then
      waituntil order.dim > 0 prio 1;
      @.targetSensorID:=order.pop;
      @.backwards:= getDirection(?,3,
        @.targetSensorID);
      @.stopped:=false;
    end;
  end;
end;
end;
```


4. The Transporter gets an order, drives off, and stops in front of a machine. The transporter has to load the part from the machine and drive forward to the sink. The transporter does not yet know the machine; the method must read the machine from the sensor list. This is accomplished with the method `read (id)`.

Program the method `drive_control`; in addition to example above

```
(sensorID : integer)
is
do
  if sensorID = @.targetSensorID then
    @.stopped:=true;
    if sensorID= 3 then
      waituntil order.dim > 0 prio 1;
      @.targetSensorID:=order.pop;
      @.backwards:=getDirection(?,3,
        @.targetSensorID);
      @.stopped:=false;
    else
      sensor_list.read(sensorID).cont.move(@);
      @.backwards:=false;
      @.stopped:=false;
    end;
  end;
end;
```

5. Program the method `unload` for unloading the part onto the drain, and assign it as the exit control of the track. The transporter must unload the part to the drain. Then the transporter drives forward. If an order exists, it must be read and the `targetSensorID` of the transporter must be set anew. If no order exists, then the `targetSensorID` is 3.

Program the method `unload` (exit control track)

```
is
do
  @.stopped:=true;
  @.cont.move(drain);
  if order.dim > 0 then
    -- to machine
    @.targetSensorID:=order.pop;
  else
    -- to waiting position
    @.targetSensorID:=3;
  end;
  @.backwards:=true;
  @.stopped:=false;
end;
```

8.4 The TableFile

8.4.1 Basic Behavior

The TableFile is a two-dimensional list, which allows random access to the entries via their address. TableFiles have many fields of application in simulation projects, e.g.:

- Storage of work plans and production orders
- Collection of statistical information
- Parameterization of models

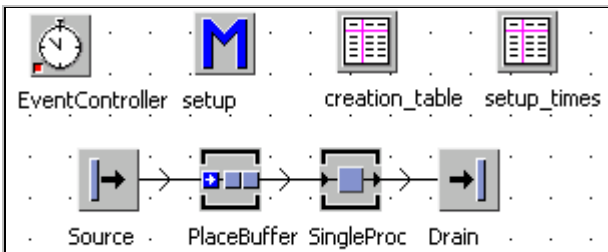
Note:

For the following example: For the distribution of setup times, depending on the actual part and the new part, you can define the setup time in a matrix.

Example 89: Lot Change

A milling center successively processes different production orders. The parts call for different setup times. Prior to the part moving onto the machine, the setup is to be set anew if necessary. The necessary information is to be centrally stored in a TableFile.

Create the following Frame:



Settings: Source interval 1:30 minutes, blocking; create three entities p1,p2,p3. The source creates the parts in a cycle. Use the TableFile creation_table for the distribution of the parts:

Interval:	Const	1:30	<input type="checkbox"/>
Start:	Const	0	<input type="checkbox"/>
Stop:	Const	0	<input type="checkbox"/>
MU selection:	Sequence Cyclical	<input type="checkbox"/>	<input type="checkbox"/> Generate as batch
Table:	creation_table	<input type="checkbox"/>	<input checked="" type="checkbox"/> Format table

Type in the following lot sizes into the TableFile creation_table:

	object 1	integer 2	string 3
string	MU	Number	Name
1	.MUs.P1	5	
2	.MUs.P2	10	
3	.MUs.P3	5	

PlaceBuffer: capacity 100 parts, no processing time; the TableFile setup_time contains the following information:

	string 0	time 1	time 2
string		setup_time	dismantling_time
1	P1	10:00.0000	5:00.0000
2	P2	13:00.0000	3:00.0000
3	P3	20:00.0000	5:00.0000

The setup time consists of the dismantling time of the old part (already located on the machine) and the setup time for the new part. Setting up from P1 to P2 might take, for example, 5+13 minutes = 18 minutes. The setup time must be assigned to the workstation before the machine starts to set up (automatically every time a MU with a different name arrives). For this reason, we have to make the following considerations: It has to be checked for which part (MU) the machine is equipped; if the machine is set up for the same part, then no action is required; if the machine is set up for another part, the set-up time is set anew and the machine starts setting-up after moving the part to the machine.

Attributes and methods for setting up a machine

Method	Description
<path>.setUp	determines whether a object is currently setting-up(true if the object sets up).
<path>setUpFor(<mu>)	triggers setting an object up for a certain MU class. The time needed depends on the value of the setup time for the object.
<path>.isSetUpFor	returns the name of the MU (string), for which the object is set up. If the object is not set up for a specific MU, the method returns an empty string.
