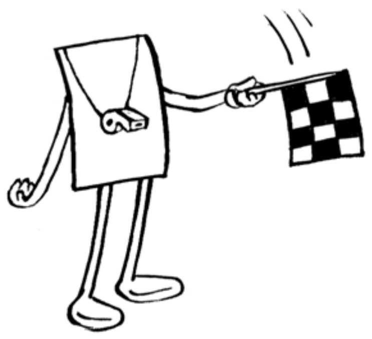


Chapter 1

Strategies

About This Chapter

This chapter provides basic information about methods and how they can be used for effective modeling. This chapter contains the following lessons:

Lesson 1: Introduction to the Object Method

Introduction

A Method is a small program, comparable to a procedure or a function in the programming languages Basic, Pascal or C++.

SimTalk, the programming language of eM-Plant, is based on the programming language **Eiffel**. It resembles other programming languages.

You can program a Method to read the attributes of the objects you inserted into your model and change these attributes. Select an object and then select **Objects > Show Attributes and Methods** to show all built-in attributes and methods that object provides.

In addition you can define any number of user-defined, custom attributes. The object Method is fully integrated into the object-oriented concept of eM-Plant. Enter your source code into the object Method. Then the built-in Interpreter processes these statements during the simulation run.

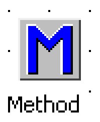
➢ A Method can:

- ❖ React to certain events during a simulation run.
- ❖ Get and set conditions.
- ❖ Execute statements.
- ❖ Modify and extend the behavior of objects.
- ❖ Prepare a model for new users with custom dialogs.

- **It allows you to:**
  - Implement different approaches, adapt your model to your needs.
  - Increase the efficiency of your simulation model.
  - Increase the flexibility of your simulation model

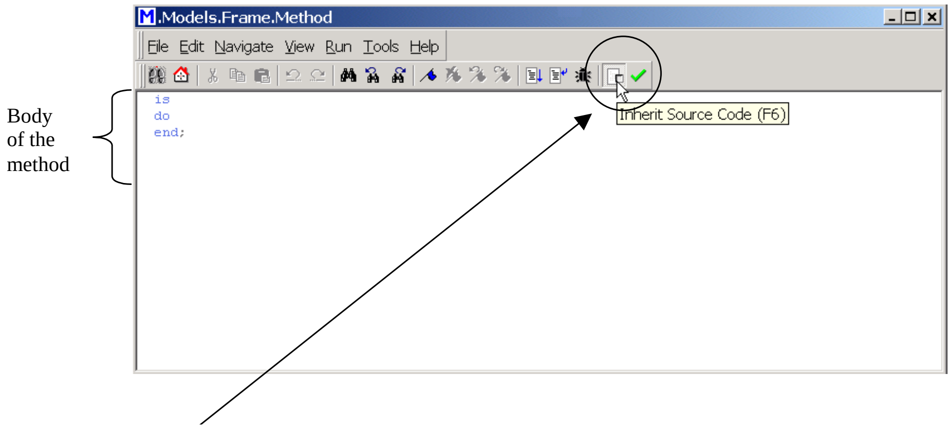
The Object Method

Use the Method to program controls that other objects start and execute during the simulation run.



Properties:

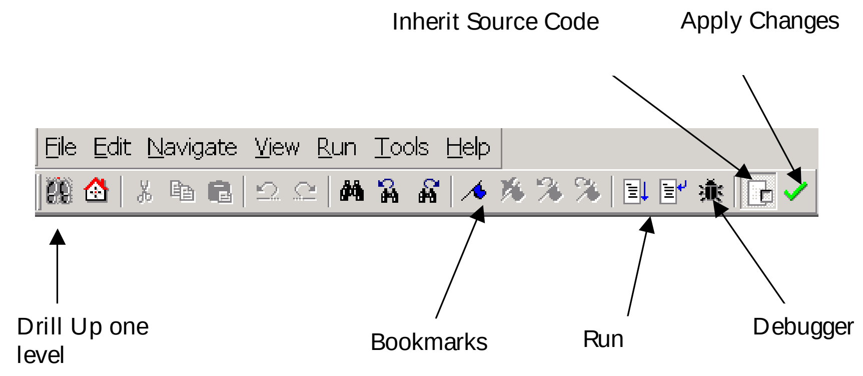
- Capacity: 0
- Information flow object



```
is
do
end;
```

When you insert the instance of a *Method* into your model, eM-Plant activates Inherit Source Code (from the class). Before you deactivate it, check if you want to change the source code in the class or in the instance.

Toolbar of the method



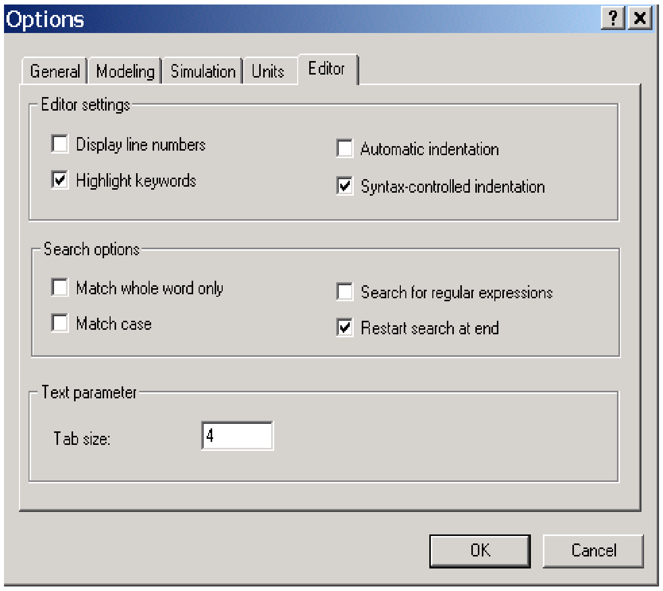
Editor

Select **Tools -> Options**

in the TUNE window to choose settings for the Method Editor.

It is recommend to activate Display line numbers

This allows to quickly change to the line that contains wrong source code in the Method Debugger.


Options

General | Modeling | Simulation | Units | Editor

Editor settings
- ☐ Display line numbers
- ☑ Highlight keywords
- ☐ Automatic indentation
- ☑ Syntax-controlled indentation

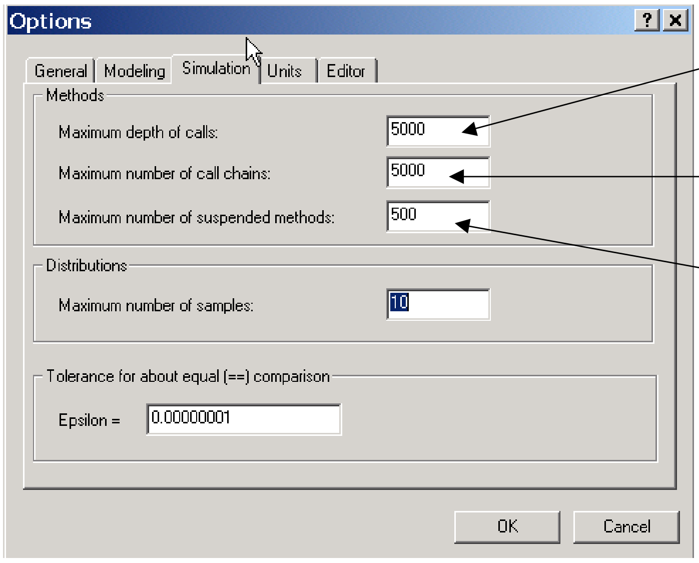
Search options
- ☐ Match whole word only
- ☐ Match case
- ☐ Search for regular expressions
- ☑ Restart search at end

Text parameter

Tab size: 4

OK Cancel

Options- Simulation



Methods

Maximum depth of calls: 5000

Maximum number of call chains: 5000

Maximum number of suspended methods: 500

Distributions

Maximum number of samples: 10

Tolerance for about equal (==) comparison

Epsilon = 0.00000001

Sets how many methods may be called by another method.

Sets the maximum number of methods that may be called at the same time.

Sets the number of methods that may be suspended at the same time and wait for an event to occur.

Lesson 2: Syntax for Writing a Method

Structure of a Method

The *Method* consists of these parts, not all of which are required.

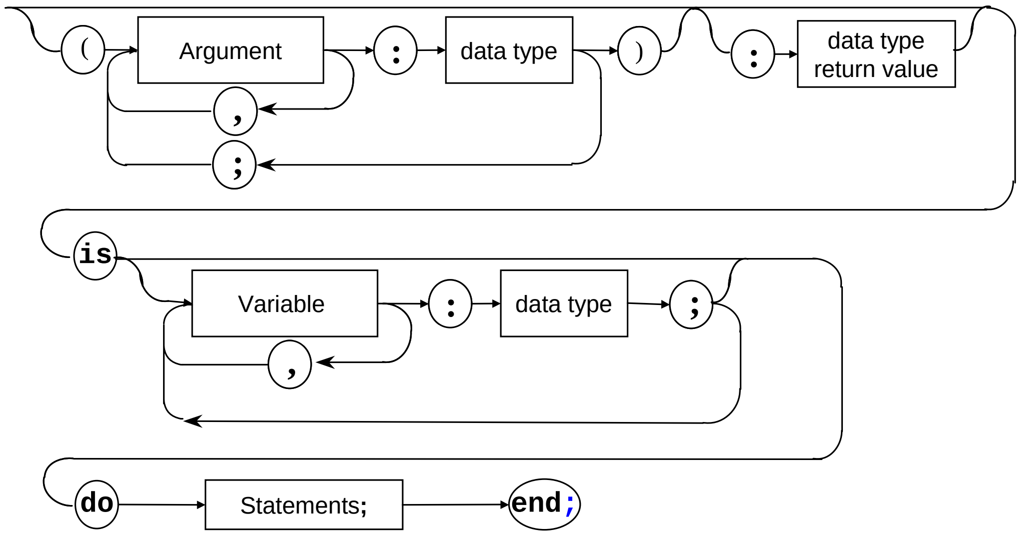
- **[Argument]**
    - Extends the functionality by adding arguments that pass additional information of the caller specifying certain actions. The caller has to pass the same number of arguments as you declared in the method. The data type passed has to be identical with the expected data type.
- **[Result]**
    - Enter the data type of the return value. It has to be assigned to the caller with the keyword **result**, when it returns a value.
- **Is**
    - Separates the declaration of arguments from the declaration of local variables.
- **[Local Variables]**
    - Local variables can only be accessed in the method in which, it is declared. Use them when you do not need data any longer after the method call.
- **Do**
    - Enter the source code proper after **do**.
- **[Statement]**

- ❖ Enter the source code, i.e. standard methods, assignments, control structures, method calls, branches, and loops.

➢ **End;**

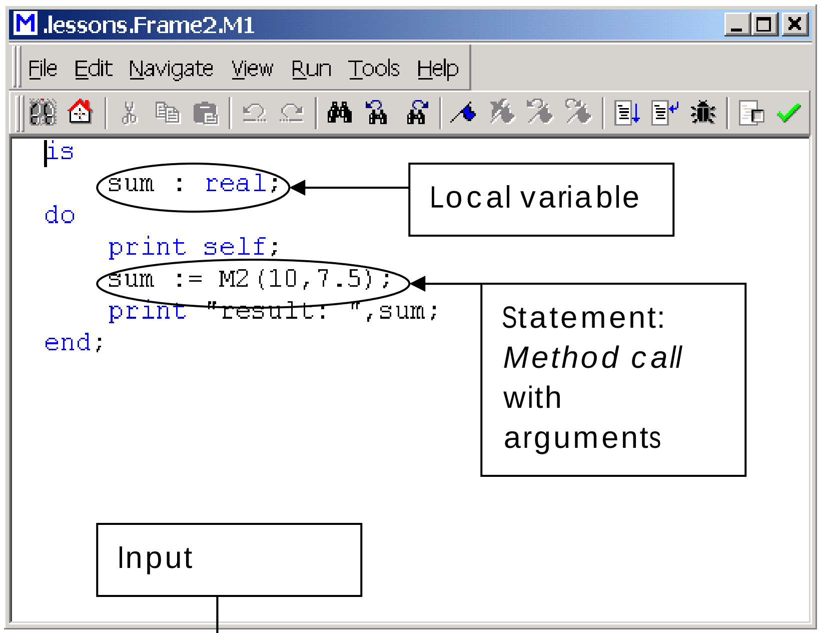
- ❖ Designates the end of the source code. After **end** you can only enter comments.

Syntax of a Method



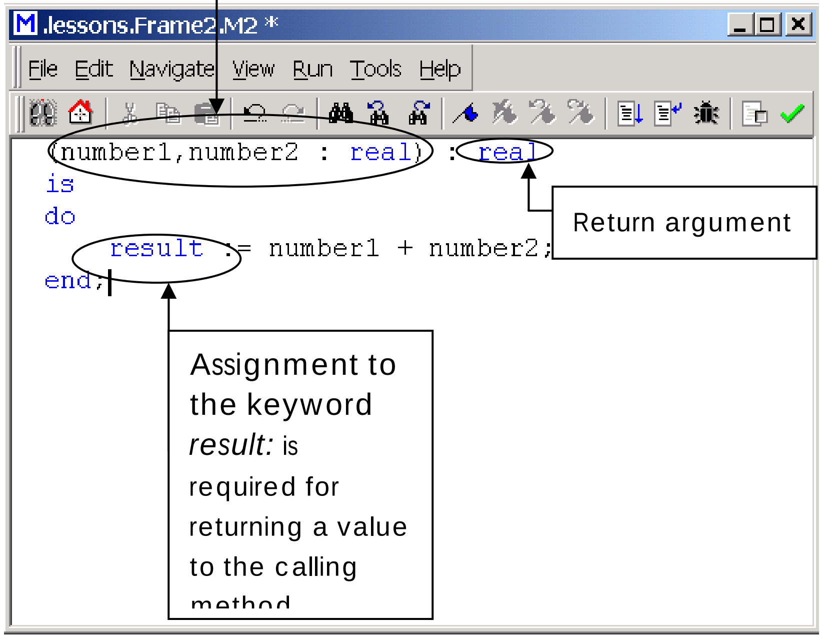
Sample Statements

The following are sample statements used in a method.


.lessons.Frame2.M1
File Edit Navigate View Run Tools Help

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

Local variable
Statement: *Method call* with arguments
Input


.lessons.Frame2.M2 *
File Edit Navigate View Run Tools Help

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

Return argument
Assignment to the keyword *result:* is required for returning a value to the calling method

SimTalk Operators

✚ **Assignment Operator**

The assignment operator **:=** assigns a new value to a variable.

<variable> **:=** <new value>;

✚ **Arithmetic Operators**

Addition, subtraction, multiplication, division (**+**, **–**, *****, **/**) and standard functions, such as trigonometric functions, logarithm, and exponential function.

✚ **Relational Operators**

Relational operators **=**, **/=**, **>**, **>=**, **<**, **<=**, **= =** compare two values. The result, TRUE or FALSE, is of data type Boolean.

Boolean operators **and**, **or**, **not** combine expressions.

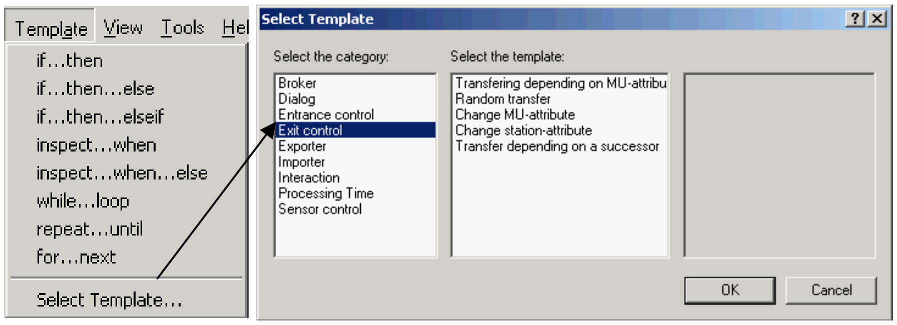
✚ **Input and Output Operators Load and save data.**

Document Conventions

- ➢ Names of attributes begin with an upper-case letter. Each new term after that begins with an upper-case letter, such as GenerationTableActive.
- ➢ Names of methods begin with a lower-case letter. Each new term after that begins with an upper-case letter, such as currIcon and absSimTime.
- ➢ Methods show keywords in blue (examples are is, do, and end) and comments in green.

Method Templates

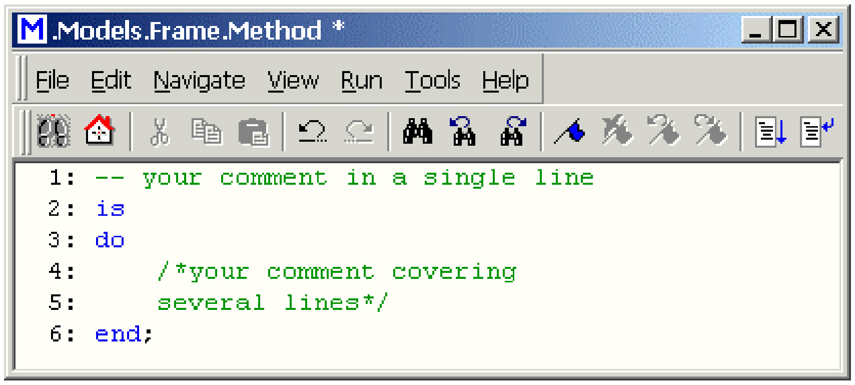
EM-Plant has templates availabe for creating methods.



Comments

Comments improve the legibility of a method and help to understand its function.

- Enter two hyphens - - to start a one-lined comment that ends at the end of the line.
- Enter a forward slash and an asterisk (/*) to start a comment line that will cover several lines and an asterisk forward slash (*/) to end the commented lines.



```
-- your comment in a single line
is
do
    /*your comment covering
    several lines*/
end;
```

Lesson 3: The Method Debugger

Introduction

eM-Plant automatically runs the debugger when the source code contains errors that prevent the method from being executed. The debugger can also be used to detect errors in the simulation model and can be used to watch how the procedures are executed.

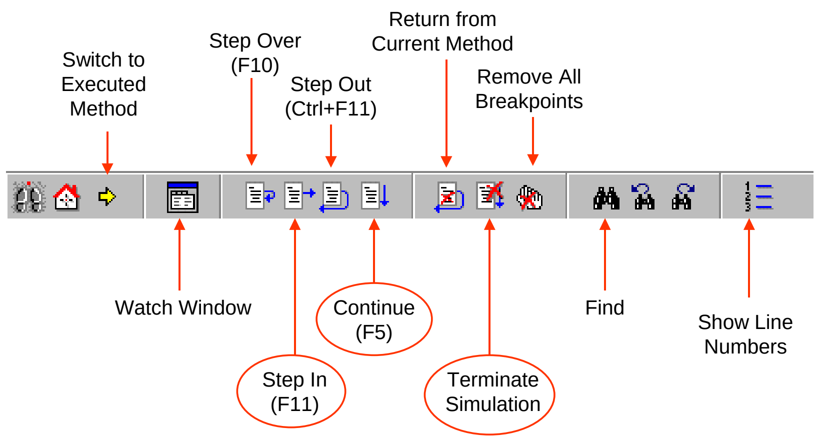
To open the debugger:

- Set breakpoints (F9).
- Select Debugger -> Open Debugger, or press F6, or hold down Shift + Alt + Ctrl while a method is being executed.

Debugger uses:

- Watch local variables, arguments and information about the calling object and the MU that triggers an action in the Watch Window.
- Track how the Method is executed step-by-step.
- Detect errors using the messages the debugger shows.

