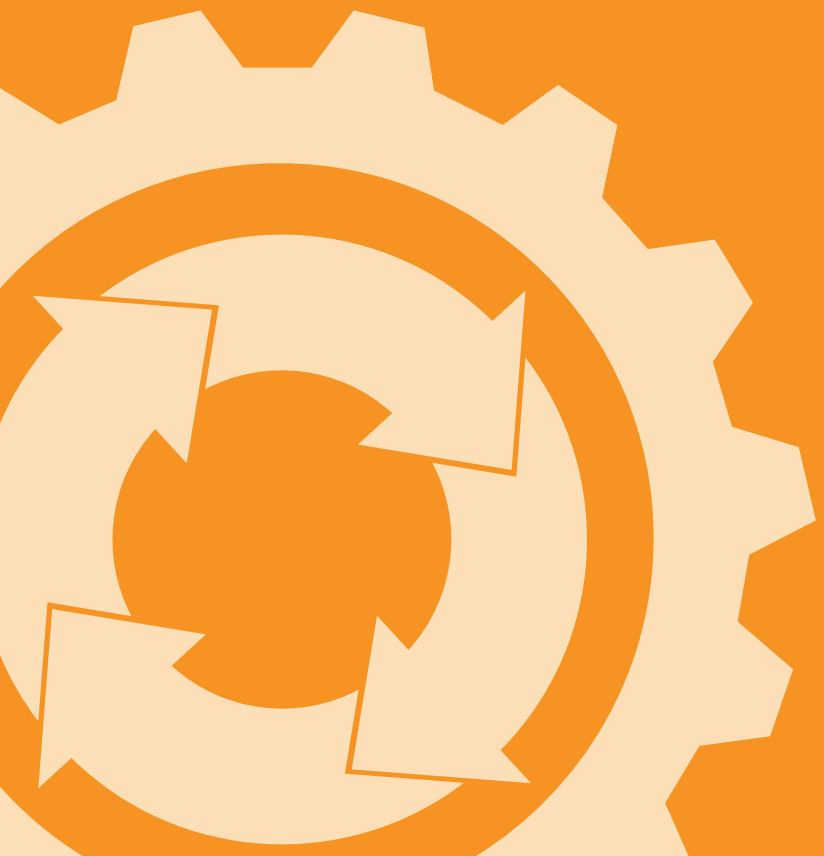
PRAGMADEV PROCESS

# USER MANUAL



PRAGMADEV
modeling and testing tools

# Contents

**1 Introduction** **6**

**2 Supported BPMN constructs** **7**

**3 Project manager** **9**
3.1 Preferences . . . 9
3.1.1 General preferences . . . 10
3.1.2 Printing preferences . . . 12
3.1.3 Executor preferences . . . 13
3.1.4 Licensing preferences . . . 15
3.2 File manipulations . . . 15
3.3 Checking the models . . . 18

**4 BPMN editor** **20**
4.1 Symbols . . . 20
4.1.1 Full BPMN mode . . . 21
4.1.2 Simplified BPMN mode . . . 23
4.2 Hierarchy . . . 23
4.3 Pools with sticky lanes . . . 24
4.4 Link with MEGA HOPEX . . . 27
4.5 Editor . . . 27
4.5.1 Diagrams . . . 27
4.5.2 Selection modes . . . 28
4.5.2.1 Select only . . . 28
4.5.2.2 Select or edit . . . 29
4.5.3 Flow and successor creation using symbol hover buttons . . . 29
4.5.4 Symbol & link insertion keyboard shortcuts . . . 31
4.5.5 Re-select last tool . . . 32
4.5.6 Automatic sequence flow creation . . . 32
4.5.7 Handling broken segments . . . 33
4.5.8 Modifying symbol types . . . 36
4.5.9 Modifying link types . . . 38
4.5.10 Connecting Call activities . . . 39
4.5.11 Printing and exporting . . . 42
4.6 Heatmap . . . 42
4.6.1 File format . . . 42
4.6.2 Display . . . 43

**5 Executor** **45**
5.1 Underlying principles . . . 45
5.2 Controlling the executor . . . 46
5.2.1 Via the graphical user interface . . . 46
5.2.2 Via the command line interface . . . 47
5.3 Behavior . . . 50
5.3.1 Start . . . 50
5.3.2 Sequence flows . . . 50
5.3.3 Message flows . . . 51
5.3.3.1 Implicit resolution . . . 51
5.3.3.2 Explicit resolution . . . 54
5.3.4 Call activities . . . 59
5.3.5 Gateways . . . 59
5.3.5.1 Inclusive . . . 59
5.3.5.2 Exclusive . . . 61
5.3.5.3 Parallel . . . 63
5.3.5.4 Event . . . 65
5.4 Execution tree . . . 70
5.5 Coverage . . . 72
5.5.1 General information . . . 72
5.5.2 Highlight non-covered symbols . . . 74
5.6 Execution traces . . . 75
5.6.1 Recording . . . 75
5.6.2 Replay . . . 75
5.6.2.1 Single-trace execution . . . 77
5.6.2.2 Multi-trace execution . . . 78

**6 MSC and PSC Editor** **79**
6.1 Overview . . . 79
6.2 MSC & PSC reference guide . . . 79
6.2.1 General diagram format . . . 79
6.2.2 Links . . . 80
6.2.2.1 Message links . . . 80
6.2.2.2 PSC-specific normal, required and failed message syntax 81
6.2.2.3 Sequence flow . . . 82
6.2.3 Main symbols . . . 82
6.2.3.1 Lifeline . . . 82
6.2.3.2 Lifeline components . . . 82
6.2.3.3 Collapsed lifelines . . . 87
6.2.3.4 Inline expressions . . . 88
6.2.3.5 Absolute times . . . 91
6.2.3.6 BPMN signals throws & catches (PragmaDev extension) 92
6.2.3.7 Enabled and disabled gates (PragmaDev extension) . . . 93
6.2.3.8 Comments . . . 94
6.2.3.9 Texts . . . 94

6.3 MSC editor . . . 95
6.3.1 Specific tools . . . 95
6.3.2 Symbol creation . . . 97
6.3.3 Manipulating components in lifelines . . . 99
6.3.4 Big diagrams handling . . . 100
6.3.5 MSC symbol and link properties . . . 101
6.3.6 Message parameters display . . . 101
6.3.7 Conformance checking: diagram diff & property match . . . 102
6.3.7.1 Basic MSC diff: trace vs. trace, spec. vs. spec., ... . . . 103
6.3.7.2 Spec vs. trace comparison . . . 106
6.3.7.3 Property match . . . 107

**7 Explorer 110**
7.1 Architecture . . . 110
7.2 Properties . . . 111
7.3 Launch an exploration . . . 111
7.4 Result analysis . . . 114
7.4.1 Full state space exploration . . . 114
7.4.2 Uncovered elements . . . 115
7.4.3 Property verification . . . 116

**8 Simulator 119**
8.1 Principles . . . 119
8.2 Simulation scenarios . . . 119
8.3 Simulation attributes . . . 121
8.3.1 Time parameters . . . 123
8.3.2 Cost parameters . . . 125
8.3.3 Control parameters . . . 126
8.3.4 Result requests . . . 129
8.3.5 Resource parameters . . . 130
8.3.5.1 Basic resource selection . . . 130
8.3.5.2 Advanced resource selection (XPATH expression) . . . 131
8.4 External BPSim data . . . 133
8.5 Running a simulation . . . 134
8.5.1 Via the graphical user interface . . . 134
8.5.2 Via the command line interface . . . 137
8.6 Simulation results . . . 138
8.6.1 Summary . . . 140
8.6.1.1 Graphs . . . 140
8.6.1.2 Exporting . . . 142
8.6.2 Details . . . 143
8.6.3 Logs . . . 145
8.6.3.1 Global mode . . . 145
8.6.3.2 Selection mode . . . 147
8.6.3.3 Log files . . . 149
8.6.4 Resource logs . . . 151

8.6.5 Resource wait time heatmap . . . 153
8.7 Critical path . . . 155
8.7.1 Principles . . . 155
8.7.2 Configuration . . . 155
8.7.3 Heat map . . . 155

**9 Resources Editor 157**
9.1 Overview . . . 157
9.2 Resource definition files . . . 159
9.3 “Roles” tab . . . 160
9.4 “Resources” tab . . . 161
9.4.1 Resource attributes . . . 162
9.4.2 Resource quantities . . . 162
9.4.3 Resource roles . . . 175

**10 Glossary 178**

# 1 Introduction

Complex organizations or systems operations are based on processes described in graphical models. The most popular notation is BPMN. It describes what the different participants in a process do and how they interact with each other. These processes must be thoroughly discussed before they are applied in a real situation. Any misunderstanding of the process might lead to a catastrophic situation in operation.

PragmaDev Process is a set of tools:

- A project manager
  The project manager will gather all the files of your project in one place for easy access.
- BPMN Editor
  The BPMN editor allows you to design your process graphically. It is also possible to import an existing diagram from another tool through the XML standard BPMN.
- BPMN Executor
  The BPMN executor animates your BPMN. The possible path of execution will be graphically displayed.
- BPMN Tracer
  When executing the model step by step it is possible to trace the different steps in an MSC. One or several scenario can be replayed automatically against the model.
- BPMN Explorer
  The Explorer relies on a third party tool called OBP (Observer Based Prover) developed by ENSTA Bretagne. This tool will automatically try all possible scenarios in the model. There are two possible outcomes of this exploration:
  - Overall number of possible steps in the model.
    If that number is too high compared to the size and complexity of the model, it probably means the model is wrong.
  - Verification of a property
    The property to be verified is expressed with a PSC (Property Sequence Chart) that is a sort of MSC. The Verifier will search if there is a scenario that verifies the property.

# 2 Supported BPMN constructs

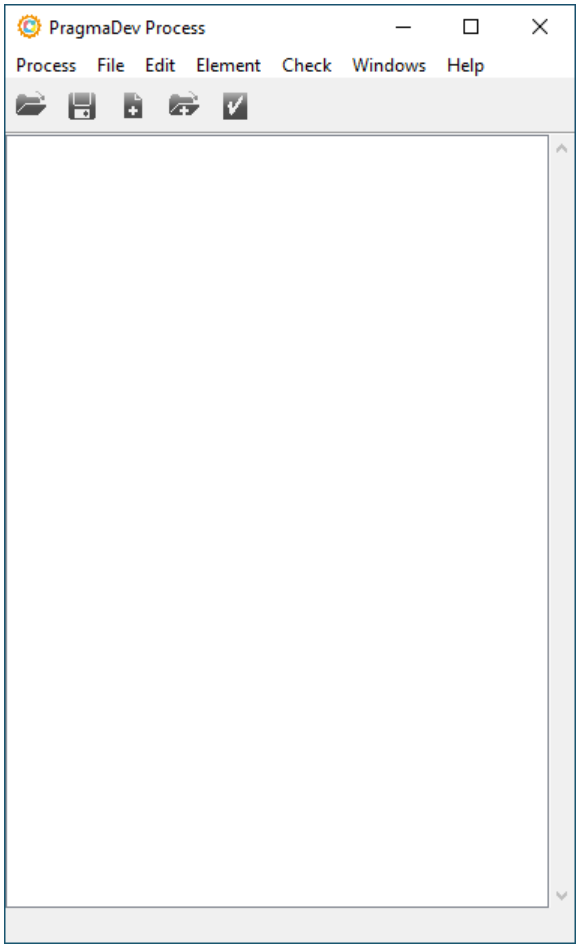
PragmaDev Process is a BPMN Viewer (VI), Editor (ED), and Executor (EX). There are a few restrictions to these features as some BPMN constructs can be viewed but not edited or executed. A summary of supported constructs is given in table 2.1.

Table 2.1: Supported BPMN constructs.

| BPMN construct | VI | ED | EX |
|---|---|---|---|
| Pool, Lane | YES | YES | YES |
| Task | YES | YES | YES |
| Sub-Process (Transaction, Ad-Hoc, and Event) | YES | NO | NO |
| Call-Activity | YES | YES | YES |
| Data (Object, Input, Output, and Store) | YES | NO | NO |
| Start Event | | | |
| None, Message, Timer, Signal | YES | YES | YES |
| Conditional, Multiple, Parallel Multiple | YES | NO | NO |
| End Event | | | |
| None, Message, Signal, Terminate | YES | YES | YES |
| Escalation, Error, Compensation, Multiple, Cancel | YES | NO | NO |
| Intermediate Event (Throw) | | | |
| None, Message, Signal | YES | YES | YES |
| Escalation, Compensation, Link, Multiple | YES | NO | NO |
| Intermediate Event (Catch) | | | |
| None, Message, Timer, Signal | YES | YES | YES |
| Link, Conditional, Multiple, Parallel Multiple | YES | NO | NO |
| Boundary Event | | | |
| None, Message, Timer, Signal | YES | YES | YES |
| Conditional, Escalation, Error, Compensation, Multiple, Cancel | YES | NO | NO |
| Gateway | | | |
| Exclusive, Inclusive, Parallel, Event | YES | YES | YES |
| Complex, Event Start, Parallel Event Start | YES | NO | NO |
| Group | YES | NO | - |
| Text Annotation | YES | YES | - |
| Choreography, Conversation | NO | NO | NO |
| Sequence Flow, Message Flow | YES | YES | YES |
| Association | YES | NO | NO |

# 3 Project manager

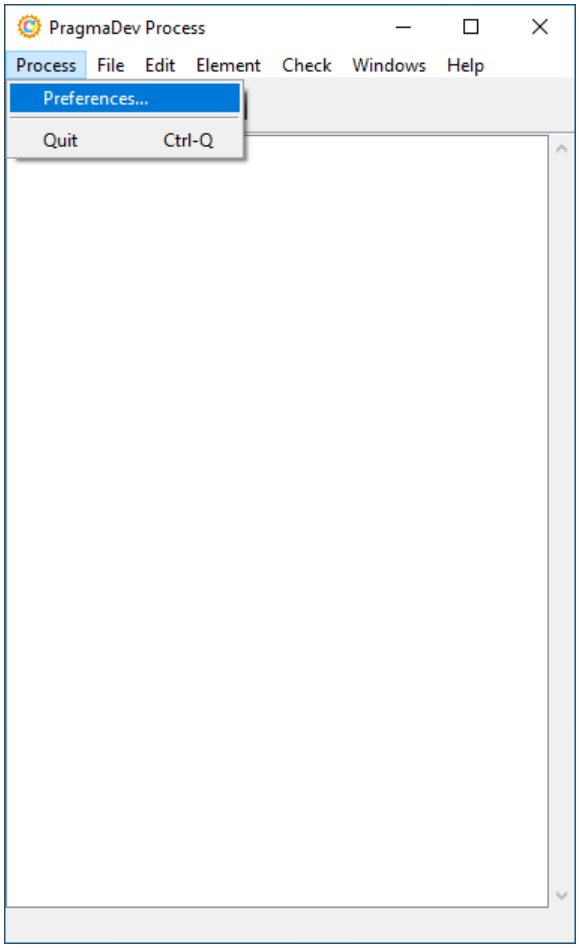
When starting the tool, after setting the licensing mechanism described in the Installation manual, the Project manager pops up.




A project allows you to gather all the files related to the on going project: BPMN, traces, and properties. By default an empty unnamed new project is created at startup.

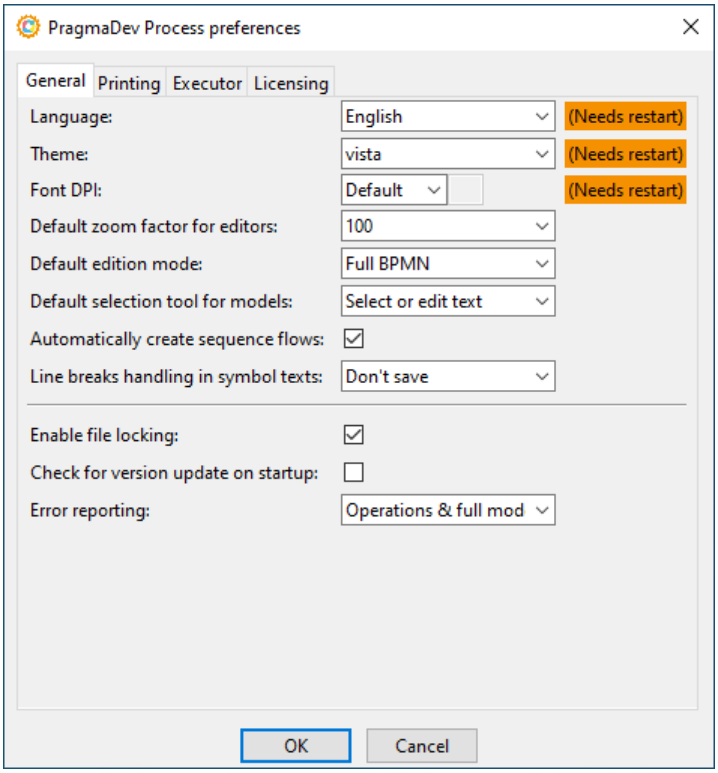
## 3.1 Preferences

The first menu in the tool gives access to the preferences window:




### 3.1.1 General preferences

In the *General* tab, you can change the options for the application as a whole and for the BPMN model editor:



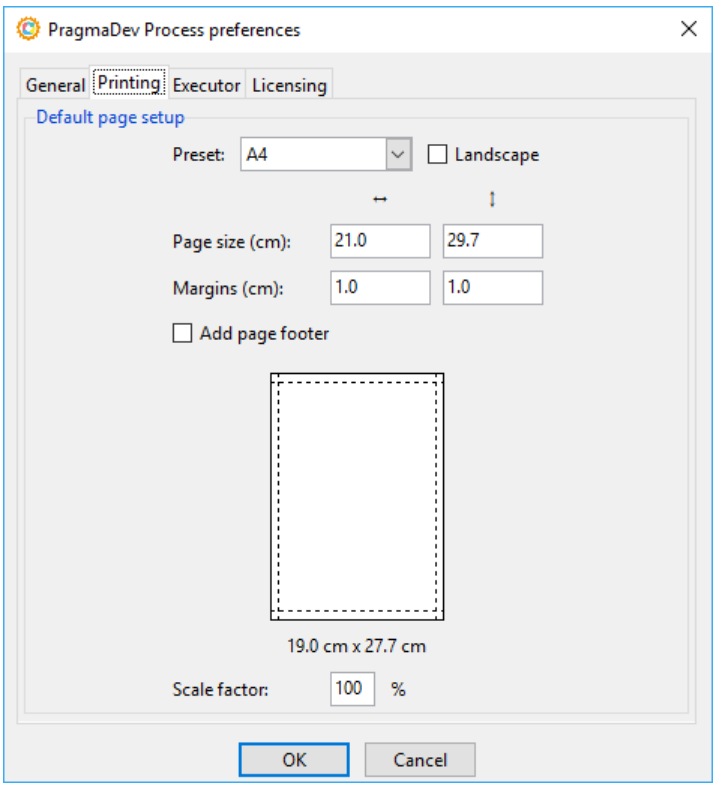

- *Language* is the language used for all texts in the whole application.
- *Theme* is the theme used for the GUI; this is mostly useful for Linux users.
- *Font DPI* should be used only if there are issues with the text in the application, e.g text that is way too small or too large. This usually means that the resolution configured for fonts has been badly guessed by the GUI toolkit. If you have such an issue, select *Forced* instead of *Default* in the menu and enter a resolution in dots per inch in the field. The most common values for the resolution are 96 and 72; in some cases, values 92 or 80 can give good results too.
- *Default selection tool for models* allows to select how the selection tool will behave in the BPMN model editor. See "Selection modes" on page 28 for an explanation of the available modes.
- *Automatically create sequence flows* allows to automatically set the same option in all opened BPMN editors. This will automatically create a sequence flow from the last inserted symbol to any new created one whenever possible. See "Automatic sequence flow creation" on page 32.
- *Line breaks handling in symbol texts* controls how line feed characters are handled for symbols that have their texts inside them (e.g, activities) and for symbols that have their texts under them (e.g start, end & boundary events and gateways). Two modes are available:
  - In *Manual* mode, all texts are wrapped to the appropriate width for the symbol by actually inserting line feed characters. These characters are then kept in the saved model, so if another line feed character is inserted by the user, it is kept too. The width used for activities is the width of the symbol itself; for events and gateways, the width used is 4 times the width of the symbol.

  - In *Don't save* mode, the wrapping is done the same way, but the line feed characters are replaced by spaces each time the text is modified. So the wrapping is actually done each time the symbol is displayed or redisplayed. This also means that any explicitely inserted line break will be lost when the text is validated.

  Note that this is done only for the symbols listed above: no wrapping happens for any other symbol, including pools, lanes and also text annotations, where explicit line breaks must usually be kept.
- *Enable file locking* controls whether a lock is set on the files opened from the project manager. This is useful if the project is shared between several users and editing the same models at the same time could lead to conflicts.
- *Check for version update on startup* controls whether a check is made when the application starts to figure out if a new version is available. If there is one, a dialog will appear offering to download it.
- *Error reporting* controls the level of information will be sent if an unexpected error ever happens in the application. The choices are "Operations only - no models" to send a history of the operations performed during the session, but not any content of the edited models, and "Operations & full models" to send the contents of the models as well.

### 3.1.2 Printing preferences

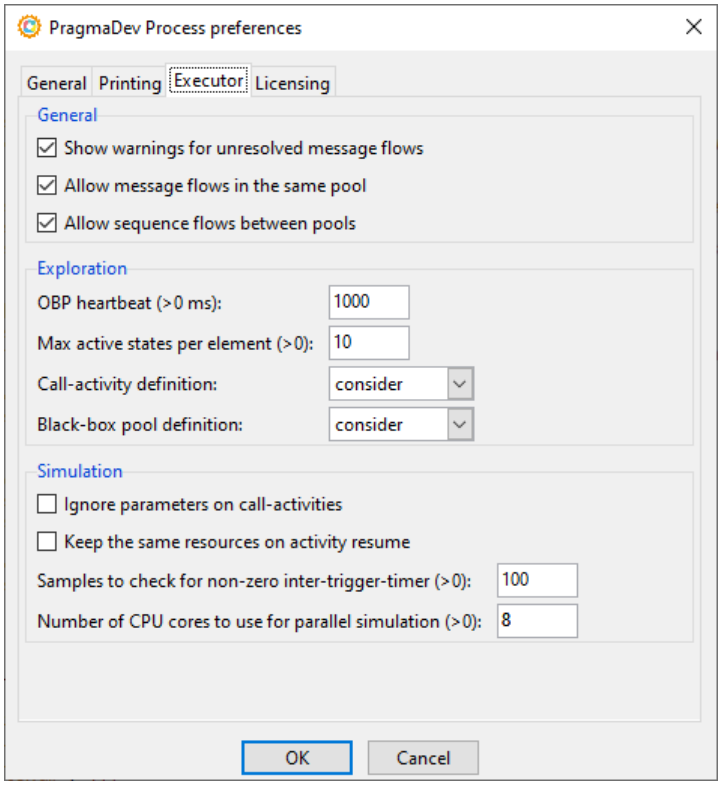
The *Printing* tab allows to define the default page setup that will be used when printing BPMN models:



The *Preset* menu contains predefined standard page sizes. The horizontal and vertical margins will be taken on both sides of the page: left & right, and top & bottom. If a footer is added, it will take a little more space at the bottom of the page and will contain the name of the diagram or model and the page number.

### 3.1.3 Executor preferences

The *Executor* tab contains some rules that can be enabled or disabled during BPMN semantics check and execution, the default values used with OBP and options for simulation:




The following *General* options are available:

- *Show warnings for unresolved message flows*
  When enabled, unresolved messages flows will generate warnings during a semantics check (or execution). This option concerns message flows incoming or outgoing an empty pool or lane (back-box). Message resolution is explained in "Explicit resolution".
- *Allow message flows in the same pool*
  The BPMN semantic does not allow message flows within the same pool. This option allows to override this rule.
- *Allow sequence flows between pools*
  The BPMN semantic does not allow sequence flows to cross pool boundaries, i.e., they are allowed only within the same pool. This option allows to override this rule.

The OBP related values are explained in "Launch an exploration". They are:

- *OBP heartbeat*
  Allows to change the default refresh value of the status information returned by OBP during exploration.
- *Max active states per element*
  Upper limit for the number of active states in a BPMN element (except message flows).
- *Call-activity definition*
  Exploration can be limited to certain scenarios by defining how a call-activity definition is handled.
- *Black-box pool definition*
  Exploration can be limited to certain scenarios by defining how a black-box pool definition is handled.

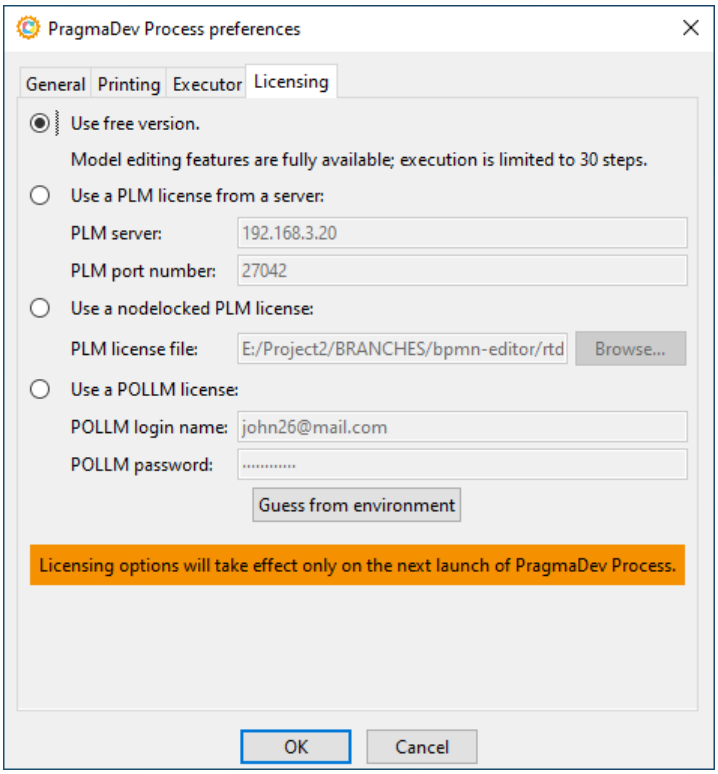
The *Simulation* options are the following:

- *Ignore parameters on call activities* applies to call activities that have both defined time and/or cost parameters and a called process. In this case, there are two possibilities, and this option selects which one will be used:
  - If the option is unchecked, the call activity is considered as an ordinary task and its simulation parameters are taken into account. The linked process is not executed.
  - If the option is checked, the call activity's simulation parameters are ignored and the linked process is called. The times & costs for the call activity itself are then computed from those of the tasks in the called process.

  Note that this option gives the default value for all call activities. It is possible to force the behavior on any call activity by using its *load definition* simulation parameter, as described in "Control parameters" on page 126.
- *Keep resources for activities* controls the resource allocation for activities that have to be suspended because the resources they use become unavailable. If the option is checked, such an activity will resume only when the same set of resources it started with becomes available again. If the option is not checked, the activity can resume using any available set that matches its resource requirements, even if it's not the one it started with.
- *Samples to check for non-zero inter-trigger-timer*
  When the scenario duration is set but trigger-count is not defined, the inter-trigger-timer should not be zero. In this case, if inter-trigger-timer is defined as a distribution, and all the samples taken from the distribution are zero, then an error is raised during semantic check. This is done to prevent the creation of an infinite number of triggers, which would block simulation.
- *Number of CPU cores to use for parallel simulation*
  The execution workload of a simulation is distributed between the set number of CPU cores. This enables running scenario replications in parallel to speedup simulation. The value should be set according to the available CPU cores in the machine. Note that setting the value higher that the maximum number of physical CPU cores (not threads) may degrade performance.