<path>.setupTime	sets/gets the set up time of the object

<code><path>.automaticSetUp</code>	sets/gets whether the set up process is triggered automatically when another MU class arrives at the object
--	---

Program the method set up (as the exit control front for the PlaceBuffer):

```
is
  former : string;
do
  -- read the set-up time from the table
  -- set the attribute setUpTime
  former:=singleProc.isSetUpFor;
  if former="" then
  --the first part only set-up
  singleProc.setUpTime:=
  setup_times["setup_time",@.name];
  else
  -- former part dismantling_time
  -- recent part setup_time
  singleProc.setUpTime:=setup_times["setup_time",
  @.name] + setup_times["dismantling_time",former];
  end;
  @.move;
end;
```

8.4.2 Methods and Attributes of the TableFile

Delete

Syntax:

```
<path>.delete or <path>.delete(<range>)
```

This method deletes all entries or the specified range in the table/list. The following notation applies to ranges in TableFiles:

- One cell: table[column,row]
- Range: {column1, row1}...{column2, row2}

The range specification consists of two direct specifications, which are separated by two periods. The first specification defines the upper left corner, while the second determines the lower right corner of the range. All entries in the rectangular area will be evaluated. When you enter {*,*} as a second indication, the table evaluates to the largest valid column and row index.

Samples:

```
{2,2}...{3,3}
{"front","door"}...{"rear","door"}
{3,1}...{*,*}
```

Notation	Range
$\{1,2\} \dots \{3,5\}$	from column 1 to column 3 and row 2 to row 5
$\{1,*\} \dots \{4,*\}$	all rows in column 1 to column 4
$\{2,3\} \dots \{*,3\}$	in row 3 all columns starting with column 2
$\{2,3\} \dots \{*,*\}$	all the columns from column 2 and all rows from row 3
$\{*,*\} \dots \{3,5\}$	all columns to column 3, and all lines to line 5

Additional Methods of the TableFile are

Method	Description
<code><path>.copy</code> <code><path>.copy(<range>)</code>	Copies the contents of the cells to the clipboard
<code><path>.initialize</code> <code>(<ranges>, <values>)</code>	Preallocates the specified areas with the passed values, existing contents will be overwritten
<code><path>[<column>, <row>]</code> <code><path>[<column>, <row>] := <value>;</code>	Read/write access; the TableFile allows random access via column and row indices. The index starts and ends with a square bracket. Within the brackets, you first enter the column and then the row of the cell you want to access. If you assign a value, the data type of the value must match the data type of the cell.
<code><path>.yDim</code>	Returns the number of entries (lines).
<code><path>.xDim</code>	Returns the number of columns (which contain values)
<code><path>.dim</code>	returns the product of columns and rows
<code><path>.find(<range>, <value>)</code>	Sets the cursor into the cell, which contains the value. You can determine the coordinates with the cursor (<code>table.cursorX</code> and <code>table.cursorY</code>). Before searching make sure that the cursor is located in the correct position. For this you can use <code><path>.setCursor(<column>, <row>)</code> .

<code><path>.insertRow(<value>)</code>	Adds a new empty row at the position <code><value></code>
<code><path>.writeRow(<position>, <value1>, <value2> ...)</code>	Replaces all entries in the row at the specified position by the passed arguments

In addition, the TableFile provides methods for inserting and removing columns and rows. You find more information about this in the help and under the heading statistics.

8.4.3 Calculating within Tables

Example 90: Calculating Machine-Hour Rates

Basics: Calculating the machine-hour rate allows a more accurate allocation of common costs and thus a more accurate calculation. The goal is the apportionment of the machine-related indirect production costs to one hour of machine running time. Calculating the machine-hour rate consists of the following components (sample):

Machine-dependent indirect production costs	Fixed amount per month	Variable costs per hour
1. Imputed depreciation ³	4.500	
2. Imputed interest ⁴	900	
3. Imputed rent ⁵	500	
4. Energy costs per hour		0.25
5. Tooling costs		10.00
6. Repair/Maintenance		10.00
7. Fuel costs		2.50
Indirect production cost per hour	5900/number of hours machine running time	22.75
Machine-hour rate	5900/number of hours per month + 22.75	

³ Replacement value/asset depreciation range in months.

⁴ Cost value/2 * imputed rate/100.

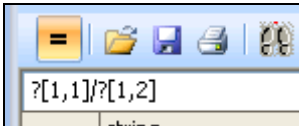
⁵ Footprint of the machine in m² * imputed rent per month.

We calculate the machine-hour rate in a TableFile. First, create a sample of the machine-hour rate calculation in the class library (the calculation scheme is the same for all machines, only the values will change). Create the following table:

	string 0	real 1
string		
1	Replacement value	2000000.00
2	asset depreciation range in months	120.00
3	Imputed depreciation	

Calculations in Tables

You can enter values directly into the table cells. Alternatively, you can specify formulas, which calculate the values in the table cells. Tables and lists therefore have two modes: In formula mode, you can enter formulas; in input mode, you can enter values and the values of the formulas are displayed. You can switch to the formula mode with the formula button (to the left of Open):



Calculated fields are shown with a light blue background. A formula has the following basic structure:

PathTableFile[c,r] operator PathTableFile[c,r]

This is a bit cumbersome in relation to the same table, so in calculations within a table you can use the anonymous identifier „?” as a substitute for the path.