The Debugger Toolbar



Breakpoints executing a method step-by-step

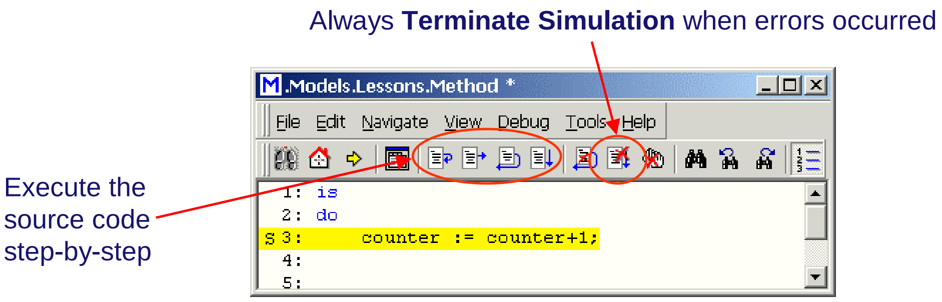
Set breakpoints in the method to watch how it is executed or to execute it step-by-step.

Setting breakpoints:

- Open the Method window.
- Place the curser in the line where the breakpoint is to occur.
- Press F9.

When the source code reaches this line while executing, eM-Plant opens the debugger window and selects the breakpoint.

Click the buttons on the toolbar to continue executing the program or to abort it.

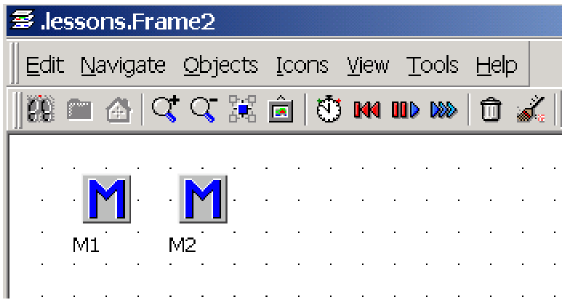

Always **Terminate Simulation** when errors occurred
Execute the source code step-by-step
.Models.Lessons.Method *
File Edit Navigate View Debug Tools Help

```
1: is
2: do
S 3:     counter := counter+1;
4:
5:
```

Exercise: Defining a method

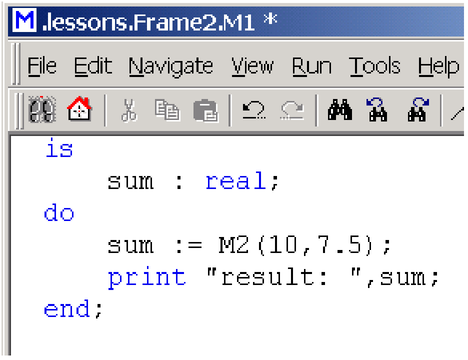
Objective: To write a method and view the results in the console.

1. Add the frame, Frame_2, to the folder Exercises.
2. Insert two Method objects.



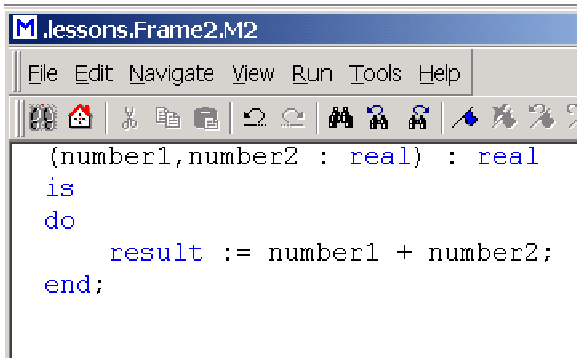
3 Enter the source code as displayed in the image below.

For M1:



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

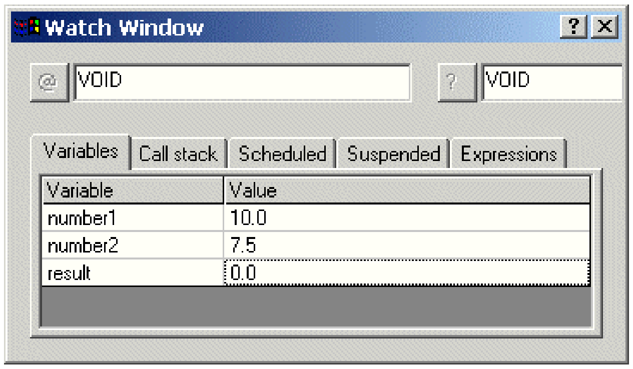
For M2:



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

4 Start the method M1 with the menu commands of the context menu, or run (F5).

5 Run the method step-by-step, open the watch window and watch the values.



Variable	Value
number1	10.0
number2	7.5
result	0.0

6 Set breakpoints and rerun the Method.

Lesson 4: Names, Identifiers and Paths

Introduction

Each object has an individual address to uniquely identify the object residing in the hierarchy level, within the name space, or within another name space.

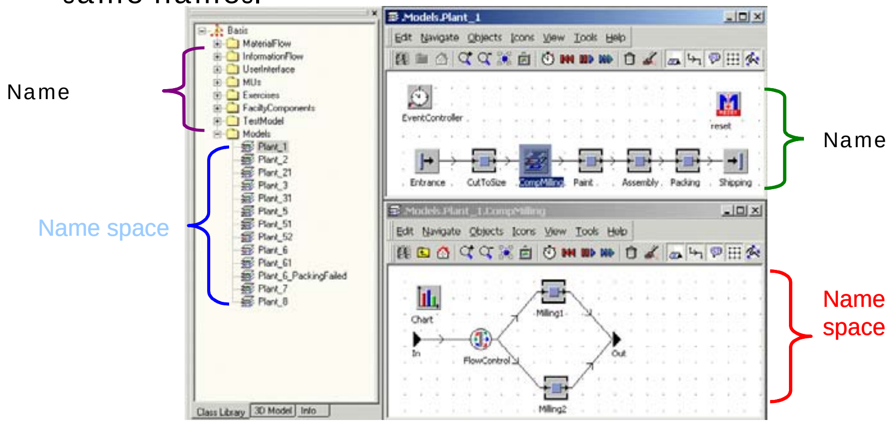
The relative path calls the object when it is located within the same name space.

The absolute path calls the object when it is located on another hierarchy level.

Name Space

All objects on the same hierarchy level are located within the same name space.

- Within a name space all objects must have unique names.
- Within different name spaces objects can have the same names.



Absolute Path

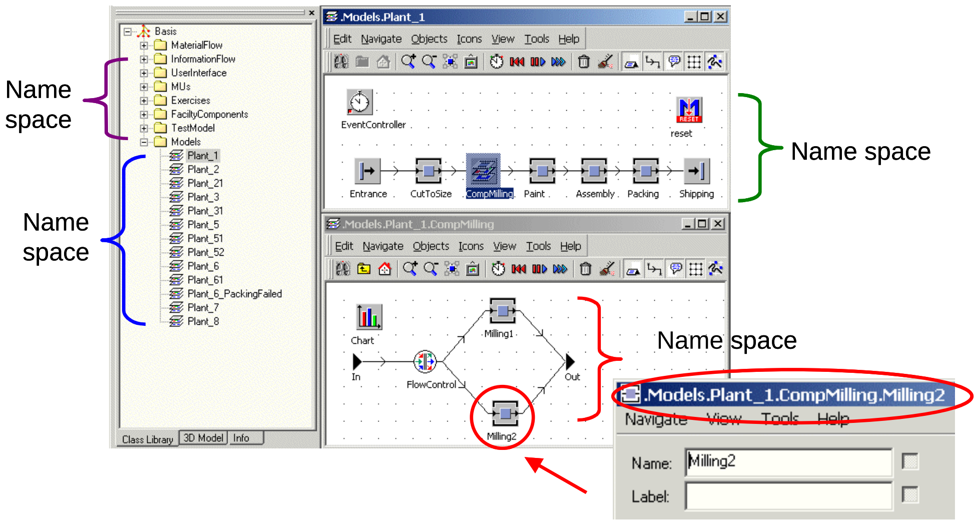
Identifies the exact location of an object, irrespective to its hierarchy level within the frame.

- The absolute path is required to call an object located on another hierarchy level.
- The absolute path always starts in the Class Library and then proceeds through the folders and frames to the object.
- The absolute always begins with a period followed by a name, a period, a name, a period until it has reached the name of the object.

  .<name>.<name>...<object>

- When you place an object onto another hierarchy level eM-Plant has to rename the absolute path.

Example of the Absolute path to the Milling2 object:



Name: Milling2
Label:

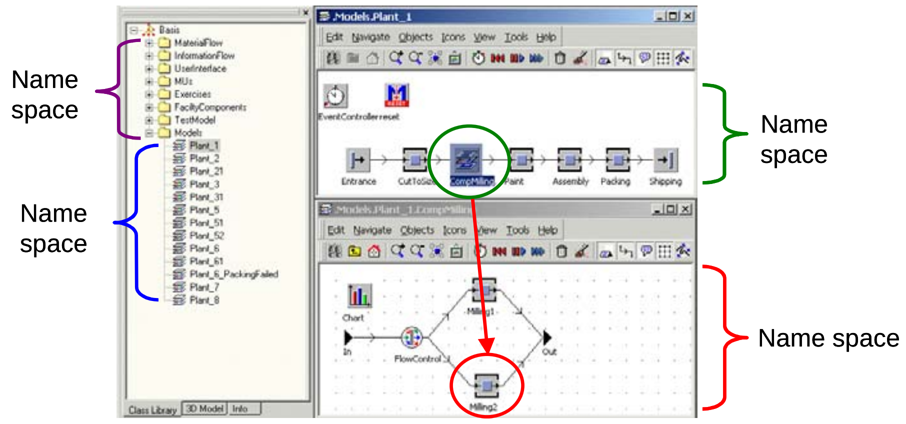
. Models.Plant_1.CompMilling.Milling2

The Relative Path

Calls an object within the same name space without having to enter the hierarchy level.

- Ths path never starts with a period.
- Enables you to fleibly and universally deploy Methods and objects.
- Is correct even after the structure of the model has been changed.

Example of the Relative path to the Milling2 object:



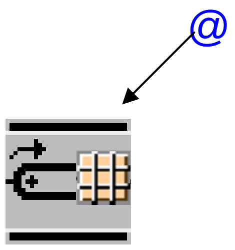
CompMilling.Milling2

Lesson 5: Anonymous Indentifiers

Introduction

The paths to your objects should be as univeral as possible, to enable you to deploy the called object as flexibly as possible.For this you can employ a number of **anonymous identifiers**, which serve as placeholders.

Calling the MU Triggering the Method — @



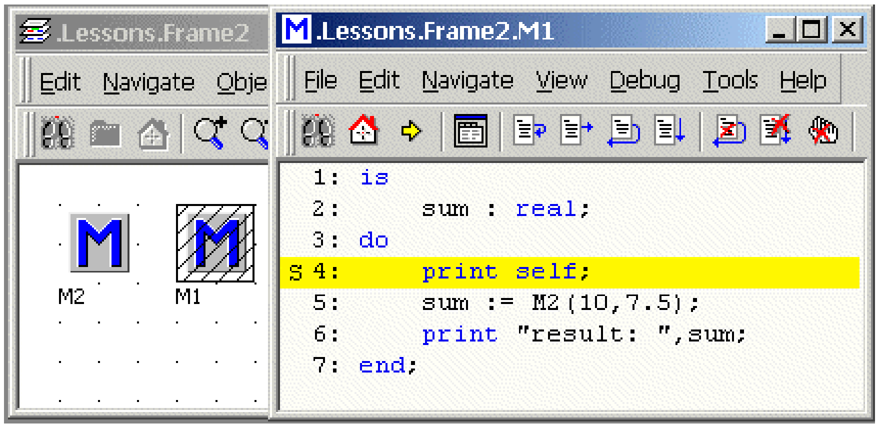
The anonymous identifier @ uniquely identifies the MU, such as **table_top:6712**, that triggered the *Method* when entering or exiting the object.

eM-Plant even uniquely assigns the MU, when several controls are triggered. The MU might also be located on another object.

Calling the Active Method – Self

The anonymous identifier *self* returns the path of the currently executed method.

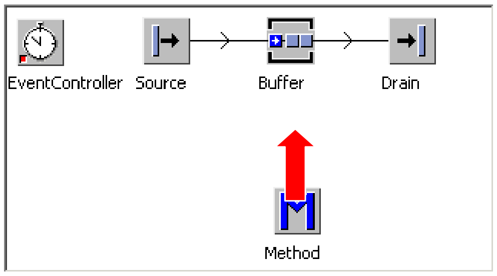
- **Self;** returns the path to the name of the Method.
- **Self.name;** returns only the name of the Method.

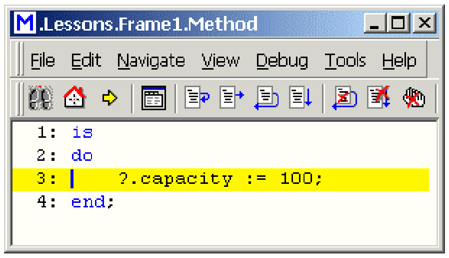


```
is
    sum : real;
do
    print self;
    sum := M2(10,7.5);
    print "result: ",sum;
end;
```

Calling the Object Calling the Method — ?

The anonymous identifier **?** is the material flow object or the control (*Method*) that called the *Method*. Employing **?** allows a *Method* to be used without modifications by several objects.





```
is
do
    ?.capacity := 100;
end;
```

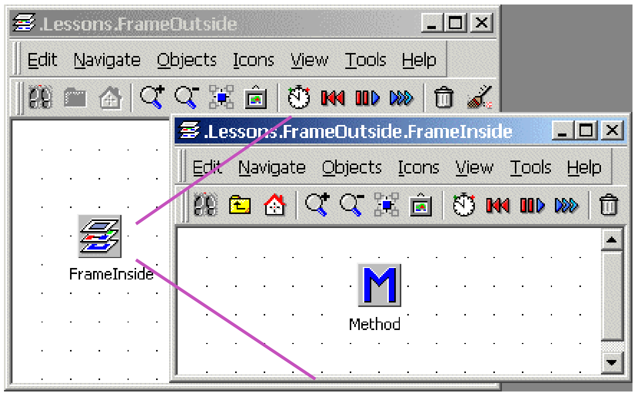
Calling the Active Frame – Current

The anonymous identifier *current* returns the frame the method object is located in. This is used to enter the location of a frame into tables and lists or passes it as an argument to methods in other frames.

Calling the Location of the Active Frame – Location

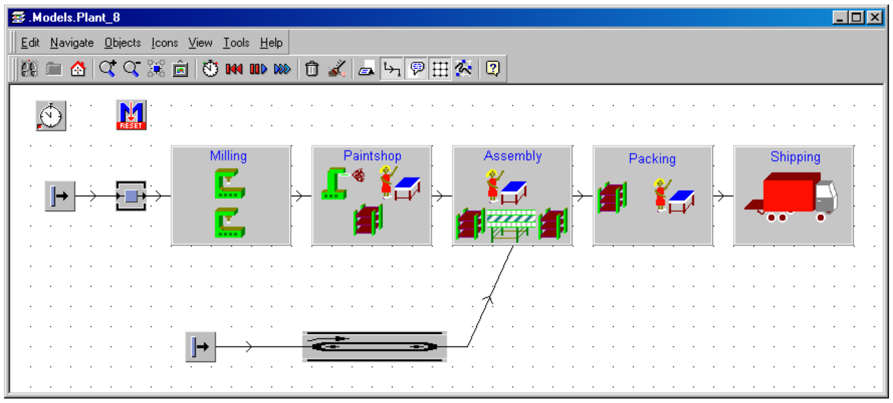
The method *location* returns the object located directly above the object designated by the path. The location of the MU is the object it is currently located on.

- **<object>.location** returns the path to the frame where the object is located.
- **@.location** returns the location of the object where the MU is located.


.Lessons.FrameOutside
Edit Navigate Objects Icons View Tools Help
FrameInside
.Lessons.FrameOutside.FrameInside
Edit Navigate Objects Icons View Tools Help
Method

Calling the Topmost Level of the Hierarchy – root

The anonymous identifier root returns the topmost frame in the hierarchy of frames. This identifier is especially useful when you do not know the name of the root frame (the frame located at the top of the hierarchy).

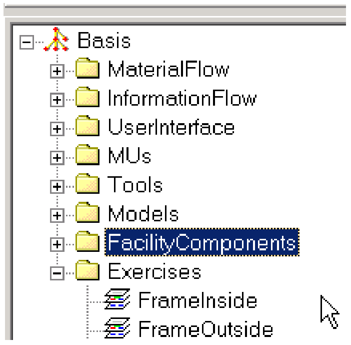


Exercise: Using the Anonymous Identifiers

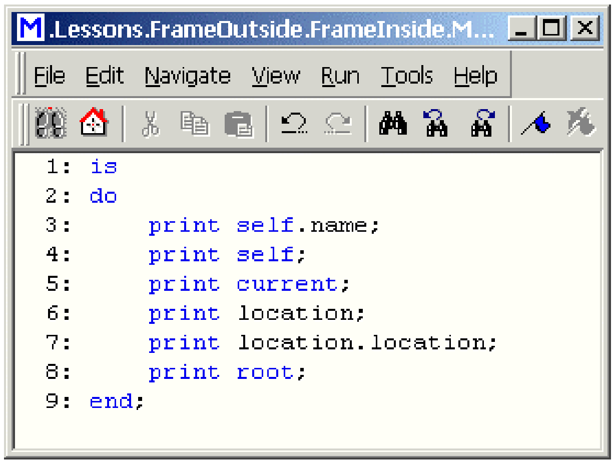
Objective: To use the anonymous identifiers.

1 Add the frame FrameOutside to the Exercises folder.

2 Add the frame FrameInside to the same folder.

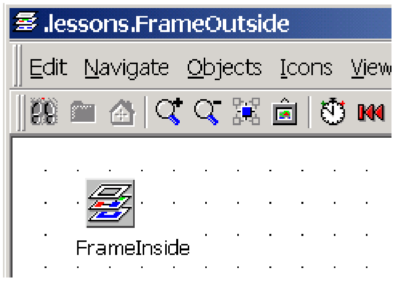


3 Insert a method into the frame FrameInside and enter the source code as follows:


.Lessons.FrameOutside.FrameInside.M...
File Edit Navigate View Run Tools Help

```
is
do
      print self.name;
      print self;
      print current;
      print location;
      print location.location;
      print root;
end;
```

4 Open the frame FrameOutside and drag the frame FrameInside into it.



5 Run the Method and watch the values in the console.

6 Open the Method in the Frame FrameInside, run it step-by-step and watch what the Console displays. You can also use the values in local variables and watch them in the Watch Window of the Debugger.

NOTES:
Nesting the methods demonstrates the hierarchy level of the anonymous identifier.

Lesson 6: Scheduling Method Calls

Events in the Model - Calling Methods

An event taking place at a certain time in the model can call a Method:

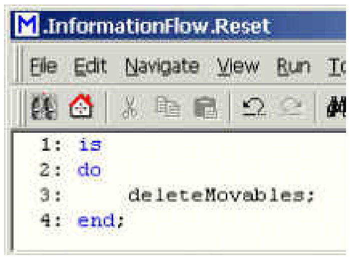
- ➢ Clicking this button of the EventController calls this Method:
  - ❖ **Reset**:Calls all Methods named reset located in the open model.
  - ❖ **Init**: Calls all Methods named init after resetting the model.
- ➢ eM-Plant executes the Method autoexec that has to be located in the class library, when opening the model file.
- ➢ eM-Plant calls all Methods named endSim at the end of a simulation run.
  - ❖ The simulation ends when the EventController has processed all events from the List of Scheduled Events or when eM-Plant reaches the end time you entered in the text box **End** on the tab **Settings**.
- ➢ The **Trigger** and the **Generator** can also call a Method.

The Method Reset

Whenever you click **Reset** in the EventController, eM-Plant scans the entire model, including all Frames you inserted, for Methods named reset and executes them.

Our reset Method deletes all MUs in the model.

Insert a Method object into your model, and change the name to **reset**. Enter the text into the method from the following image.