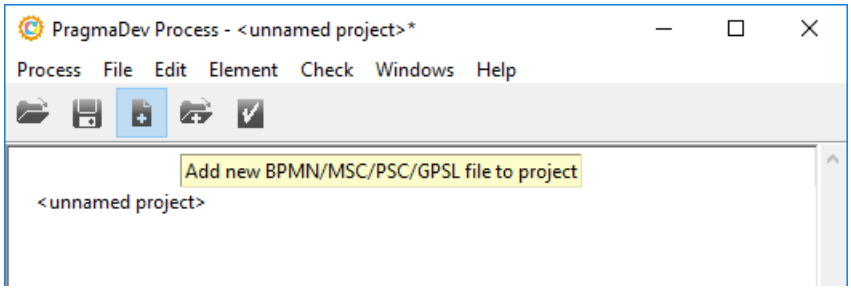
### 3.1.4 Licensing preferences

You can also change the licensing mechanism as explained in the Installation Manual:

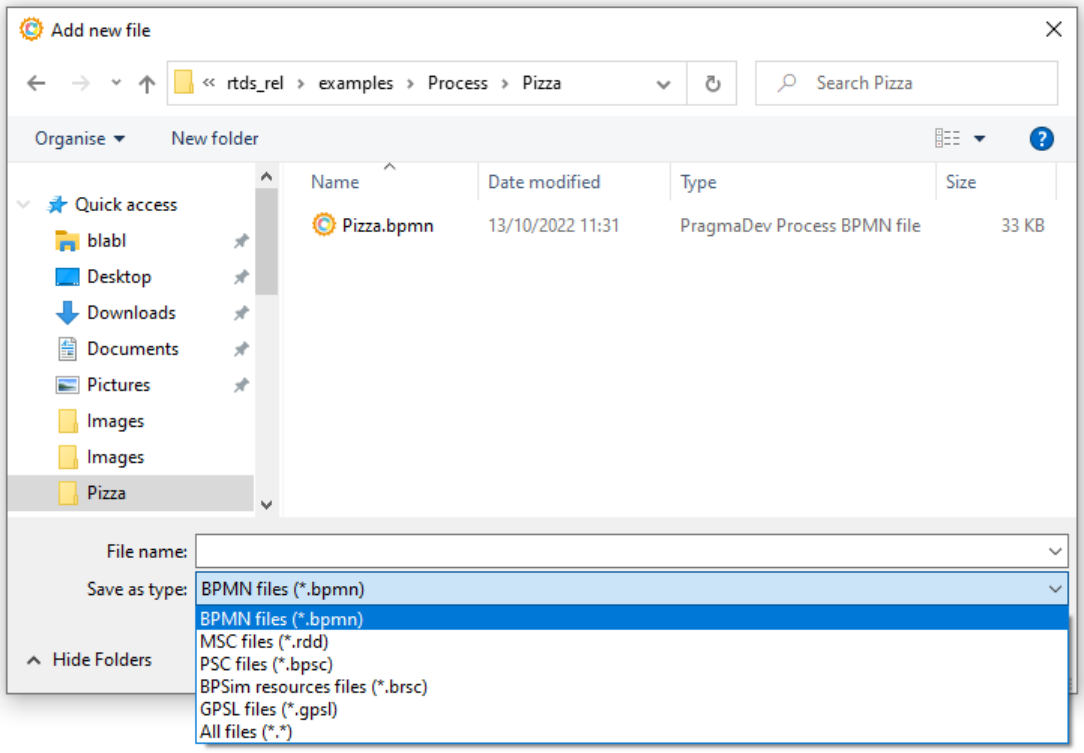



## 3.2 File manipulations

A file is added to the project with the 'Add file' button:



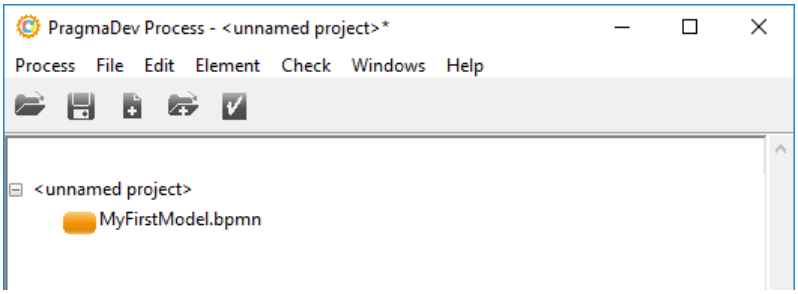

A file browsing window will open, allowing to add files with the `bpmn`, `rdd`, `bpsc`, `brsc` & `gpsl` extensions:




The file types are the following:

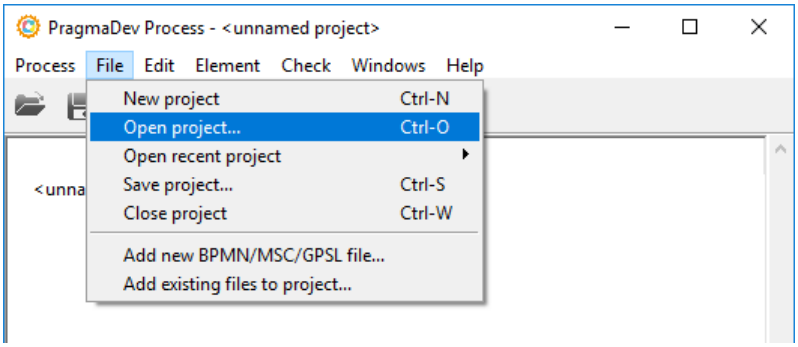
- Files with the `bpmn` extension are BPMN models. They are displayed and edited in the BPMN editor described on page 20.
- Files with the `rdd` & `bpsc` extensions are MSC and PSC diagrams respectively. They have different purposes, but very similar formats and are displayed and edited in the same editor, described in "MSC and PSC Editor" on page 79.
- Files with the `brsc` extension are BPMN resource definitions. They are displayed and edited in the Resources editor, as described on page 157.
- Files with the `gpsl` extension are textual property files. They are opened in the default textual file editor defined via the operating system.

A new file is added to the project:

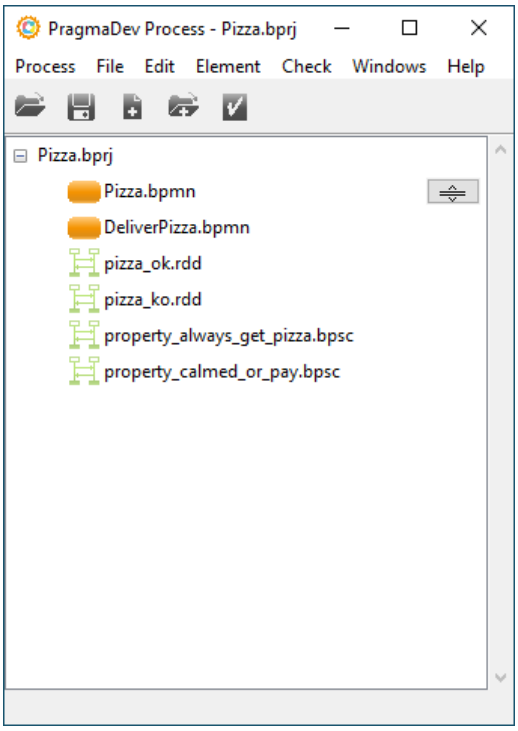



The button allows to add an existing file to the project. The available extensions are the same, with the addition of `bhmp`, which are heatmaps that can displayed with a BPMN model in the BPMN editor. The format for heatmaps and how they are displayed with BPMN models are described in "Heatmap" page 42.

A project can also be directly opened:




The selected project will display its associated files:



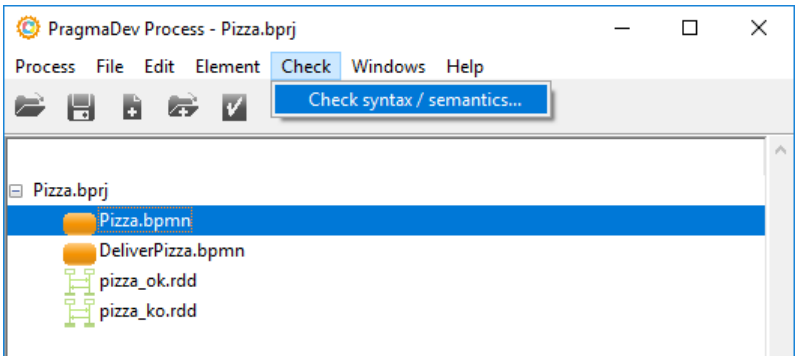

Files can be moved in the project by using the move handles that appear when hovering over an element. Entries in the *Edit* menu allow to force an order for the items in the project:

- If the *Keep items grouped by type* option is checked, project items will be grouped by type: BPMN models first, then MSC traces and PSC diagrams, then textual GPSL properties and finally code coverage results.
- If the *Keep groups sorted by file* option is also checked, items in each group will be sorted by their file name. This will deactivate the arrow buttons since items cannot be reordered in this mode.

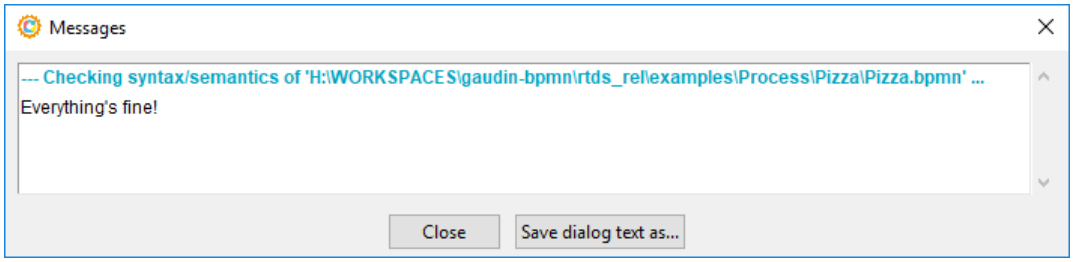
Note that checking any of these options will actually modify the item order in the project, so the project will need to be saved.

## 3.3 Checking the models

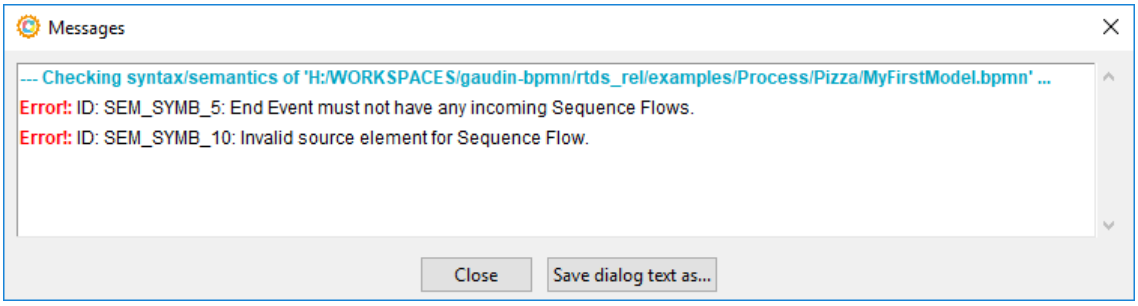
It is possible to check the syntax of a model from the Project manager. Select the diagram and go to the Check menu:



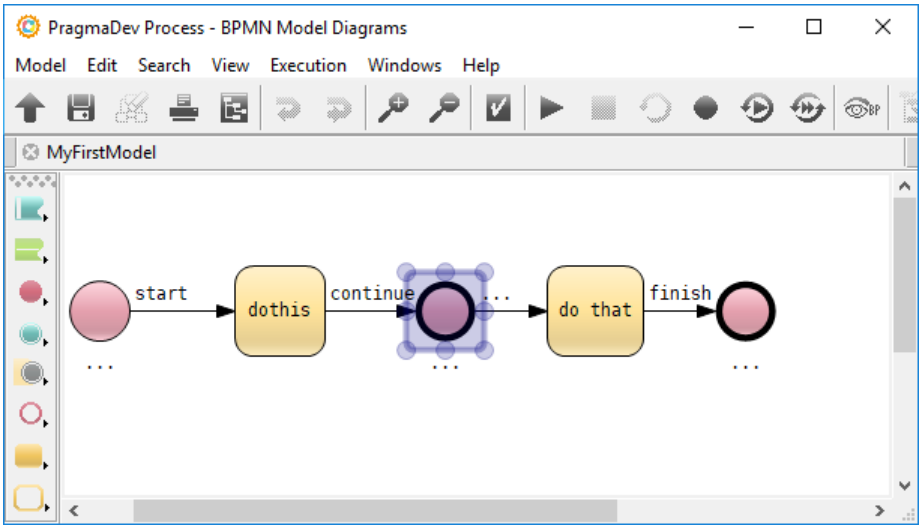

In the case no errors are found:




In the case errors are found:



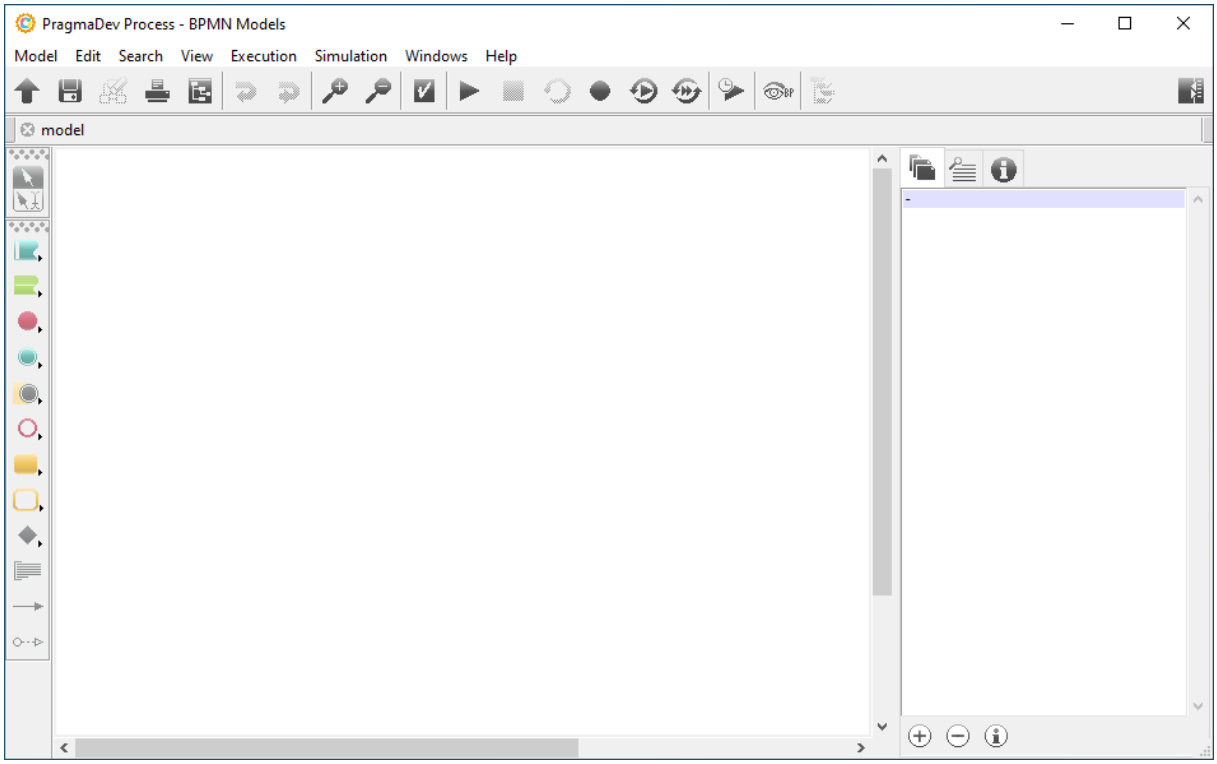

A double click on the error line will open the diagram and selected the symbol related to the error:


PragmaDev Process - BPMN Model Diagrams
Model
Edit
Search
View
Execution
Windows
Help
MyFirstModel
start
dothis
continue
do that
finish

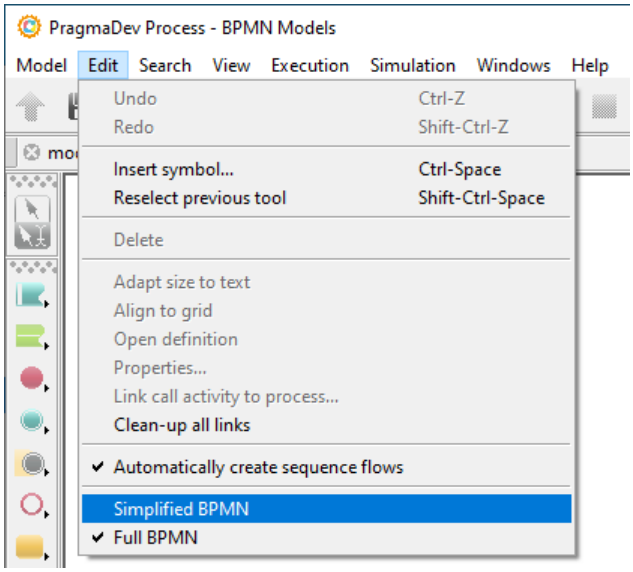
# 4 BPMN editor

## 4.1 Symbols

A double click on any BPMN diagram in the Project manager will open the BPMN editor:




The different symbols are organized in categories on the left side of the editor. The available categories and the symbols they contain depend on the BPMN mode chosen in the "Edit" menu:




In "Simplified BPMN" mode, only the most common symbol types are directly available in the toolbar; in "Full BPMN" mode, all symbol types supported in PragmaDev Process are shown.

Below is the list of categories and symbols available in each category and for each mode.

### 4.1.1 Full BPMN mode

- Pools

  - Horizontal pool
  - Vertical pool
- Lane

  - Horizontal lanes
  - Vertical lanes
- Start events

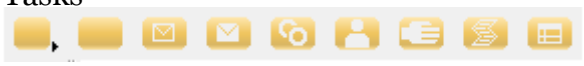
  - Plain start event
  - Message catch start event
  - Timer start event
  - Signal catch start event
- Intermediate events

  - Plain intermediate event
  - Message throw intermediate event

  - Message catch intermediate event
  - Timer intermediate event
  - Signal throw intermediate event
  - Signal catch intermediate event
- Boundary events

  - Message boundary event
  - Timer boundary event
  - Signal boundary event
- End events

  - Plain end event
  - Message throw end event
  - Signal throw end event
  - Terminate end event
- Tasks



  - Plain task
  - Message send task
  - Message receive task
  - Service task
  - User task
  - Manual task
  - Script task
  - Business
- Call activities

  - Plain call activity
  - User call activity
  - Manual call activity
  - Script call activity
  - Business rule call activity
  - Process call activity
- Gateways

  - Exclusive gateway
  - Parallel gateway

  - Inclusive gateway
  - Event gateway
- Text annotation

  Note that these cannot be linked to other symbols yet.
- Sequence flow

- Message flow

### 4.1.2 Simplified BPMN mode

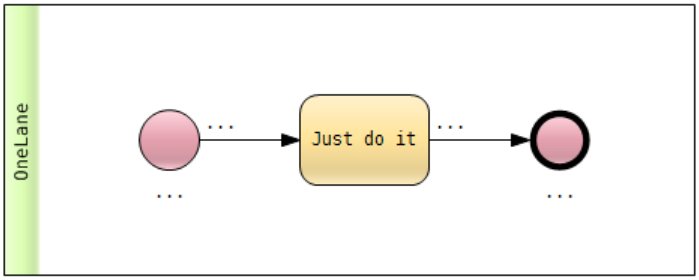
The available symbols are the same as for the Full BPMN mode, except in the following categories:

- For start events, only the plain, message catch and timer ones are available.
- For intermediate events, only message catch, message throw and timer ones are available.
- For boundary events, only message catch and timer ones are available.
- For end events, only plain and terminate ones are available.
- For taks, only plain ones are available.
- For call activities, only process ones are available.

Note that it is still possible to create any kind of symbol in Simplified BPMN mode by creating a symbol, then changing its type via its "General properties": the "Shape kind" selector will always contain all supported types, regardless of the current BPMN mode.

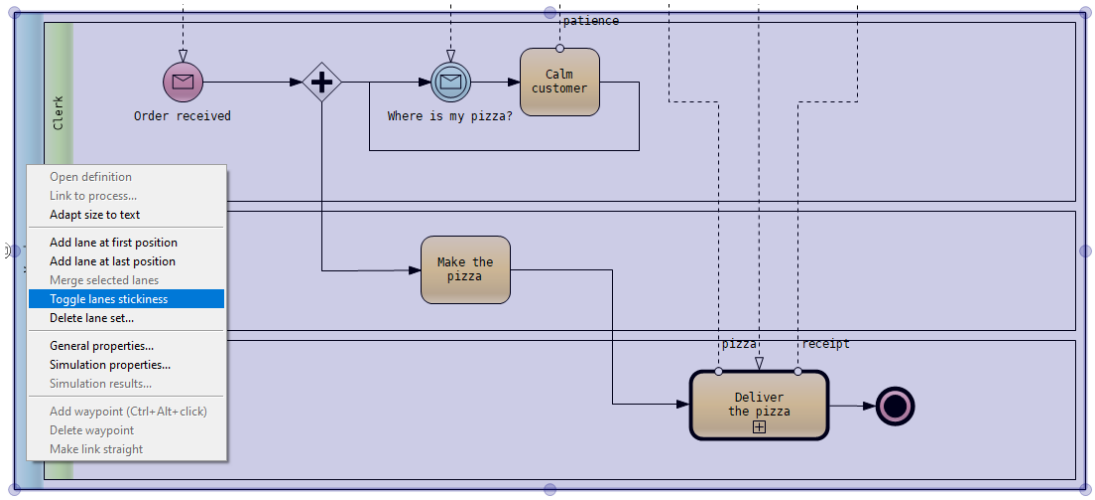
## 4.2 Hierarchy

BPMN has container symbols that include other symbols. Typically Pools and Lanes are containers for execution symbols. Drawing a symbol in a container symbol will automatically associate the contained symbol with its container. Moving the container will move all the contained symbols. Graphically dragging a symbol out of the container will dissociate the symbols. In the example below, the start, the task and the end symbols are contained in the `OneLane` lane. Moving the lane symbol will move all the contained symbols.

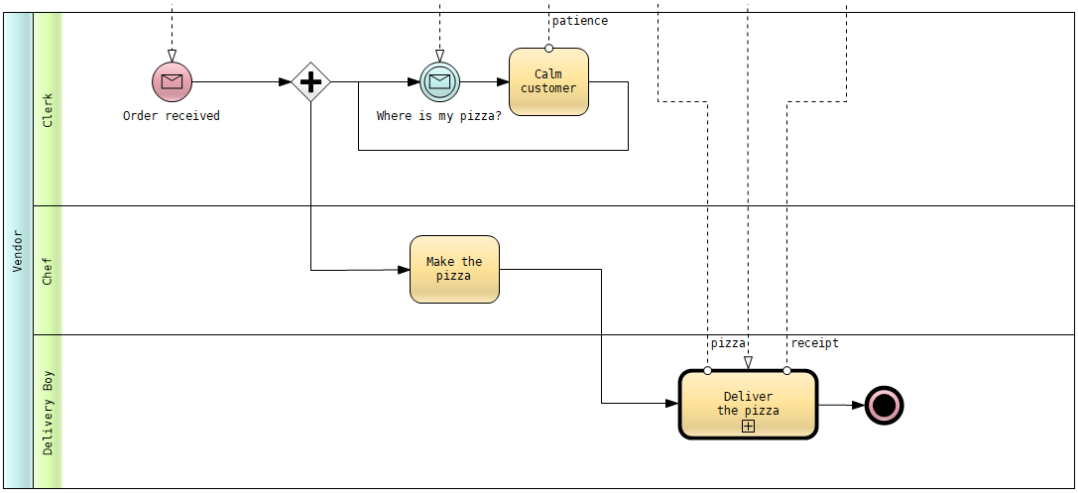



## 4.3 Pools with sticky lanes

PragmaDev Process offers a way to makes lanes within a pool stick to their parent's sides. For participants created in PragmaDev Process, this option is automatically on. For BPMN models created in other tools, it has to be turned on explicitely on the participant:

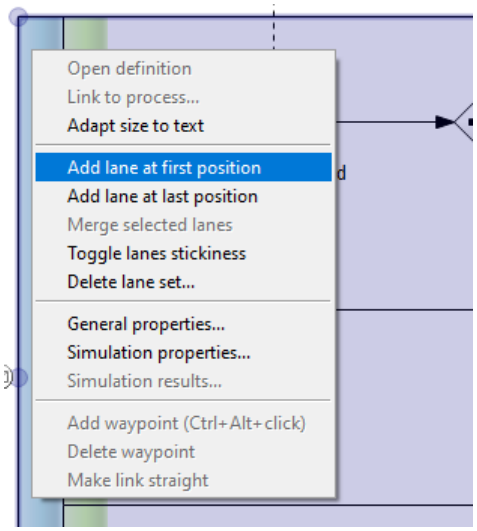



Once it's turned on, the lanes will expand to stick to the pool's borders:

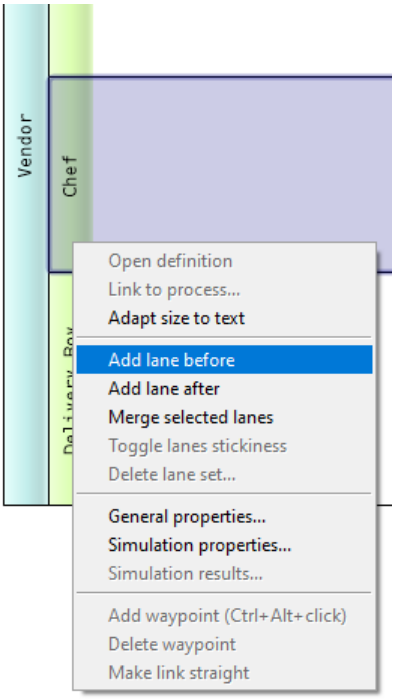


If a pool has sticky lanes, some operations will behave differently:

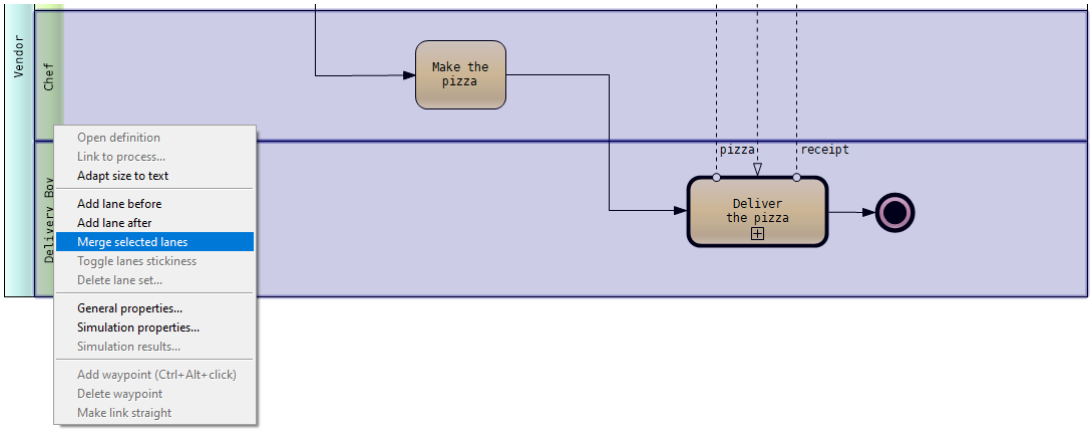
- If a lane symbol is dropped in the pool, it will insert itself in the lane set and expand automatically to the pool's borders;
- A lane can also be added at the first or last position in a pool via the pool's contextual menu:



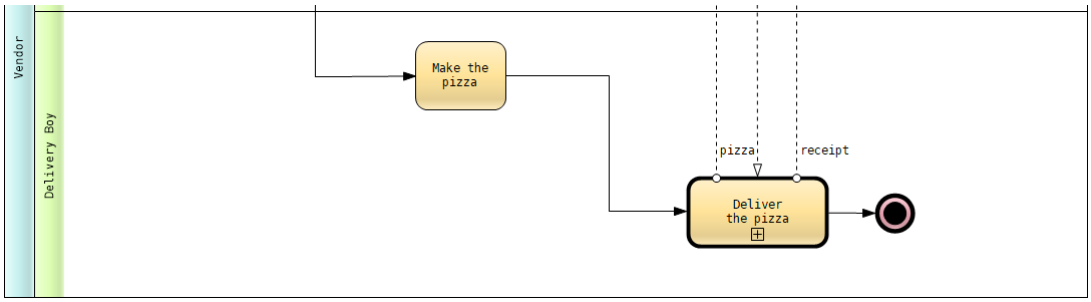

or after or before an existing lane in a pool via the lane's contextual menu:




- Sticky lanes can only be deleted if they are empty. So to delete a lane, all the symbols it contains have to be deleted first, or moved to another lane. If the whole contents of a lane has to be moved to another one, it is also possible to merge two or more consecutive lanes into a single one:

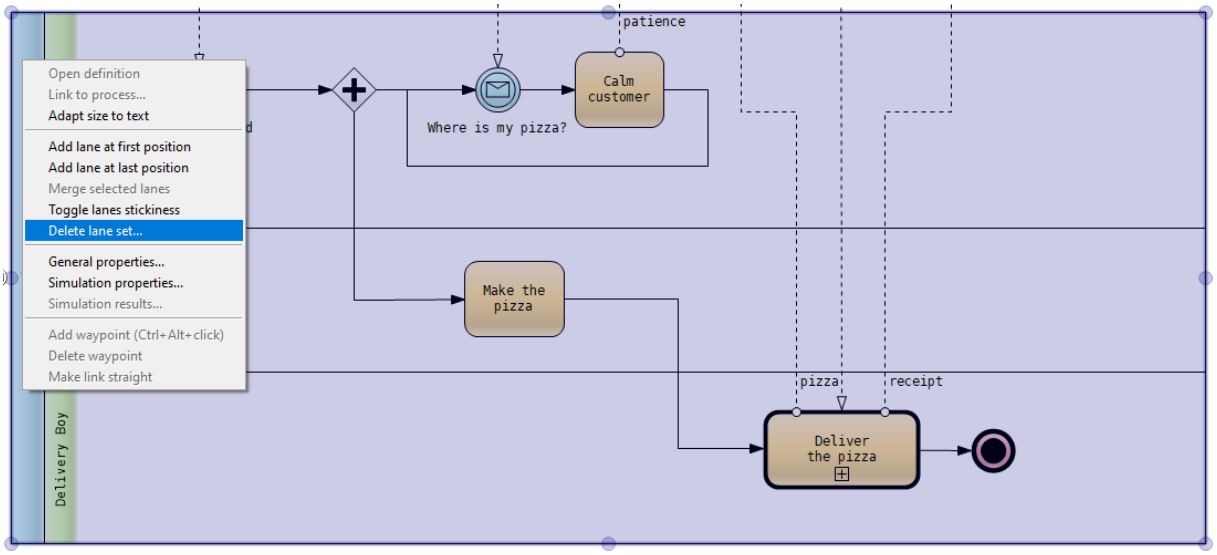



Once the merge is done, all the symbols will end up in a single lane:

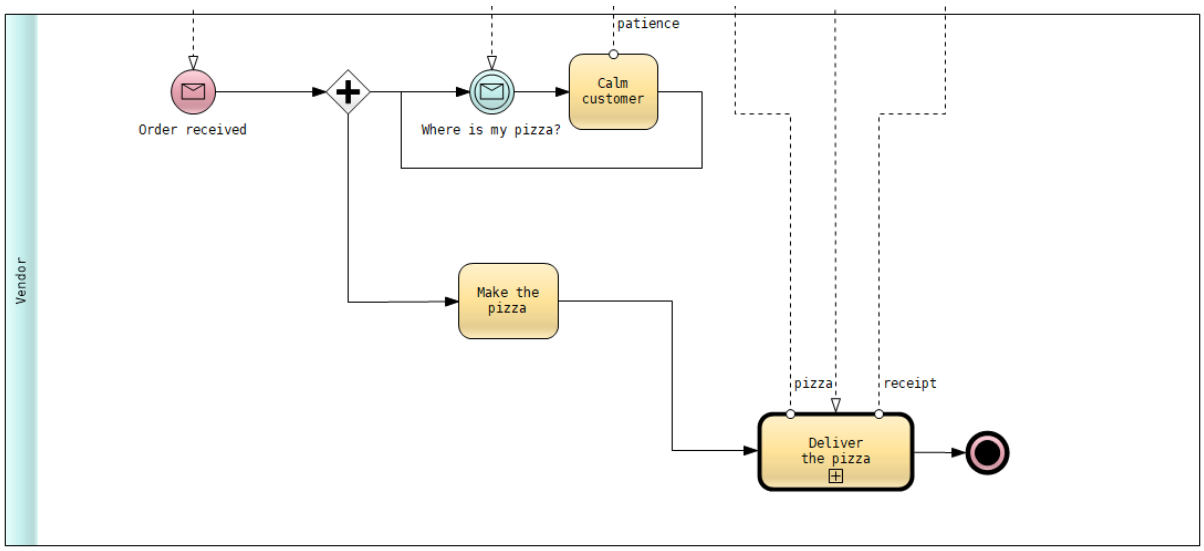



Note that the final lane will always have the name of the lane selected last before merging them.

- To move all symbols in the lanes of a pool to the pool itself and discard the lanes, it is also possible to delete the lane set of a pool:




A confirmation dialog will be displayed; if the lane set deletion is confirmed, all symbols will be moved to the pool and all lanes will be removed:




## 4.4 Link with MEGA HOPEX

PragmaDev Process has a specific link with HOPEX tool as we are one of MEGA's partners. Users of MEGA HOPEX can launch the PragmaDev Process executor from HOPEX on a set of selected diagrams. Diagrams are automatically exported in BPMN and can be viewed in PragmaDev Process viewer. In that specific situation the models can not be edited, they can only be viewed. If a modification should be made on the model it should be on the source model in HOPEX. For that matter click on the "Go to source model" button at the top of the editor:

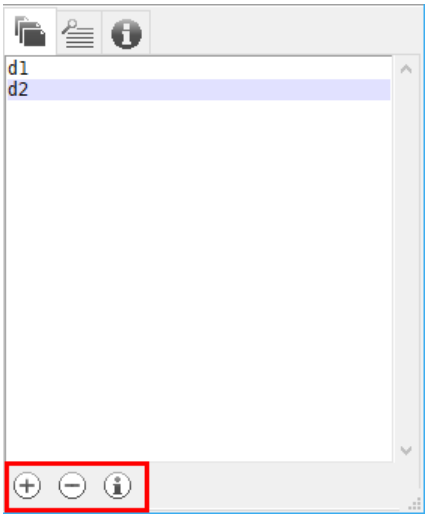
Please note this feature only works with HOPEX web front end.

## 4.5 Editor

The BPMN editor is quite straight forward but there are a few particular features that deserve to be explained.

### 4.5.1 Diagrams

A BPMN model may contain several diagrams. Diagrams can be added, removed, or edited using the buttons in the right panel:



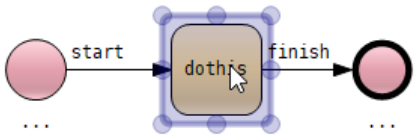

The panel can be shown or hidden with *Toggle browser* button .

## 4.5.2 Selection modes

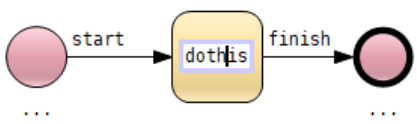
The editor provides 2 modes for selecting and editing. It is possible to toggle from one to the other with this button:

### 4.5.2.1 Select only

This is the default mode. Symbols have a graphical shape and some text inside. To select the graphical shape of the symbol you can click anywhere in the symbol including the text area:



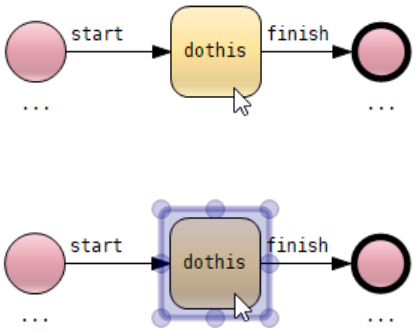

To edit the text, double click on the text:



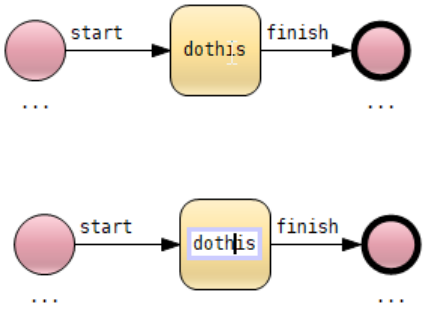

This mode is more efficient if most of the work is to graphically re-organize the diagram.

#### 4.5.2.2 Select or edit

Symbols have a graphical shape and some text inside. To select the graphical shape of the symbol you should click between the edge of the symbol and the text:



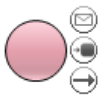

Clicking on the text will actually edit the text:



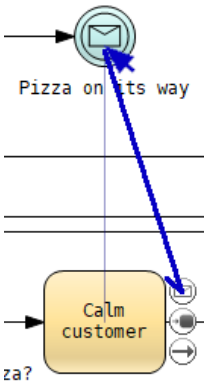

This mode is more efficient if most of the work is to edit the text contents of the symbols.

### 4.5.3 Flow and successor creation using symbol hover buttons

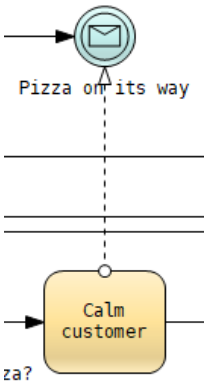
Once a symbol has been created in the model, it is possible to create its successors or the message or sequence flows starting from it by using the buttons that appear when the mouse pointer hovers over the symbol:



The top and bottom hover buttons will create respectively a message and a sequence flow starting from the symbol. The mouse button has to be pressed when in the button, then dragged to the target symbol:




After the mouse button is released, the link is created:



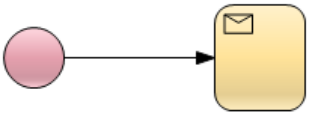

Creating a broken link can also be done by clicking on the hover button, then shift-clicking on each of the waypoints for the link to create.

The other hover button which appears in the middle allows to directly create a successor for the symbol with the sequence flow connecting them. It is used just as the flow creation buttons: the mouse button has to be pressed when in the button, then dragged to the position where the successor must be created:

When the button is released, a circular menu will appear, allowing to choose the actual type for the successor to create:

The red cross will cancel the successor creation. The other items will select the main type for the successor to create. Depending on the current BPMN mode (see "Symbols" on page 20) and the selected type, a second circular menu may appear, allowing to select the actual type for the successor. For example, in "Full BPMN" mode, if the main type for tasks is chosen, the following secondary menu appears:

If for example a message receive task is chosen, the successor is created with this type, along with the sequence flow connecting it:



### 4.5.4 Symbol & link insertion keyboard shortcuts

In addition to selecting them in the left-hand side button bar or creating them via the hover buttons, symbols and links can also be created by using keyboard shortcuts. This avoids having to get the mouse back to the button bar for every created symbol or link. All these keyboard shortcuts are introduced by Control + Space (Command + Control + Space on macOS), then a letter for each symbol type:

- Ctrl + Space, then **p** creates a pool;
- Ctrl + Space, then **l** creates a lane;
- Ctrl + Space, then **s** creates a start event;
- Ctrl + Space, then **i** creates a intermediate event;
- Ctrl + Space, then **b** creates a boundary event;
- Ctrl + Space, then **e** creates an end event;
- Ctrl + Space, then **t** creates a task;
- Ctrl + Space, then **c** creates a call activity;
- Ctrl + Space, then **g** creates a gateway;
- Ctrl + Space, then **a** creates a text annotation;
- Ctrl + Space, then **S** (i.e Shift + s) creates a sequence flow;
- Ctrl + Space, then **M** (i.e Shift + m) creates a message flow.

The insertion then goes on just as if the corresponding tool had been selected in the button bar: a symbol "ghost" appears and can be positionned at the location where the symbol must be created. When the location is clicked, since only the main symbol

type has been specified via the keyboard shortcut, the creation actually behaves like the creation of a successor via the hover buttons as described in "Flow and successor creation using symbol hover buttons" on page 29: a circular menu appears and allows to select the actual type for the symbol.

So for example, if a gateway is inserted via Ctrl+Space, then g, after clicking at the position where the gateway must be inserted, the following menu appears:

Selecting one kind of gateway from the menu will actually insert the gateway at the clicked location.

### 4.5.5 Re-select last tool

When using the same tool again and again, it might be tedious to select it again and again. For that situation pressing Ctrl + Shift + Space bar actually re-selects the last used tool.

### 4.5.6 Automatic sequence flow creation

The option *Automatically create sequence flows* in the *Edit* menu allows to create sequence flows automatically between created symbols.

When this option is on and a symbol is created, once its text is validated, it will be automatically selected. If a new symbol insertion is started, the "ghost" displayed to mark the position for the symbol to create will also include a link from the selected symbol:

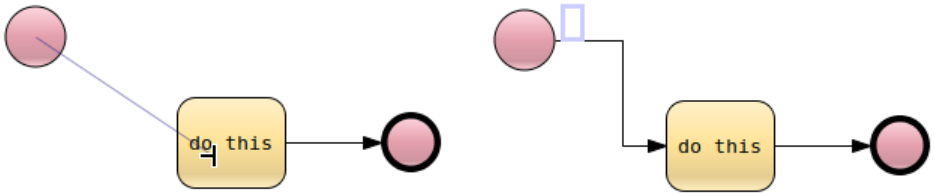
Once the symbol is actually created, a sequence flow will be created between the previous symbol and the new one:

Note that the text for the automatically created sequence flow will always be empty.

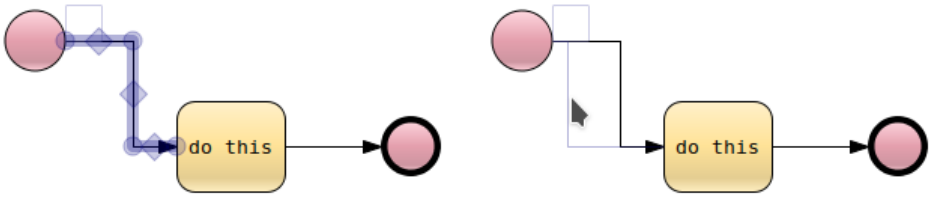
This works too if the previous symbol's text is opened for edition and a new symbol creation is triggered via its keyboard shortcut. This allows to create models very quickly and efficiently. An option is available in the application preferences' *General* tab to enable this feature by default in the editor (see "Preferences" on page 9).

### 4.5.7 Handling broken segments

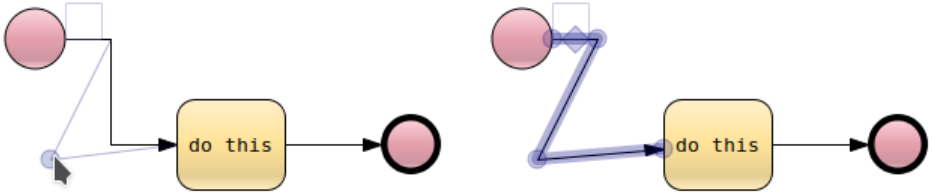
When connecting two symbols with a sequence or a message flow, the created link will be a succession of horizontal and vertical segments by default:




If you'd prefer a different path than the default one, but want to keep a succession of horizontal and vertical segments, you can select the link and move the segments by using the diamond-shaped handles:

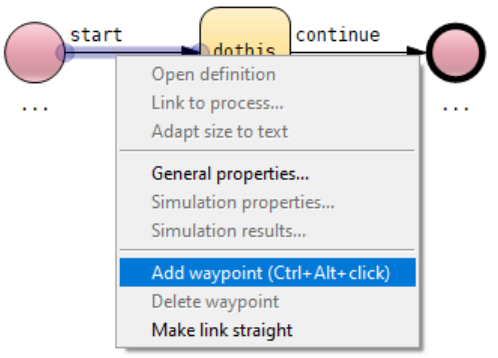


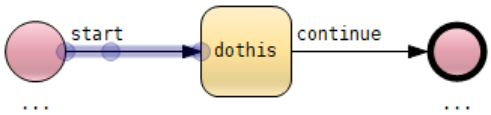

The corner points - also called "waypoints in BPMN - can also be moved via the circular handles around them:



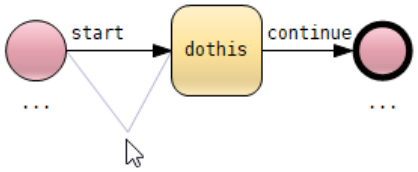

Note that the diamond-shaped handles do not appear on segments that are neither horizontal, nor vertical.

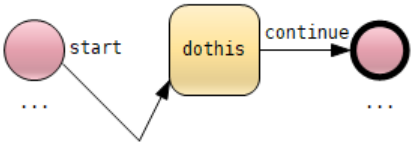
It is also possible to add a waypoint via the contextual menu that appears when right-clicking on the link:



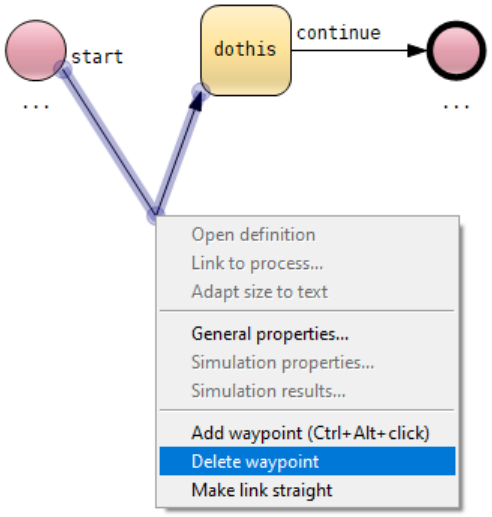




You can just drag the newly created waypoint to where you want the segment to break:



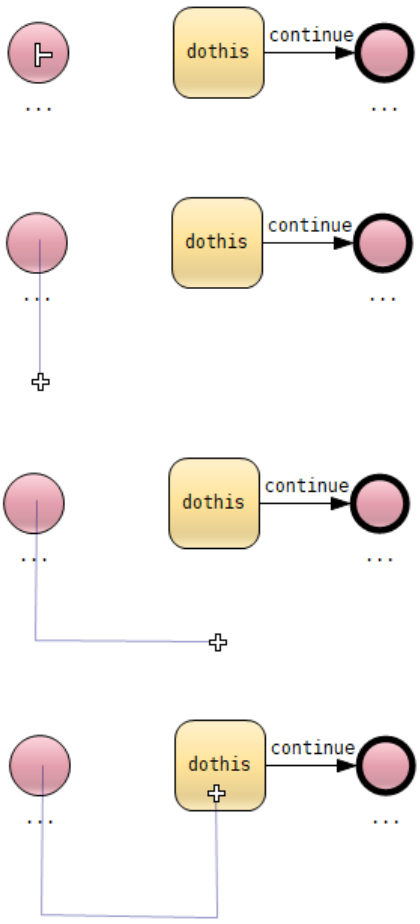



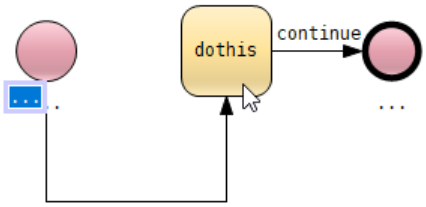

To delete a waypoint, select the link, right click on the waypoint to get the contextual menu and select "Delete waypoint":



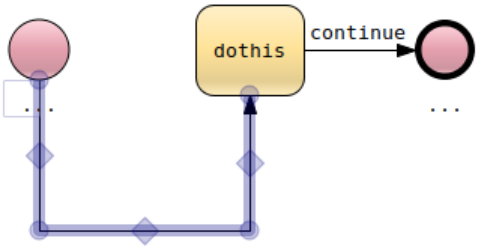

It is also possible to ask the editor to remove all waypoints in the segment to get a default straight link via the "Make link straight" entry in the contextual menu.

A link with a specific path can also be created from the start; to do so, hold the Shift key down while creating the link and click where the waypoints should be:



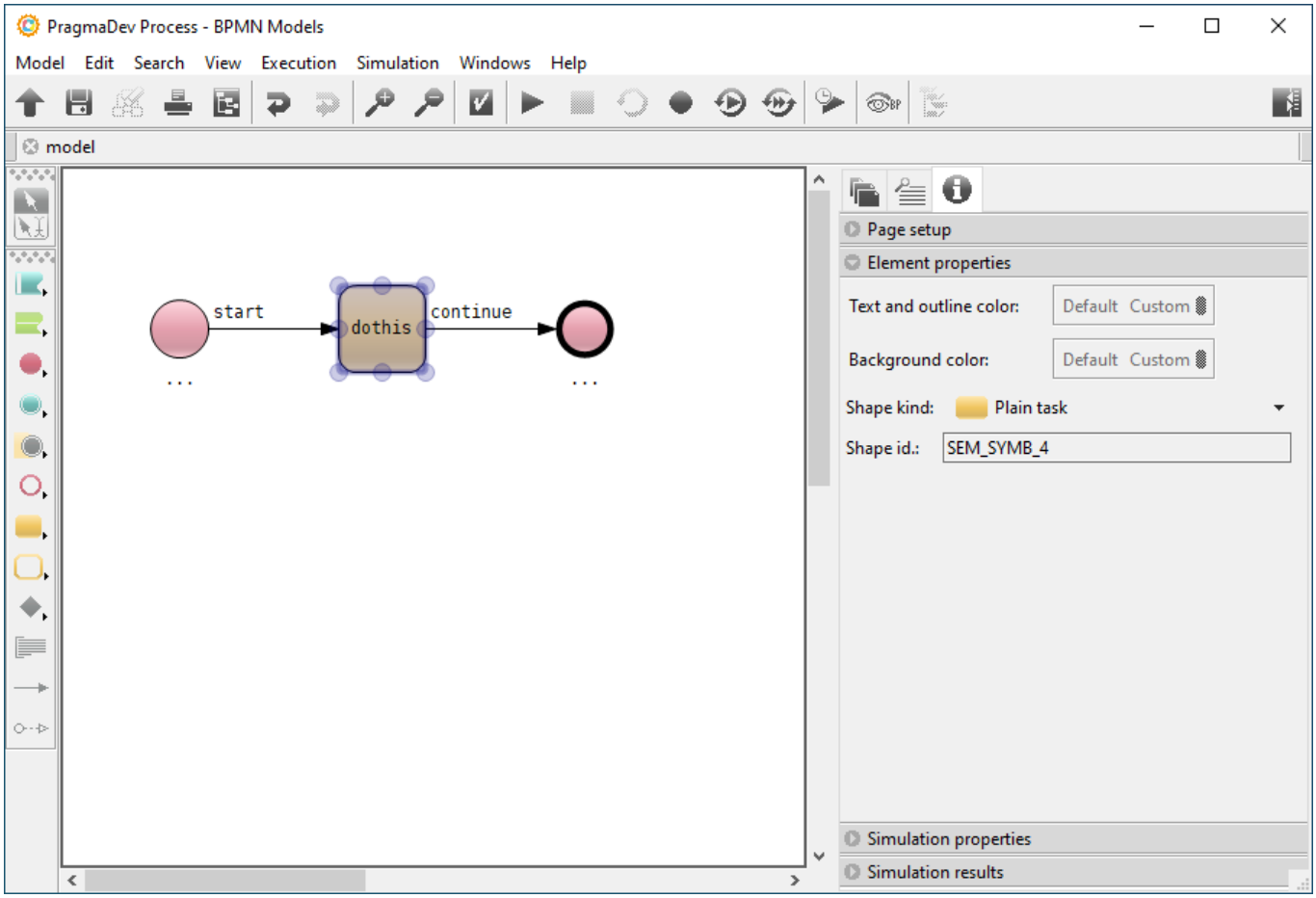



Note that any horizontal or vertical segment will show a diamond-shaped symbol when the link is selected:

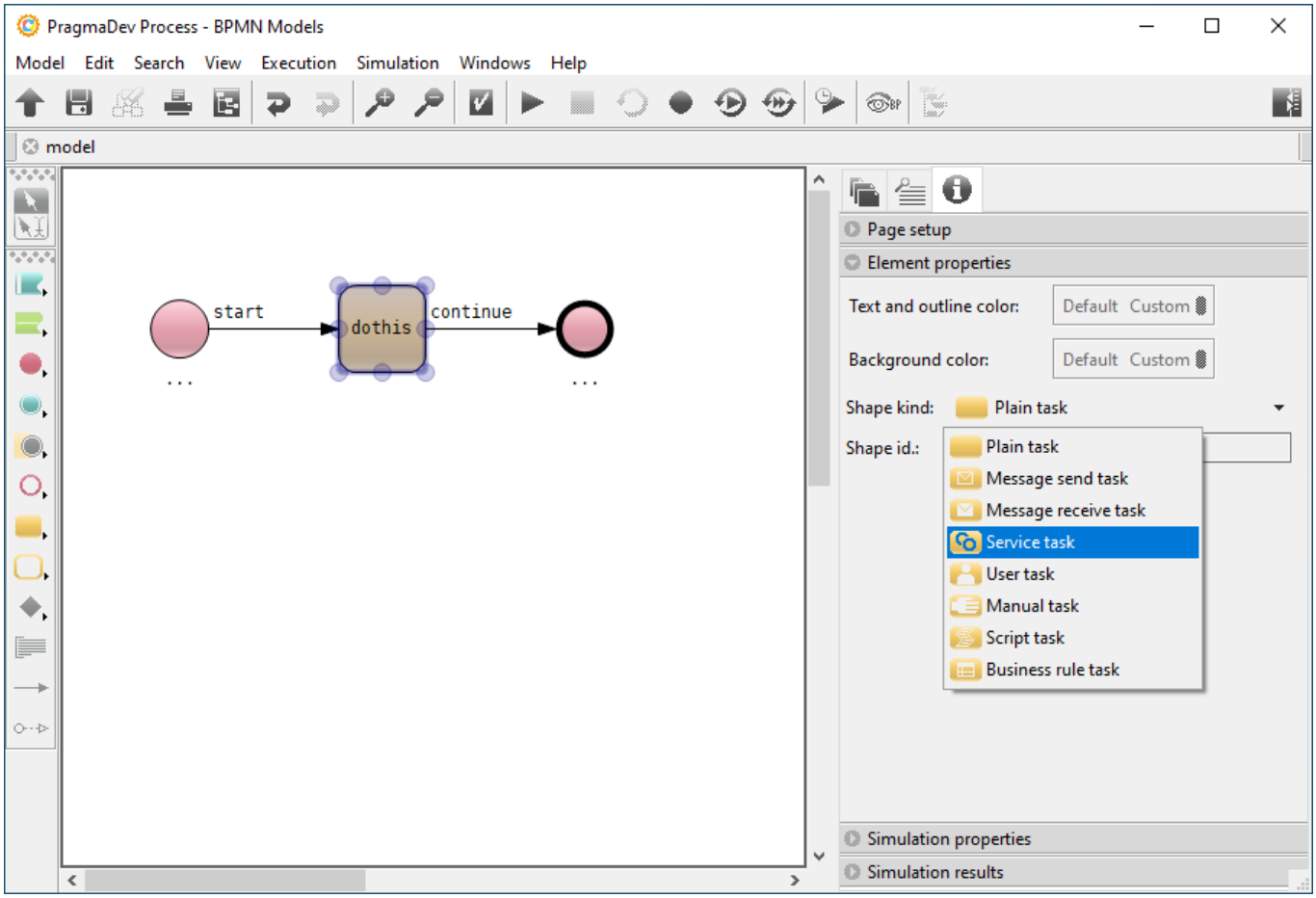



### 4.5.8 Modifying symbol types

When selecting a symbol in the editor, some complementary information is displayed in the right panel. If the panel is not displayed, it can be accessed directly by right-clicking the symbol and then *General properties...*:

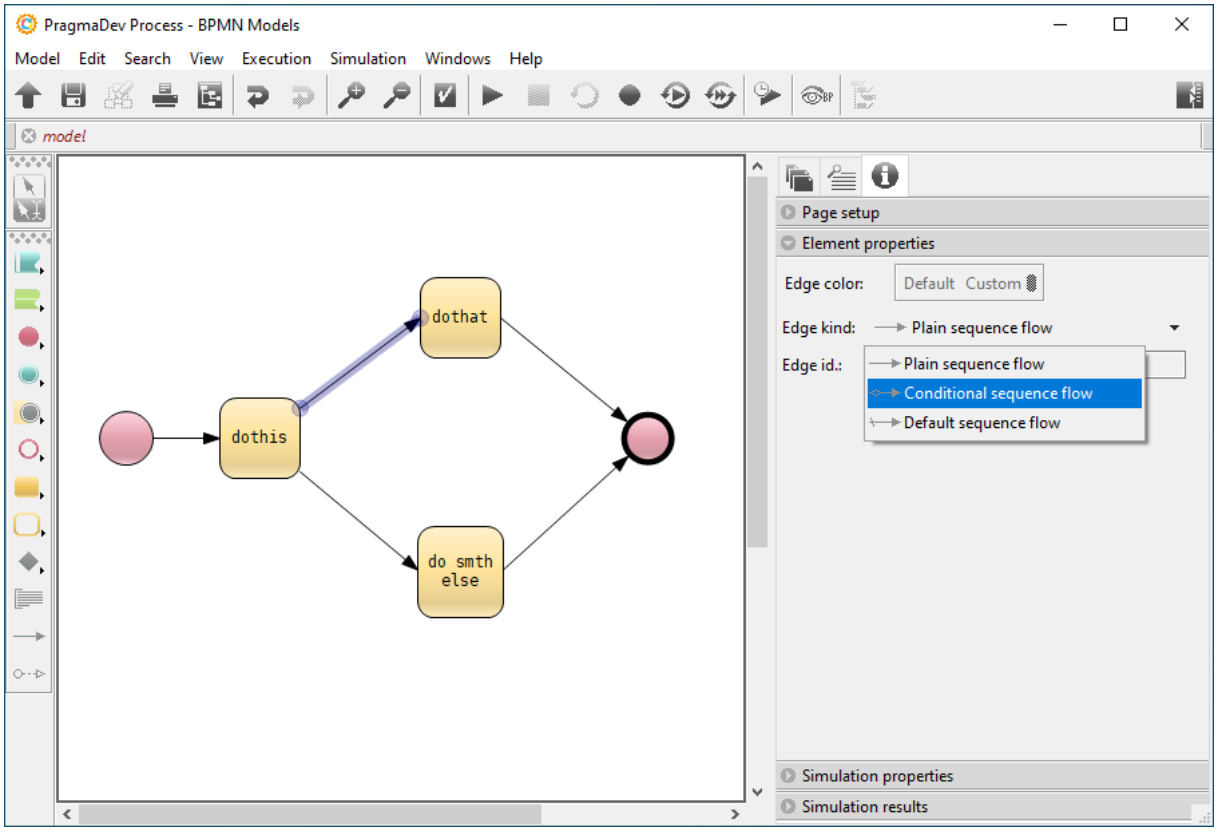



Click on the *Shape kind* to change the symbol type:




### 4.5.9 Modifying link types

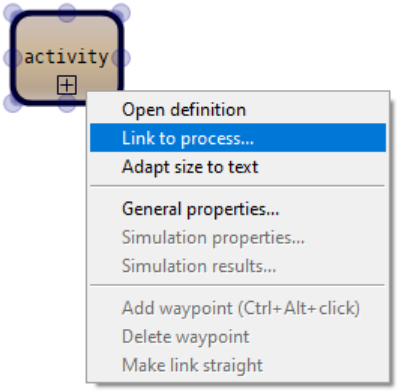
Right-clicking on a sequence flow and then *General properties...* will display some complementary information in the right panel. Click on the *Edge kind* to change the link type:



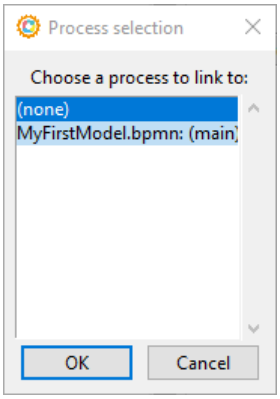

## 4.5.10 Connecting Call activities

A Call activity references another process.

To link the Call activity to the other process description right click on the symbol:



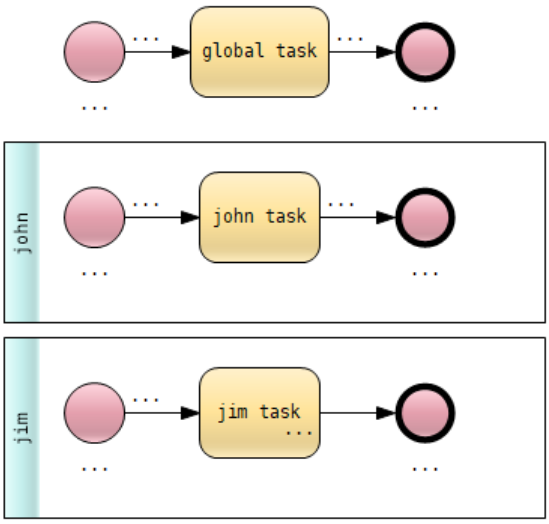
This will open a selection window that will list the possible processes:



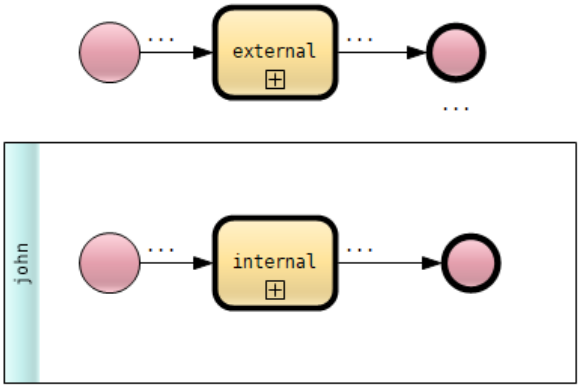

Now it is important to note the list of possible choices depend on the context of the process as well as the context of the caller. The possible connection will list:

- processes that are in the same pool as the caller,
- processes that are in no pool (implicitly the pool of the caller).

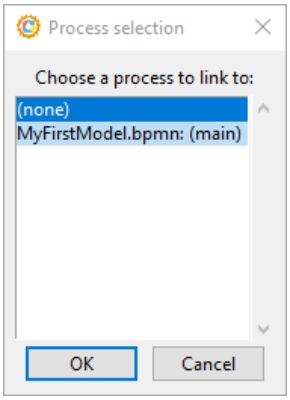
To illustrate this let's take the example of three processes described in MyFirstModel diagram, one is defined outside of any pool, one is defined in john pool, and the last one is defined in jim pool:




In another diagram we have two call activities, one outside of any pool, and one in john's pool:

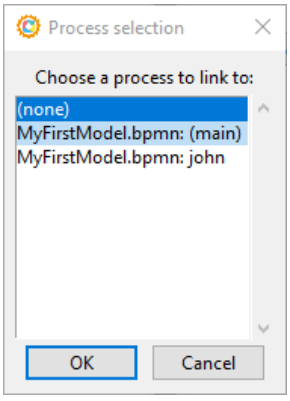



The possible links for external are:




The only possible choice is (main) of MyFirstModel meaning the process defined out of the pools.

The possible links for internal are:




The two possible choices are:

- the process that has been defined outside of any pool,
- the process defined in john's lane.

Once the link is done it will be used for navigation: double click will open the process definition, and for execution: click on the call activity will execute the process.

### 4.5.11 Printing and exporting

Diagrams in a BPMN model can be exported as an image or printed in a paginated way. Models can also be printed, which will print all of the diagrams in them in a single document.

Exporting a diagram as an image is done via the entries in the *Export diagram as* sub-menu in the *Model* menu. The available image formats are PNG, PDF, Encapsulated PostScript, CGM v2 and CGM v3. It will ask for the name of the output file and export the whole diagram as a single page in the specified file.

Printing a diagram is done via the entries in the *Print current diagram* sub-menu in the *Model* menu. The diagram can be printed to the printer or as a PDF file. This will take into account the page setup defined via the entry *Diagram page setup* in the *Model* menu and split the diagram in pages if needed.

The same feature is available for the model as a whole; the printed document will then include all the diagrams in the model, split into pages.

Notes:

- Exporting a diagram to PDF and printing it as a PDF file will not produce the same document. In the first case, the PDF file will always have exactly one page, that will have the size of the diagram itself, and no page borders or footers will ever appear. In the second case, the page size will be the one defined in the diagram page setup, and the document can have several pages, each with borders and footers.
- The page setup is for the moment not remembered with the BPMN model. The only way to remember a page setup is via the application preferences, but it will be the same for all models.

## 4.6 Heatmap

### 4.6.1 File format

The file format for a heatmap is described in:

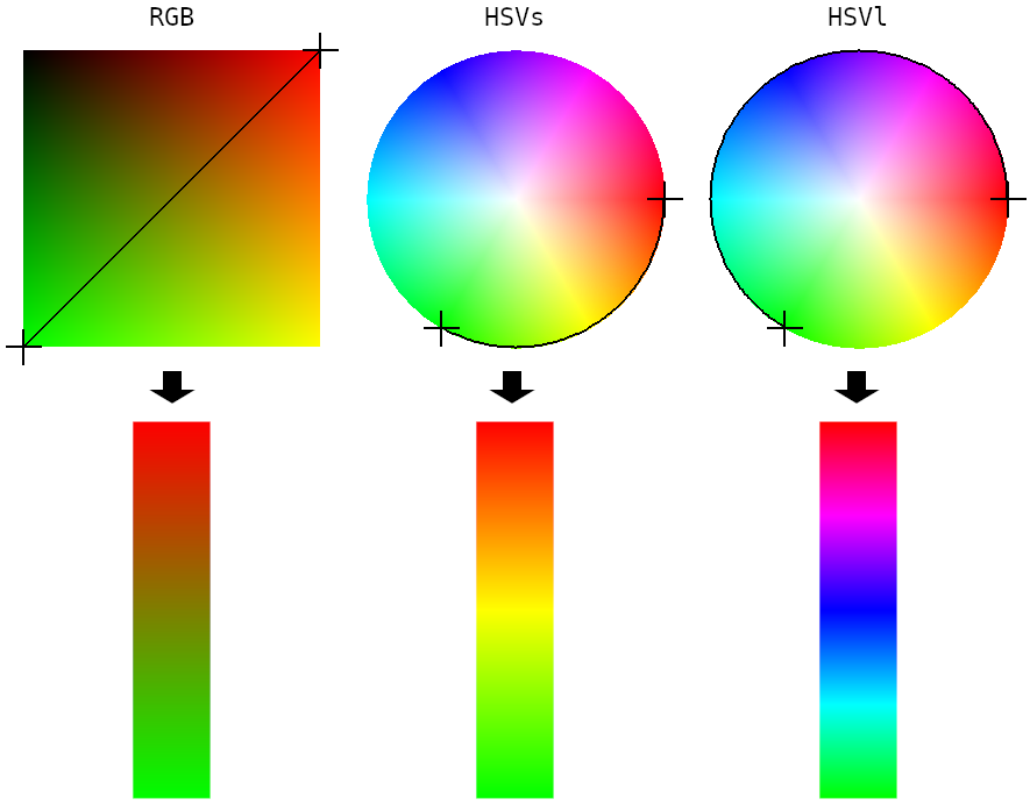
`$PRAGMADEV_PROCESS_HOME/rtds_rel/share/bpmn/heatmap.dtd`.

A simple example file is listed below that relates to the Pizza example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Heatmap SYSTEM "heatmap.dtd">
<Heatmap title="Example heatmap" min_value="0" max_value="30">
<Colors min_color="#00fbff" max_color="#FF0000" init_alpha="0.75" gradient_color_space="HSVl"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_10" value="0" label="Very low"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_18" value="5" label="Sequence flow"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_11" value="10" label="Average"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_19" value="15" label="Sequence flow"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_12" value="16" label="Event gateway"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_30" value="17" label="Sequence flow"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_24" value="20" label="Message catch"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_31" value="25" label="Sequence flow"/>
<Element model_file="Pizza.bpmn" id="SEM_SYMB_26" value="30" label="Plain task"/>
</Heatmap>
```

The tags and attributes are pretty much self explanatory:

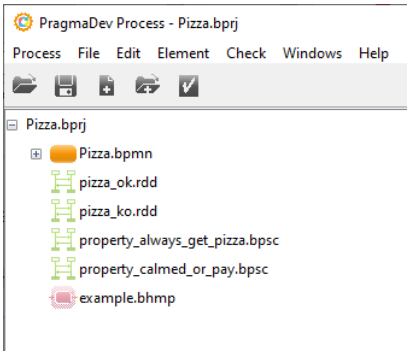
- `Heatmap` has a `title`, a `min_value` and a `max_value`. If any of these values is not specified, it is figured out from the values for the elements.
- `Colors` have a `min_color` and a `max_color`, which default to green & red. The `init_alpha` is the value for the alpha channel for the color of the part of the "halo" around the elements that is closest to the element itself. This is a number between 0 and 1, 0 meaning fully transparent and 1 fully opaque; the default value is 0.75. The `gradient_color_space` is the color space to use to interpolate the colors for the elements. The possible values are "RGB", and HSV in two variants: one for the shortest path between the colors in the HSV colors space ("HSVs"), and the other one for the longest path ("HSVl"). The default is "HSVs". Here is how the gradient between the default colors green & red will look in the 3 possible color spaces:



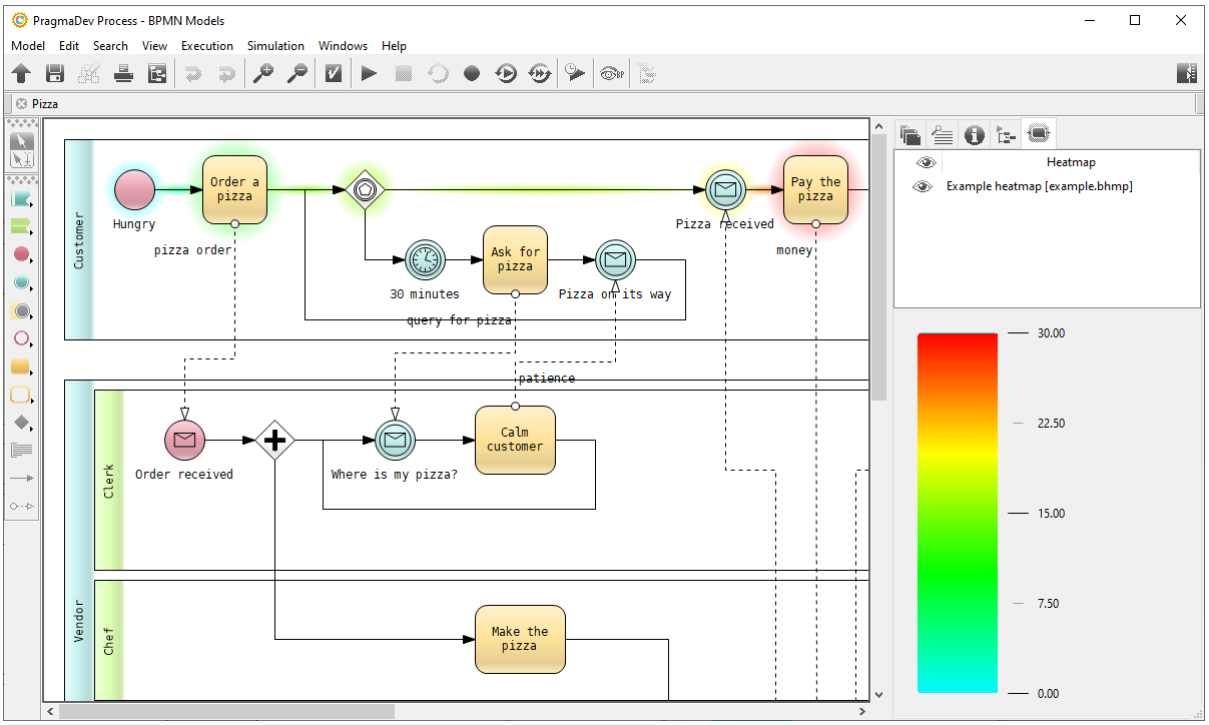

- `Element` has a `model_file` (a directory must be specified if the model is not in the same directory as the heatmap; the directory can be relative), an `id` which is the BPMN identifier of the element in the model, a heat `value` and a `label` which value will be displayed as a tool tip in the editor.

### 4.6.2 Display

The heat map above needs to be loaded in the project through the File / Add existing files to project... so that the heat map appears in the project.




When opening the BPMN diagram the heat maps that are in the Project Manager will be listed in the heat map tab in the right panel of the window. Click on one of the heat maps to display it.



# 5 Executor

## 5.1 Underlying principles

Each BPMN element in the model has a state of execution:

- *None*
  The element does not accept any action from the user, and it has never been enabled or disabled.
- *Active*
  The element is waiting for either an enabling or disabling action from the user.
- *Ready*
  An enabling action was issued on the element, but the element cannot be enabled yet because it depends on the state of other elements.
- *Enabled*
  An enabling action was previously issued on the element, and all enabling conditions have been fulfilled (i.e., the other elements it depends on are in the required state).
- *Disabled*
  A disabling action was issued on the element.

There are two types of actions, that can be issued only on *Active* elements:

- *Enabling*: click on active element.
- *Disabling*: right-click on active element.

In general, these actions can be issued on sequence flows, message flows, and process call-activities.

Since an element can be enabled or disabled several times (i.e., many flows of execution can go through the same element), each element has actually a list of states of execution. During execution the most recent state of the element is displayed in color as show in the following table.

Table 5.1: Color representation of execution states.

| State | Color | Example |
|---|---|---|
| None | No color | |
| Active | Blue | |
| Ready | Orange | |
| Enabled | Green | |
| Disabled | No color | |

Having the most recent state shown, an implicit priority is created in displaying execution states, i.e., *Active* has the highest priority, then *Ready*, and last *Enabled* and *Disabled*. For example, given an element that is *Active* in a flow and *Ready* in another flow, its *Active* state will be shown in blue color.

## 5.2 Controlling the executor

### 5.2.1 Via the graphical user interface

The executor is controlled from the diagram editor with the following tools:

- It is started with the *Start* button .
- Once started, it can be stopped with the *Stop* button .
- Once started it can be reset with the *Reset* button .
- The execution scenario can be recorded with the *Record* button .
- A single execution scenario can be replayed with the *Replay* button .
- Several execution scenarios can be replayed with the *Replay all* button .
- Last execution step can be undone with the *Undo* button .
- Last undone execution action can be re-executed with the *Redo* button .

### 5.2.2 Via the command line interface

A BPMN model can also be executed via a command line. The command line interface for PragmaDev Process uses a single command called `pragmaprocesscommand`, which accepts a sub-command, then the options and arguments for this sub-command. The sub-command for a simulation is `execute`, so simulating a model via the command line interface is done via:

```
pragmaprocesscommand execute \
        [ -m ] \
        [ -s ] \
        [ -v ] \
        [ -d <diagram name or index> ] \
        [ -p <port number> ] \
        <BPMN model file name>
```

The command arguments & options are the following:

- `<BPMN model file name>` is the name of the model to execute. It must be present in the command.
- If the `-d` option is present, it must be followed by either the index or the name of the main diagram in the model, i.e the one on which the execution must be started. If it is not specified, the executor is launched on the first diagram in the model.
- If the `-p` option is present, it must be followed by a number between 1024 and 65535. This is the port where the executor will be listening for connection. If the option is not specified, a random free port will be chosen.
- If the `-m` option is present, message flows will be allowed in the same pool (see "Executor preferences"). If the option is not specified, message flows will not be allowed in the same pool, which is the standard BPMN semantics.
- If the `-s` option is present, sequence flows may cross pool boundaries (see "Executor preferences"). If the option is not specified, sequence flows will not be allowed between different pools, which is the standard BPMN semantics.
- If the `-v` option is present, additional information regarding execution will be displayed on STDOUT. If the option is not specified, no addition information will be shown.

The executor will be started in server mode waiting for connection on the chosen port. A single client can connect and control the executor using via a socket; multiple connections are not supported. The client can send commands (with parameters) to the executor. The executor will always reply with the result of the requested command. Command requests and replies are serialized using JSON. The available command requests and possible answers are listed in table 5.2, and the action representation in table 5.3.

Table 5.2: PragmaDev Process Executor CLI commands.

| Command | | | Description |
|---|---|---|---|
| **Get state** | R | `{`<br>`  "cmd"  : "get_state",`<br>`  "args" : []`<br>`}` | Request current execution state. |
| | A | `{`<br>`  "status" : "ok",`<br>`  "args"   : [<number>]`<br>`}` | Return the current execution state represented by a number. |
| **Set state** | R | `{`<br>`  "cmd"  : "set_state",`<br>`  "args" : [<number>]`<br>`}` | Set current execution state. The list of arguments contains a single number representing the state. |
| | A | `{`<br>`  "status" : "ok",`<br>`  "args"   : []`<br>`}` | Acknowledgement. |
| **Get actions** | R | `{`<br>`  "cmd"  : "get_actions",`<br>`  "args" : []`<br>`}` | Request possible actions in the current execution state. |
| | A | `{`<br>`  "status" : "ok",`<br>`  "args"   : [<action>, ...]`<br>`}` | Return the possible actions in the current execution state. The list of arguments contains all actions. |
| **Execute action** | R | `{`<br>`  "cmd"  : "execute_action",`<br>`  "args" : [<action>]`<br>`}` | Request execution of the given action in the current execution state. The list of arguments contains the action to execute. |
| | A | `{`<br>`  "status" : "ok",`<br>`  "args"   : []`<br>`}` | Acknowledgement. |
| **Reset** | R | `{`<br>`  "cmd"  : "reset",`<br>`  "args" : []`<br>`}` | Reset executor. |
| | A | `{`<br>`  "status" : "ok",`<br>`  "args"   : []`<br>`}` | Acknowledgement. |

R=Request, A=Answer

Table 5.3: Action representation.

| | |
|---|---|
| { | |
| "action_id" : <number>, | Action identifier. |
| "element_id" : [<string>, ...], | BPMN identifiers of elements the action applies to. This is a list of sequence or message flow identifiers.<br>This field takes a single value (not a list) when action is passed to `execute_action`:<br>`"element_id" : <string>` |
| "instance_id" : <number>, | Diagram instance identifier (during execution). |
| "source_element_id" : <string>, | BPMN identifier of the source element. This is the source element of the sequence or message flows. |
| "source_element_type" : <string>, | Type of the source element. |
| "target_element_id" : [<string>, ...], | List of BPMN identifiers of the target elements. This is the list of identifiers of every target element of the sequence or message flows.<br>This field takes a single value (not a list) when action is passed to `execute_action`:<br>`"target_element_id" : <string>` |
| "target_element_type" : [<string>, ...], | List of target element types. There is one target element type for each element target id. |
| "action_type" : <string> | Type of action; either "enable" or "disable". |
| } | |

All fields are present in actions returned by `get_actions`. However, when passing an action to execute via the `execute_action` command, some fields may be omitted (`source_element_type` and `target_element_type` can always be omitted). Also, the fields `element_id` and `target_element_id` should be passed as a single value, not a list. The actual action that will be executed is identified based on the fields present:

- If `action_id` is present, then only this field is used to identify the action. All other fields can be omitted.
- If `action_id` is not present, but `element_id` is, then the later is used for identifying the action. The remaining fields are optional.
- If neither `action_id` nor `element_id` are present, then either `source_element_id` or `target_element_id` is used to identify the action. The remaining fields are optional.

Note that the fields which are set should uniquely identify the action, i.e., if more than one action is possible for execution at a given state, then the command will fail. When any of the commands (requests) in table 5.2 fails, the answer will be:

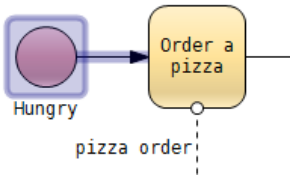
```
{
  "status" : "ko",
  "args"   : [<string>]
}
```

with the list of arguments containing a single value, i.e., the error message describing the failure.

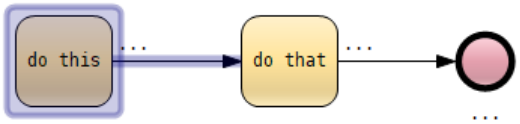
## 5.3 Behavior

### 5.3.1 Start

When starting, the executor will look for explicit and implicit starting points in the model. An explicit starting point is a sequence flow following a *Start* symbol:




Any outgoing sequence flow from a task without incoming flows can be an implicit starting point. Here is an example:




### 5.3.2 Sequence flows

Sequence flows can be either enabled or disabled. However, even though most sequence flows can be enabled (if they are *Active*), disabling an *Active* sequence flow follows the following rules:

- A normal (uncontrolled) sequence flow can be disabled only if it is an outgoing flow of an inclusive gateway, and at least one other outgoing flow of the gateway is still *Active* or has been enabled.
- In absence of normal (uncontrolled) flows, an outgoing conditional sequence flow can be disabled only if at least one other outgoing conditional flow is still *Active* or has been enabled.
- Default sequence flows can be always disabled if they are *Active*.

### 5.3.3 Message flows

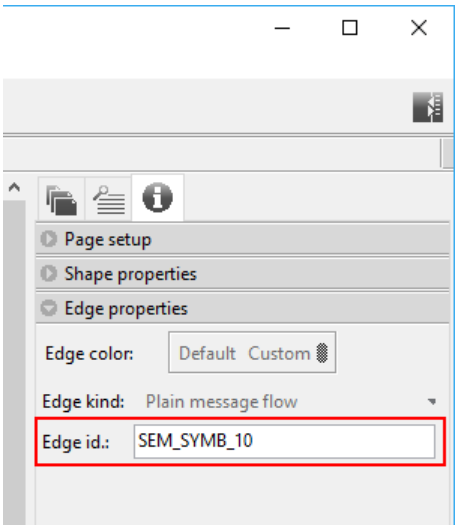