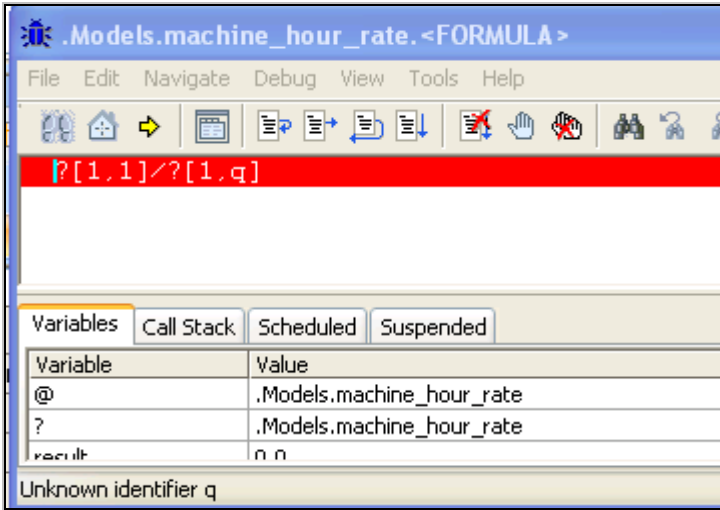
Sample:

The imputed depreciation cost is calculated as follows:

Replacement value/asset depreciation range in months. As formula in the table in the example above you would enter:

?[1,1]/?[1,2]

If your formula is wrong, Plant Simulation shows an error message.



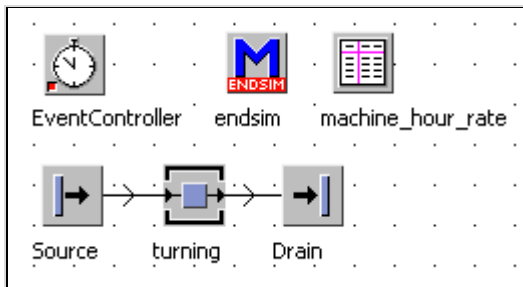
It is important to consider the data types in calculations. The result cell has a certain data type (determined by the data type of the column in the table). The result must also have this data type; otherwise, Plant Simulation will show an error message. A reasonable simplification is the calculation in a single data type (e.g., real) and formatting of the output (e.g., money). The calculation of the machine-hour rate results in the following table (and the associated formulas):

	0	1
1	Replacement value	2000000.00
2	Asset depreciation range in months	120.00
3	Imputed depreciation	?[1,1]/?[1,2]
4	Cost value	1500000.00
5	Imputed rate (%)	7.00
6	Imputed rent	?[1,4]*?[1,5]/200
7	Footprint of the machine in m ²	20.00
8	Imputed rent per month and m ²	13.00
9	Imputed rent per month	?[1,7]*?[1,8]
10	Total fixed costs per month	?[1,3]+?[1,6]+?[1,9]
11		
12	Energy costs per hour	0.25
13	Tooling costs per hour	10.00

14	Repair/Maintenance per hour	10.00
15	Fuel costs per hour	2.50
16	Total indirect production costs per hour	?[1,12]+?[1,13]+?[1,14]+?[1,15]
17		
18	Monthly machine running time in hours	1.00
19		
20	Machine-hour rate	?[1,10]/?[1,18]+?[1,16]

The cell [1,18] must be calculated in the simulation. The result of the simulation is the actual occupancy of the machine and with the calculation in the table the machine-hour rate (taking account of breaks, occupancy, maintenance, and whatever else you take into account in the simulation).

Create the following Frame:



At the end of the simulation, the method `EndSim` reads (set a month as the end of the simulation in the event controller) the working time of the object `turning` from the statistics data and writes it to the table `machine-hour rate` (cell [1,18]). Then, the table calculates the machine-hour rate.

Program the method `endSim`:

```
is
do
  -- set the machine working time in the table
  machine_hour_rate[1,18] :=
    turning.statWorkingTime/3600;
end;
```

Note:

The values in the table retain their values between the simulation runs. At the beginning of the simulation, you therefore need to initialize all required values (in the example above, the cell [1,8]).

8.5 The TimeSequence

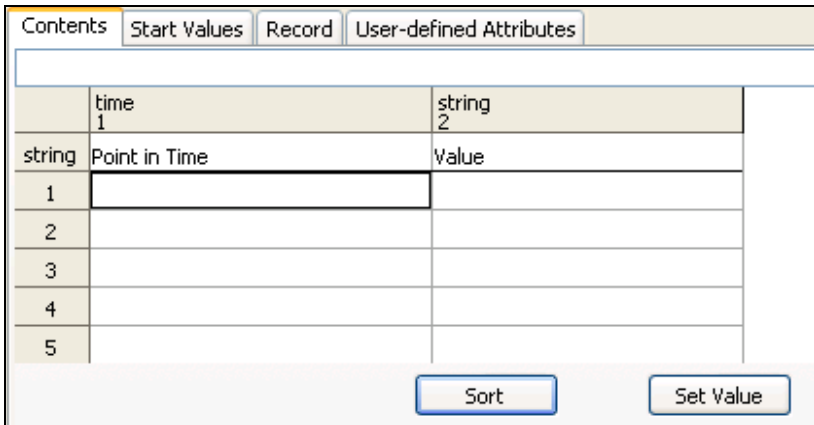
8.5.1 Basic Behavior

You can use the TimeSequence for recording and managing temporary value progressions (stocks, machine output...). The TimeSequence has two columns: Point in time (1st column) and value (2nd column). You can enter values into the TimeSequence with SimTalk, or the TimeSequence can record values by itself.