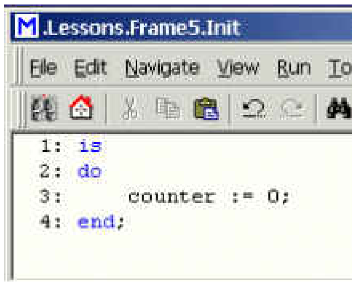


```
is
do
	deleteMovables;
end;
```

The Method init

Whenever you click **Init** in the EventController, eM-Plant executes all Methods named init located in the model. Init Methods can initialize your model, for example set the values of variables, place MUs onto material flow objects, etc.

Insert a Method object into your model, and enter **init** into the text box **Name** in the dialog **Rename**. Insert the text as follows:



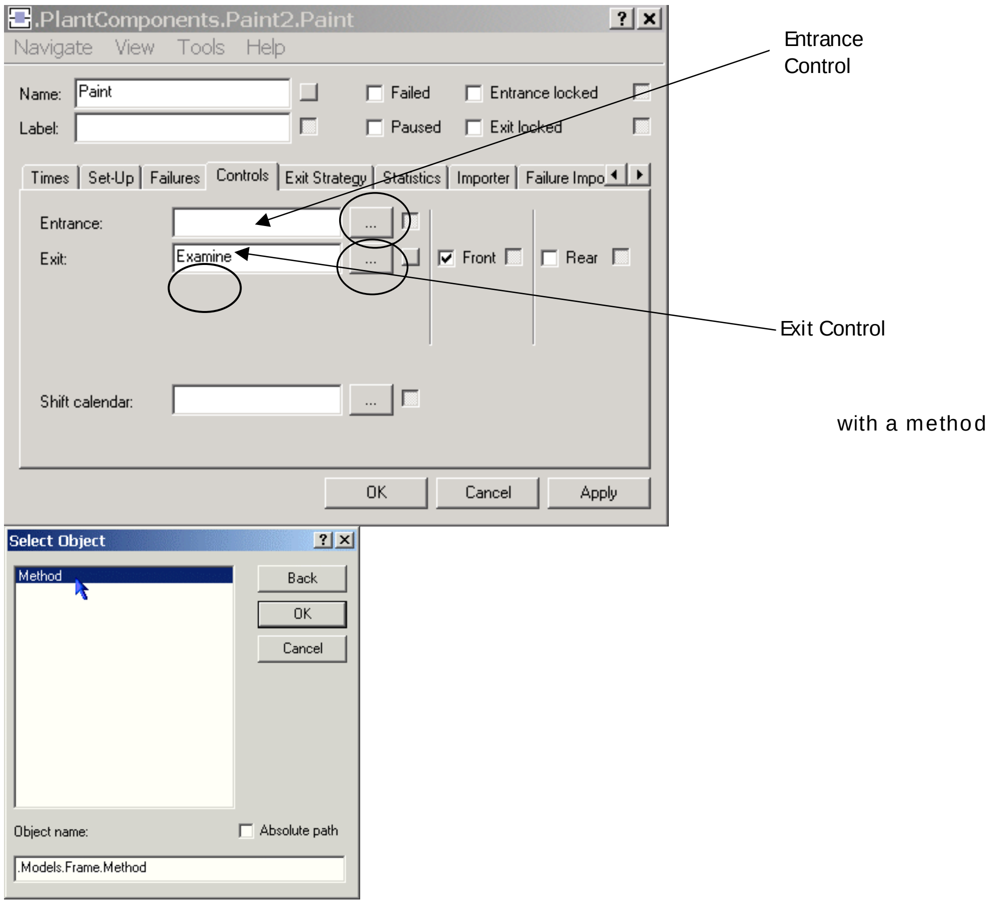
```
is
do
	counter := 0;
end;
```

Lesson 7: Entrance and Exit Controls

Introduction

You can program and select an **Entrance** and/or an **Exit control** for each material flow object. You might compare them to a photo sensor placed at the entrance/exit of the material flow object.

You can attach a *Method* to each of these **photo sensors** that eM-Plant executes as soon as an MU enters or exits the object.


.PlantComponents.Paint2.Paint
Navigate View Tools Help
Times Set-Up Failures Controls Exit Strategy Statistics Importer Failure Impo
OK Cancel Apply
Entrance Control
Exit Control
Select Object
Back
OK
Cancel

Name: Paint
Label:
☐ Failed ☐ Entrance locked
☐ Paused ☐ Exit locked

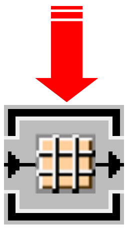
Entrance:
Exit: Examine ☑ Front ☐ Rear
Shift calendar:

with a method

Method

Object name: ☐ Absolute path
.Models.Frame.Method

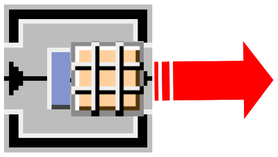
Entrance Control of Location Oriented Objects



This is activated as soon as an MU enters the object, the MU is located all the way on the object, irrespective to its length.

NOTE
Once set, the time the MU remains on the object cannot be changed, unless a formula is entered as a processing time.

Front-Activated Exit Control of the Location Oriented Objects

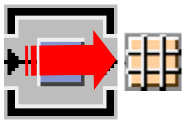


This is called when the MU intends to exit the object. The MU is still located on the object.

The exit control may be activated several times: when the MU cannot exit the object or because the successor is full. eM-Plant calls the exit control again once the successor is unblocked.

NOTE
The front-activated exit control overrides the default exit strategy of eM-Plant, the MUs must be moved manually by methods.

Rear-Activated Exit Control of Location Oriented Objects

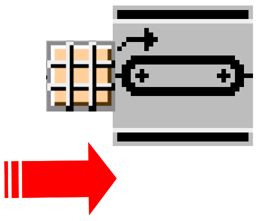


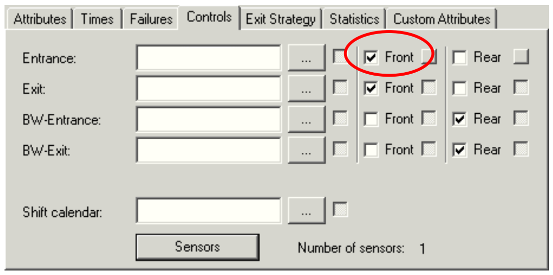
This is called once the MU exits the object. The MU is located on the successor.

NOTE
The rear-activated exit control is called only once and it does not override the default strategy.

Front_Activated Entrance Control of Length Oriented Objects

This is called as soon as an MU wants to enter the object. The front
of the MU is located on the object when the method is called.




Attributes | Times | Failures | Controls | Exit Strategy | Statistics | Custom Attributes

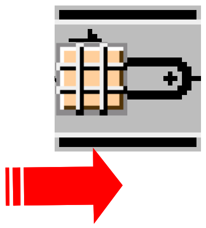
Entrance:		☐	[✓ Front] ☐	☐ Rear ☐
Exit:		☐	[✓ Front] ☐	☐ Rear ☐
BW-Entrance:		☐	☐ Front ☐	[✓ Rear] ☐
BW-Exit:		☐	☐ Front ☐	[✓ Rear] ☐
Shift calendar:		☐		

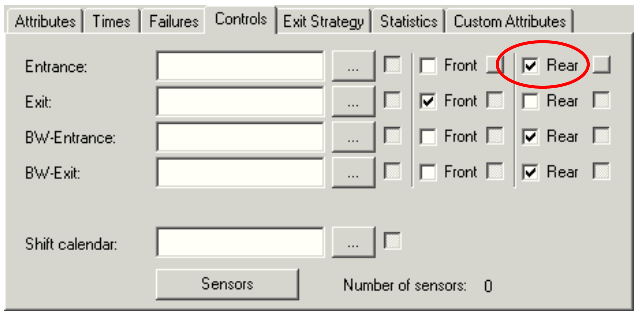
Sensors   Number of sensors: 1

NOTE
Once set, the time the MU remains on the object cannot be changed, unless a formula is entered as a processing time.

Rear-Activated Entrance Control of Length Oriented Objects

This is called once the entire length of the MU is located on the object.





		Front		Rear	
Entrance:	☐	☐ Front	☐	[✓ ☑ Rear]	☐
Exit:	☐	☑ Front	☐	☐ Rear	☐
BW-Entrance:	☐	☐ Front	☐	☑ Rear	☐
BW-Exit:	☐	☐ Front	☐	☑ Rear	☐
Shift calendar:	☐				

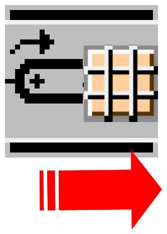
Sensors   Number of sensors: 0

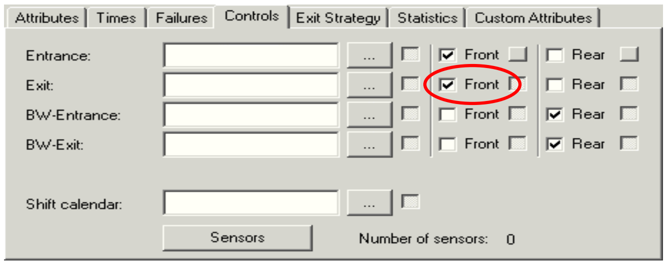
NOTE

The point-in-time the MU enters the object until the point-in-time the rear-activated entrance control is activated is the time the MU needs to entirely move onto the object.

Front_Activated Exit Control of Length Oriented Objects

This is called once the MU intends to exit the object. The MU is
located entirely on the object.



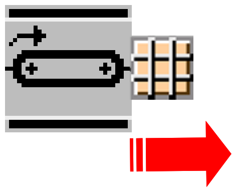


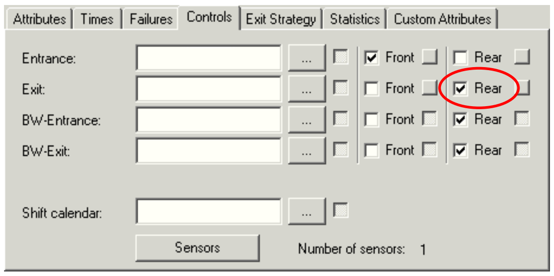
		Front		Rear	
Entrance:	☐	☑ Front	☐	☐ Rear	☐
Exit:	☐	[✓ ☑ Front]	☐	☐ Rear	☐
BW-Entrance:	☐	☐ Front	☐	☑ Rear	☐
BW-Exit:	☐	☐ Front	☐	☑ Rear	☐
Shift calendar:	☐				

Sensors   Number of sensors: 0

Rear_Activated Exit Control of Length Oriented Objects

This is called once the MU has exited the object. The MU is located on the succeeding object.



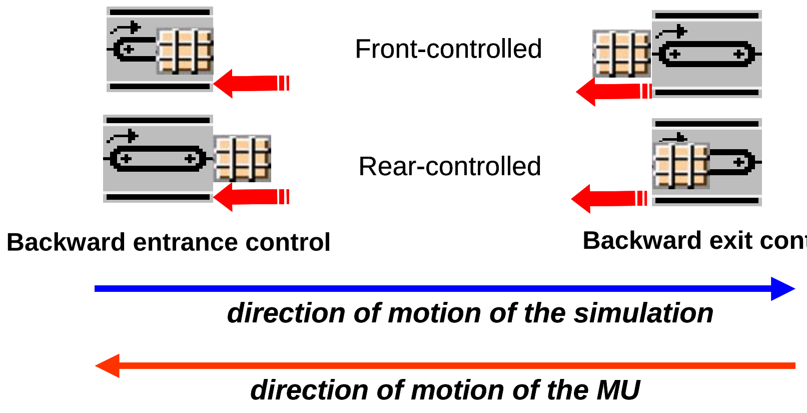


Entrance:		✓ Front	Rear
Exit:		Front	✓ Rear
BW-Entrance:		Front	✓ Rear
BW-Exit:		Front	✓ Rear
Shift calendar:			

Number of sensors: 1

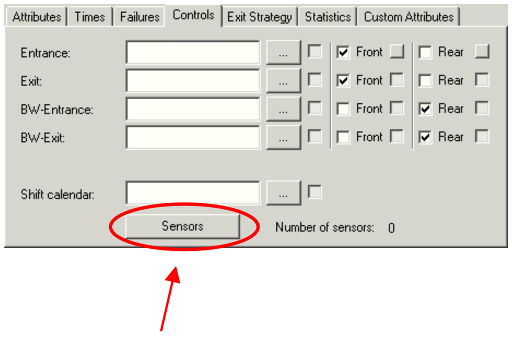
Backward Entrance and Exit Controls

The MU does not change its orientation (i.e. it does not turn around and move with its frint end against the direction of motion of the simulation).


Front-controlled
Rear-controlled
Backward entrance control
Backward exit control
direction of motion of the simulation
direction of motion of the MU

Sensors of Length Oriented Objects

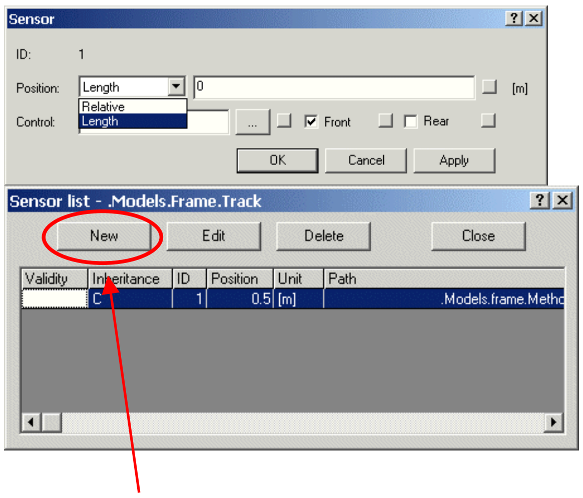
For the length oriented objects you can define sensor controls for any location on them.



Number of sensors: 0

Entrance:	☑ Front	☐ Rear
Exit:	☑ Front	☐ Rear
BW-Entrance:	☐ Front	☑ Rear
BW-Exit:	☐ Front	☑ Rear

Click Sensors to open the sensor list.



ID: 1

Position: Length 0 [m]

Control: ☑ Front ☐ Rear

Sensor list - .Models.Frame.Track

Validity	Inheritance	ID	Position	Unit	Path
	C	1	0.5	[m]	.Models.frame.Metho

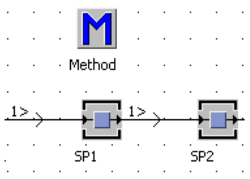
Click New to open the dialog **Sensor**.
Select and enter settings there.

Moving an MU to the Next Object

The method move moves the MU from one object to another object. eM-Plant moves the MU onto a location oriented object with the speed entered.

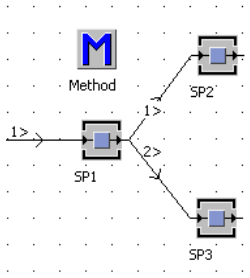
move to SP2:

```
@.move;
@.move(SP2);
@.move(1);
```



move to SP3:

```
@.move(SP3);
@.move(2);
```



insert

The method ***insert*** moves the MU completely to the destination object. It has to provide enough space to accomodate the MU.

Example: `@.insert(track,3.3);`

transfer

The method ***transfer*** transfers the MU to the next material flow object. The method *transfer* completely removes the MU from the present object, even if the successor does not provide enough space. The remainder of the MU hangs in the air.

Example: `@.tranfer(track,3.3);`

The Methods ***move***, ***insert***, ***transfer*** return true, when the MU was moved, false, when it was not moved.

Predecessors and Successors

Use the methods:

- succ(<integer>)
- pred(<integer>)

to check the predecessor and/or successor of an object. This is helpful for modeling an exit strategy.

Starting with the path of the called object the method accesses its successor or predecessor provided the objects are connected. The number is the number of the connector.

Example:

- @.move(source.pred(1));

- @.move(?.succ(1));
- if milling.pred(1).name = „CutToSize“ then...

Exercise 1: Entrance and Exit Control of a SingleProc

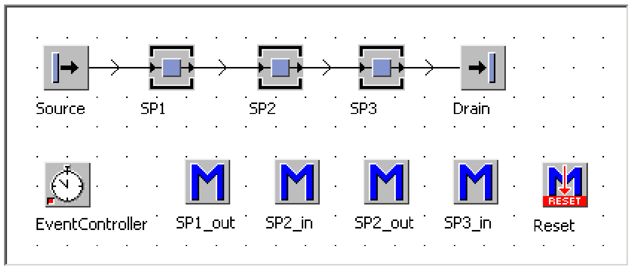
Objective: To insert entrance and exit controls in single processes.

1 Add the frame Frame_4 to the Exercises folder.

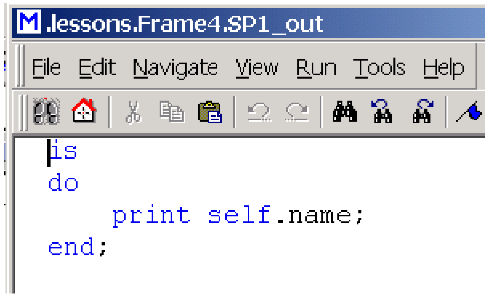
2 Add the following objects:

- a source
- 3 SingleProc’s
- a drain
- an EventController
- 4 Method objects

3 Name each according to the image below:



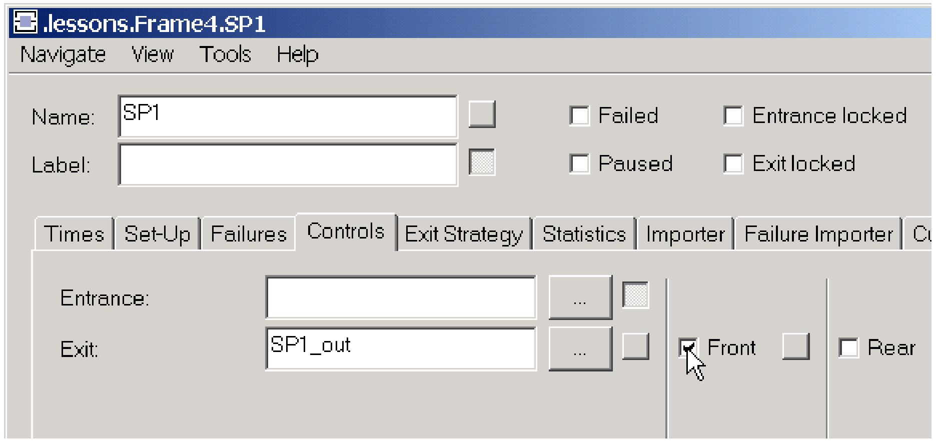
4 Enter **print self.name** into the methods.



```
is
do
    print self.name;
end;
```

5 select the following controls for each method:

- SP1_Out: front-activated exit control.