Message flows may accept only enabling action, i.e., they cannot be disabled. The *resolution* of a message flow is the identification of both its *endpoints* (sender and receiver). The Executor supports two kinds of message flow resolution:

- Implicit
  Both sender and receiver are flow elements (e.g., task, event, etc.). The Executor automatically identifies these flow elements as message flow endpoints. Message flows DO NOT accept any action from the user; they are enabled automatically by the executor.
- Explicit
  Either the sender or the receiver is a *black-box* participant (i.e., an empty pool with no flow elements). The Executor expects user action for identifying the endpoints. A message flow MAY accept an enabling action only if its source is a *black-box* (NOT a flow element).

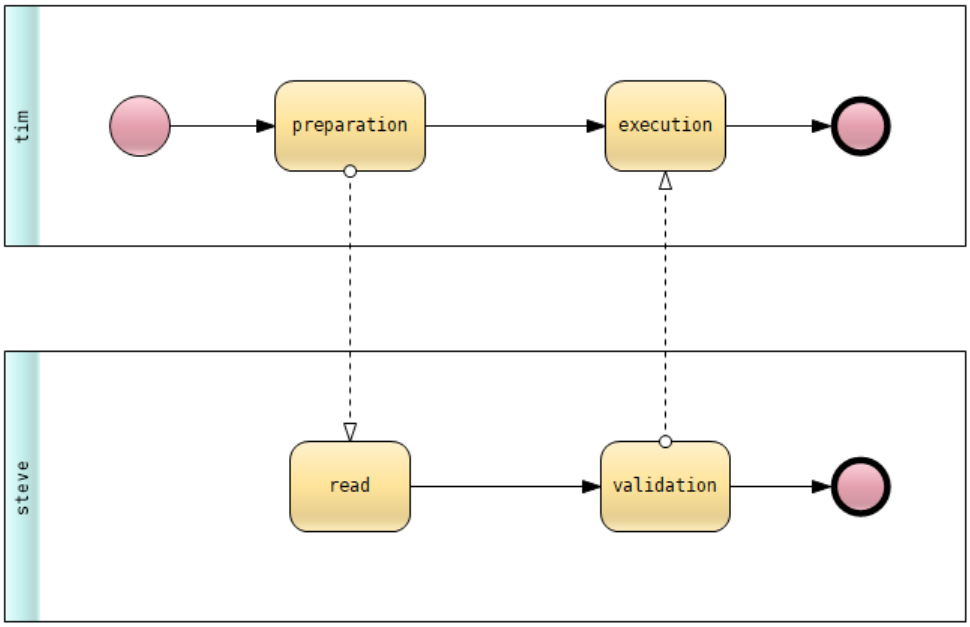
#### 5.3.3.1 Implicit resolution

The general message flow semantic is as follows:

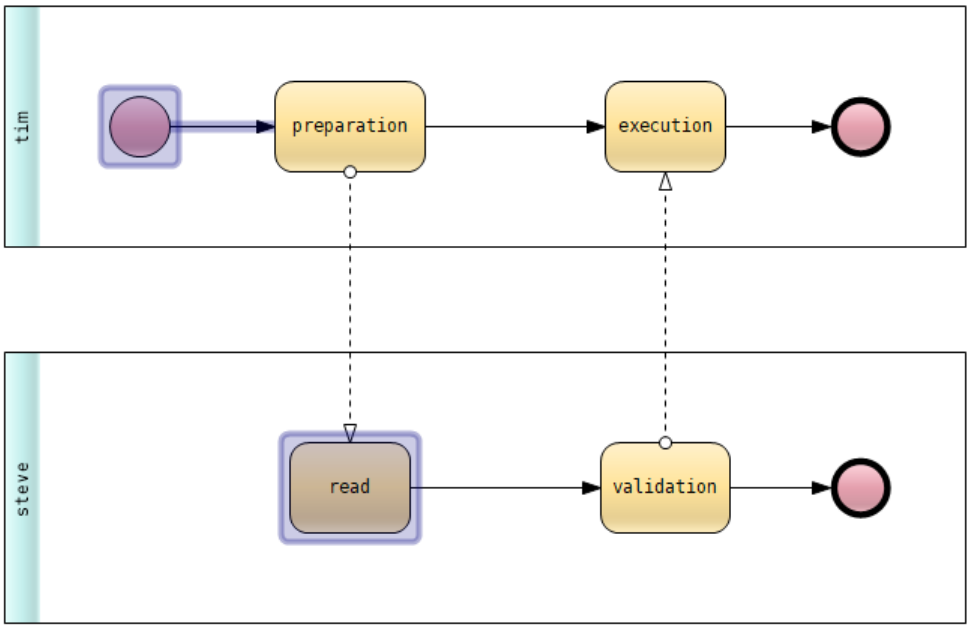
- All incoming message flows must be present (*Ready*) before executing the receiver of said message flows, i.e., outgoing sequence flows become *Active* when enabling actions have be issued on all incoming message flows.
- An automatic (implicit) enabling action is issued on all outgoing message flows when an enabling (or disabling) action is issued on any *Active* outgoing sequence flow. To ensure deterministic behavior, the order of enabling of the message flows is based on their BPMN ID (found in the *Properties inspector* in the right panel of the editor):



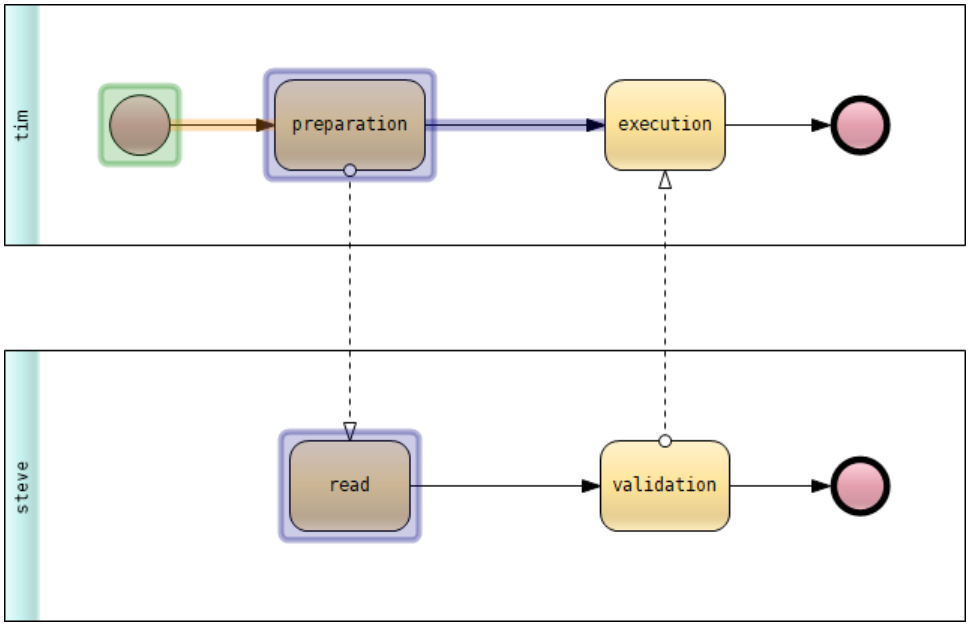

To illustrate this let's consider the following example:



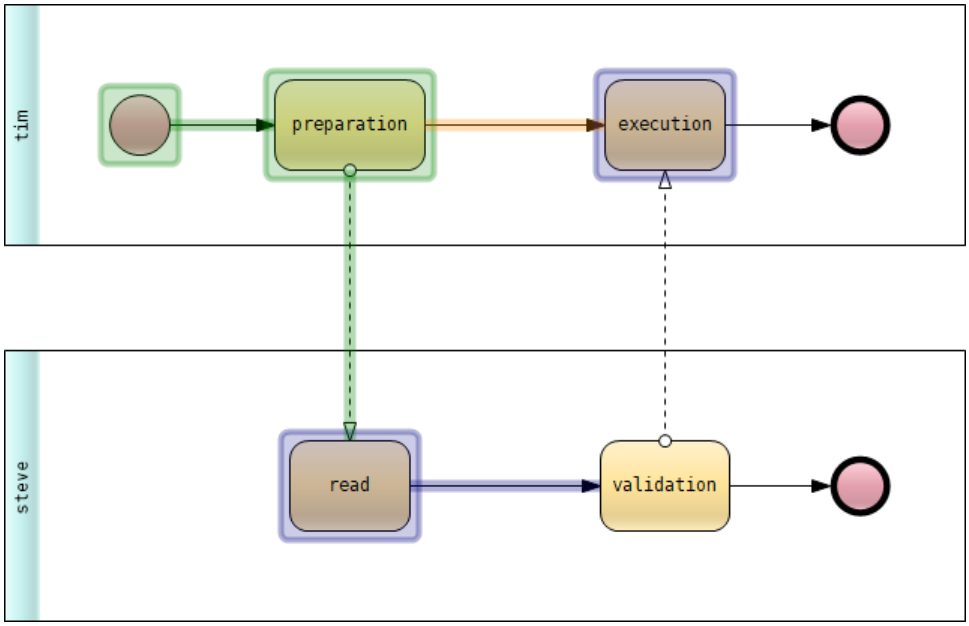

When executing the first possible action is to go from the *Start* symbol to the *Task*:



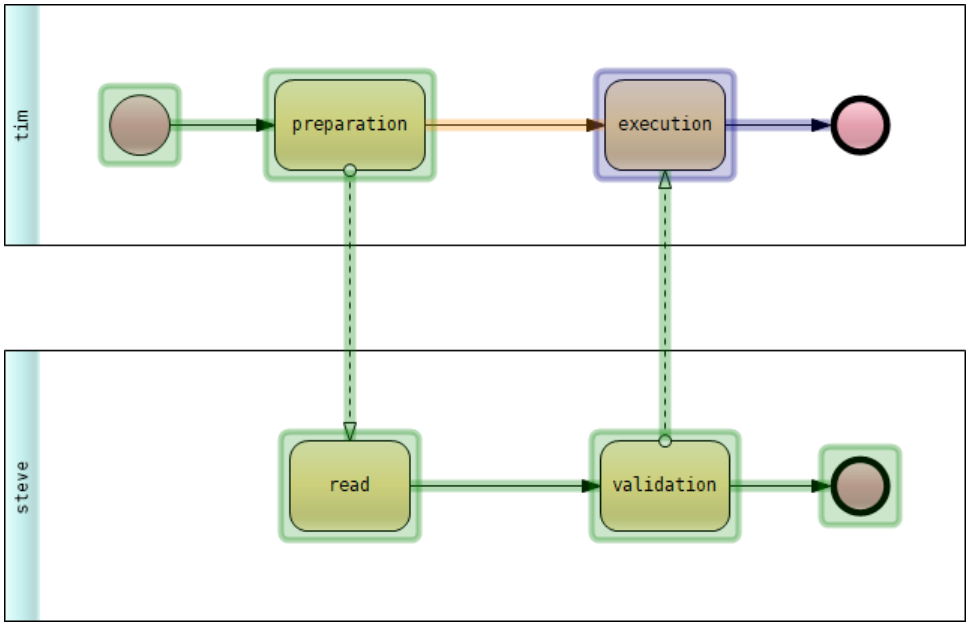

Then the only possible action is the sequence flow outgoing the *Task*:




Enabling the sequence flow will automatically send the message, `steve` can proceed now while `tim` is waiting for the second message:




Again, enabling the sequence flow will automatically send the message:



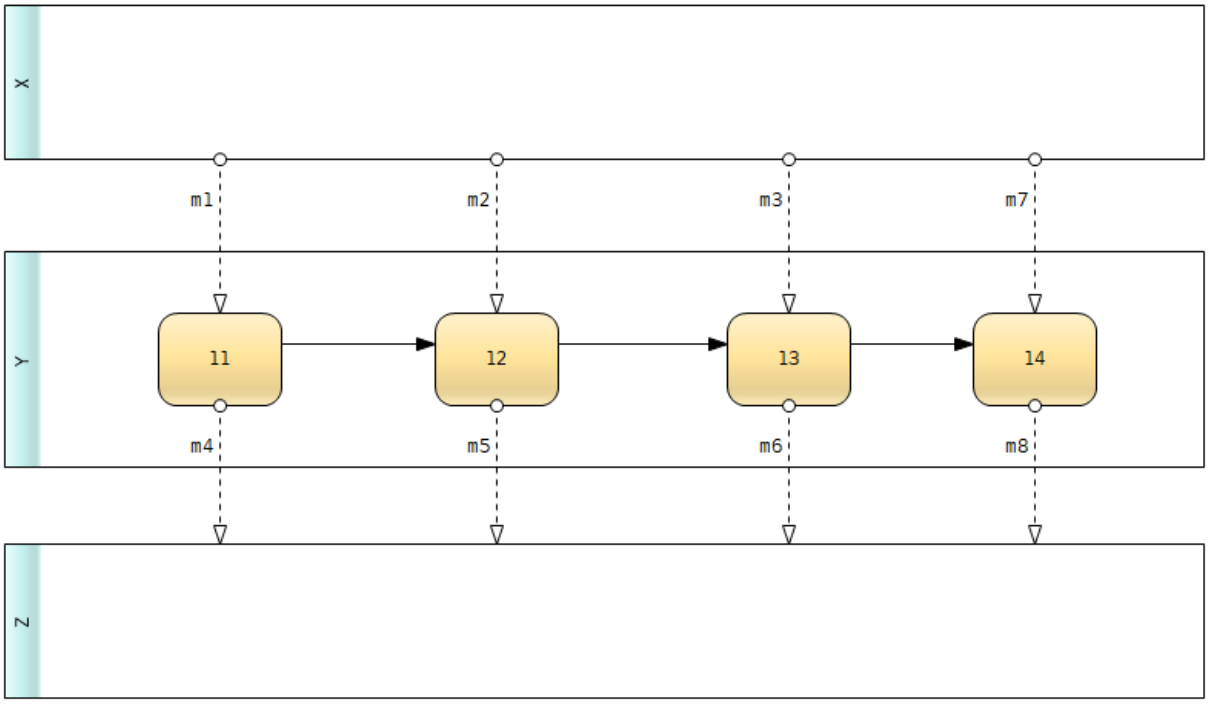

#### 5.3.3.2 Explicit resolution

Large models are described in several diagrams written by different modelers. In these situations the pools that are defined by the other modelers are represented by empty pools. We call these empty pools *black-boxes*. During execution these black-boxes might be defined or not, and even if they are defined one might not necessarily want to execute them. The Executor provides a way to handle all these situations with the concept of *gates*.[1] A gate is a small rectangle on the border of the pool where the message endpoint is found. The means that the message flow goes through the gate. At startup, for each gate, the Executor will look for a process that handles the same message flow from and to the same pools. If it finds such a definition, the gates can be enabled or disabled by the user (hence explicit resolution). If enabled it will link both processes, and if disabled it will act like the black-box is undefined. When the black box is undefined, it is up to the user to send as many messages as wanted. An undefined black-box will always receive all messages sent to it.

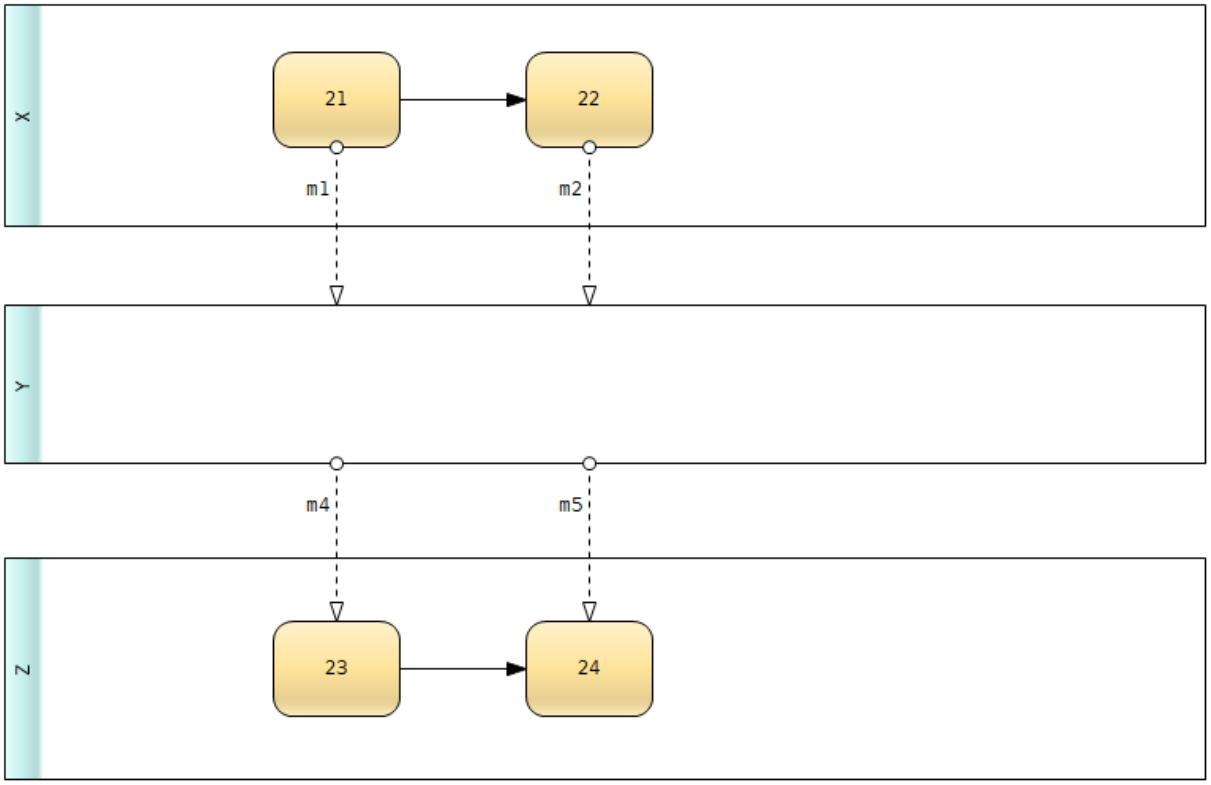
**Note**: the only situation where message flows are enabled (sent) explicitly by the user is when they are outgoing an undefined black-box. In all other cases the message flows are enabled automatically (impicitly) by the executor.

These concepts are illustrated by the following example with three diagrams. The first diagram describes a process in pool Y with messages exchanged with pool X and pool Z.

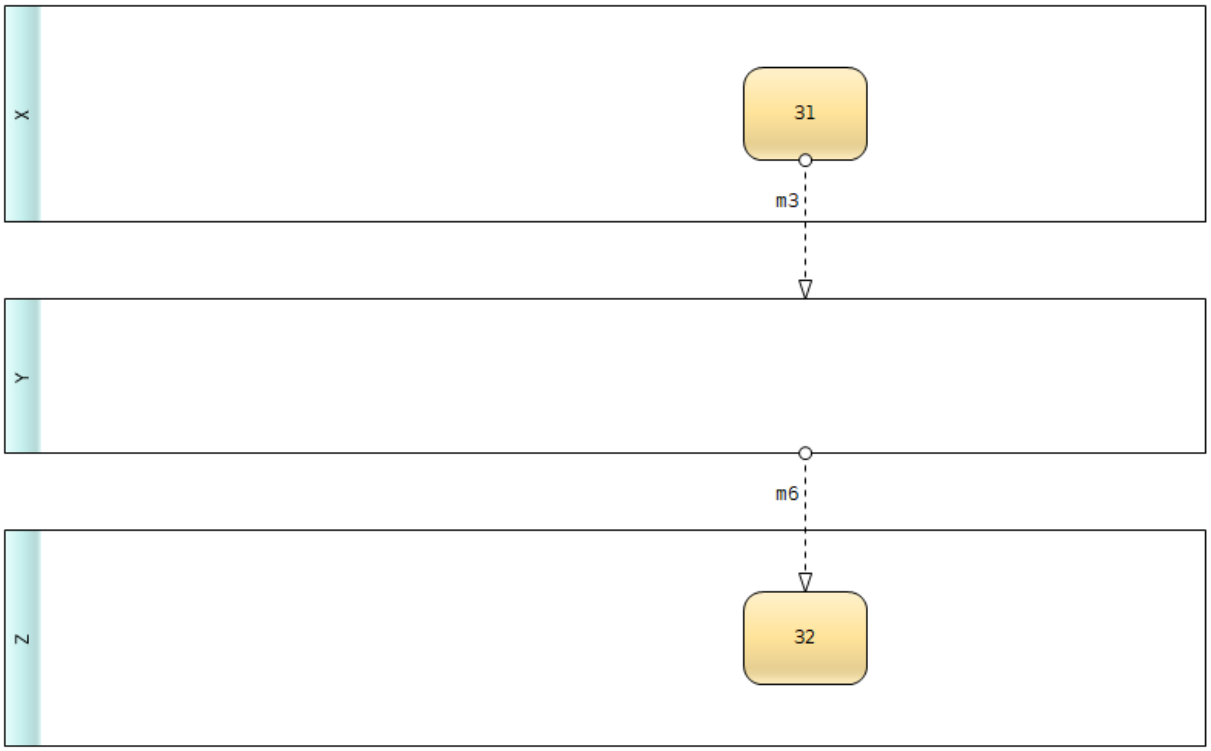
[1]The gate is not a BPMN symbol; it is a hidden symbol that may become visible only during execution to support explicit message flow resolution.



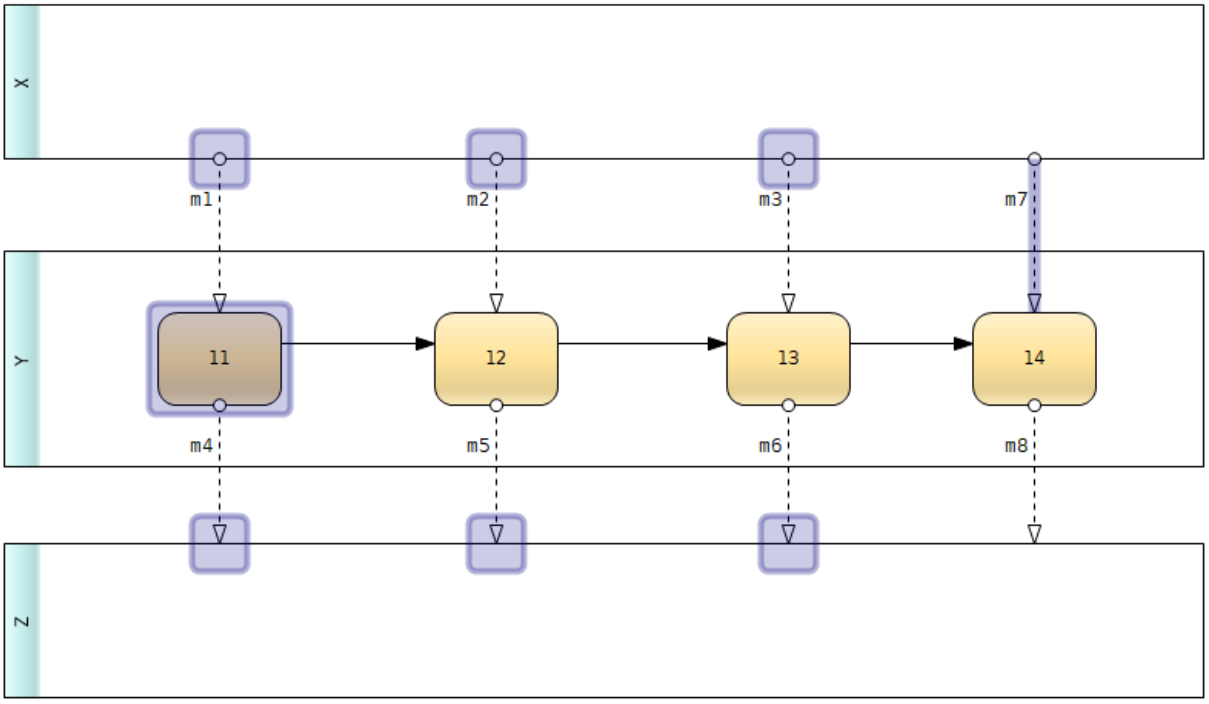

The `m1`, `m2`, `m4`, and `m5` interactions between `X` and `Y`, and `X` and `Z` are described in a second diagram:



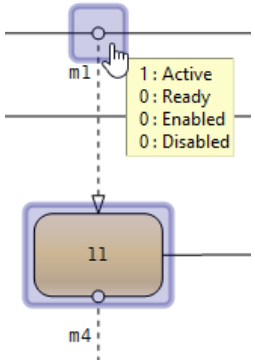

The `m3` and `m6` interactions are described in a third diagram:



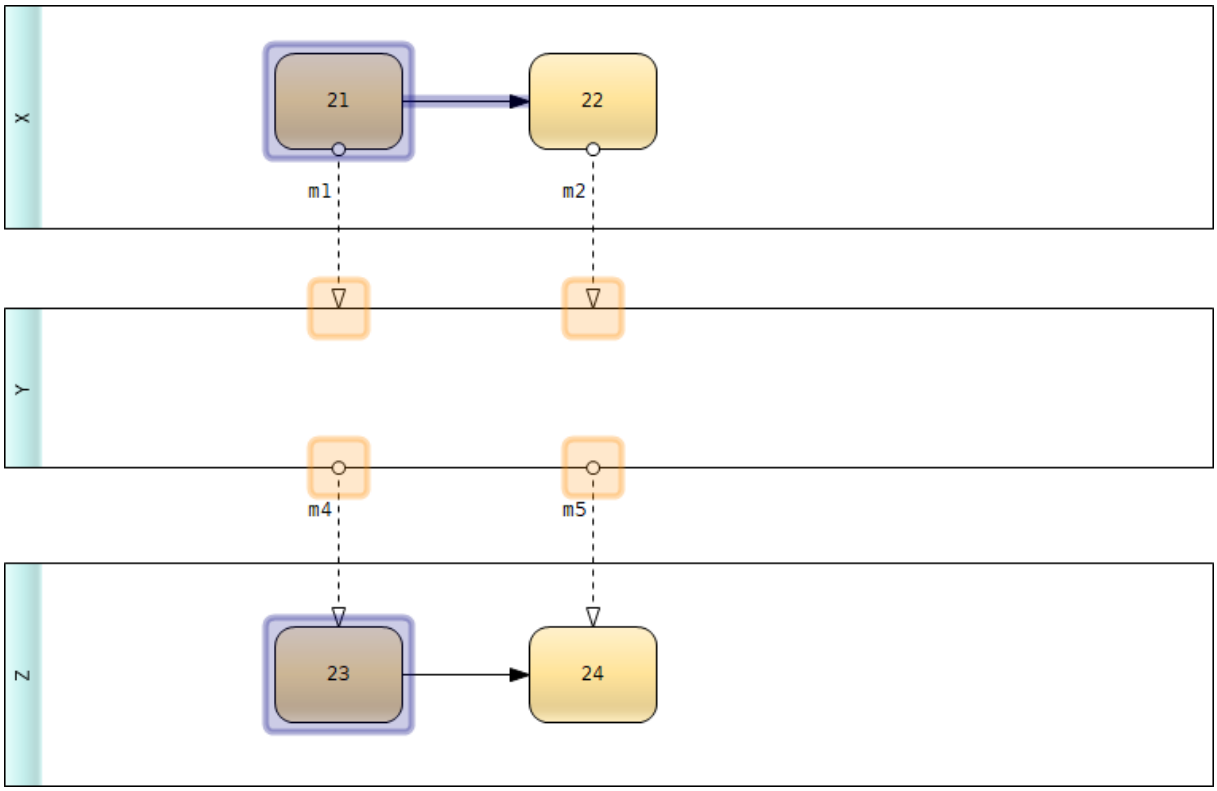

When starting the Executor the links between the three diagrams are analyzed. The `m1`, `m2`, `m3`, `m4`, `m5`, and `m6` related gates are *Active* meaning a diagram definition has been found for these message flows. It is possible to either enable or disable them.



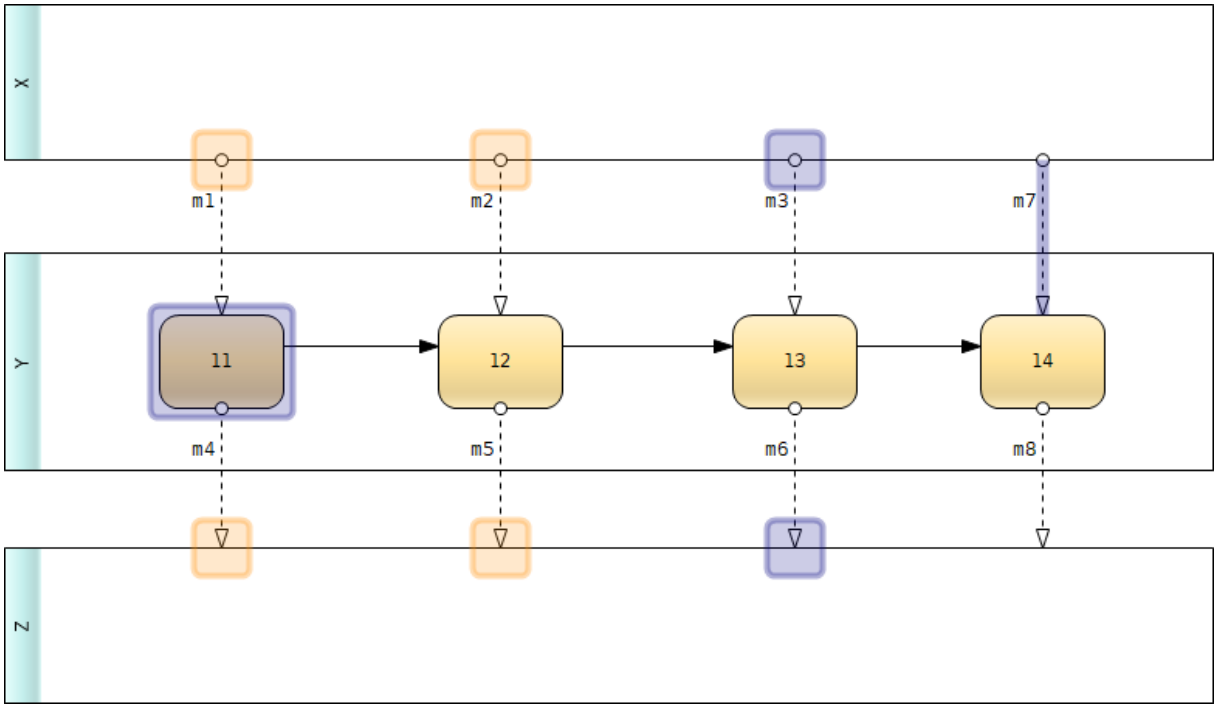

If we enable the `m1` related gate with a simple click on it:



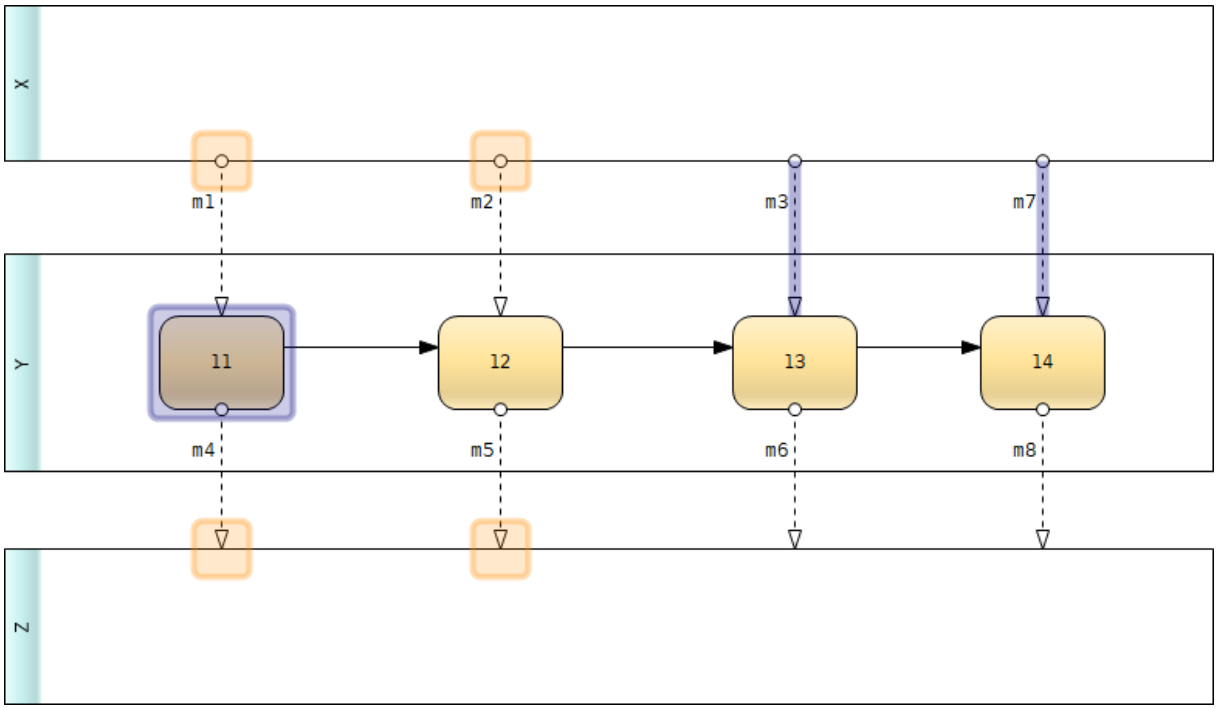

It will automatically include the related diagram in the execution:




The first diagram will display the following:




If we disable the `m3` related gate with a right click on it, the related diagram (the third one) is then ignored and pool `X` is considered a black-box for `m3`. There is no diagram that describes the `m7` interaction, so for `m7` pool `X` is also considered a black-box. In practice that means the message flows can be always enabled in the Executor:



### 5.3.4 Call activities

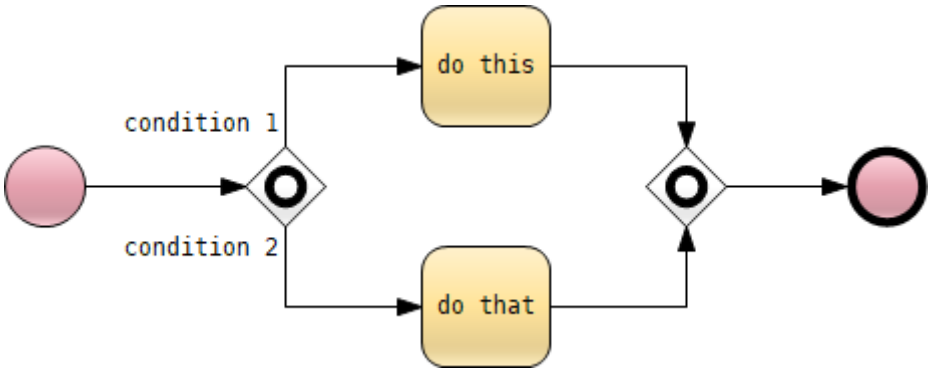
Process call-activities accept enabling or disabling actions during execution. Enabling a process call-activity means including the referenced process' diagram (if it exists) in the execution; this can be seen as a *step-into* the call-activity. Disabling a process call-activity means ignoring the referenced process and treating the activity as a simple task; this can be seen as a *step-over* the call-activity. So, when a process call-activity becomes *A*ctive during execution, a left-click will step-into it, while a right-click will step-over it.

### 5.3.5 Gateways

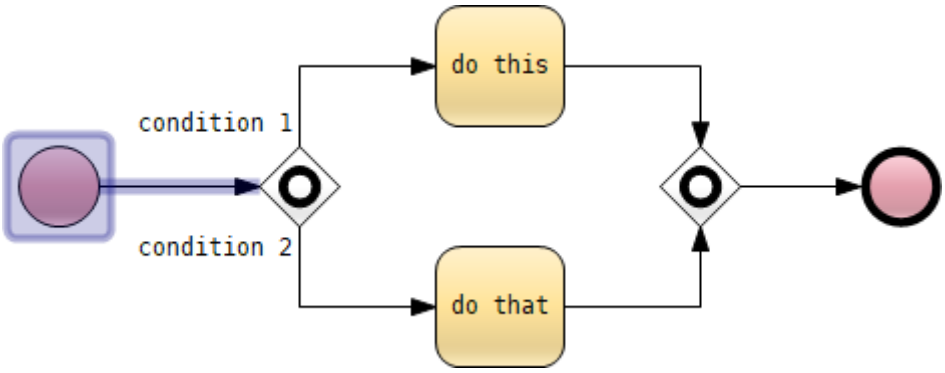
The behavior of the gateways is conform to the standard. As a reminder some typical cases are described in the following paragraphs.

#### 5.3.5.1 Inclusive

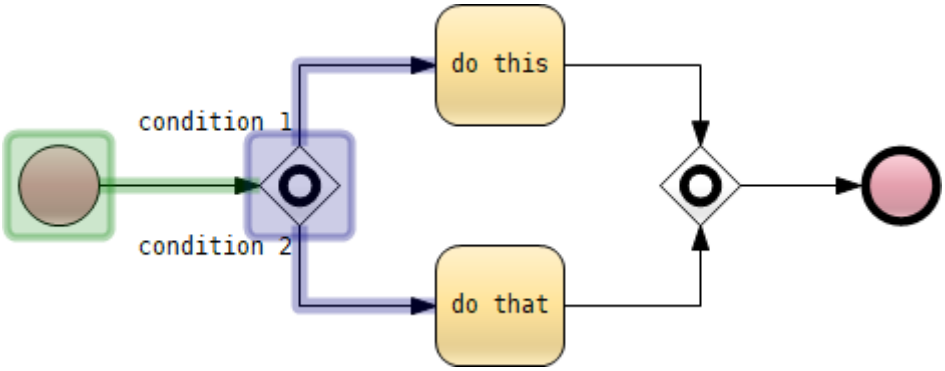
Let's consider a forking and a merging inclusive gateway:



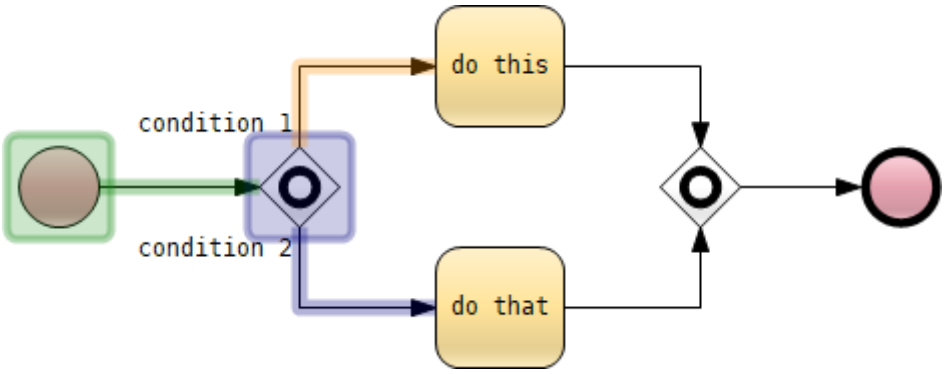

The forking gateway allows to enable or disable `condition 1` and `condition 2`. Not both branches can be disabled because no further action could be done. To enable a branch left click on it, to disable a branch right click on it. Let's execute our example, in the first step the start sequence flow can be enabled:



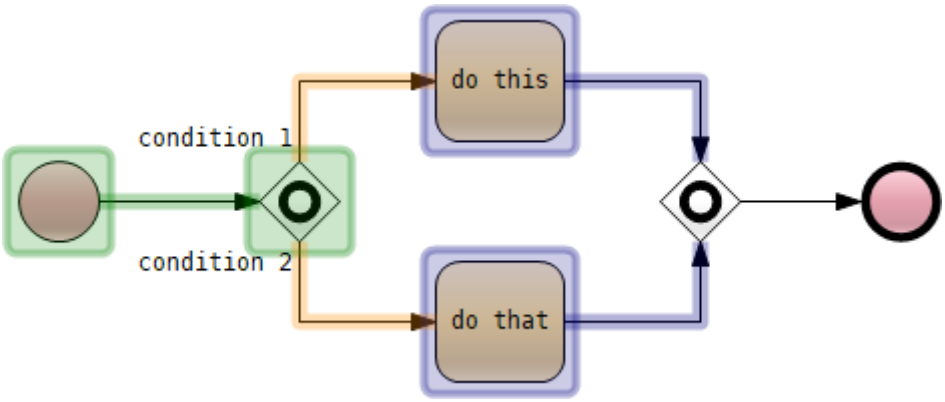

Then it is possible to enable any of the branches:



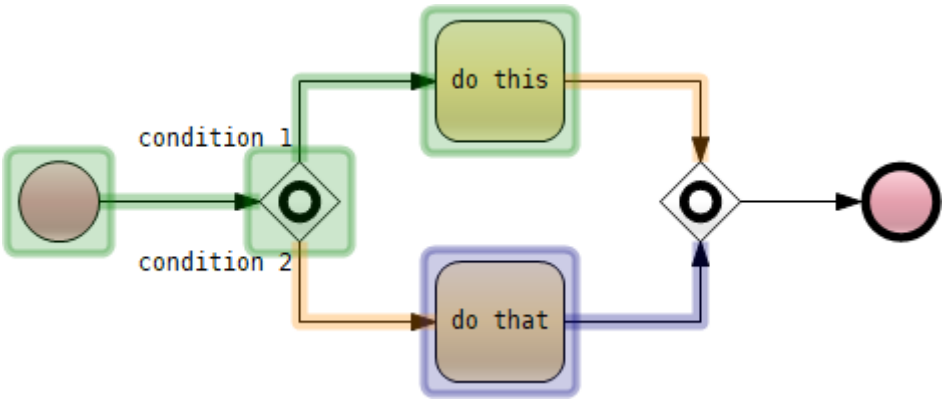

Let's enable `condition 1` branch:



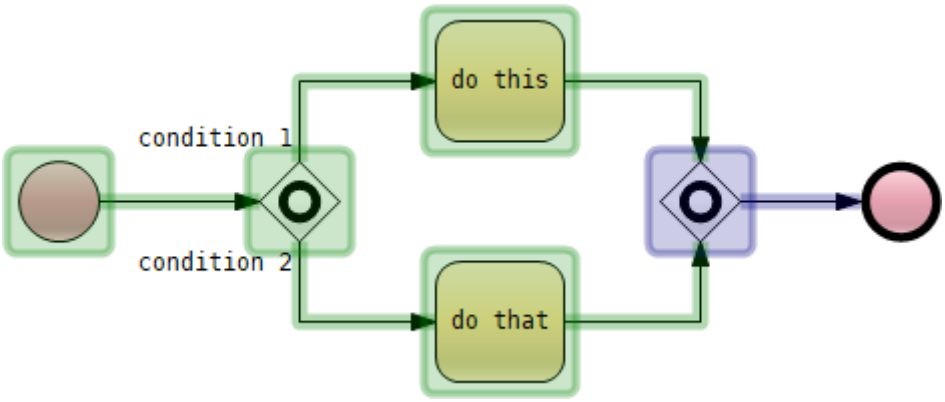

and then the `condition 2` branch:



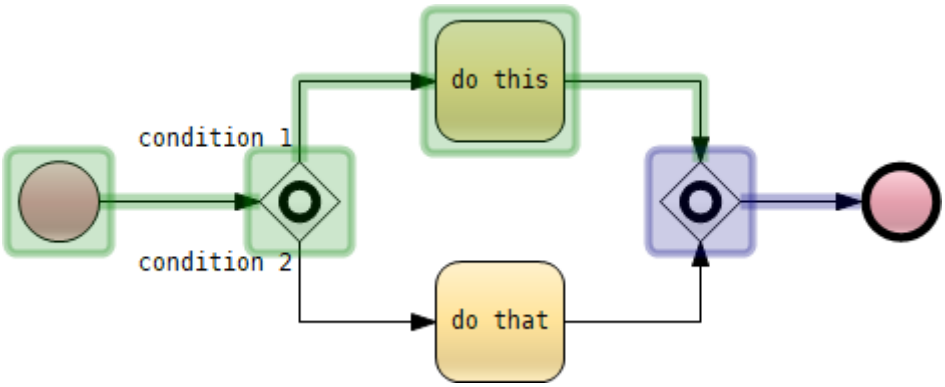

Let's continue with the sequence flow outgoing the `do this` task:



The merging inclusive gateway is waiting for the activation of the sequence flow outgoing the `do that` task. Doing so will activate the outgoing sequence flow of the gateway:

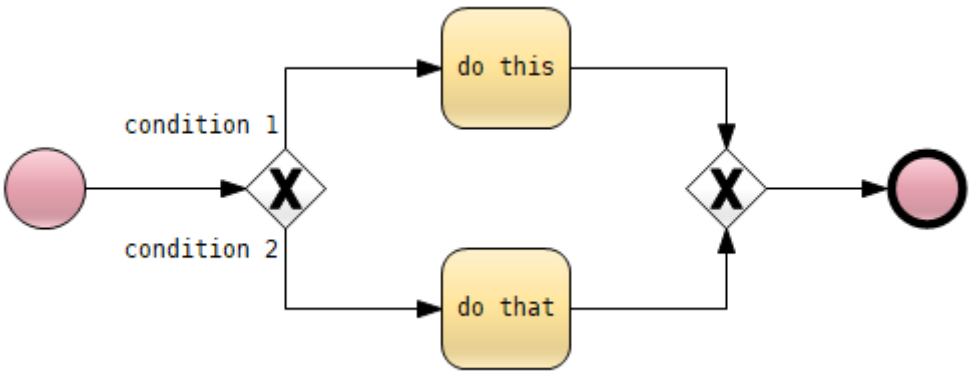