8.5.2 Settings

Tab Content

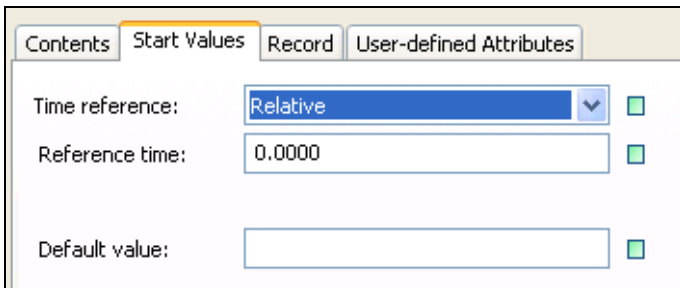
The tab Contents shows the recorded values. You can sort the values in ascending order according to time. The button Set sets empty fields to a default value.



On this tab, you must specify the data types to be stored. This is analogous to the tables:

1. Turn off inheritance (**FORMAT – INHERIT FORMAT**).
2. Then click the right mouse button on the column header of the second column, select Format from the menu.
3. Select the data type, and click OK.

Tab Start Values



Time reference: You can specify whether Plant Simulation shows time-related data in absolute format (datetime) or in relative format (time).

Reference time: Enter the start of the recording of the values (time, date). The time values are shown relative to this reference value (which shifts the values of the time axis).

Tab Record

The screenshot shows the 'Record' tab of the TimeSequence dialog. It has four tabs: 'Contents', 'Start Values', 'Record', and 'User-defined Attributes'. The 'Record' tab is active. It contains the following fields and controls:

- Value:** A text input field containing the path `.Models.Frame.Buffer.NumMU`. To its right is a button with three dots and a small square icon.
- Mode:** A dropdown menu currently showing 'Sample' with a downward arrow and a small square icon.
- Interval:** A text input field containing '1:00.0000' and a small square icon.
- Active:** A checked checkbox.

Here, you can select the settings that are required for collecting the data.

Value: Enter the relative or absolute path to the value (method, variable, attribute), whose course over time the TimeSequence will record. You might, for example, record the number of parts in the object buffer. The method is `buffer.numMU`. You can select the value using the button next to the input field methods, attributes, and variables.

This screenshot shows the 'Record' tab of the TimeSequence dialog with a 'Select Object' dialog box open. The 'Select Object' dialog has a title bar with a question mark and a close button. It contains a list of 'Attributes and methods':

- NumChildren
- NumMU (highlighted)
- NumMUParts
- NumPartsSinceSetup

Buttons for 'Back' and 'OK' are visible on the right side of the dialog.

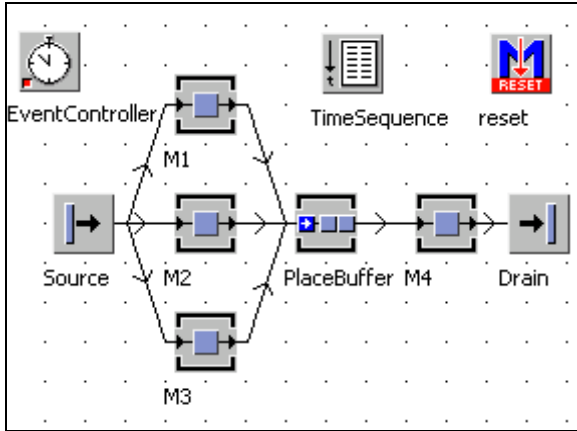
Mode: Watch means that values are entered after each change in value. This may possibly lead to a slowdown of your simulation. Sample means that values at certain time intervals are entered (e.g., every 30 minutes). In watch mode, only observable values will be recorded.

Active: Use this to activate or deactivate the TimeSequence.

Example 91: TimeSequence

A process is to be balanced. Three machines supply a fourth machine with parts. The machines M1, M2, M3 have very low availabilities (time-consuming tool testing and adjustments). We are looking for the maximum output of the line, the processing time of M4 and the required buffer size.

Create the following Frame:

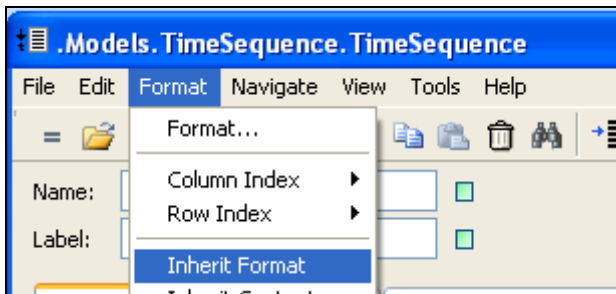


Settings: Source interval 50 seconds, blocking; M1,M2,M3 processing time 1 minute, 50% availability, 45 minutes MTTR; PlaceBuffer capacity 10,000 parts, 0 second processing time; M4 40 seconds processing time, 75% availability, 25 minutes MTTR.