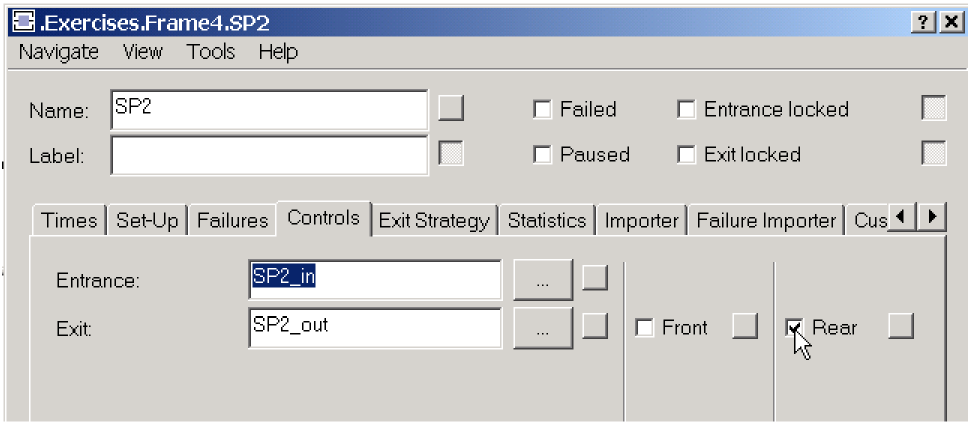


Name: SP1
Label:
Failed
Paused
Entrance locked
Exit locked
Entrance:
Exit: SP1_out
[✓ Front]
Rear

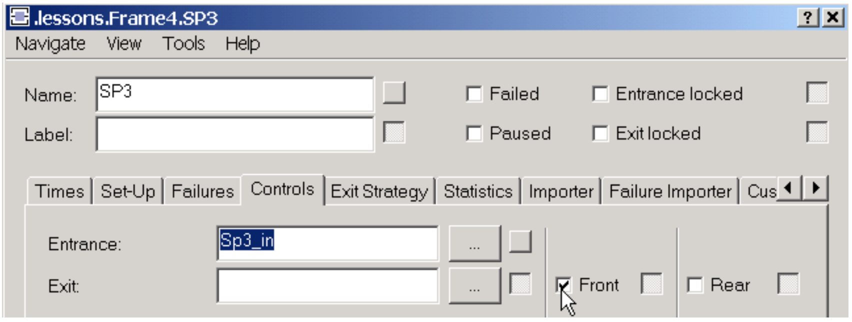
NOTE
Keep in mind how the front activated exit control works.

- SP2_In: front-activated entrance control.
- SP2_Out: rear-activated exit control.

Last printed: 10/26/2008 2:03:00 PM
Last saved: 2/19/2004 1:54:00 PM



 SP3_In: front-activated entrance control.



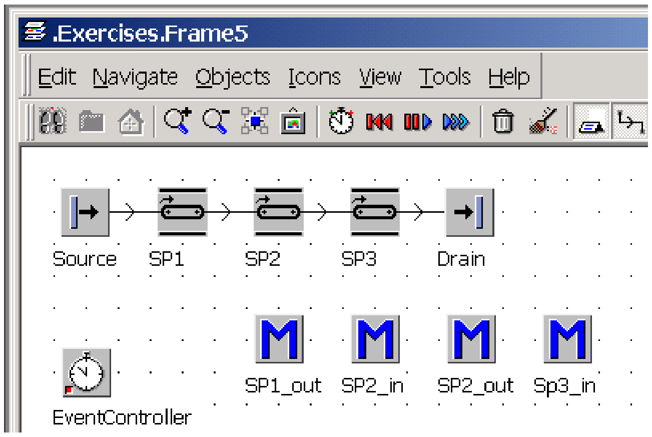
6 Run the simulation and observe the console window.

Exercise 2: Entrance and Exit Control of the Line Object

Objective: To view entrance and exit controls on length oriented objects.

1 Copy the contents of Frame_4 and paste it into Frame_5.

2 Replace the SingleProc's with Lines



3 Insert the controls as follows:

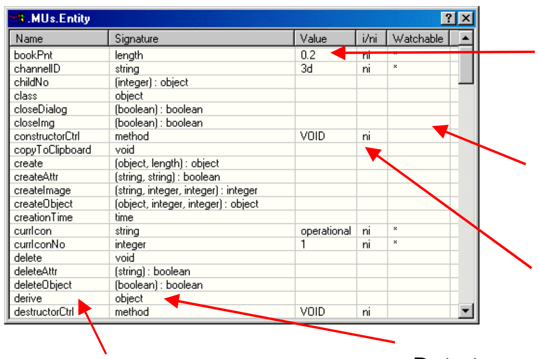
SP1_Out:	rear-activated exit control.
SP2_In:	rear-activated entrance control.
SP2_out:	rear-activated exit control.
SP3_In:	front-activated entrance control.

4 Run the simulation and observe the console window.

Lesson 7: Attributes

Introduction

The attributes of a class or an instance can be viewed using the Show Attributes and Methods from the context menu.



Name	Signature	Value	i/ni	Watchable
bookPnt	length	0.2	ni	*
channelID	string	3d	ni	*
childNo	(integer) : object			
class	object			
closeDialog	(boolean) : boolean			
closeImg	(boolean) : boolean			
constructorCtrl	method	VOID	ni	
copyToClipboard	void			
create	(object, length) : object			
createAttr	(string, string) : boolean			
createImage	(string, integer, integer) : integer			
createObject	(object, integer, integer) : object			
creationTime	time			
currIcon	string	operational	ni	*
currIconNo	integer	1	ni	*
delete	void			
deleteAttr	(string) : boolean			
deleteObject	(boolean) : boolean			
derive	object			
destructorCtrl	method	VOID	ni	

Value. Only attributes have a value. Double-click to edit.

Shows if eM-Plant can watch the attribute or the state.

Shows if the value is inherited (i) or not inherited (ni).

Name of the attribute or method.

Data type of the attribute, or data type of the argument(s).

Assigning a Value

A value can be assigned by entering the value to the right of the assignment operator: <expression> := <value>.

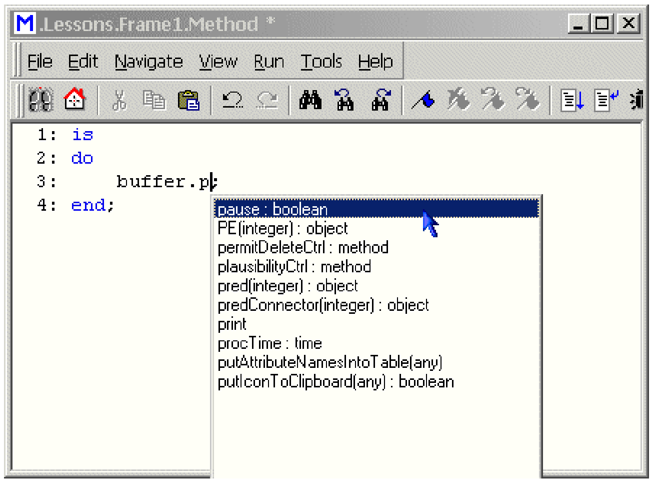
Examples: buffer.capacity := 8;

SingleProc.Pause := true;

@.currIcon := „TableTop“;

Use auto-complete when entering source code. Enter the first one or two characters of an object, an attribute or a method and press **esc** or select **Edit -> Auto Complete** or hit the keys **Ctrl + Space**.

eM-Plant completes the word, and if necessary it will add the corresponding characters – period, left parantheses, assignment operator and right parantheses.


.Lessons.Frame1.Method *
File Edit Navigate View Run Tools Help

```
is
do
    buffer.p
end;
```

pause : boolean
PE(integer) : object
permitDeleteCtrl : method
plausibilityCtrl : method
pred(integer) : object
predConnector(integer) : object
print
procTime : time
putAttributeNamesIntoTable(any)
putIconToClipboard(any) : boolean

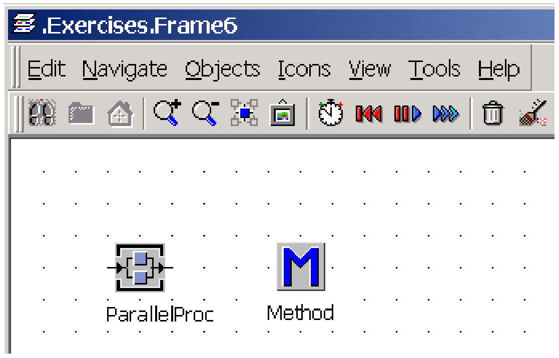
If the name is not unique, eM-Plant opens a window where you select an item, press the up or down arrow to select the desired expression and press enter.

Exercise: Changing the Attributes of an Object with a Method

Objective: to change the attributes of a ParallelProc by using a Method.

1 Add the frame Frame_6 to the Exercises folder.

2 Insert a ParallelProc and a Method object.



3 Change the following attributes using the method and by searching the Show Attributes and Methods window.

- ❖ Process time = 120 seconds
- ❖ x-dimension = 5
- ❖ y-dimension = 3
- ❖ Lock the Entrance of the ParallelProc (Entrance Locked)

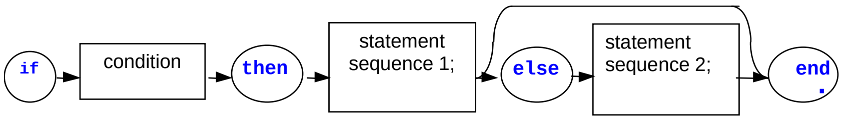
4 Close the Show Attributes and Methods window.

5 Run the method.

6 Verify the changes in the Show Attributes and Methods window as well as on the ParallelProc object.

Lesson 8: Conditional Statements

Branching using If Then



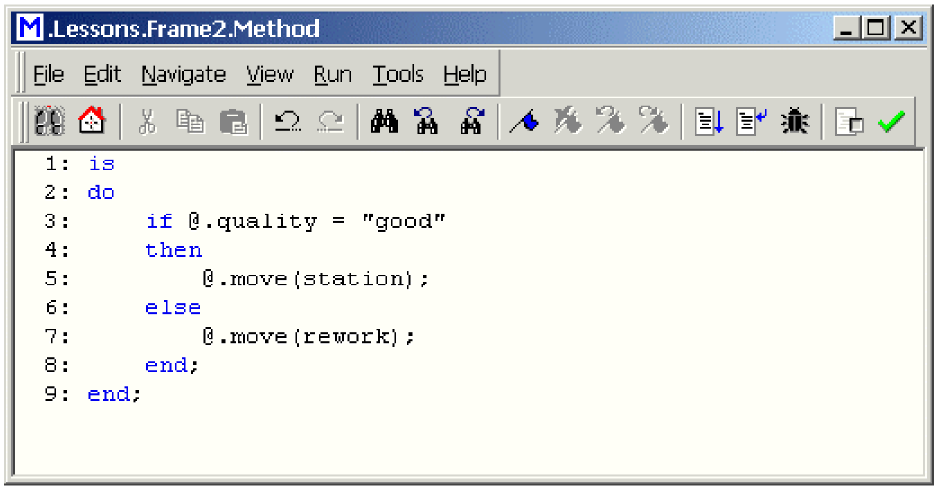
U

se the *if* statement to make the execution of a *Method* depend on the result of a condition.

If the condition returns true, the *Method* executes statement sequence 1. If the condition returns false, the *Method* executes statement sequence 2.

If you do not need it, you can also omit the else branch with statement sequence 2.

Example of an If statement:

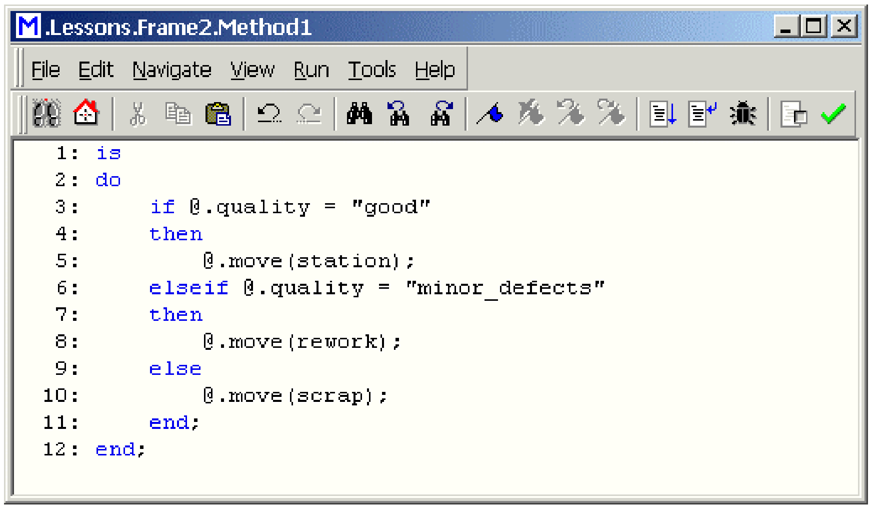


```
is
do
     if @.quality = "good"
     then
          @.move(station);
     else
          @.move(rework);
     end;
end;
```

Branching using If Elseif

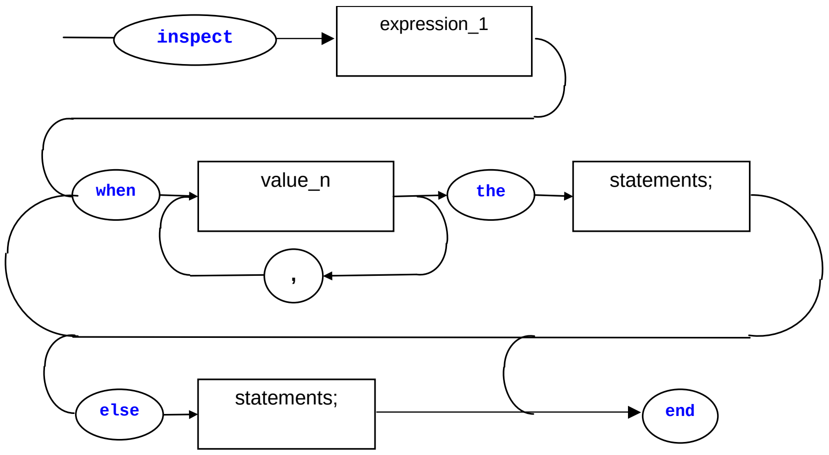
Using the *if then elseif* statement, the Method can react to different conditions. eM-Plant analyzes the conditions one after the other until one condition is true. Then it executes the respective statement sequence. If no condition is true, it executes the optional else branch. When the else branch is not present, eM-Plant continues after the final end statement.

Example of an ElseIf statement:

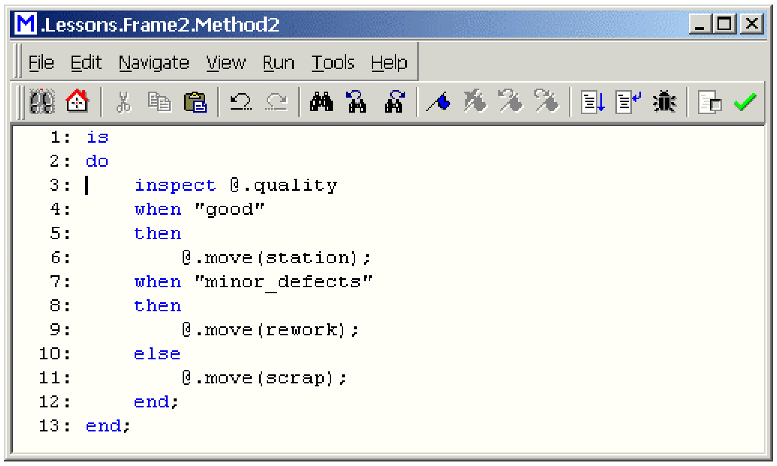


```
is
do
    if @.quality = "good"
    then
        @.move(station);
    elseif @.quality = "minor_defects"
    then
        @.move(rework);
    else
        @.move(scrap);
    end;
end;
```

The Inspect Statement

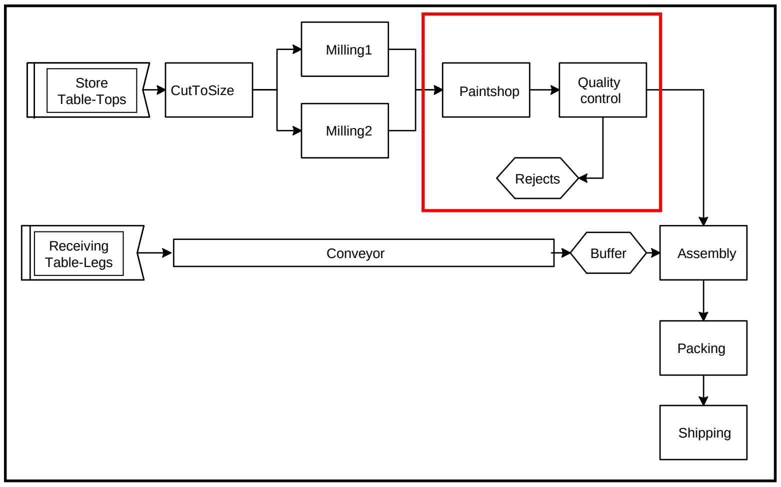


The *inspect* statement enables eM-Plant to make easier and clearer choices when presented with several possibilities. This way you do not have to use lengthy *if-then-elseif*-chains.Example of an Inspect Statement:



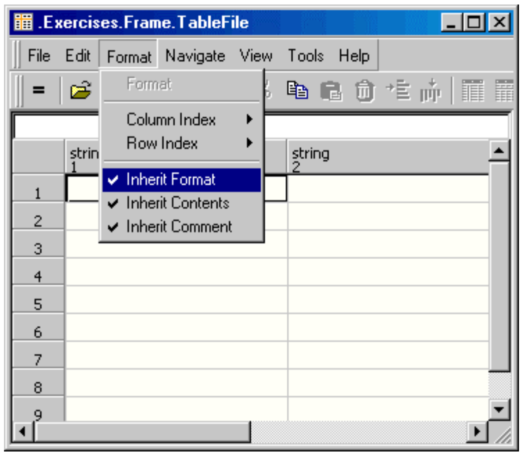
```
is
do
    inspect @.quality
    when "good"
    then
        @.move(station);
    when "minor_defects"
    then
        @.move(rework);
    else
        @.move(scrap);
    end;
end;
```

Lesson 9: Tables and Global Variables



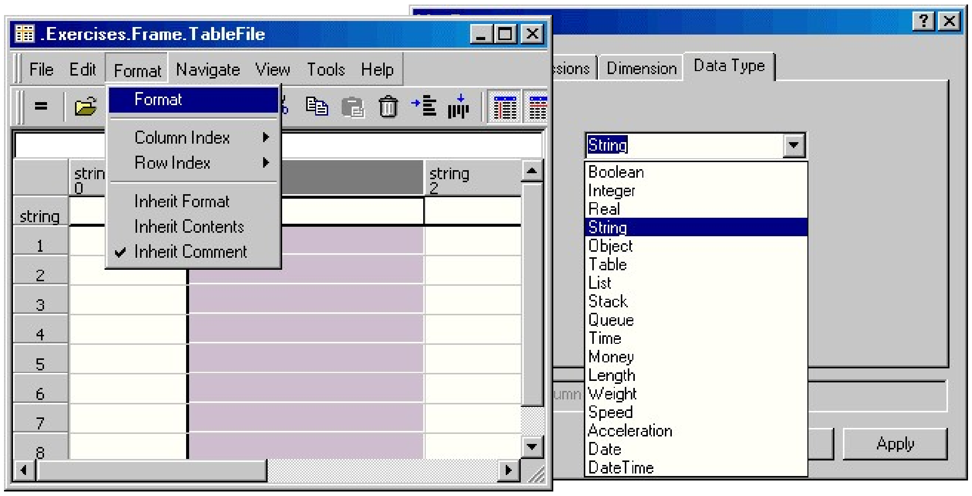
Introduction

The TableFile is a list with several columns that can be modified for content or format by deactivating the inherit commands from the Format menu.

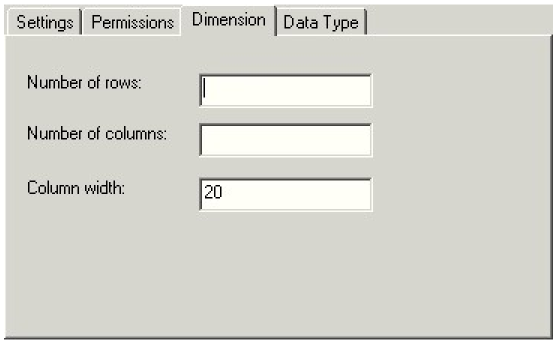


Formatting the TableFile

Before selecting a data type, select the entire table or a column of you choice and select **Format -> Format -> Data** and select the data type. Enter the dimension of the table:



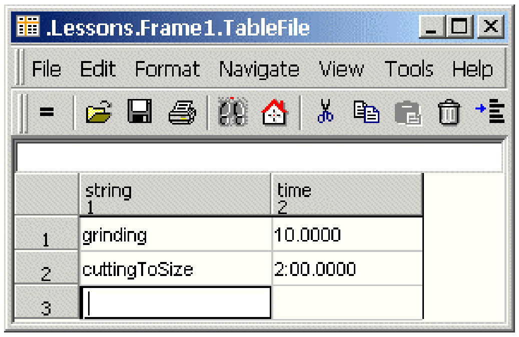
Last printed: 10/26/2008 2:03:00 PM
Last saved: 2/19/2004 1:54:00 PM



Number of rows:
Number of columns:
Column width: 20

Accessing the TableFile with a Method

When accessing the TableFile, eM-Plant first calls the column and then the row.