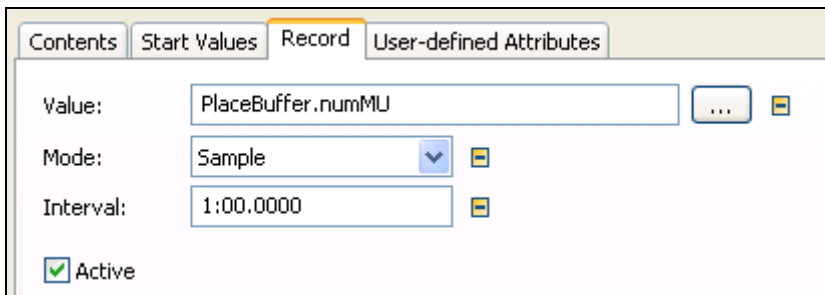
The course of stock in the PlaceBuffer is to be recorded in the TimeSequence.

Follow these steps:

1. Turn off inheritance: Format – Inherit Format (remove the check mark).



2. Click the tab Record, and select the following settings:



3. Start the simulation. The current time of the EventController and the stock in the PlaceBuffer will be entered into the TimeSequence every minute.

Contents		
	Start Values	Record
0.0000		
	time 1	integer 2
string	Point in Time	Value
131	2:10:00.0000	20
132	2:11:00.0000	22

You can easily export the values of the TimeSequence (e.g., as a text file). First select the format of the text file:

FILE – FORMAT ...

The 'Text File' dialog box has a blue title bar with a question mark and a close button. It contains the following settings:

- Format: (empty)
- Column separator: ;
- Line separator: LF
- Decimal separator: ,
- Time format: S

Save the table with: **FILE – SAVE AS TEXT ...**

Note:

The EventController does not reset the TimeSequence. You must delete the previous content of the TimeSequence inside a reset or init method.

Example of a reset method:

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

The methods and attributes of TimeSequence are those of the TableFile.

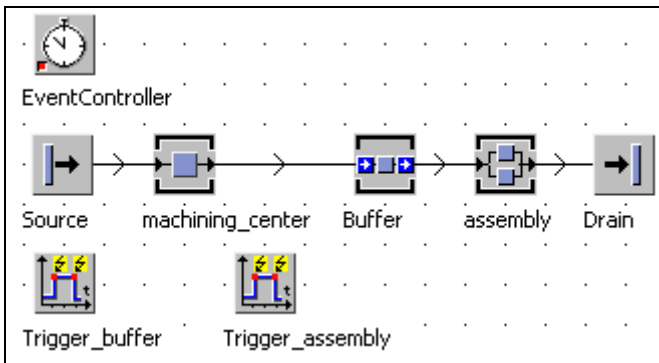
8.6 The Trigger

8.6.1 Basic Behavior

The trigger can change values of attributes and global variables during the simulation according to a defined pattern and perform method calls. In addition, the trigger can control a source, so that this starts to produce MUs from a certain moment in time on.

Example 92: Trigger

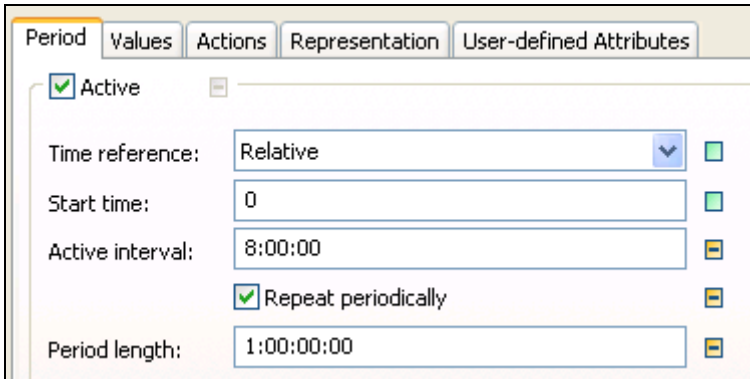
A machining center produces parts in three shifts (24 h) with a processing time of 1 minute. The following assembly produces one shift with three parallel places, and two shifts with one place. The assembly time is 1:40 minutes. The parts not yet assembled are collected in a buffer. Create the following Frame:



Settings: Source 1 minute interval, blocking; machining_center 1 minute processing time; buffer 0 second processing time, capacity 1000 parts.

After 8 h simulation time, the property assembly.XDim: = 1 (after another 16 hours according to 3 again) must be set. Select these settings in the object Trigger_assembly.

Tab Period



Active: Select whether the trigger is active or not during the simulation run.

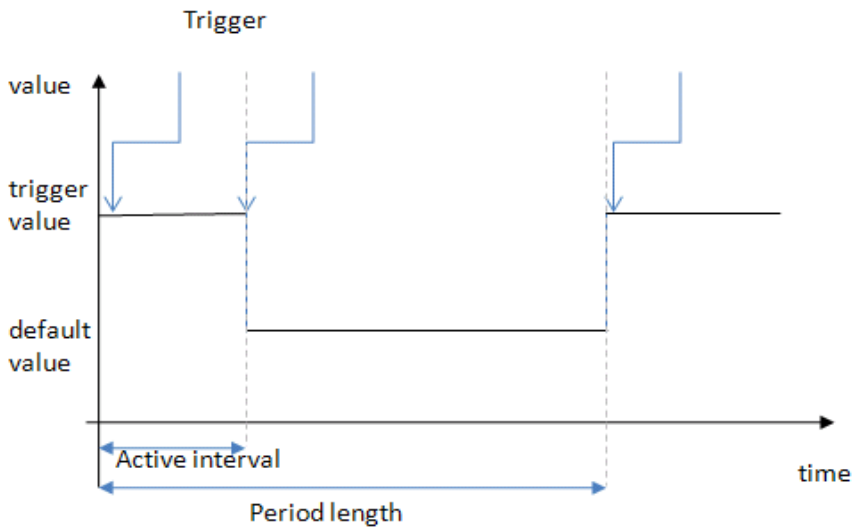
Time reference: You can select a relative start time (0:00) or an absolute time (date).