.Lessons.Frame1.TableFile
File Edit Format Navigate View Tools Help

	string 1	time 2
1	grinding	10.0000
2	cuttingToSize	2:00.0000
3		

➢ Reading from a table: print TableFile [1,2}; (column 1 row 2).

➢ Writing to a table: TableFile [1,3]: = “grinding”;

TableFile [2,3]: = 180;

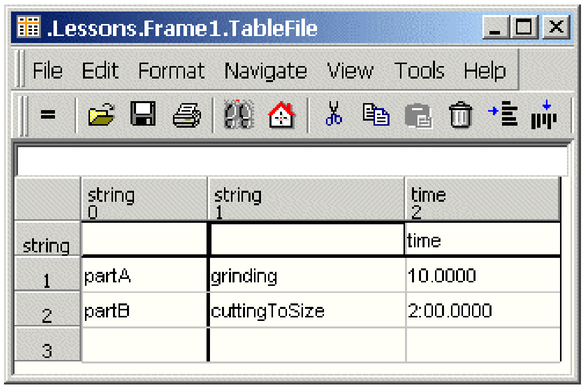
Here we assign the contents of a cell from the table to a variable, attribute, etc…

✚ <Object>.procTime: = TableFile [2,1];

Accessing a TableFile Using a Custom Index

You can also call a cell using the column and row index. Here we assign the processing time to a material flow object by using the contents of the cell in column 2, row 1 of the table.

✚ <Object>.procTime: = TableFile [“time”, “PartA”];


.Lessons.Frame1.TableFile
File Edit Format Navigate View Tools Help

	string 0	string 1	time 2
string			time
1	partA	grinding	10.0000
2	partB	cuttingToSize	2:00.0000
3			

Methods for Lists and Tables

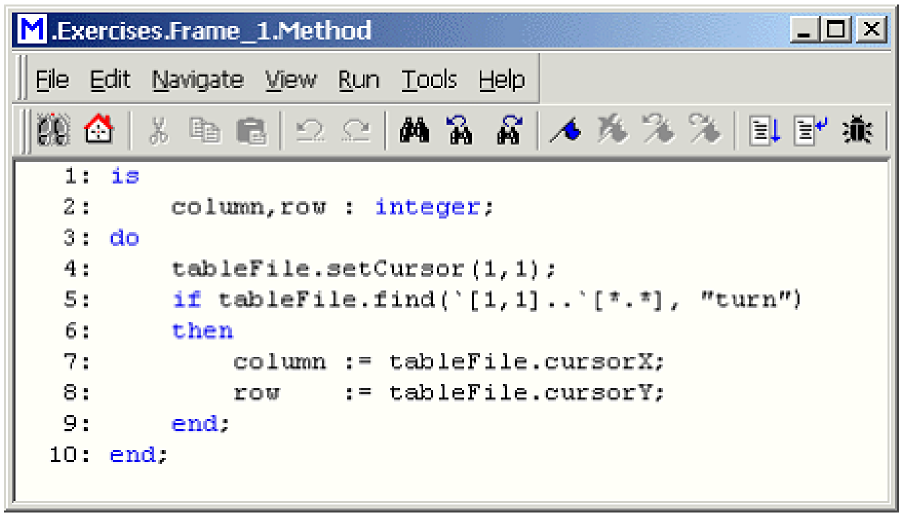
<TableFile>.setCursor(1,1);	Sets the cursor to the designated cell.
<...>.find(‘[1,1]..’[*,*],<value>);	Finds the value in the designated range.
<...>.CursorX;, <...>.CursorY;	Sets the cursor to a column (X) or to a row (Y)
<...>.xDim;, <...>.yDim;	Returns the last column or row with an entry.
<...>.sort(3,”ascending”);	Sorts the table

	depending on one or several columns in ascending order.
<...>.meanValue('[1,*]);	Returns the arithmetic mean of all values of data type real or integer in the range.
<...>.delete('[1,1]..[*,1]);	Deletes the range or the entire contents.
<local_variable>.create;	Creates the table as a local variable, without contents.

Searching a TableFile

eM-Plant searches the table starting at the position of the cursor and returns a "true" value when the item is found and places the cursor within that cell. If the item is not found a value of "false" is given

Here is an example of a Method searching a TableFile:


.Exercises.Frame_1.Method
File Edit Navigate View Run Tools Help

```
is
    column,row : integer;
do
    tableFile.setCursor(1,1);
    if tableFile.find(`[1,1]..`[*.*], "turn")
    then
        column := tableFile.cursorX;
        row    := tableFile.cursorY;
    end;
end;
```

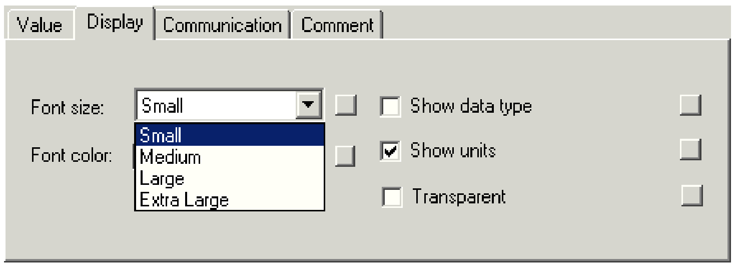
The Variable

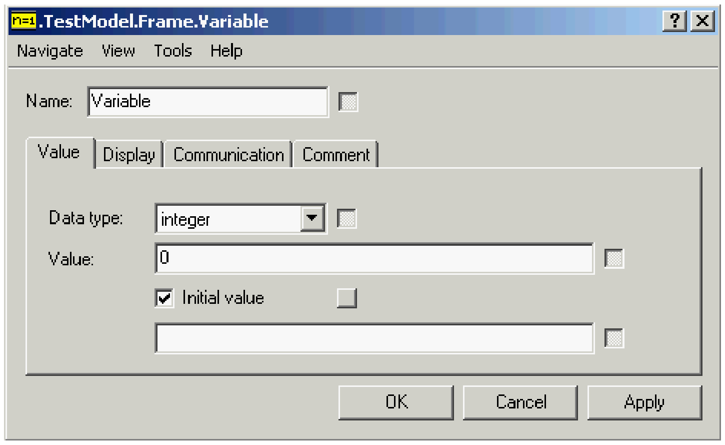
Capacity = 0

Information Flow Object

Use the variable to store values over an extended period of time during your simulation run. You can call it as any other object. The range of values depends on the data type. Other objects or methods can call the Variable.

To reset the value to an initial value during the init phase select the checkbox and enter the initial value.





Name: Variable

Data type: integer

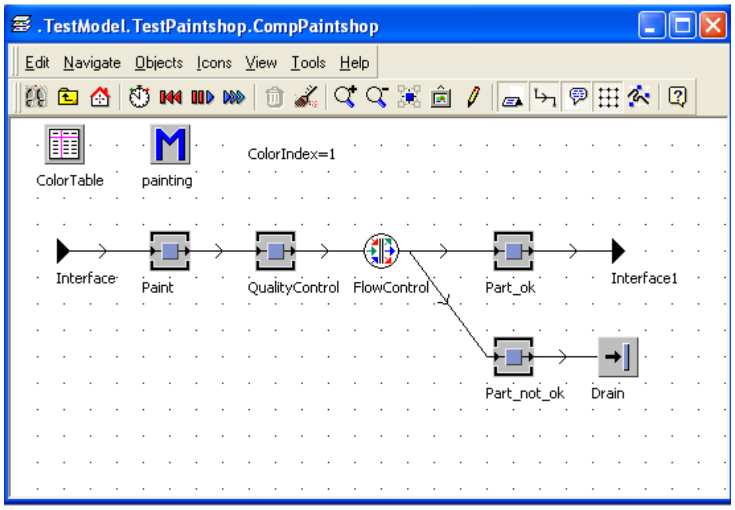
Value: 0

☑ Initial value

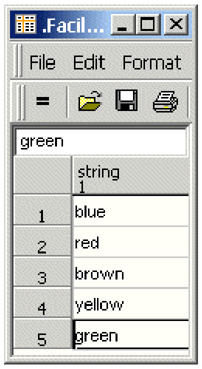
Exercise 1: Painting the TableTops

Objective: To write a method that will paint the TableTops.

1 Open the frame CompPaintShop in the FacilityComponents folder and add a TableFile (name it ColorTable) and a Method (named Painting).



2 Enter the colors from the table into the ColorTable.



	string 1
1	blue
2	red
3	brown
4	yellow
5	green

3 Into which control do you enter the calling Method?

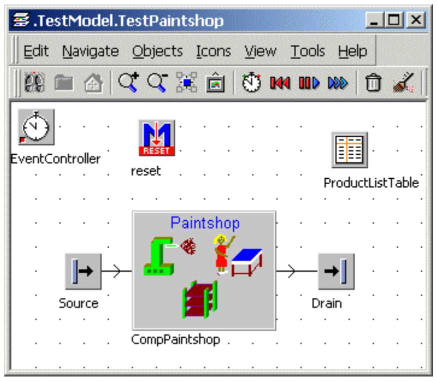
4 Program the method so that eM-Plant paints the TableTops with the colors from the table. Hint: breakdown the problem into small steps.

5 If required, insert all objects still missing.

Exercise 2: Test Painting

Objective: To test the new paint shop before inserting it into the project.

1 Open the frame TestPaintShop in the TestModels folder. Once you extended the frame PaintShop in the class of he of the frame PaintShop (within the folder FacilityComponents), the test frame TestPaintShop displays those changes.



2 Start the simulation and watch how the TableTops are painted in the frame.

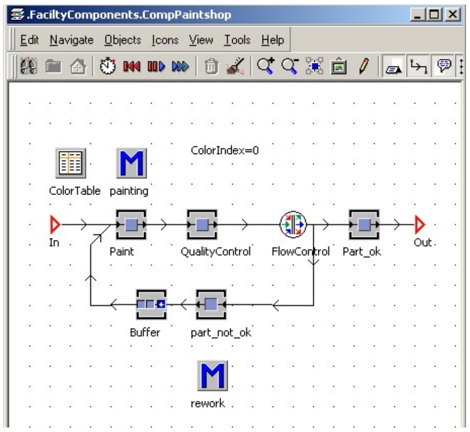
Exercise 3: The Paint Cycle

Objective: To determine a paint cycle to correct for the “bad” TableTops.

1 Open the CompPaintShop.

2 Delete the drain and insert a buffer with a capacity of 4.

3 Close the cycle to the station QualityControl by inserting a connector (see image below).



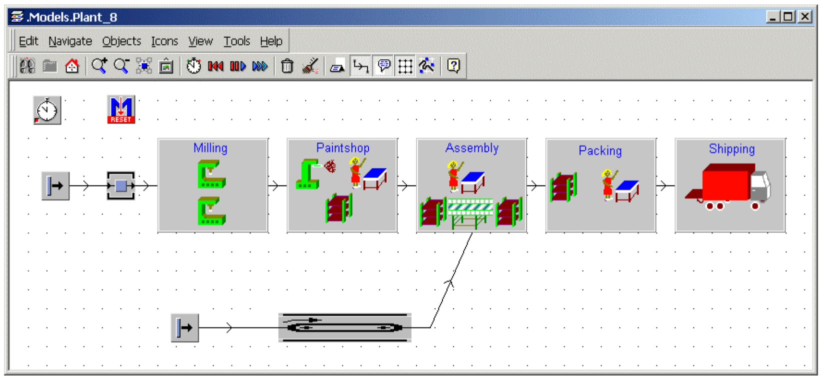
4 Insert a method and program it to set the attribute Quality to “good” and change the name to TableTop_good.

Exercise 4: Test and Add the Paint frame to Plant_8

Objective: To test the TestPaintShop for functionality and to insert the new frame into Plant_8.

1 Test the PaintShop in the frame TestPaintShop.

2 Insert the PaintShop frame into Plant_8.



The StackFile, QueueFile and CardFile

The *StackFile*, *QueueFile* and *CardFile* are lists with one column. They cannot have empty cells. They provide different features for reading, inserting and cutting the contents of a cell.

- *StackFile* (corresponds to the data type stack) inserts entries at the top and removes the contents of the cell added first (the LIFO method: Last In First Out).
- *QueueFile* (corresponds to the data type queue) saves entries added in the order inserted and removes the element waiting in the queue the longest.
- *CardFile* (corresponds to the data type list) is accessed in a random manner using the position of the cell, the following cells move up.

Method for Writing into StackFiles, QueueFiles and CardFiles.

<…>.push(<value>);	Inserts the value into the topmost cell of the StackFile or the QueueFile.
<…>.insert (3,<value>);	Inserts the value into the cell in a row (3) of the CardFile and moves the contents of this cell and the following cell down one.
<…>.[];	Removes the contents of the last cell of the StackFile and removes the contents of the first cell of the QueueFile. The following cells move up one position.

<…>.[3];	Reads and removes the contents of the cell in a row (3) of the CardFile. The following cells move up one position.
<…>.pop;	Reads the content of the last cell of the StackFile and reads the content of the first cell in the QueueFile and removes them.
<…>.read (3);	Reads the content of a row (3) of the CardFile without removing it.
<…>.dim;	Returns the last row with any entry.

Lesson 10: Operations Converting Data Types

Introduction

A condition is an expression that returns a Boolean value. Use conditions in conditional branching and loops.

Logical Operators

NOT	Not True -> False Not False -> True

=	False = False -> True
	True = True -> True

/=	False /= True -> True

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

Integer Operators

		Return value	
+	addition		
-	subtraction	integer	
*	multiplication		
//	integer division	integer	17//5 = 3
\\	modulo operator		17\\5 = 2
/	division	integer, real	
=	equal		
/=	not equal		
>	greater than	boolean	
<	less than		
>=	greater than or equal to		
<=	less than or equal to		

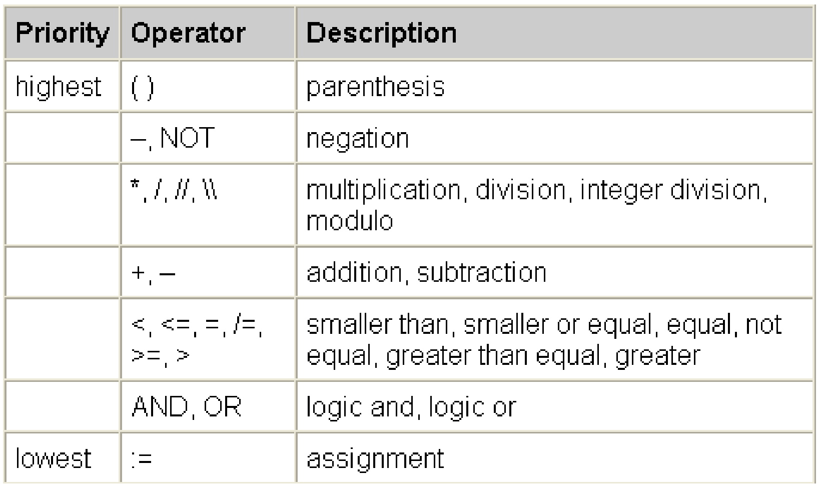
Real Operators

		Return value
+	addition	
-	subtraction	real
*	multiplication	
/	division	
=	equal	
/=	not equal	
>	greater than	
<	less than	
>=	greater than or equal to	boolean
<=	less than or equal to	
= =	about equal	
<= =	less than or about equal to	
>= =	greater than or about equal to	

String Operators

Operator	Name	Return value
+	addition	string
=	equal	boolean
/=	not equal	boolean
==	about equal	boolean
	toLower	string
	toUpper	string
	copy	string
	incl	string
	omit	string
	strlen	integer
	pos	integer

Operator Precedence



Priority	Operator	Description
highest	( )	parenthesis
	–, NOT	negation
	*, /, //, \\	multiplication, division, integer division, modulo
	+, –	addition, subtraction
	<, <=, =, /=, >=, >	smaller than, smaller or equal, equal, not equal, greater than equal, greater
	AND, OR	logic and, logic or
lowest	:=	assignment

Methods for Converting Data Types

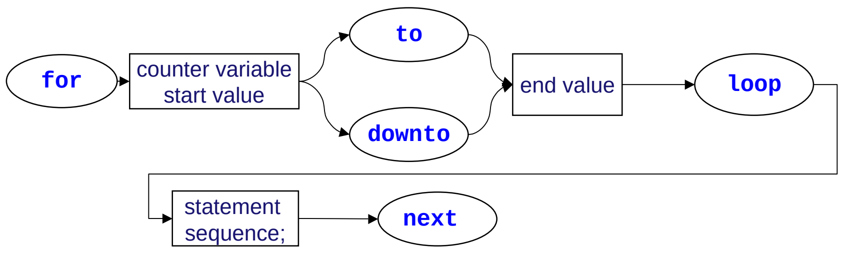
Data types can perform certain actions. Not all operations can be applied to all data types.

Before concatenating different data types, you might have to convert them to make them compatible. Only then can eM-Plant execute the operations.

Method	Returns
to_str(<...>)	string
num_to_bool(<integer>)	boolean
bool_to_num(<boolean>)	real
str_to_num(<string>)	real
str_to_bool(<string>)	boolean
str_to_time(<string>)	time
str_to_date(<string>)	date
str_to_datetime(<string>)	datetime
str_to_length(<string>)	length
str_to_weight(<string>)	weight
str_to_speed(<string>)	speed
str_to_obj(<string>)	object

Lesson 11: Loops

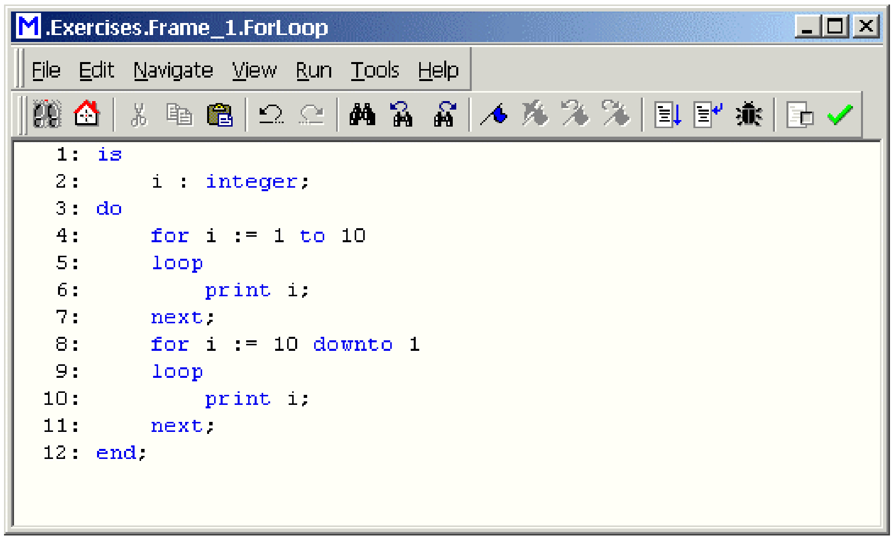
Introduction



The For Loop

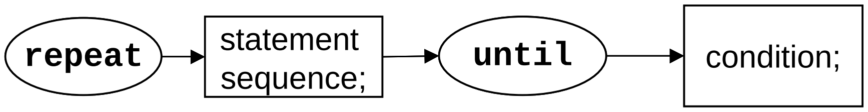
The *for* loop passes over a range between a beginning and an end value. The loop variable has to be of data type *integer* and denotes the beginning of the loop. Once the loop is closed, eM-Plant adds 1 to the variable (setting *to*) or deducts 1 (setting *downto*). It executes the loop until it reaches the end values.