Start time: When should be the trigger for the first time active?

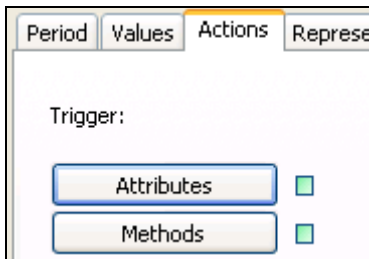
Active interval: After what period should be set the value back to the default value (defined in a time line, e.g., 8 hours)?

Repeat periodically: The trigger is active again after the expiration of the period length.

Period length: Sets the duration of a trigger period (e.g., one day or 1:00:00:00.0000).



Tab Actions



You can start methods or attributes.

Button Attributes: Type into the list which attributes you want to control. An error message appears in the console when Plant Simulation could not execute the

action. Before you can type values into the table, you must turn off inheritance and click Apply.

.Models.Trigger.Assembly			
	Object	Attribute	Error Message
1	.Models.Trigger.Assembly	xDim	XDim attribute could not be set!

Tab Values

Period Values Actions Representation User-defin

Trigger type: Input

Values

Enter the progress of the value, which the trigger controls, into a TimeSequence.

Button Values:

Before you can type in values, you first have to turn off inheritance in the value table (as table).

Enter the following values in the table:

Contents Start Values

	time	integer
	1	2
string	Point in Time	Value
1	0.0000	3
2	1:00:00:00.0000	3
3		

Set the default value (here 1) on the tab Start values.

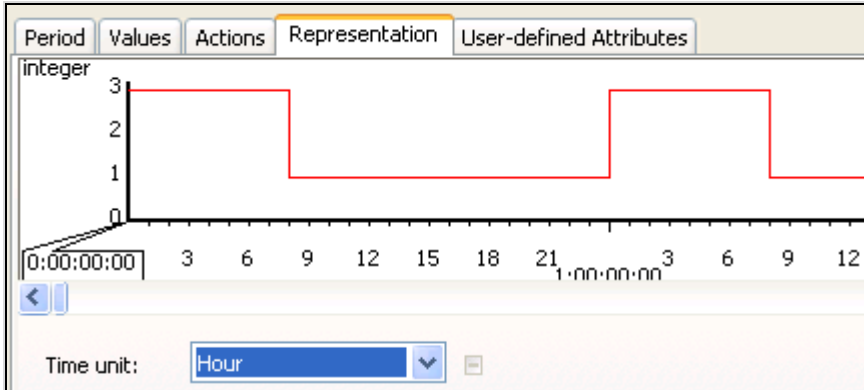
Contents Start Values

Time reference: Relative

Reference time: 0.0000

Default value: 1

You can check the distribution you set on the tab Representation:



If you run the simulation for a while, you get an error message. Plant simulation cannot reduce the dimension of the parallel station, when parts are located on the respective places. Prior to the reduction of the capacity, the object assembly needs to be emptied. This can, for example, be achieved by temporarily locking the exit of the buffer (e.g., 2 minutes before shift change). The following settings are needed in the object Trigger buffer: Actions: Attribute `buffer.exitLocked`, start time: 7:58:00, active interval 2:00, period length one day, repeat periodically, data type Boolean, values 7:58:00 true, default value false.

8.7 The ShiftCalendar

You can use triggers to set the attribute `Pause` at certain intervals to true or false to model a shift system. It is easier to accomplish this with the object `ShiftCalendar`. Every material flow object, which “deals with” entities has the following times:

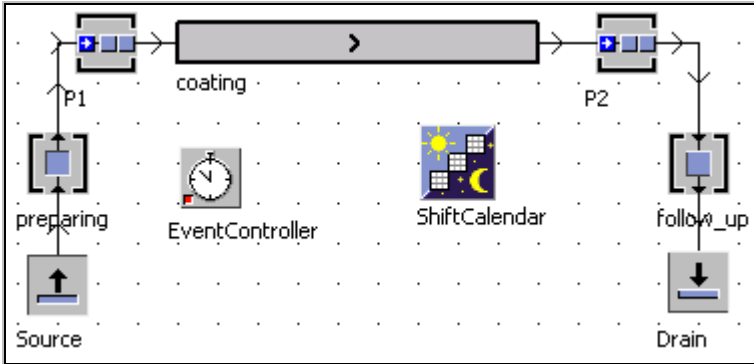
- Planned (working within the shifts)
- Unplanned (times outside the shifts, e.g., weekend)
- Paused (pause within the shifts)

The `ShiftCalendar` sets these times using a `TimeSequence`. You can use one `ShiftCalendar` for the entire simulation, or, in extreme cases, create its own `ShiftCalendar` for each machine.