Here is an example of a for loop where i is the loop variable and 1 is the step size:


.Exercises.Frame_1.ForLoop
File Edit Navigate View Run Tools Help

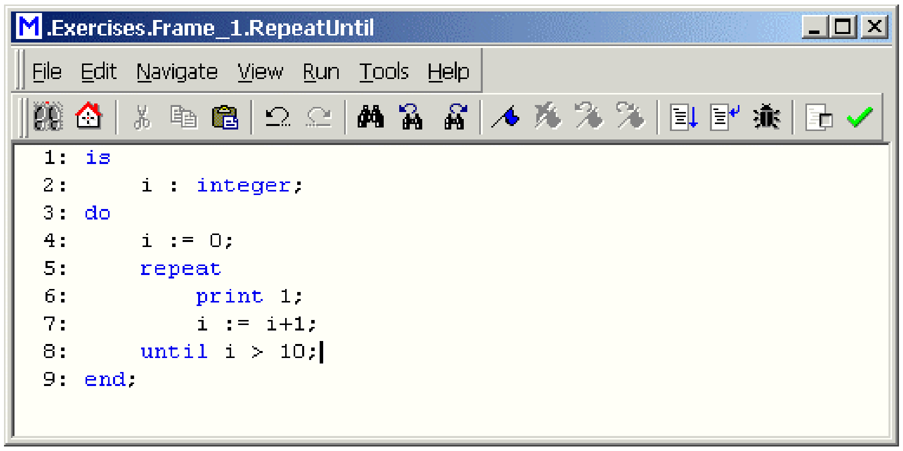
```
is
	i : integer;
do
	for i := 1 to 10
	loop
		print i;
	next;
	for i := 10 downto 1
	loop
		print i;
	next;
end;
```

The Repeat Until Loop



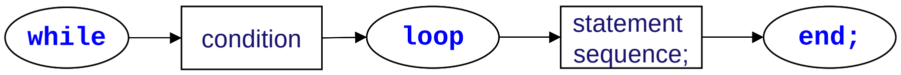
eM-Plant executes the repeat loop at least once and exits when the condition is true. If the condition never is true, eM-Plant executes the loop infinitely. You can interrupt an infinite loop by pressing Shift + Alt + Ctrl, this opens the debugger.

Here is an example of a Repeat Until loop:


.Exercises.Frame_1.RepeatUntil
File Edit Navigate View Run Tools Help

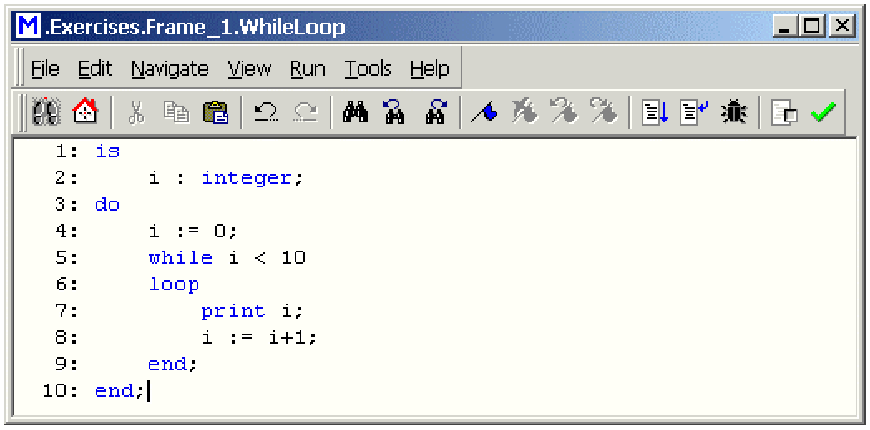
```
is
    i : integer;
do
    i := 0;
    repeat
        print 1;
        i := i+1;
    until i > 10;
end;
```

The While Loop



eM-Plant executes the while loop until the condition is true. Once the statement sequence is completely processed, eM-Plant checks again if the condition is true.

Here is an example of a While Loop:


.Exercises.Frame_1.WhileLoop
File Edit Navigate View Run Tools Help

```
is
    i : integer;
do
    i := 0;
    while i < 10
    loop
        print i;
        i := i+1;
    end;
end;
```

Example: Loops

Objective: To create methods using the loops discussed.

1 Add the frame Loops to the Exercises folder.

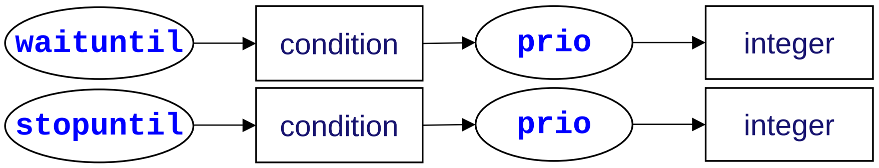
2 Add a Method and program a loop that passes over a range of values between 1 and 10.

3 Print the values of the loop to the console.

4 Step through the Method and watch the results pf the loop variable in the Watch window..

Lesson 12: Conditional Suspensions – waituntil and stopuntil

Introduction



Use the waituntil statement to suspend a method according to a condition.

If the condition is not true, eM-Plant interrupts the execution of the method program. It suspends the method and saves the entire call chain, including all arguments and local variables. It then monitors the condition while the simulation continues.

As soon as the condition is true, eM-Plant interrupts the execution of the active methods and continues executing the once suspended method at the place in its source code where it was suspended.

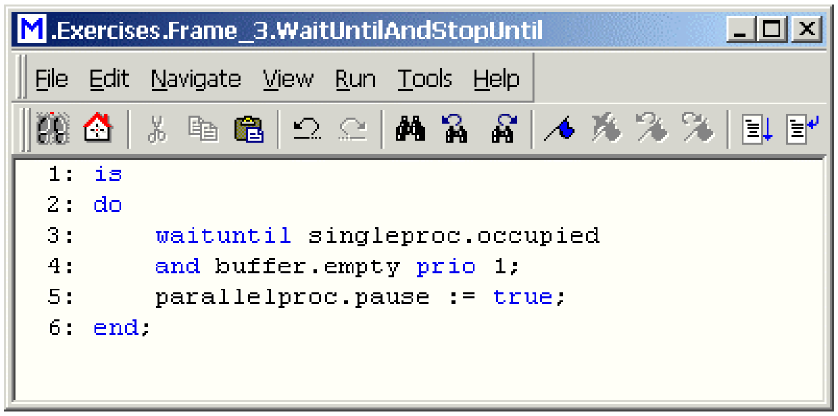
The condition has to be watchable; it has to return a Boolean expression and has to be true.

When several suspended methods wait for the same condition to be true, eM-Plant reactivates them at the same time and executes them according to priority (prio) that was entered.

For methods suspended with waituntil, the interpreter re-evaluates the condition before executing the individual methods. The interpreter again suspends the other methods, which have to wait until the condition comes true again.

For methods suspended with stopuntil, eM-Plant does not re-evaluate the condition again before executing the individual methods. It reactivates all of the methods at the same time and executes them one after the other according to their priority.

Here is an example of the waituntil and stopuntil methods:

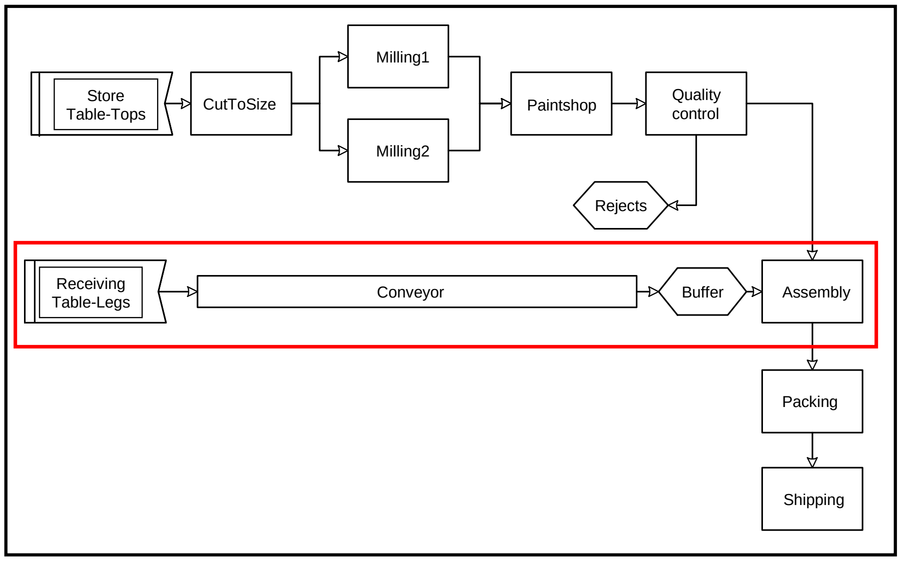

.Exercises.Frame_3.WaitUntilAndStopUntil
File Edit Navigate View Run Tools Help

```
is
do
    waituntil singleproc.occupied
    and buffer.empty prio 1;
    parallelproc.pause := true;
end;
```

NOTES:
Enter the maximum number of suspended methods under Options -> Simulation.

Lesson 13: The Contents of an Object Loading and Unloading

Introduction

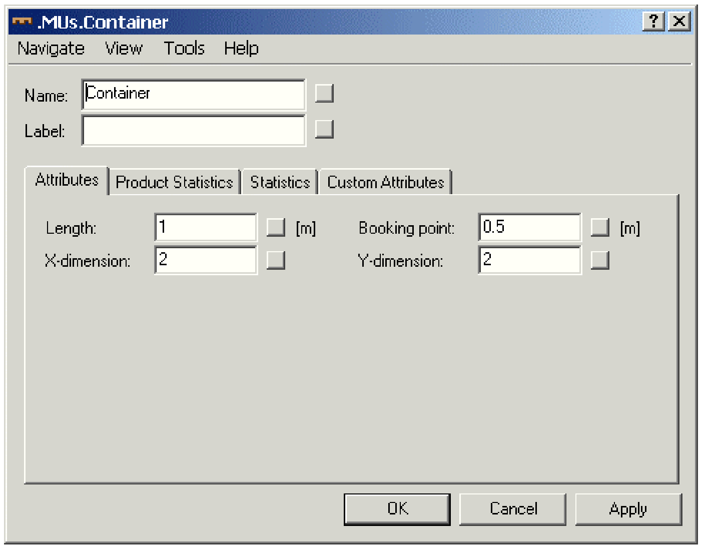


The Container

The container has a matrix based loading space. It can load and transport any moving unit.

Enter the capacity on the x-axis and the y-axis. The container does not take the actual length of the loaded object into consideration.

Containers and transporters that the containers load may themselves, be loaded with other moving units.



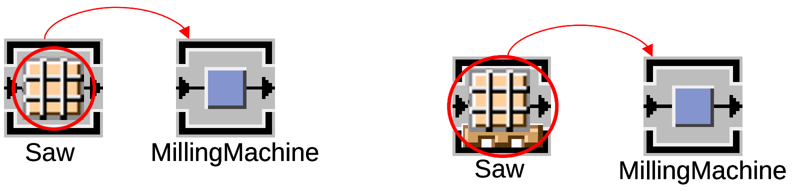
Name: Container
Label:
Length: 1 [m]
Booking point: 0.5 [m]
X-dimension: 2
Y-dimension: 2

Accessing the Contents of an Object

Use the method cont to call the contents of the material flow objects and the container. Enter <object>.cont or:

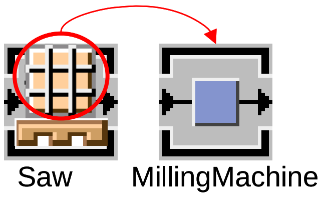
- @.move(millingmachine);
- saw.cont.move(millingmachine);

The method cont always returns the next MU that is ready to exit the object, even when it loaded other MUs (container).



Accessing the contents of a loaded container by calling the contents of the station and the contents of he container.

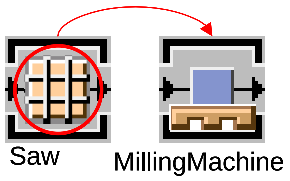
- @.cont.move(millingmachine);
- saw.cont.cont.move(millingmachine);



Use the method cont to move another MU to the container.

➢ @.move(millingmachine);

➢ saw.cont.move(millingmachine);



This also applies when moving containers loaded onto other containers.

The contents of a Material Flow Object

Before accessing the contents of an object, make sure that the object has contents. Use one of the state queries for the contents of a material flow object for this.

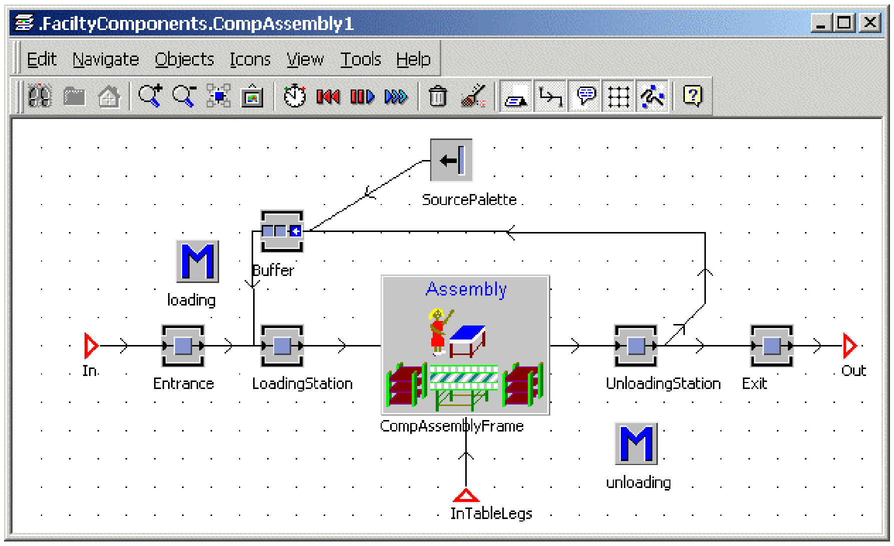
A material flow object can be in different states, which you can set and get with attributes. The respective method returns true when the object is in that state, false, when it is not in that state.

Methods for the Contents of a Material Flow Object/MU:

`<object>.empty;`	returns TRUE, when no MU is located on the station, FALSE when an MU is located on it.
`<...>.full;`	returns TRUE, when all stations of a *ParallelProc* or a *Buffer* are occupied, FALSE when this is not the case.
`<...>.occupied;`	returns TRUE, when at least one MU is located on the station, FALSE when no MU is located on it.
`<...>[2,1].occupied;`	returns TRUE, when an MU is located on the station denoted by the integer coordinates.
`<...>.cont.finished;`	returns TRUE, when the processing time of the MU has elapsed and the MU is ready to exit the object.
`<...>.ready;`	returns TRUE if the object is occupied and an MU is ready to exit. Combination of **`occupied`** and **`cont.finished`**. (Note: cannot be watched with **`waituntil`**.)

Exercise 1: Creating the Assembly Loading Cycle

Objective: To create the assembly loading cycle.



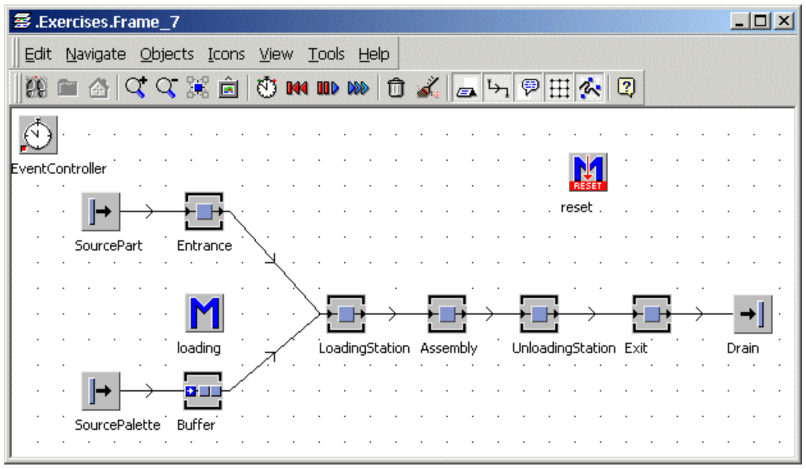
We are going to load the TableTops onto palettes, which transport them to the assembly station. Here the TableLegs are added to the TableTops. Then the assembled tables move out of the assembly station, while the palettes are fed back to the buffer to be used again.

Required to model successfully:

- ❖ Load the containers
- ❖ Unload the containers
- ❖ Loading and Unloading Stations
- ❖ Assembly process

1 Add the frame Frame_7 to Exercises folder.

2 Insert the stations so they match the image below.



3 The source part produces TableTops and the SourcePalette creates containers.

4 Insert a buffer with a capacity of 10 parts and a method named loading.

5 Program the method to load the TableTops onto the containers; think about where the coinciding events take place and where to react to them.

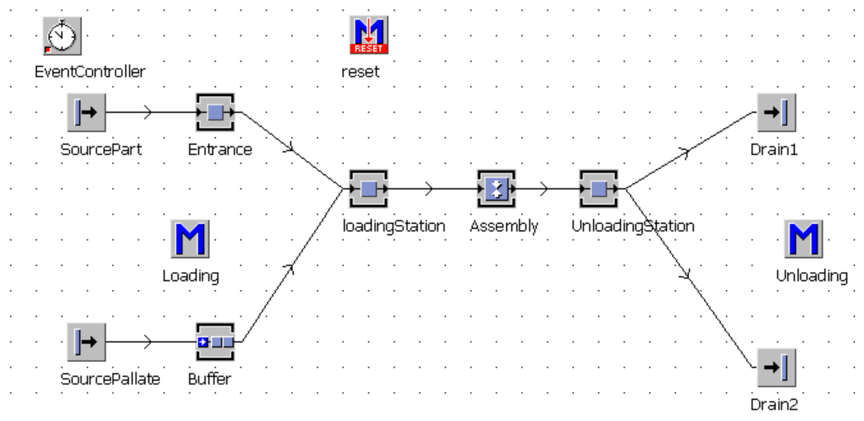
6 Test the loading process. Keep in mind that it has to work even when the MUs accumulate.

Exercise 2: Creating the Assembly Unloading Cycle

Objective: To create an assembly unloading cycle.

1 Duplicate the frame Frame_7 in the Exercises folder and name it Frame_8.

2 Add the objects necessary and connect them as shown in the image below.



3 The method is to move the TableTops to the drain1 and the containers to the drain2.

4 Before programming the method, think about where to react to coordinate the coinciding events.

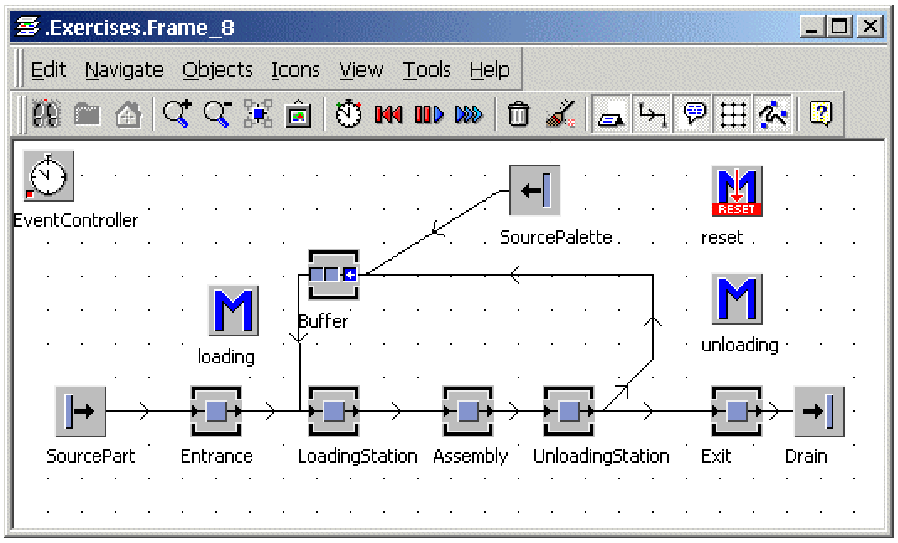
5 Make the MUs accumulate to test the unloading process and to make sure that it works.