Example 93: ShiftCalendar

You are to simulate a continuous process (coating), which has a workplace to prepare and a workplace for follow-up jobs. A coating process takes 8 hours (the facility is 75 m long), the preparing and follow-up job each take 2:30 min. The coating facility works 24 hours a day, 7 days a week. The preparing and follow-up workplaces work according to the following shift system: Beginning of the first shift,

Monday 6.00 clock, end of the last shift: Saturday 6:00 clock. Morning shift start at 6 clock, 14 clock end, break 9:00 to 9:15 clock, 12:00 to 12:30. Middle shift: start 14 clock, 22 clock end, Break 17.00 to 17.15 and 20:00 to 20:30, Night shift: Start 22.00 until 6 clock; breaks analogous to the middle shift. What is the maximum output? Create the following Frame:



Settings: Length of the entities: 0.2 meters, P1, P2 processing time 0 seconds, capacity 10,000 parts each.

Insert a ShiftCalendar object into the frame. First switch off inheritance on the tab shift times (click on the green icon on the right side + Apply). Then enter the shift times into the table.

Shift Times		Calendar	Resources	User-defined Attributes							
	Shift	From	To	Mo	Tu	We	Th	Fr	S	S	Pauses
1	Shift-1	6:00	14:00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	9:00-9:15; 12:00-12:30
2	Shift-2	14:00	22:00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	17:00-17:15; 20:00-20:30
3	Shift-3	22:00	06:00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	01:00-01:15; 04:00-04:30

Assign the ShiftCalendar to the objects on the tab Controls – Shift calendar.

Shift calendar:

The tab Resources of the ShiftCalendar provides an overview over the stations, which use the ShiftCalendar.

Shift Times	Calendar	Resources
Objects		
1		preparing
2		follow_up

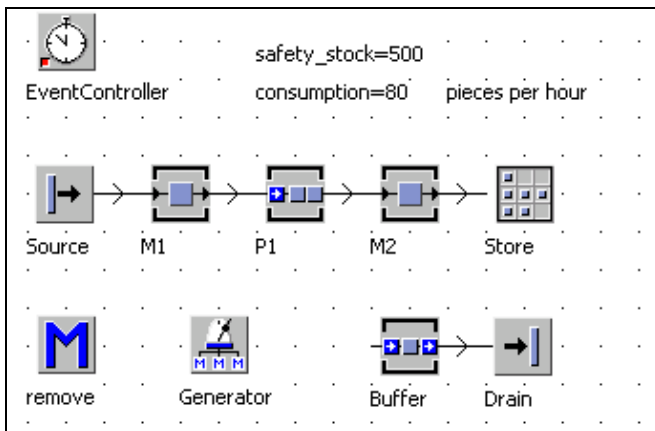
8.8 The Generator

The generator starts a method at regular intervals or after a certain time has passed. You can specify all times as a fixed time or as a statistical distribution.

Example 94: Generator, Outward Stock Movement

In the following frame, the produced parts will be not removed by a drain, they will be placed into a store (capacity 10,000). So that the store does not overflow after a short time, we need to simulate outward stock movement. The store will have an average outward stock movement of 80 units per hour. For this purpose, you need a method, which removes 80 parts per hour from the store.

Create the following Frame:



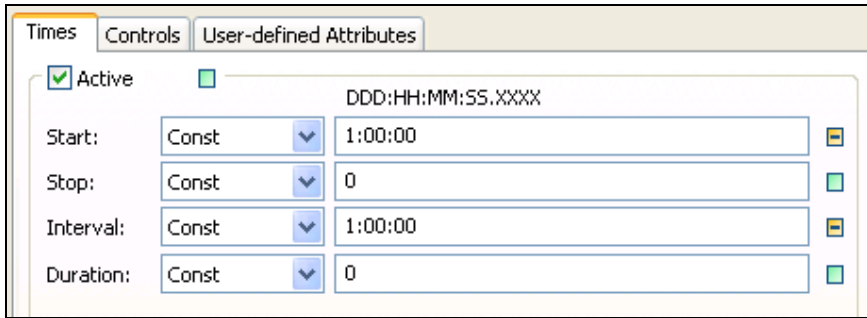
Settings: Source interval 1 minute blocking, M1 and M2 processing time 50 seconds, availability 95%, MTTR 5 hours, M1 capacity 1000 parts, store capacity 10,000 parts. Create `safety_stock` and outward stock movement (consumption) as global variables, of data type integer, in the frame

Program the method `remove` (called once per hour):

```
is
  i:integer;
do
  if store.numMU >= (consumption+safety_stock)
    then
      --remove MUs
      for i:=1 to consumption loop
        store.cont.move(buffer);
      next;
    end;
  end;
```

In the example above, the method must be called every hour. With the generator you must determine the time and the method, which should be called.

Tab Times



Active: Activate the generator.

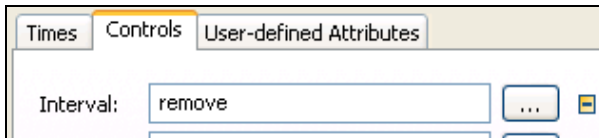
Start: Select when the interval control will be activated for the first time.

Stop: Select at which simulation time no interval control should be active.