Exercise 3: The palette cycle

Objective: To create the palette cycle.

1 Duplicate Frame_8 in the Exercises folder and name it Frame_9.

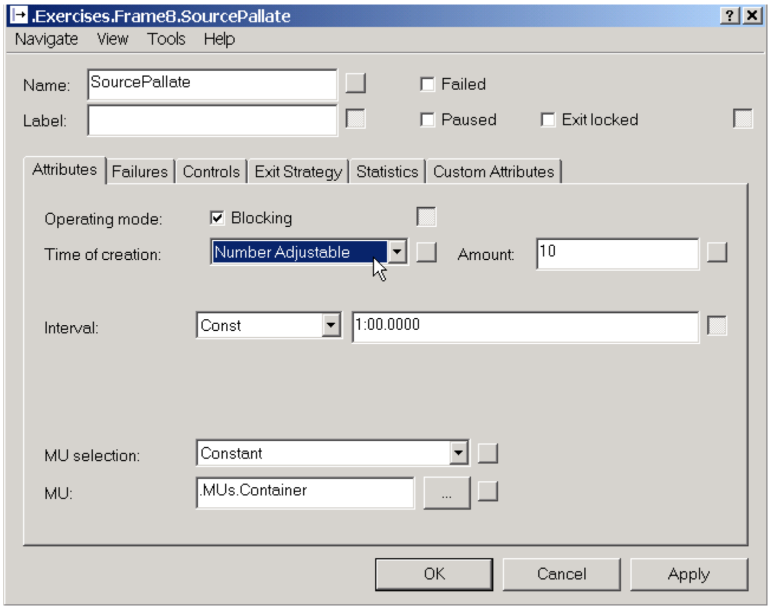
2 Delete the drain1 and reorganize the frame to feed the now unloaded containers back to the buffer (see image below).



3 We need the station exit only for unloading the TableTops from the container. It has a processing time of 0 seconds.

4 You may have to change the source code of the method unloading to accommodate the changed situation.

5 The SourcePalette produces 10 palettes at a time only (select number adjustable).


.Exercises.Frame8.SourcePallate
Navigate View Tools Help
Attributes | Failures | Controls | Exit Strategy | Statistics | Custom Attributes
OK Cancel Apply

Name: SourcePallate
Label:
Failed
Paused
Exit locked
Operating mode: ☑ Blocking
Time of creation: Number Adjustable
Amount: 10
Interval: Const 1:00.0000
MU selection: Constant
MU: .MUs.Container

6 Make the MUs accumulate to test the unloading process.

Accessing the Contents of a Location Oriented Object

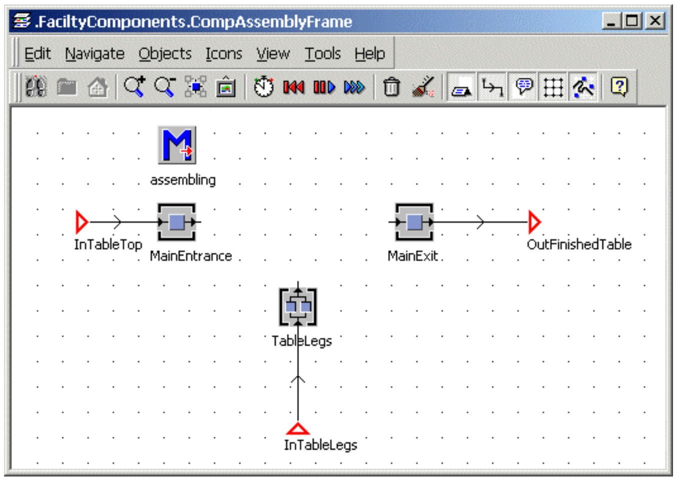
Access the contents with these methods:

<...>.mu(3);	returns the MU with the number 3 that is ready to exit.
<...>[x,y].cont;	returns the MU located at the position [x,y] of the matrix.
<singleproc>.mu(1).move;	moves the MU with the number 1, that is ready to exit on to the succeeding station.
@.move(<parallelproc>[2,2]);	moves the MU to the position [2,2] of the object.

Exercise 4: Attaching the Legs to the TableTops

Objective: To attach the TableLEgs to the TableTops using Methods.

1 Add the frame CompAssemblyFrame to the FacilityComponents folder and insert the objects to match the image below.


.FaciltyComponents.CompAssemblyFrame
Edit Navigate Objects Icons View Tools Help
assembling
InTableTop
MainEntrance
MainExit
OutFinishedTable
TableLegs
InTableLegs

2 The ParallelProc has 4 stations for the four TableLegs and no processing time.

3 Program the method to model the assembly process. How do you take care of the different events.

4 Create an Icon for the frame and animate the line.

Exercise 5: Testing the assembly and CompAssemblyComplete

Objective: To test the functionality of the Assembly processes.

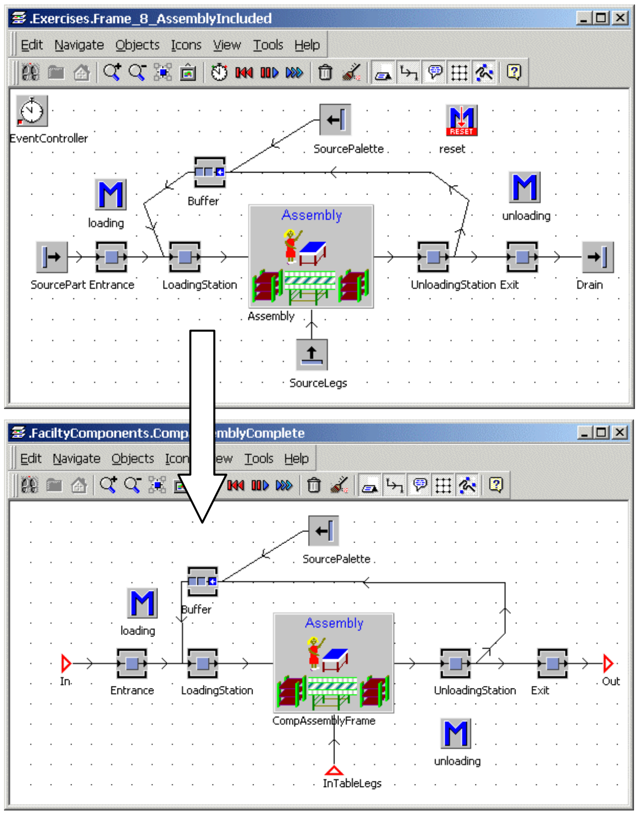
1 Duplicate Frame_9 and name it CompAssemblyComplete.

2 Replace the SingleProc Assembly with the frame CompAssemblyComplete frame just created.

3 Change the MU type to TableLegs in the SourceLegs.

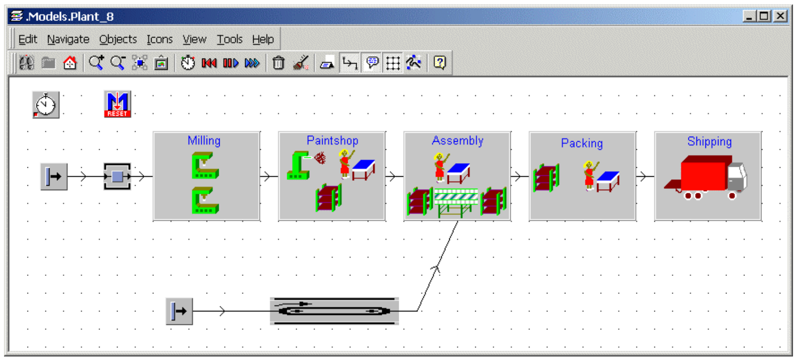
4 Test the functionality of the process.

5 Replace the SourcePart, SourceLegs and the Drain with inteface objects (see images below).



Exercise 6: TestAssembly and Plant_8

Objective: Replace the Assembly frame with the one just created and simulate the process.

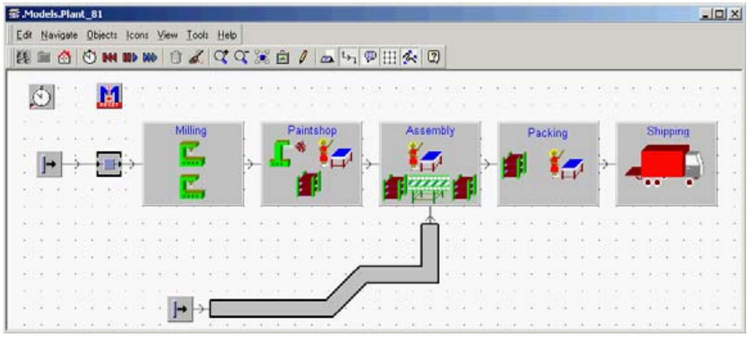


Exercise 7: Conveyor for the TableLegs

Objective: To insert a conveyor for the TableLegs.

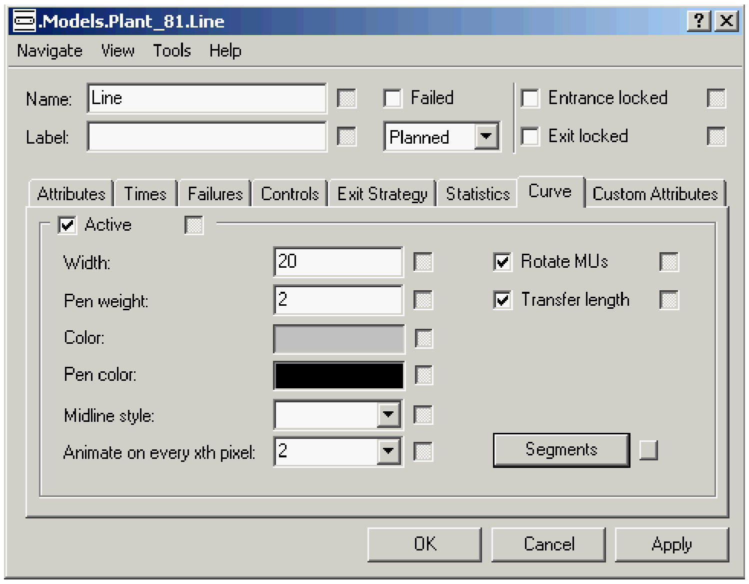
1 Add the conveyor to the simulation model. The conveyor can be any curve.

2 Edit the anchor points of the curve.



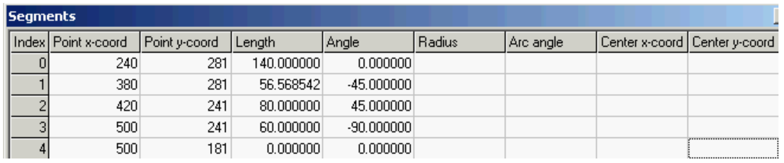
Line – Define the Curve

To insert the line as a curve, not only as an object, activate the curve mode. The segments table shows the parameters of any segment of the curve.



Name: Line
Label:
[ ] Failed
Planned
[ ] Entrance locked
[ ] Exit locked

[✓] Active
Width: 20
Pen weight: 2
Color:
Pen color:
Midline style:
Animate on every xth pixel: 2
[✓] Rotate MUs
[✓] Transfer length



Segments

Index	Point x-coord	Point y-coord	Length	Angle	Radius	Arc angle	Center x-coord	Center y-coord
0	240	281	140.000000	0.000000				
1	380	281	56.568542	-45.000000				
2	420	241	80.000000	45.000000				
3	500	241	60.000000	-90.000000				
4	500	181	0.000000	0.000000				

Line Attributes

The line calculates its length using the length of the curve. If this is not useful, this feature may be switched off on the tab *Curve*.

Lesson 14: Calling Methods; Trigger and Frame Controls

Introduction

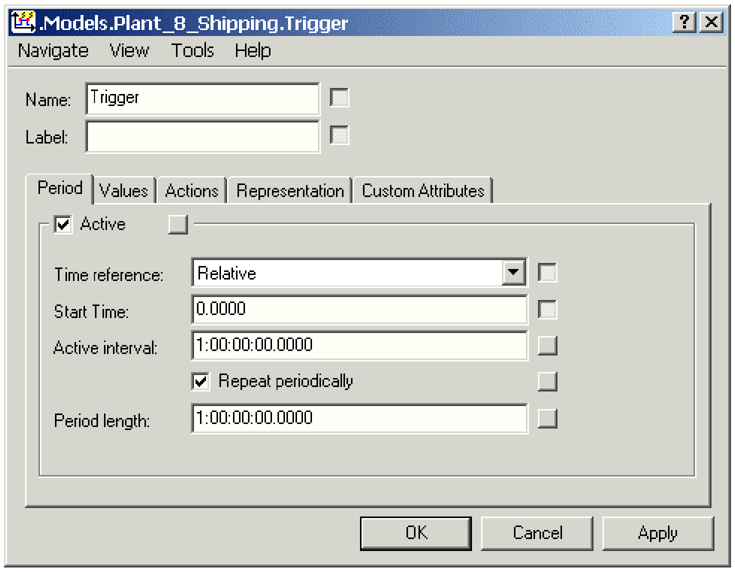
The **Trigger** changes values of attributes and global variables according to a custom pattern you define during a simulation run. It can activate methods also, which then execute the actions you programmed.

The Trigger maps times to the values of a variable. Custom attributes take values designated by the Trigger during a simulation run. Specify the values either in a single TimeSequence object or by combining several time sequences of other Triggers. In addition, a Trigger can control how and when a Source object creates MUs.

Use the trigger to control processes that take place at a certain point in time.

The trigger can control attributes or trigger methods. When the method is called, the trigger passes two arguments to the method: the last value passed by the trigger and the current value at that point in time the trigger was activated.

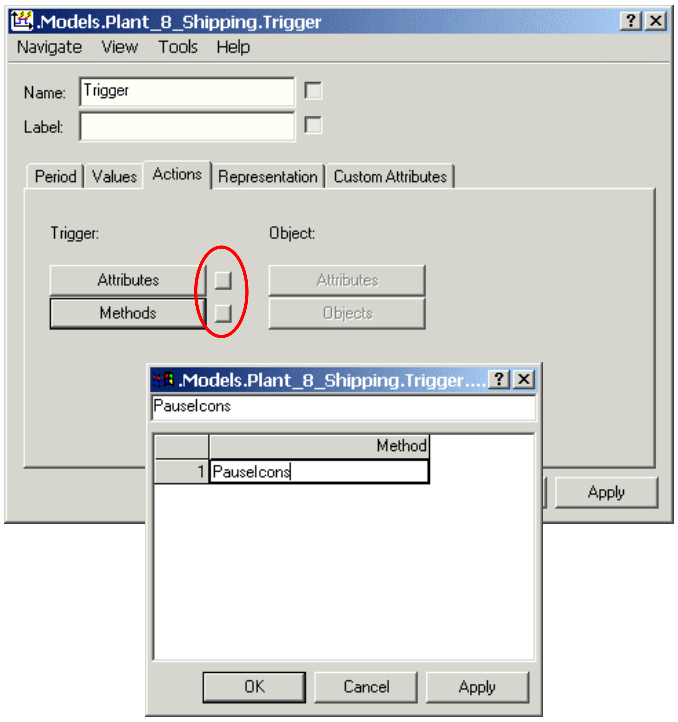
Enter the corresponding values into the values table:



Name: Trigger
Label:
[✓] Active
Time reference: Relative
Start Time: 0.0000
Active interval: 1:00:00:00.0000
[✓] Repeat periodically
Period length: 1:00:00:00.0000

Before opening the methods table, deactivate the inheritance and click Apply.

Enter the method or methods the trigger calls. eM-Plant calls the method at the time you entered and executes the program. The previous value and the new value have to be passed to the method.


.Models.Plant_8_Shipping.Trigger

Name: Trigger

Label:

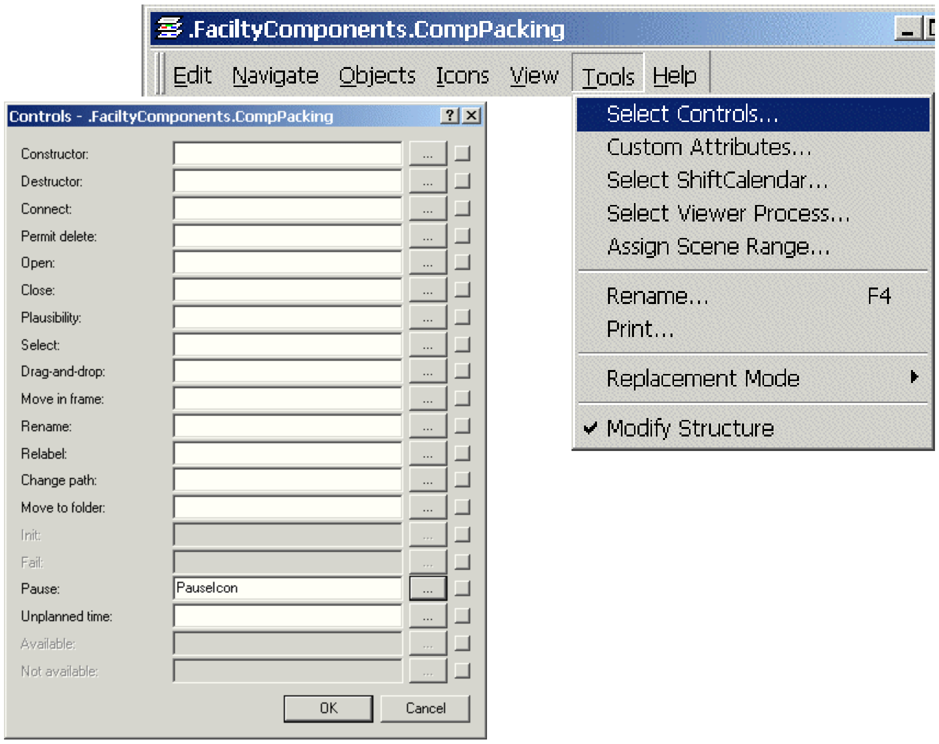
Trigger: Object:

.Models.Plant_8_Shipping.Trigger....

PauseIcons

	Method
1	PauseIcons

Method Called by a Frame Control



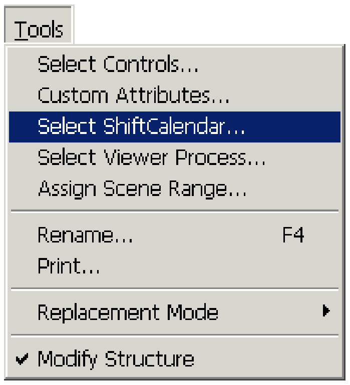
Controls - .FacilityComponents.CompPacking

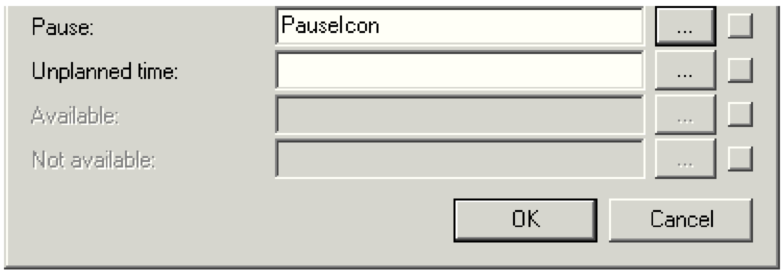
Constructor:	
Destructor:	
Connect:	
Permit delete:	
Open:	
Close:	
Plausibility:	
Select:	
Drag-and-drop:	
Move in frame:	
Rename:	
Relabel:	
Change path:	
Move to folder:	
Init:	
Fail:	
Pause:	PauseIcon
Unplanned time:	
Available:	
Not available:	

Select *Tools* -> *Select Controls* and enter the names of the methods for a number of purposes into the material flow objects or the frame.