Interval: What time should elapse between calls?

Tab Controls

Select your method on this tab:

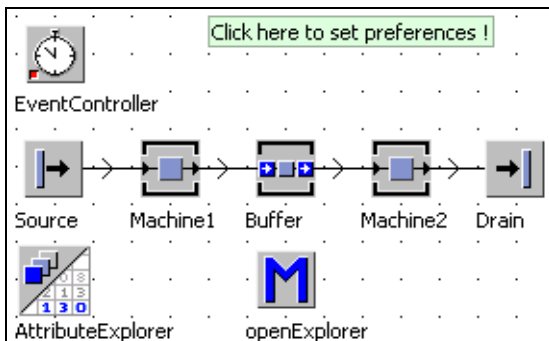


8.9 The AttributeExplorer

You can manage a variety of attributes of different objects from a single central location with the AttributeExplorer.

Example 95: AttributeExplorer

Create the following Frame:



Insert a comment with the text “Click here to set preferences!” (to open the AttributeExplorer). The processing times of Machine1 and Machine2 should be changed in a single dialog box. Therefore, add an AttributeExplorer to the frame. Open the AttributeExplorer by double-clicking it. Select the tab Objects and turn off inheritance (+ Apply). Drag the items from the frame to the list of object paths. Press Enter after each object to add a new line to the table.

Data	Objects	Attributes	Query	User-defined Attributes
	Objects			
1	.Models.AttributeExplorer.Source			
2	.Models.AttributeExplorer.Machine1			
3	.Models.AttributeExplorer.Buffer			
4	.Models.AttributeExplorer.Machine2			

Next, select the attributes that you want to view and modify. In the example above, these are the attributes interval (source), procTime (Machine1 and Machine2), and capacity (buffer). Enter these settings on the tab **ATTRIBUTES**. Use the column alias to display a different name than the attribute name in the AttributeExplorer (for instance processing time instead procTime). You can select the attribute with the button Show attributes.

First, turn off inheritance and click Apply.

Data	Objects	Attributes	Query	User-defined Attributes
	Name	Alias		
1	Interval			
2	ProcTime	Processing time		
3	capacity			

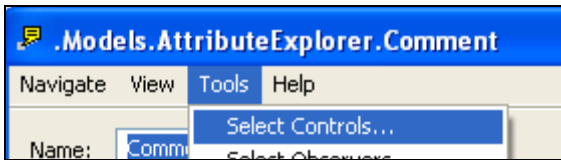
If you want to display the alias names, select the option Show attributes with alias on the tab Data. You can also select to show the paths or the labels of the objects.

Data	Objects	Attributes	Query	User-defined Attributes
<input type="radio"/> Watch <input checked="" type="radio"/> Edit <input type="radio"/> Read Only <input type="checkbox"/>				
Show objects with: Path <input type="checkbox"/>				
Show attributes with: Alias <input type="checkbox"/>				
<input type="checkbox"/> Show comment <input type="checkbox"/>				

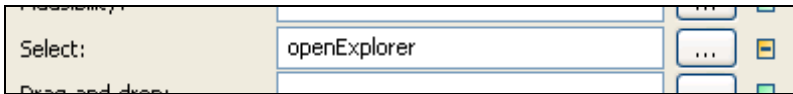
Click the button Show Explorer to open a window in which you can set all values at once

	Interval	Processing tir	capacity
.Models.AttributeExplorer.Source	2:00.0000	0.0000	1
.Models.AttributeExplorer.Machine1		1:00.0000	1
.Models.AttributeExplorer.Buffer		0.0000	4
.Models.AttributeExplorer.Machine2		1:00.0000	1

The *AttributeExplorer* itself should open when you click the comment (“Click here ...”). Open the comment. Select Tools – Select controls.



Select the following:



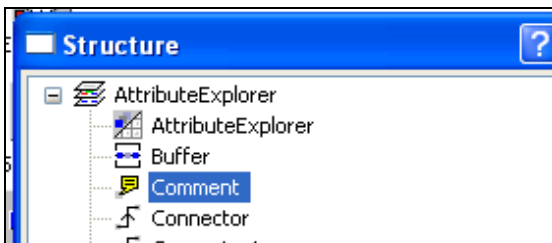
Program the method *openExplorer*:

```

is
do
  -- activate the AttributeExplorer
  attributeExplorer.Active:=true;
end;

```

When you now click on the comment, the *AttributeExplorer* opens. An issue results in this way in older versions of *Plant Simulation*. You then cannot open the dialog of the comment by double-clicking (before release 9). You can open the dialog via the structure of the frame. Click the right mouse button on the frame in the class library. Select Show structure from the menu. You can then double-click the objects in the opening window and thus open their dialogs.



8.10 The EventController

The EventController enables access to the system time. Furthermore, you can call all buttons of the EventController in SimTalk.

Methods of the EventController:

<i>Method</i>	<i>Description</i>
<code><path>.SimTime</code>	Returns the current simulation time (data type time).
<code><path>.AbsSimTime</code>	Returns the current simulation time (data type datetime).
<code><path>.start</code>	Starts the simulation
<code><path>.step</code>	
<code><path>.stop</code>	
<code><path>.reset</code>	