A shift calendar can also control the frame, select *Tools* -> *Select Shift Calendar*. Program a method that modifies the state *Pause*, which activating the shift calendar sets.

The method entered into the test box Pause is called and executed whenever the attribute Pause changes its state.



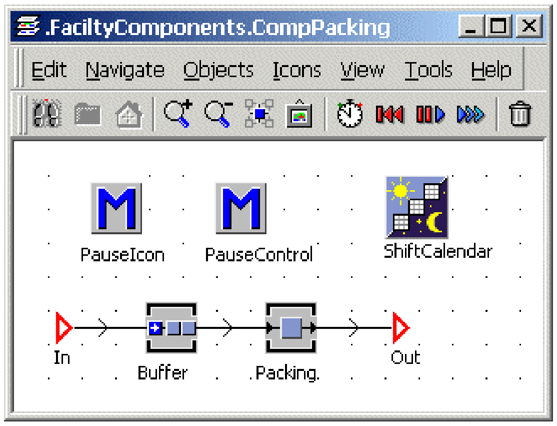


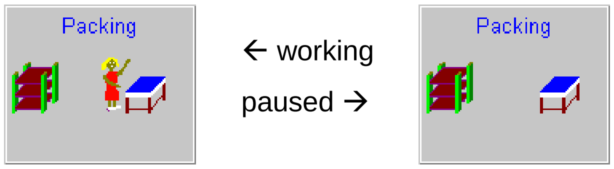
Pause: PauseIcon
Unplanned time:
Available:
Not available:

Exercise: The Frame Control

Objective: To create the packing frame using the shift calendar.

1 Create another Icon for the frame Packing in the Icon editor and enter a different name.


.FacilityComponents.CompPacking
Edit Navigate Objects Icons View Tools Help
PauseIcon
PauseControl
ShiftCalendar
In
Buffer
Packing
Out



2 Enter the method PauseIcon as a control into the frame CompPacking.

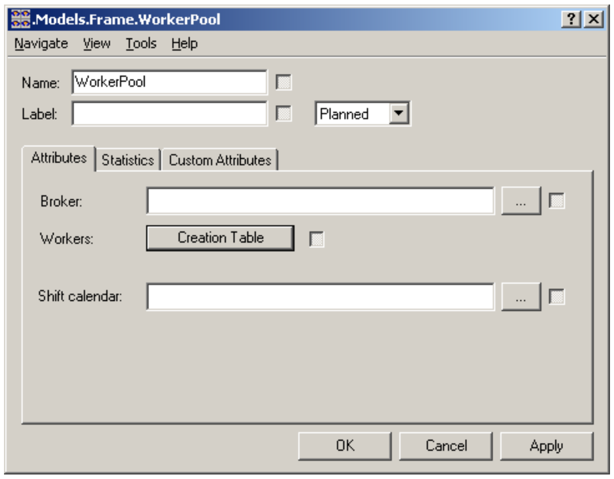
3 Activate the ShiftCalendar for the frame CompPacking and the method PauseIcon for pausing the frame.

4 Program the method that activates the corresponding Icon, whenever the pause state of the frame changes.

Lesson 15: Resource Objects

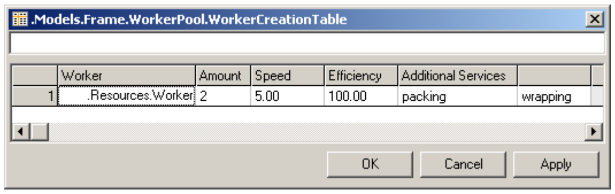
Introduction

- Workerpool: amount, skill level, walking speed, efficiency, and shift model.



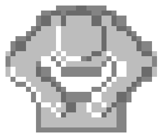
Name: WorkerPool
Label:
Planned

Broker:
Workers: Creation Table
Shift calendar:

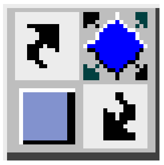


	Worker	Amount	Speed	Efficiency	Additional Services	
1	.Resources.Worker	2	5.00	100.00	packing	wrapping

- Workplace: Physical location of an operator at the station

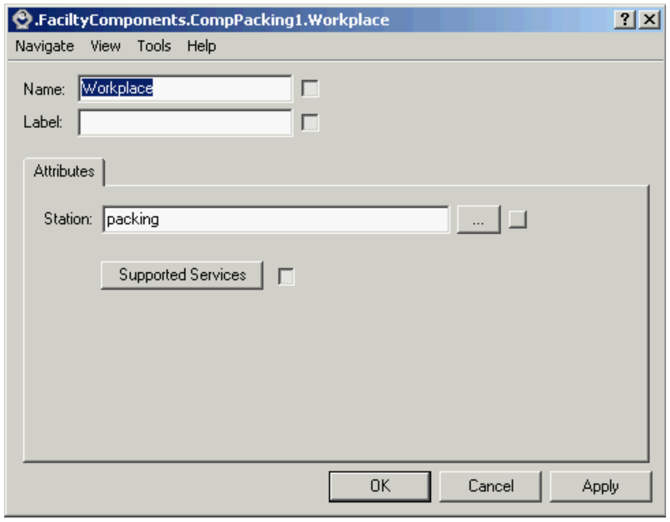


- Broker: Resource manager, receives operator requests and assigns operators to stations.



Assigning a Workplace to a Station

- Enter the path of the station in the text box Station.
- Drag the station onto the workplace and drop it there.
- Insert the workplace close to a station. This will automatically assign the station to that workplace.



.FaciltyComponents.CompPacking1.Workplace

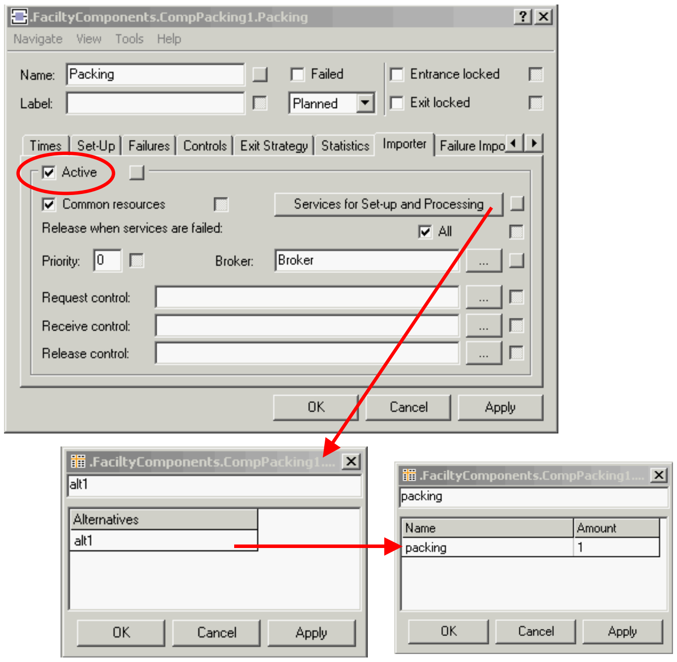
Name: Workplace

Label:

Station: packing

Defining services of a Station

- Open the station (SingleProc, ParallelProc, etc...) and select the tab Importer.
- Activate the importer. Select the broker which manages the services.
- Open the service table, enter any value for the alternatives. Enter the service and the amount needed into the suitable alternative.



Name: Packing
Label:
☐ Failed
Planned
☐ Entrance locked
☐ Exit locked
☑ Active
☑ Common resources
Release when services are failed: ☑ All
Priority: 0
Broker: Broker

alt1

Alternatives
alt1

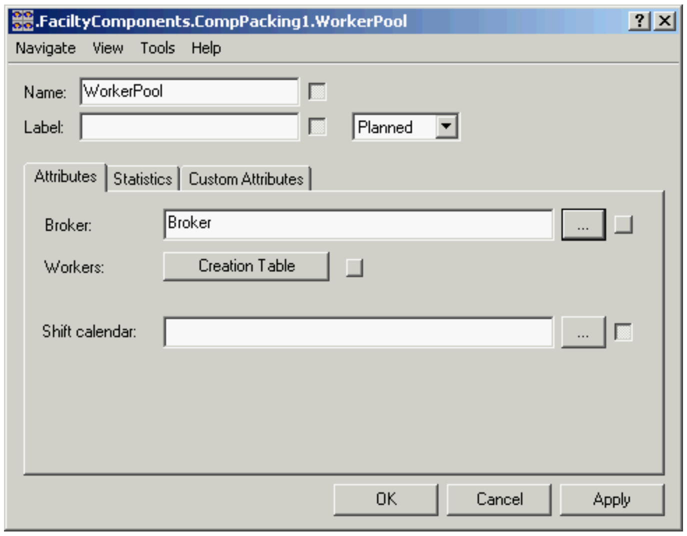
packing

Name	Amount
packing	1

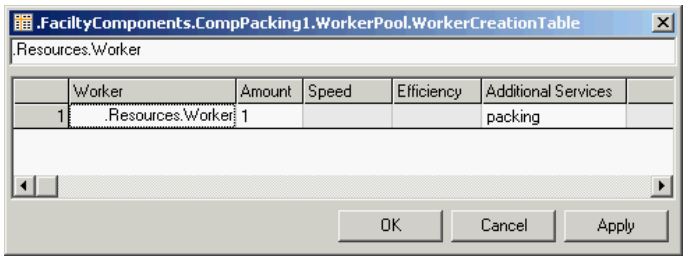
Defining the Worker from the Workerpool

- Open the workerpool.
- Enter the broker object.

- Open the Creation Table for defining the workers that are to be available.
- Enter the service the worker is providing.



Name: WorkerPool
Label:
Planned
Broker: Broker
Workers: Creation Table
Shift calendar:



.FaciltyComponents.CompPacking1.WorkerPool.WorkerCreationTable
OK
Cancel
Apply

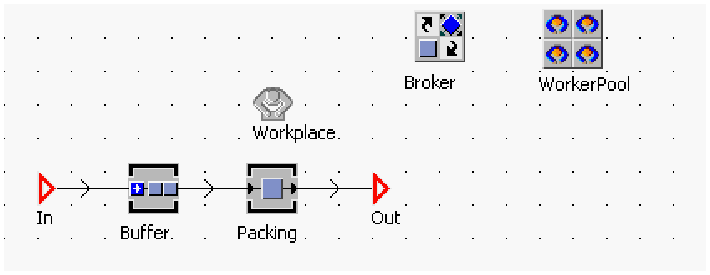
.Resources.Worker

	Worker	Amount	Speed	Efficiency	Additional Services
1	.Resources.Worker	1			packing

Exercise: Worker Assignment

Objective: To assign a worker to Packing.

1 Add the required objects.



2 Define the service packing, as a required service in the station.

3 Define the workerpool and assign the broker as a responsible resource manager.

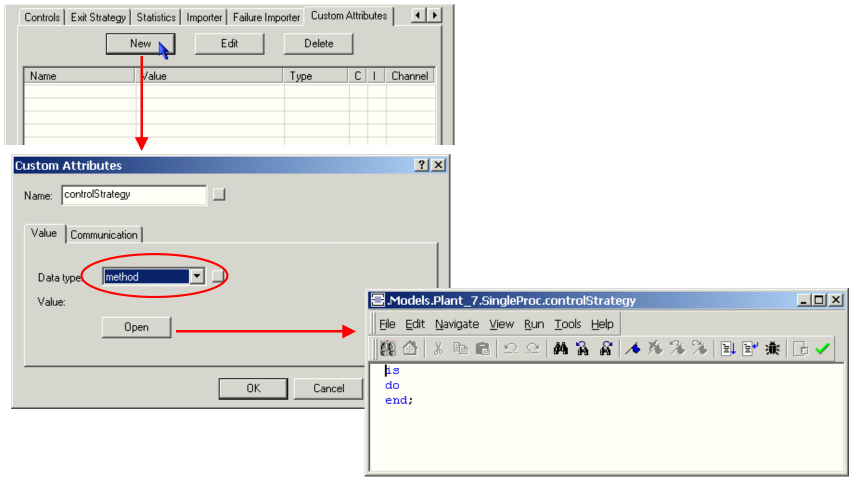
Hint: Do not forget to set the importer to active.

Lesson 16: Miscellaneous Features

A Method as a Custom Attribute

Attach a method as a cutom attribute of a data type method to an object. Program methods that are stored and handled directly by this object.

You can then program controls in the method that apply to this material flow object.



Name: controlStrategy

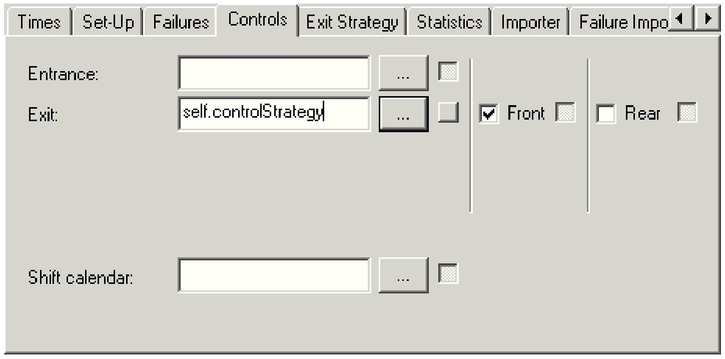
Data type: method

Value:

```
is
do
end;
```

The statement self.name_of_method calls and executes the method you programmed as a custom attribute.

This method is not a method object, but an attribute of type method of the object. This method is part of the object, even when you copy it or save it as an object.



Entrance:

Exit: self.controlStrategy

Shift calendar:

Miscellaneous Methods

- CallEvery(<path>,<method>,<argument>)
  - Calls all methods designated by the argument method in the frame <path> on all levels of the hierarchy. Arguments are passed as arguments to the calling methods.
- Ref(<method>).methCall(<time/date/datetime>,<argument>)
  - Calls a method after the number of seconds designated by <time> has passed. For <datetime> or <date> eM-Plant calculates the time span using the date of the EventController and the date you entered as the arguments <datetime> or <date>. Arguments are passed when the method is called next.
- Wait(<real>)
  - Interrupts executing a call chain for the number of seconds passed as <real>. Hte EventController receives a MethWakeup event.

Built-in Method Icons

M RESET	Reset: Resets your simulation model. It deletes all unprocessed events, resets the simulation time to 0, resets the statistics, and clears any failure of any machine.
M INIT	Init: eM-Plant activates and executes all methods named Init (even the Init controls of a Transporter).
M ENDSIM	Endsim: The simulation ends when the EventController has processed all events from the List of Scheduled Events or when eM-Plant reaches the end time you entered in the text box End on the tab Settings in the EventController.
M	Default: the icon used in the class library.
M	ExitCtrl: an exit control.
M	EntranceCtrl: an entrance control.
M	Error: a method with errors.
M	User
M	Interface
M	Dialog

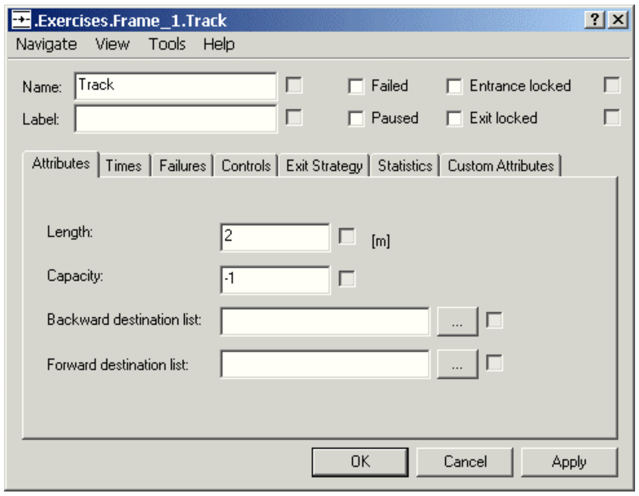
The Track

The Track is used for modeling transport lines. It also supports automatic routing. The Transporter is the only movable object able to use the Track. You might, for example, utilize both to model an AGV system. You can also change the dimension of the Track.

Properties:

- Icon
- Capacity: any
- Length oriented material flowobject

You can enter its length and its Capacity. The Transporter is the only moving material flow object able to use the track in a meaningful way. The container and entity remain where they are and will not be moved.



Name: Track
Label:
Length: 2 [m]
Capacity: -1
Backward destination list:
Forward destination list:

Enter the objects that the track can access into the destination list.

The Transporter

The Transporter is a moving object with a propulsion system of its own. You can define its loading capacity. The Transporter can hold MUs and move them about freely. It represents forklifts, AGVs, etc.

Methods of the Transporter

A source can produce a Transporter, just like an entity or a container. eM-Plant moves the transporter to the succeeding track along the connection lines or with a method.

In addition, the transporter can move on the track under its own power. When you connected the tracks on which the transporter moves with a connector, it moves from track to track.

Use this syntax to place a transporter onto a certain object and/or position:

- .MUs.transporter.create(track);
- .MUs.transporter.create(track,3.1); position in meters on track

The transporter moves on the track using the settings you selected when the simulation starts. You can also stop the transporter, make it continue on its way and make it move backwards in relation to the direction of motion.

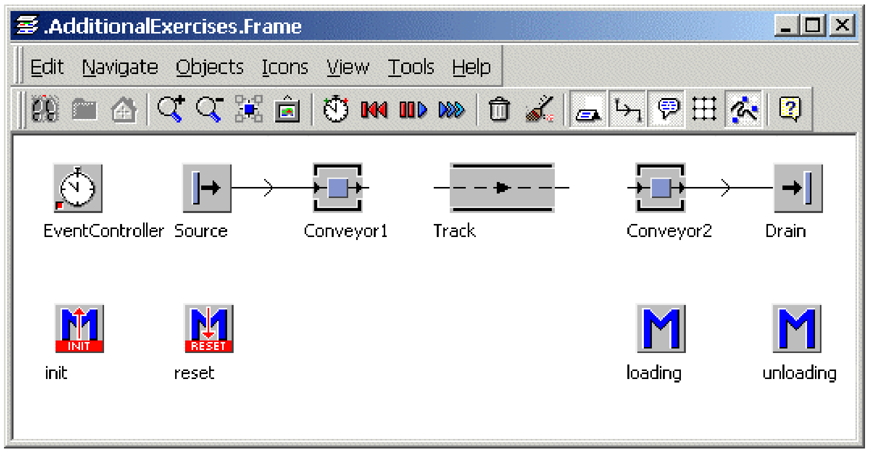
Use the method cont of the track to call the transporter. Use the identifier @ to call the transporter, provided the transporter calls the methods:

- @.stop;
- @.continue;
- @.backwards := true;
- track.cont.stop,...

Exercise: Transporting MUs

Objective: To load and unload parts and transport them across the track.

1. Create the frame AdditionalExercises in the Exercises folder.
2. Add the objects to match the image below.



3. Change the track length to 10 meters.
4. Program the method so that it creates the transporters on the track when the simulation starts.
5. Program the loading and unloading methods to coordinate the events.
6. Make sure the loading and unloading of the parts is correct.

Chapter Summary

In this chapter the following topics were discussed:

- The Object Method and the syntax for writing a Method
- The Method Debugger
- Name Space and Paths
- Anonymous Indentifiers
- Method Calls and Entrance and Exit Controls
- Attributes
- Conditional Statements
- Tables and Global Variables
- Operations Converting Data Types
- Loops and Conditional Suspensions
- The Contents of an Object Loading and Unloading
- Calling Methods and Resource Objects
- Miscellaneous Features