It would have been possible to disable `condition 2` with a right click. After that, enabling the outgoing flow of the `do this` task would have resulted in the activation of the outgoing sequence flow of the gateway:

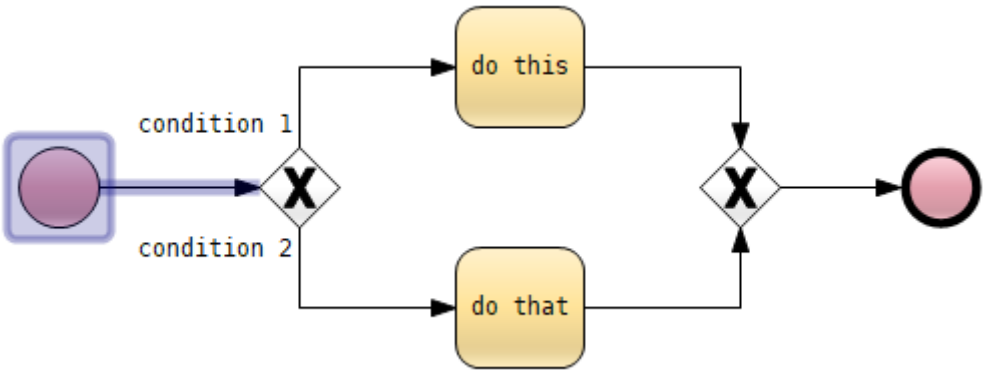



#### 5.3.5.2 Exclusive

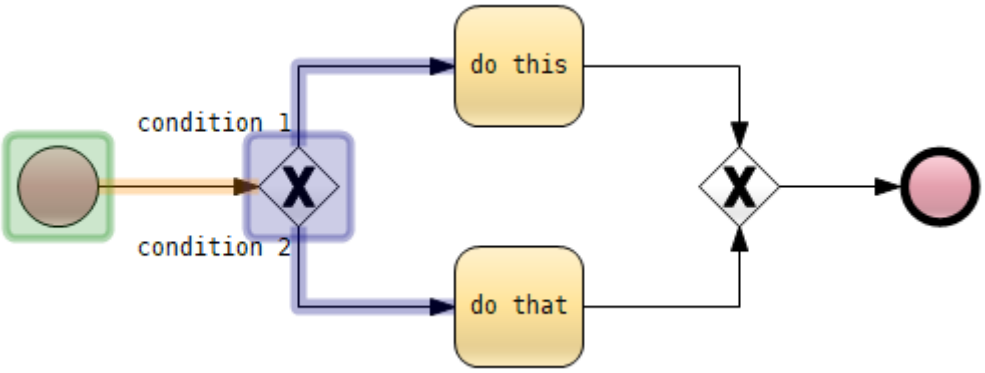
Let's consider a forking and merging exclusive gateway:



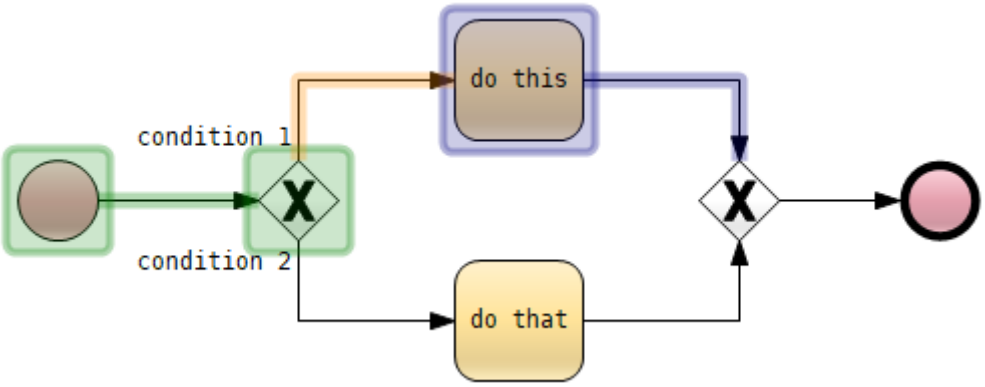

Execution of this simple diagram will first propose the sequence flow following the start event:



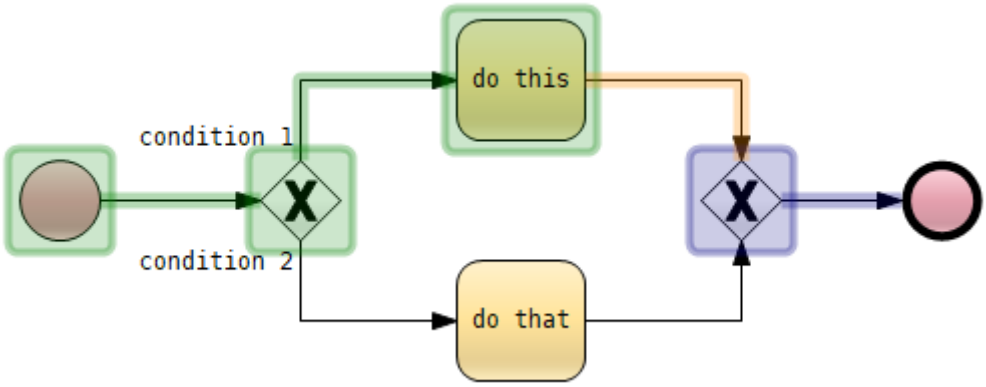

Getting to the gateway will propose any of the outgoing branches:




When enabling one of the branches, the other ones get disabled:

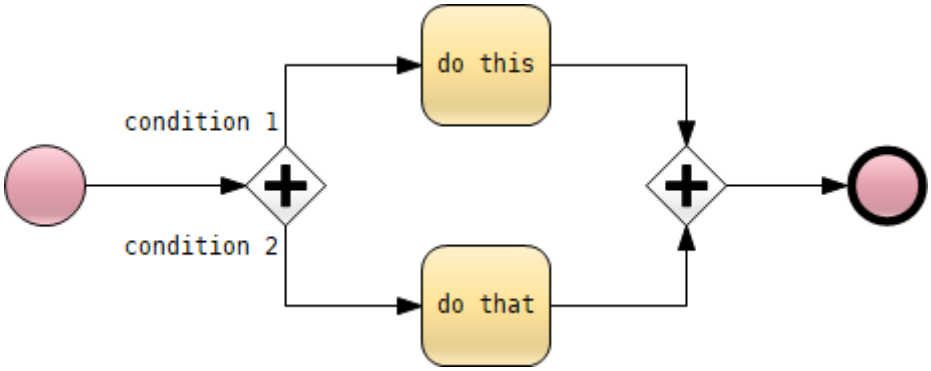



The merging gateway does not expect any other active branches to let the flow go through:

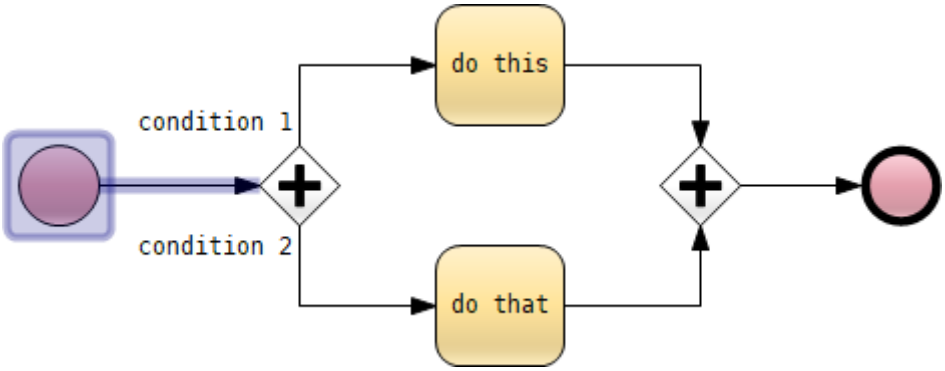


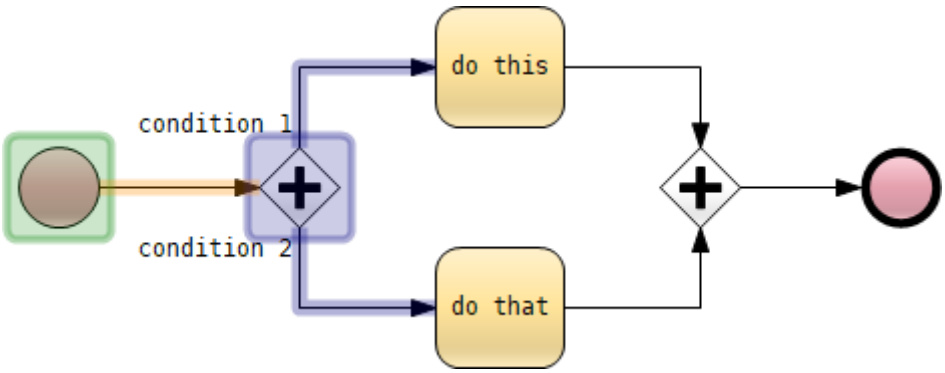

#### 5.3.5.3 Parallel

Let's take a simple parallel forking and merging gateway:

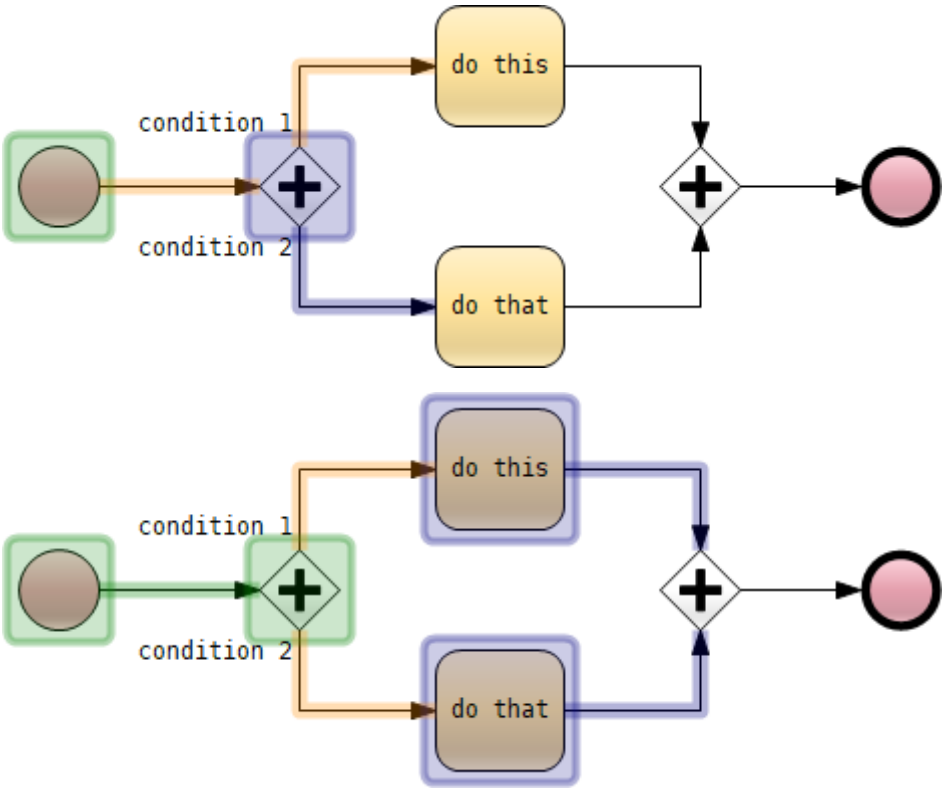



Let's proceed until the forking gateway will activate both branches:

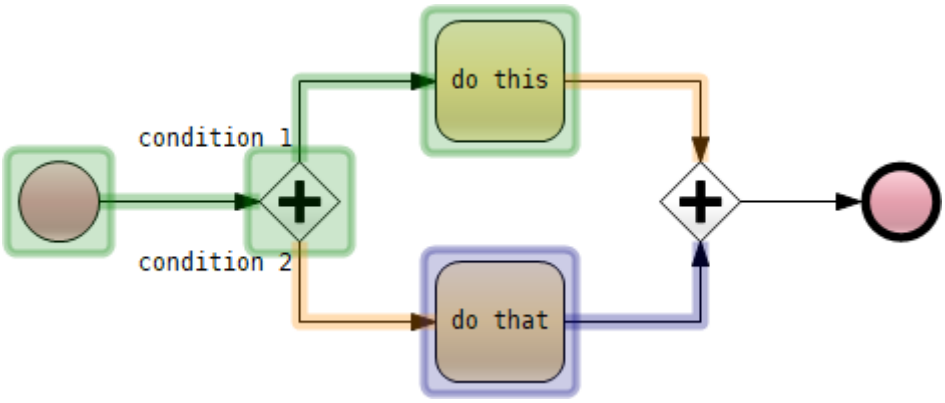


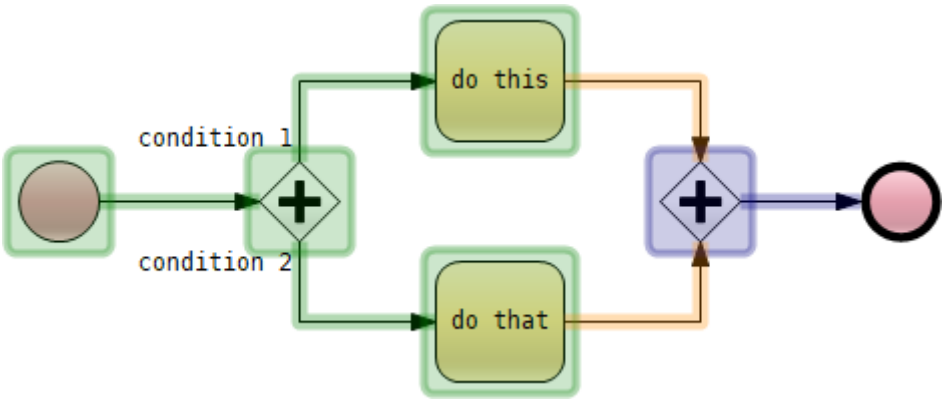



There is no way to disable a branch, both paths have to be enabled for the merging gateway to allow further execution:



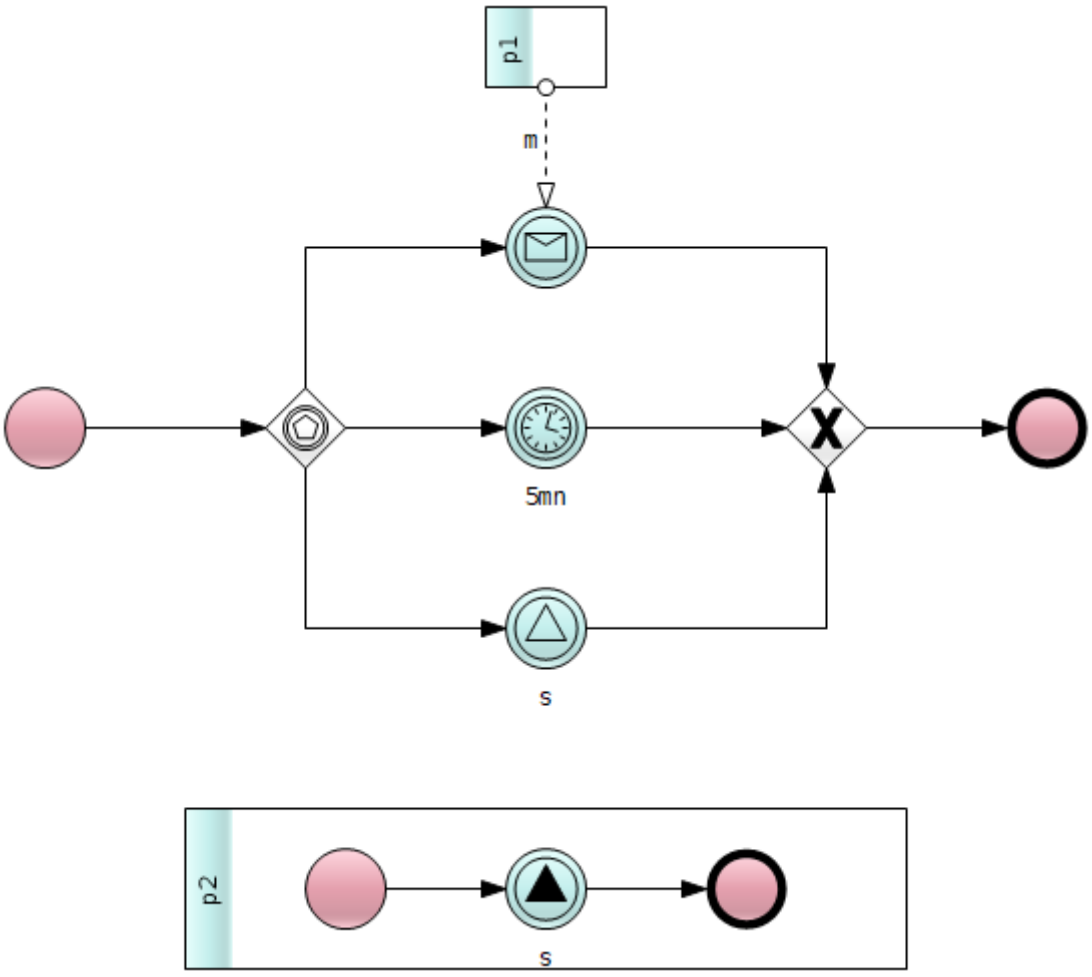

The merging gateway is waiting for both branches to allow further execution:



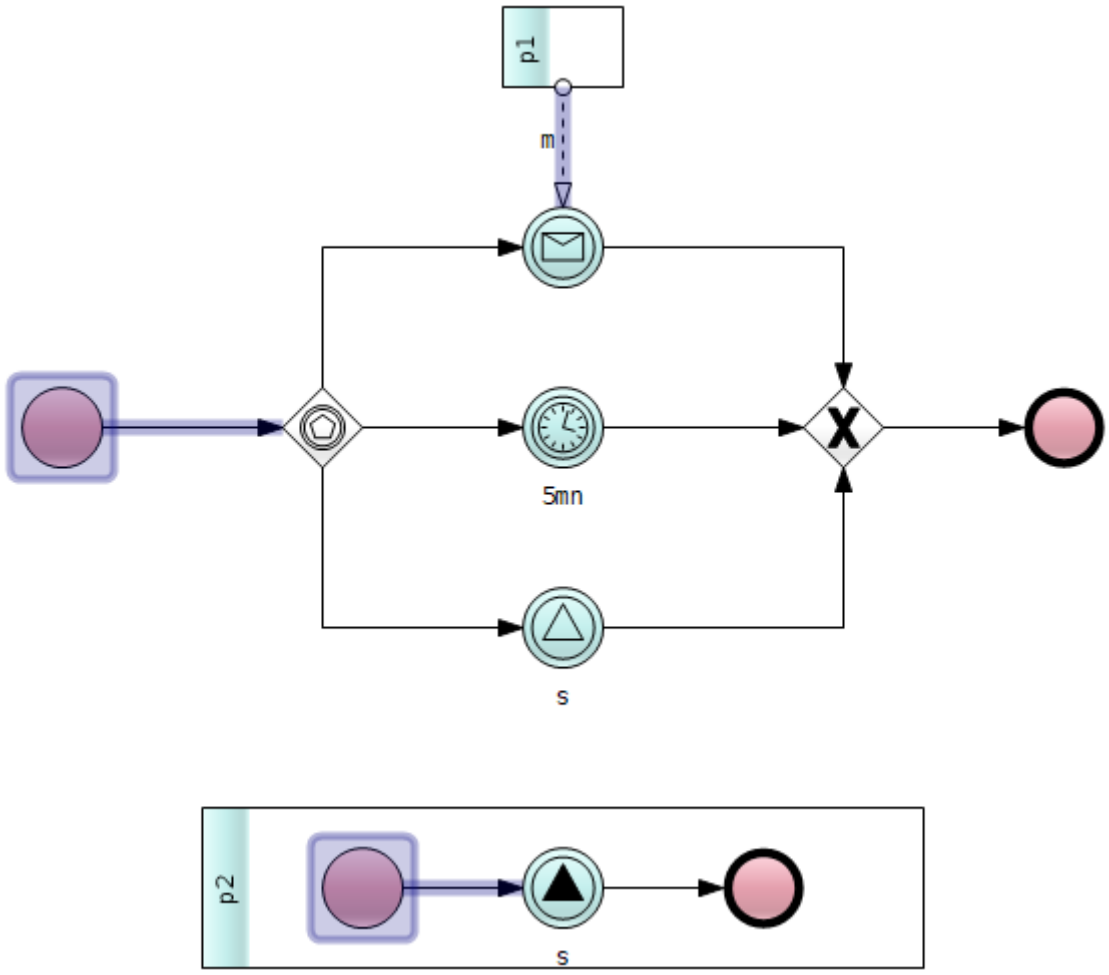



#### 5.3.5.4 Event

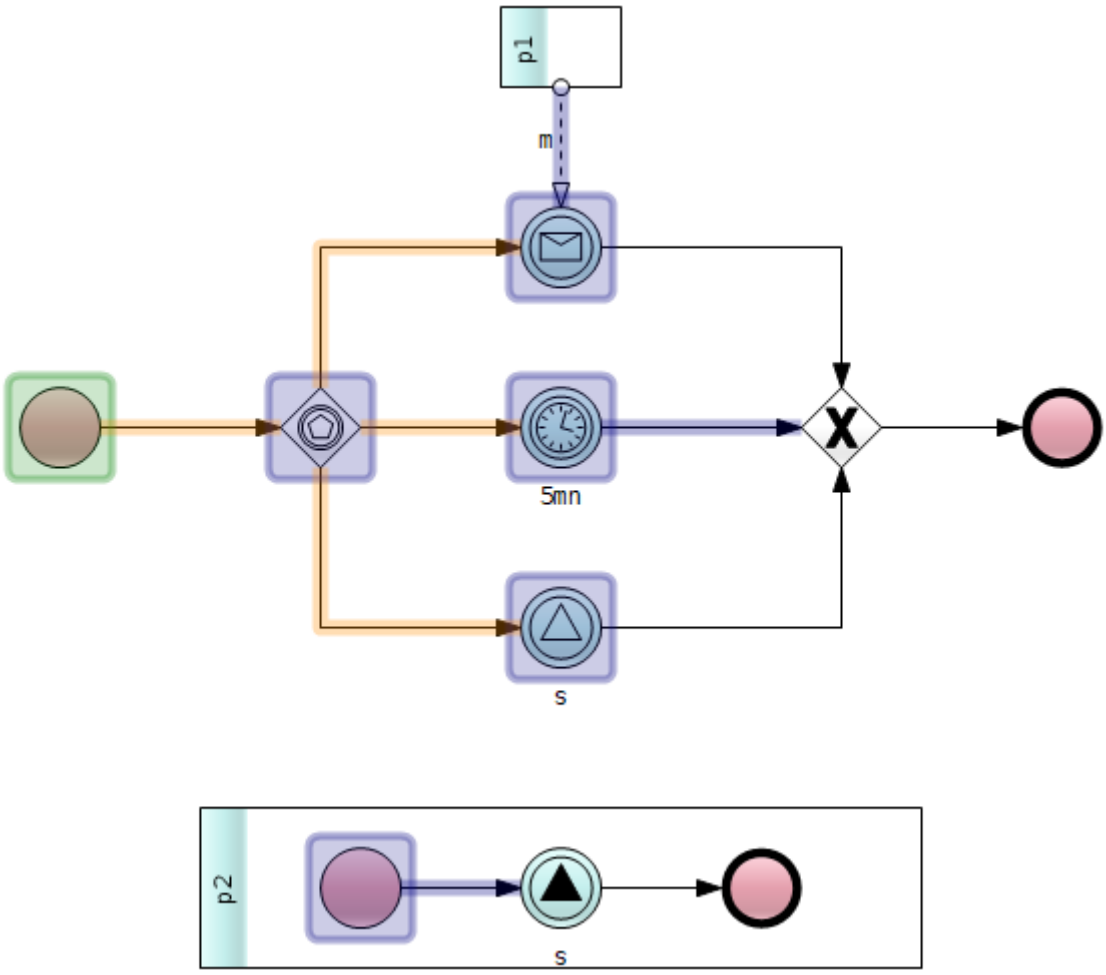
Let's take the following simple example:



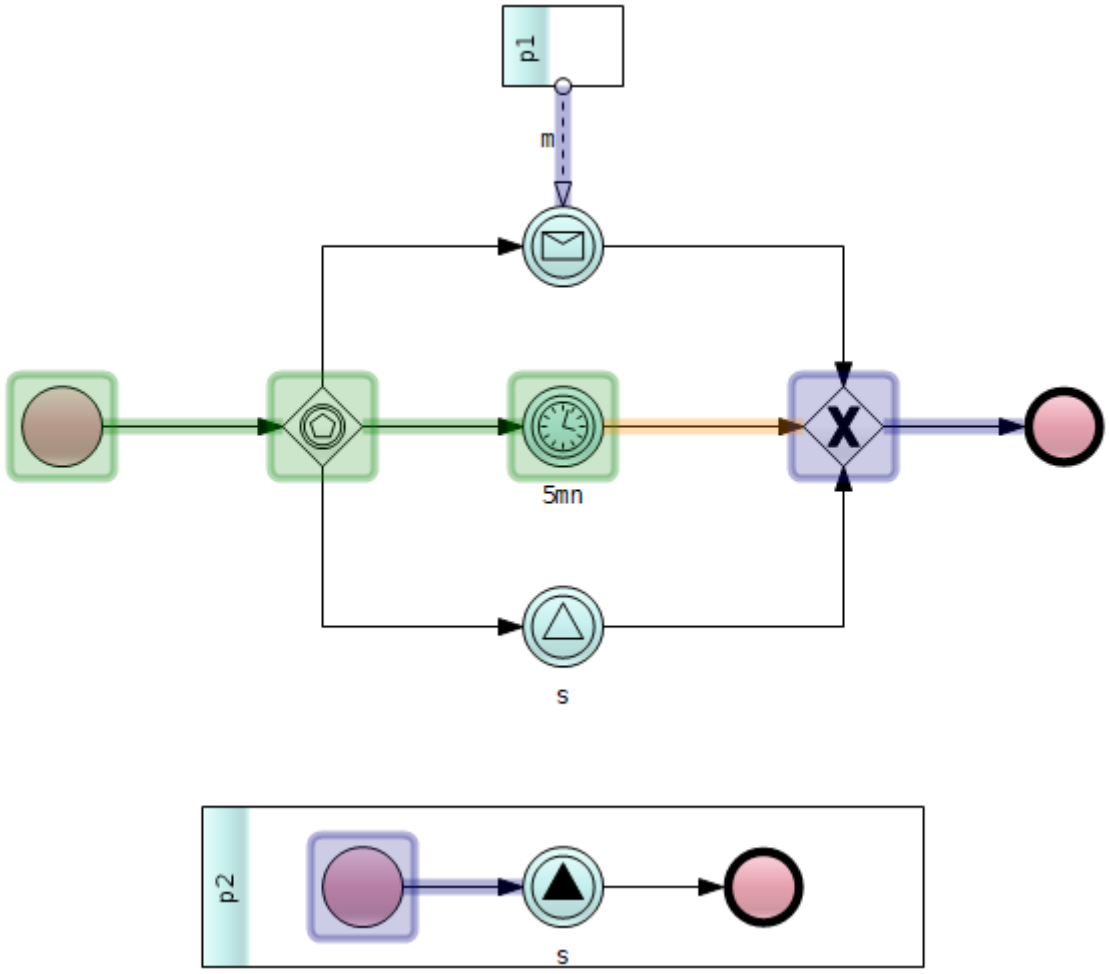

The event gateway can be triggered by the `m` message coming from pool `p1`, or a time out from the `5mn` timer, or the `s` event that could be thrown by the process in pool `p2`. Let's start execution:



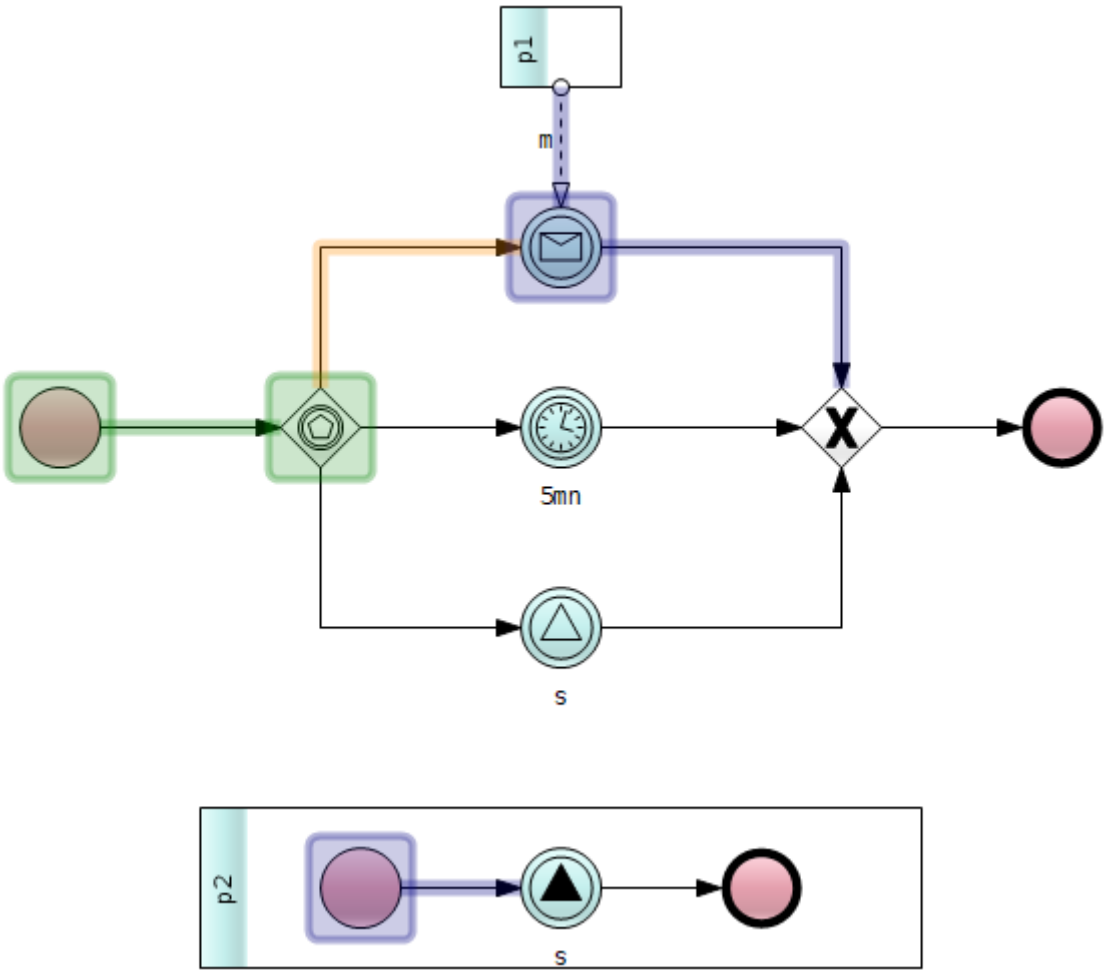

And enable the sequence flow after the start event:



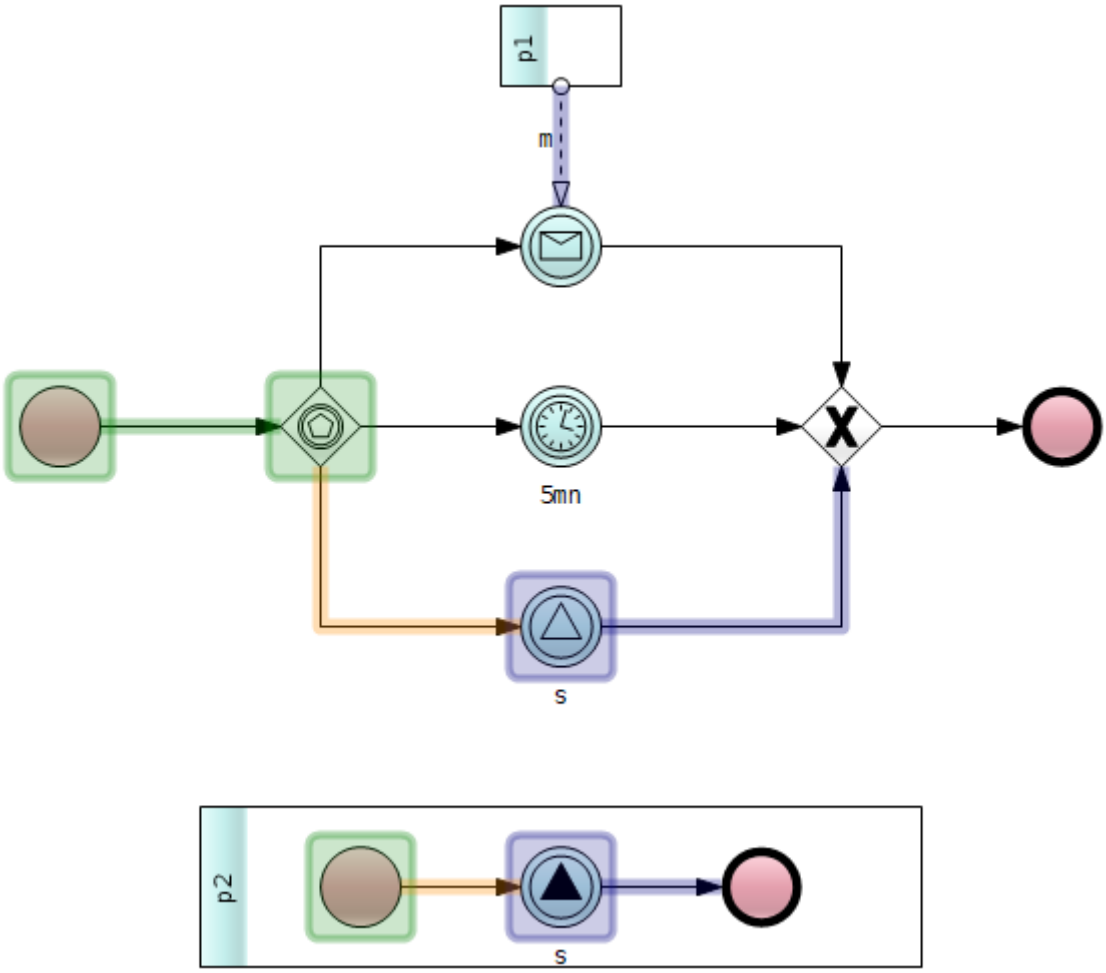

A click on the sequence after the timer will activate the sequence after the exclusive gateway. Note that all other branches are disabled:




A click on the m message flow will activate the upper sequence flow and disable the others:

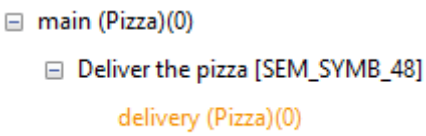



A click on the start sequence after the start event in pool p2 to throw the event that will be caught by the lower branch of the gateway. Again the other branches will be disabled:



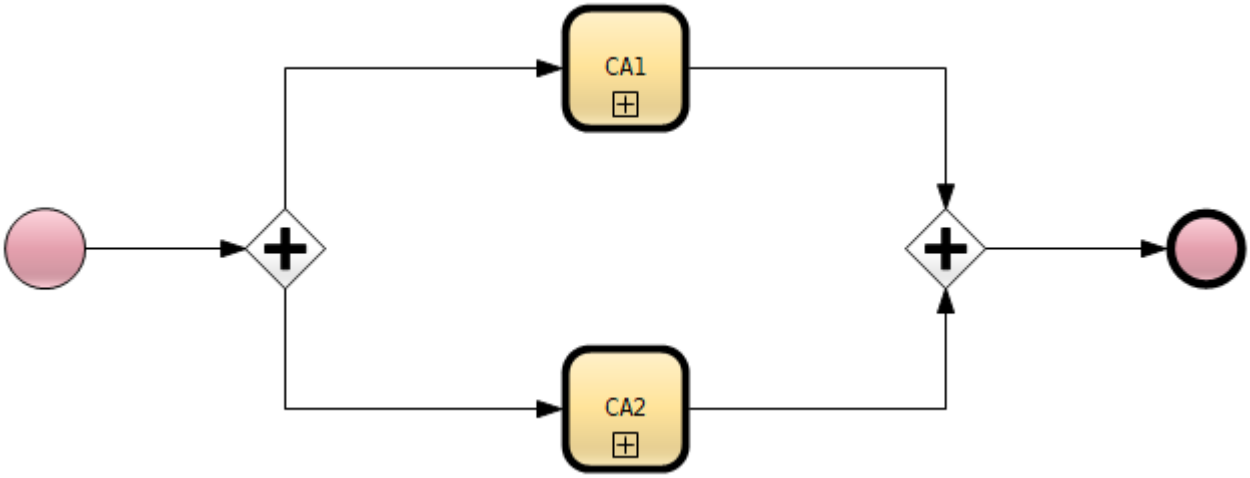

## 5.4 Execution tree

During execution, all processes can be followed via an execution tree displayed by clicking the icon at the top at the browser on the right side of the editor window. It will show a tree like this one:

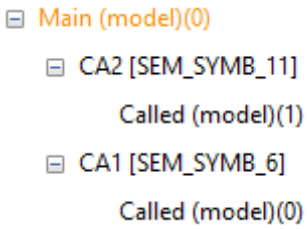



The top-level represents the main process in the model. Under it will appear all call-activity symbols for currently running call-activities, with their only child representing the process called by the call-activity. Here, the process "main" has called the process "delivery" via the symbol "Deliver the pizza". The name appearing for the process is the name of the diagram it appears in followed by the name of its parent model between parentheses.

An instance number is added to each running process to take into account the case where a process is running twice during an execution. This can happen for example in a model like the following one:

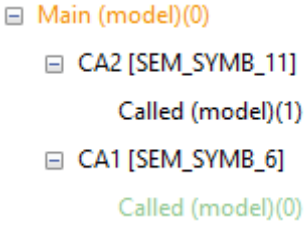



In this process, the two symbols CA1 & CA2 actually call the same process, so there may be two instances of the process running in parallel, of course in different states. The execution tree will then appear like follows:




The process in the diagram "Called" appears twice with two different instance numbers: the number 0 is the one called via the call-activity symbol "CA1", and the number 1 is the one called via the call-activity symbol "CA2".

Once a call-activity has terminated its execution, its node in the execution tree will appear green:

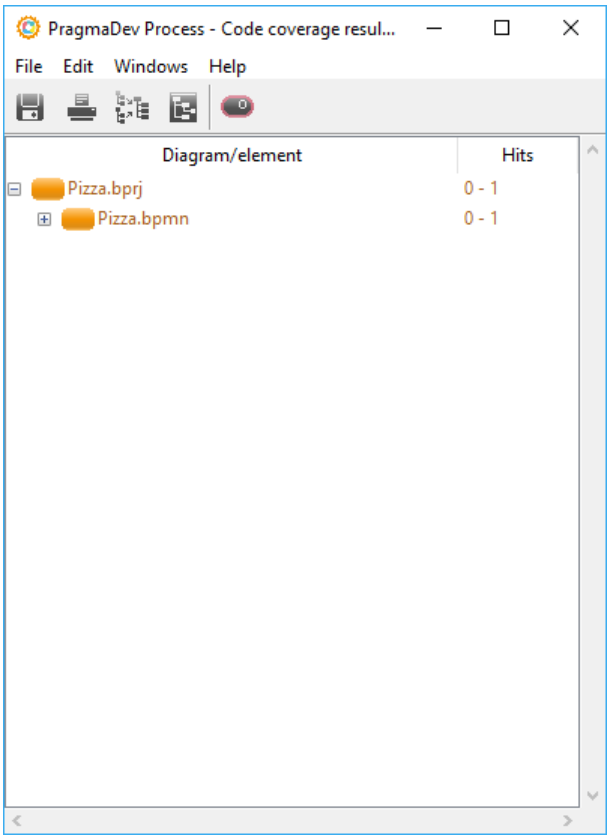



The process in orange in the tree is the one currently shown in the editor window. Clicking on a process name will open the corresponding diagram and show the execution status for the clicked instance number. Clicking on a symbol name will open its parent diagram.

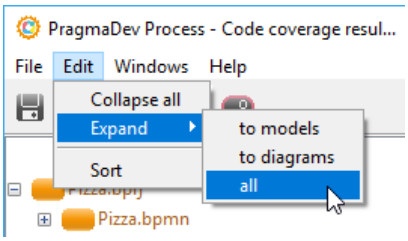
# 5.5 Coverage

## 5.5.1 General information

It is possible to generate coverage information at any moment during an interactive execution in the BPMN editor. This can be done via the menu *Execution / Generate coverage...* or the button . Coverage information will be shown in a window:

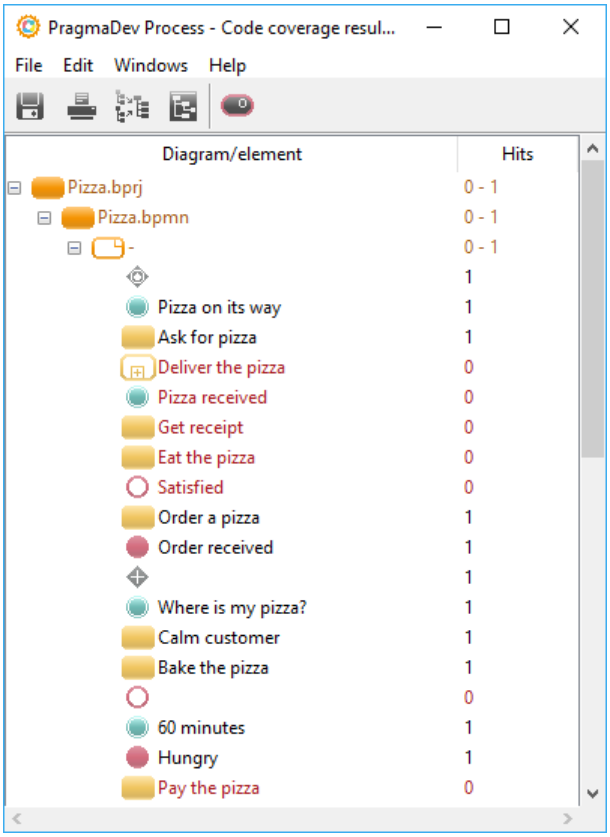



The coverage tree can be expanded via the menu *Edit / Expand*, and expansion can be done at *model*, *diagram*, or element (*all*) level via the corresponding entry in the expand sub-menu:



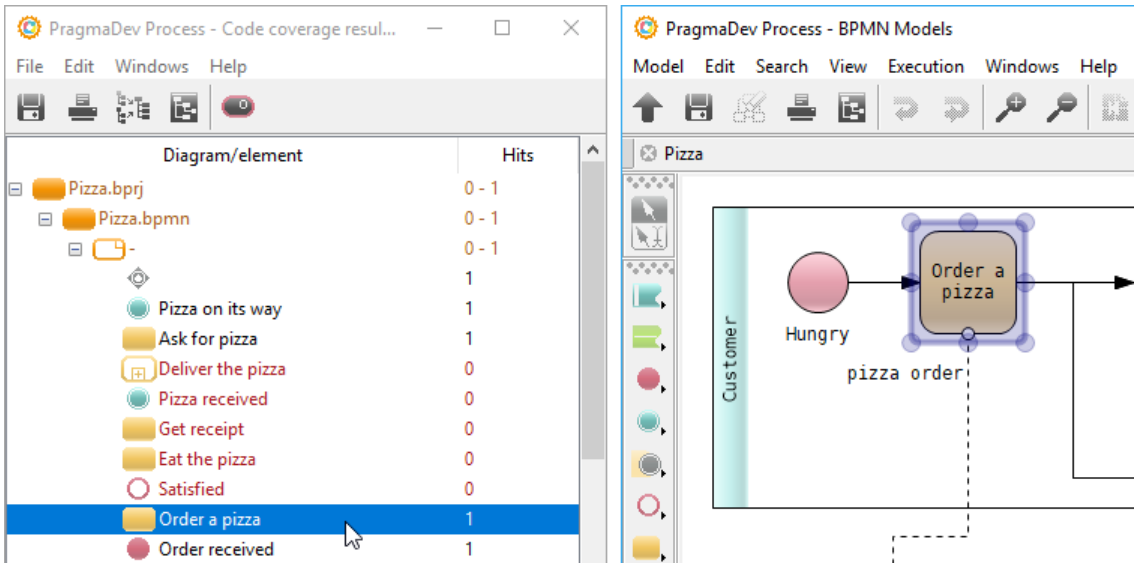

The tree can be collapsed and/or sorted also via the *Edit* menu.

When everything is expanded (i.e., *all* is selected in the expand sub-menu), the complete list of elements will be shown:




*Hits* is the number of times a given BPMN element was executed. For parent nodes in the coverage tree (e.g., model and diagram) there are two values for *Hits*. These are the minimum and maximum values of *Hits* of all BPMN elements present in the model or diagram.

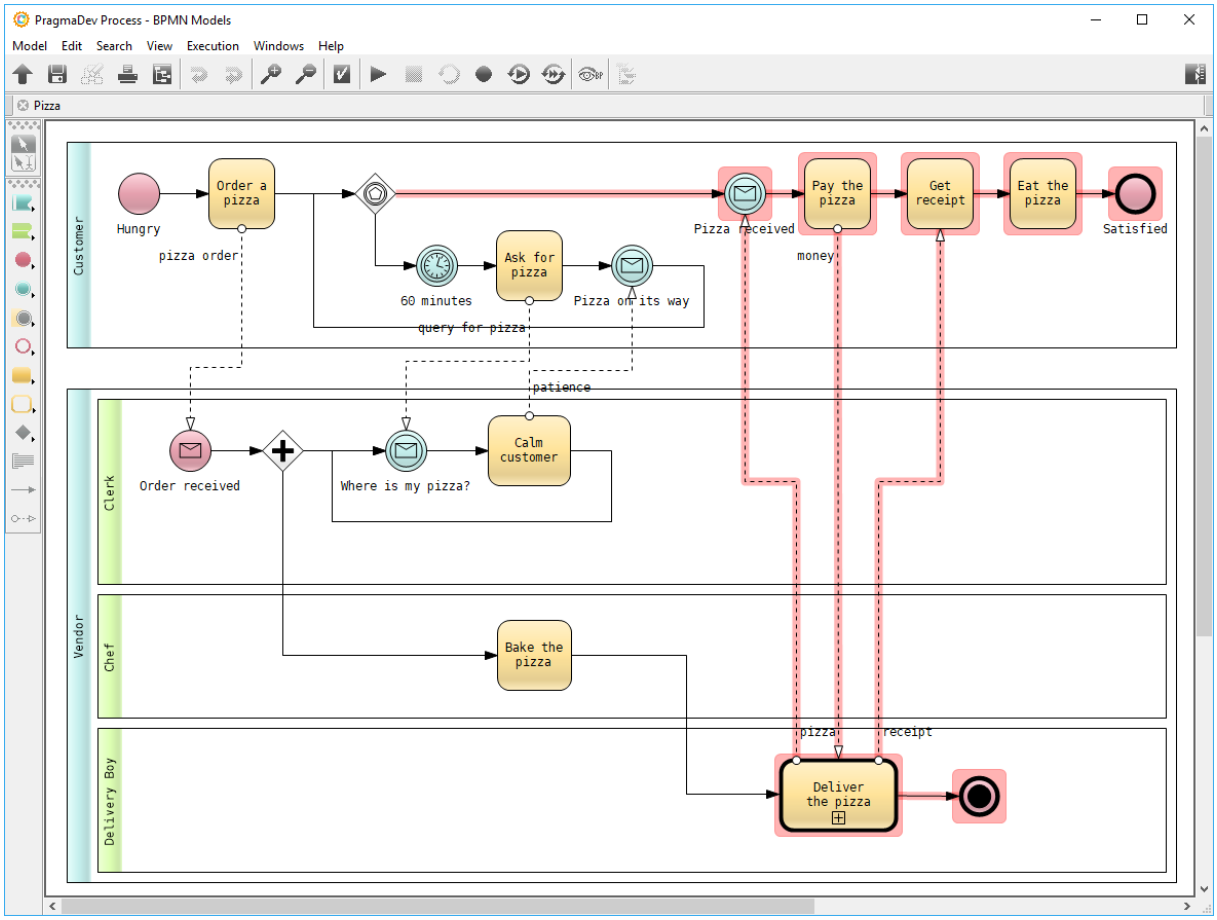
Double-clicking on a BPMN element will open its diagram in the editor and select the element:




A coverage can be merged with others via the menu *File / Merge...* or the button :

### 5.5.2 Highlight non-covered symbols

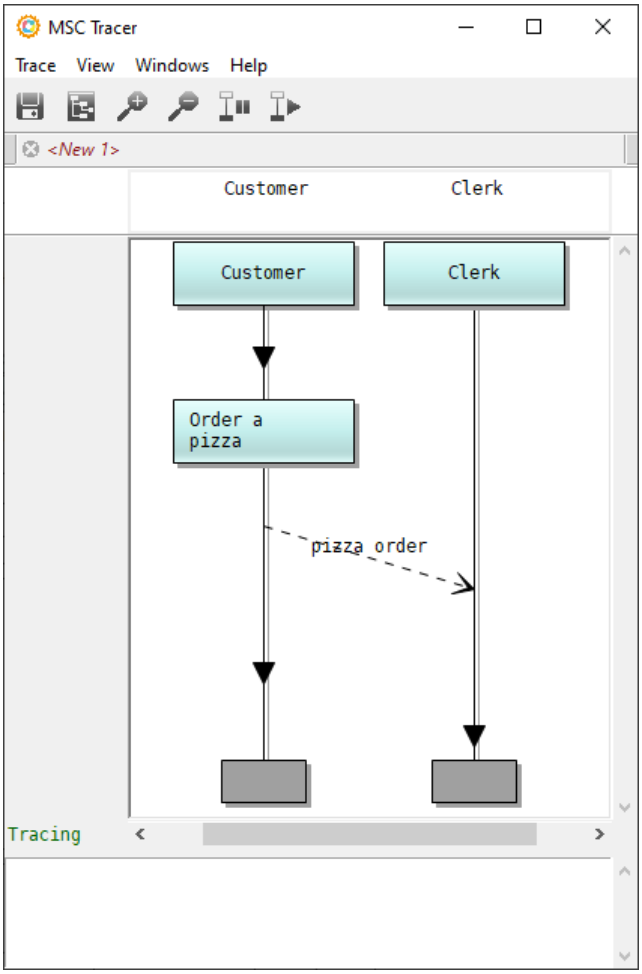
It is possible to mark (in red) in the BPMN editor all non-covered symbols from the coverage window via the toggle button :



## 5.6 Execution traces

### 5.6.1 Recording

An execution scenario can be recorded with the *Record* button . Clicking the button will open the *MSC Tracer* and record the execution events, e.g.:




The description of the events that can be traced and the BPMN element (and execution action) they represent is given in MSC & PSC reference guide.

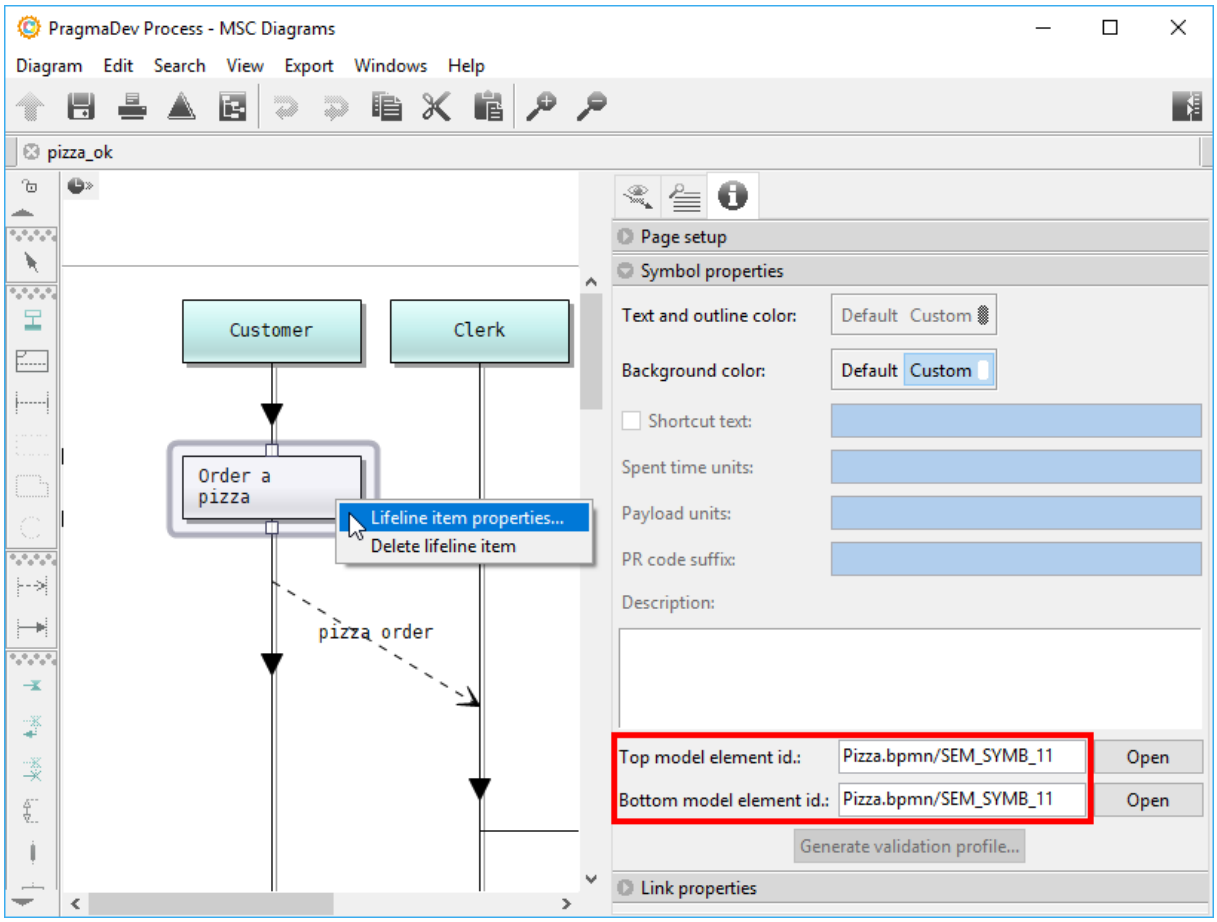
### 5.6.2 Replay

A recorded execution trace can be saved for replay. Replay means executing a trace against the model, while checking for differences between the two at every step. There two checks done at every step:

- The referenced model ID of the current element in the MSC trace is checked against the ID of the current element in the BPMN model.

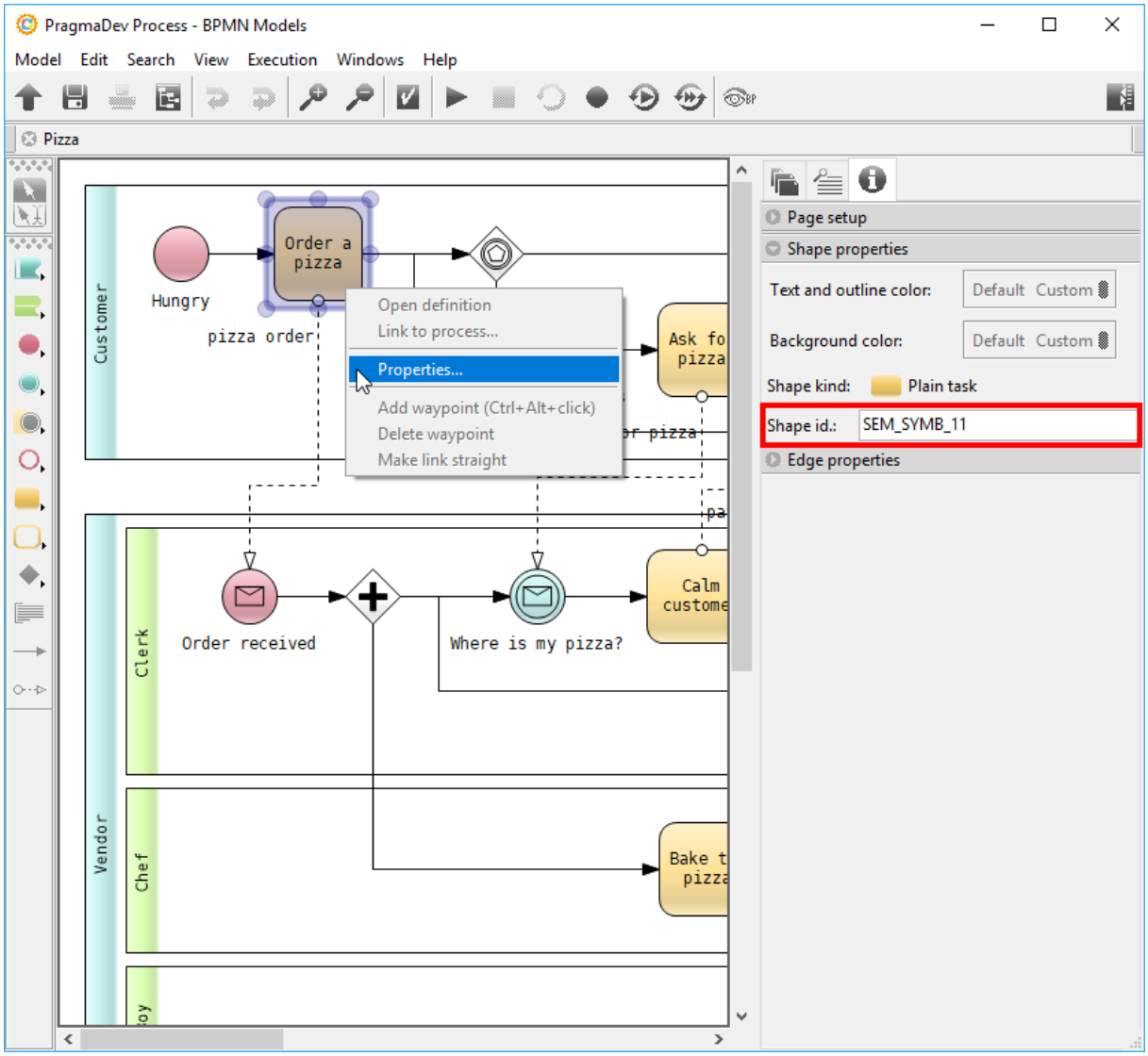
- The text of both current elements (MSC and BPMN) are checked to see whether they are the same.

The referenced model ID of an element in a trace can be accessed in the MSC editor by right-clicking the element and *Lifeline item properties* or *Link properties:*




Note that the referenced ID is preceded by `<name-of-bpmn-file>/`.

In the BPMN editor the element ID can be accessed in a similar way:




### 5.6.2.1 Single-trace execution

The *Replay* button in the BPMN editor allows replaying of a single trace. After choosing the trace to replay, the MSC editor will be opened in execution mode. The execution is controlled from the MSC editor with the following tools:

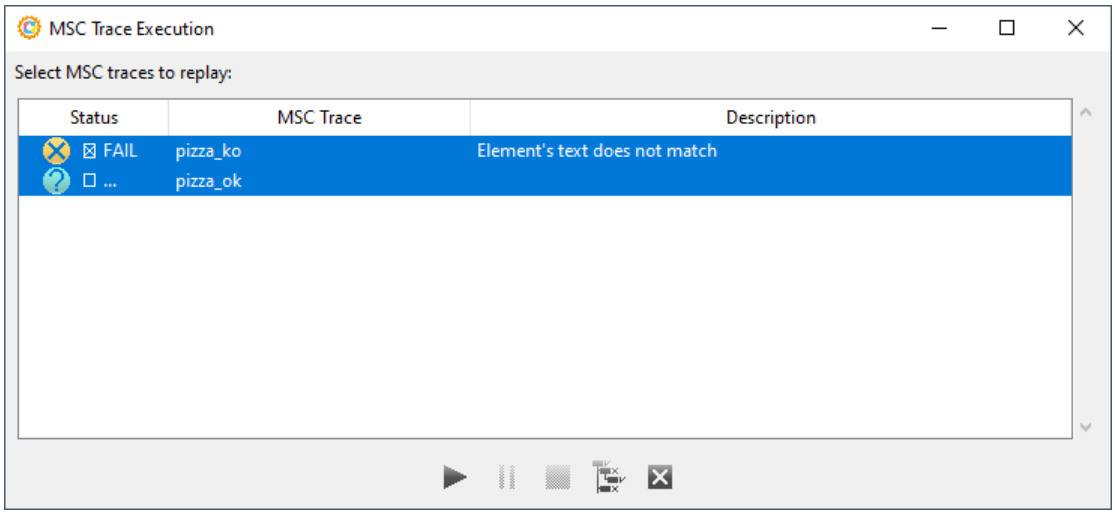
- A trace can be executed till its end *Run* button .
- A single event of the trace can be executed with the *Step* button .
- When running, execution can be paused with the *Pause* button .
- Execution mode can be exited with the *Stop* button .

- Execution is reset with the *Reset* button .

When executing a trace either step-by-step or until the end, the state of execution will be updated also in the BPMN editor. The current event in the trace will be checked and compared against the BPMN model at every execution step. The execution will stop either if a difference is found between the trace and the model or if the end of the trace was reached.

#### 5.6.2.2 Multi-trace execution

The *Replay all* button allows replaying multiple traces. Clicking the button will show the following window:




To select several traces, hold down *Ctrl* or *Shift* and click on them.

The *Status* column contains the results of execution. Possible results are:

- = NONE (...), i.e., trace has not been executed yet.
- = PASS, i.e., no differences were found between the trace and the BMPN model.
- = FAIL, i.e., differences were found between the trace and the BMPN model.
- = ERROR, i.e., could not execute trace due to errors.

If the result of execution for a given trace is FAIL, then double-clicking the trace will open it and the BPMN model in the editors, and select the symbols that caused the fail.

Coverage information from the repaly of multiple traces can be obtained via the *Generate coverage for marked traces* button . Coverage will be generated only for the most recent executed traces, which are marked with ☒.

# 6 MSC and PSC Editor

## 6.1 Overview

PragmaDev Process allows to generate execution traces when executing the BPMN model. The traces are based on a variant of the standard ITU-T MSC representation.

Online traces are created directly by the model being executed which sends trace information.

Other features of PragmaDev Process may be used in conjunction with the tracer itself to get a full-featured tracing utility with conformance checking against specification or property verification:

- Trace diagrams can be directly created for documentation purposes;
- Trace diagrams can be compared in a visual way;
- Specification MSC diagrams can be created, then can be compared to execution traces for conformance checking;
- Property diagrams can be created using the Property Sequence Chart (PSC) format, which can be:
  - matched against execution traces;
  - automatically verified with OBP explorer;
- All diagrams can be easily documented, for example by exporting them fully or partially to common image formats, allowing to insert them in a document.

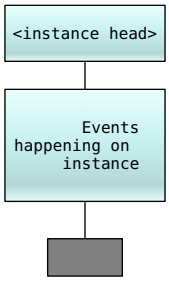
The general graphical form of the trace diagrams supported by PragmaDev Process is described in section MSC & PSC reference guide below.

The tracing feature is described in the Executor chapter. The MSC and PSC editor is described in MSC editor. The available checks that can be performed - trace against trace, specification against trace or property matches - are described in Conformance checking: diagram diff & property match.

## 6.2 MSC & PSC reference guide

### 6.2.1 General diagram format

A MSC or PSC diagram represents the interaction going on between entities called instances over time. Instances will typically be a participant or a lane. Instances are represented by symbols called lifelines, that look like follows:




The lifeline always starts with a head that specifies the instance name.

All events happening on the instance are then displayed on a vertical line under the lifeline head. These events are described below in “Lifeline components” on page 358.

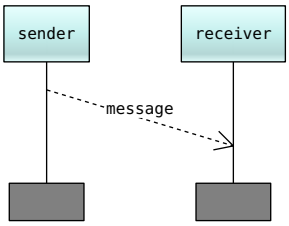
The lifeline terminates by a lifeline tail, that can take several forms depending on the status of the instance at the end of the diagram.

Lifelines are distributed along the horizontal axis, and the vertical axis represents the time, flowing from top to bottom. Events happening between lifelines are mostly represented by links, described in “Links” on page 355. Other symbols allow to further describe the diagram or add semantics to specification MSCs or PSCs; they are described in “Main symbols” on page 366.

## 6.2.2 Links

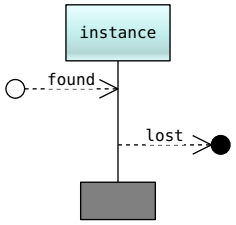
### 6.2.2.1 Message links

An asynchronous message sent by an instance and received by another is represented by a dashed line with an outlined arrow at its end:




Note that a message is a plain line in the genuine MSC format.

A message can also be received from an unknown source, or sent to an unknown target. In this case, they are called a found message and a lost message respectively:



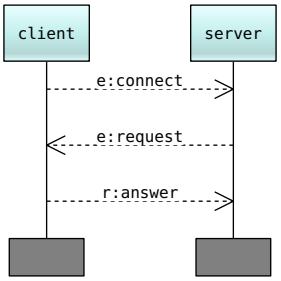

The syntax for the message link text is free.

#### 6.2.2.2 PSC-specific normal, required and failed message syntax

In PSC diagrams, texts for message links are supposed to be prefixed with one of the following:

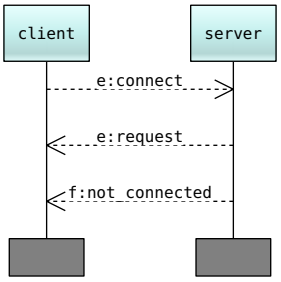
- 'e:' indicates the message is a regular one. This means that the message is part of the precondition for the property: all messages prefixed with 'e:' must appear to trigger a property match. If any of these messages do not appear in the checked diagram, the preconditions for the property are not satisfied, and no matching is attempted. Regular messages should appear first in the PSC diagram.
- 'r:' indicates a required message. This means that if all the regular messages preceding this message are present in the checked diagram, this message must be present for the property to match. If it does not, the property is violated.
- 'f:' indicates a fail message. This means that if all the regular messages preceding this message are present in the checked diagram, this message must not appear for the property to match. If it does appear, the property is violated.

Here is an example of a required message in a property:




This means that if the client has sent a connect message to the server, then sends a request message, the server must send back an answer message, or the property is violated.

Here is an example with a fail message:

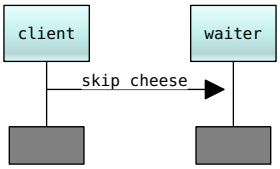



This means that if the client has sent a connect message to the server, then sends a request message, the server must not send back a not_connected message, or the property is violated.

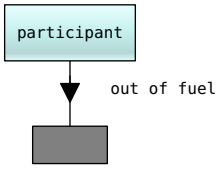
Note that in PragmaDev Process, PSC wanted or unwanted constraints will also appear in the message link text. For more details, see 6.2.3.2.

#### 6.2.2.3 Sequence flow

A sequence uses a plain arrow that can go from one participant to another:




Or that can stay on the participant:




### 6.2.3 Main symbols

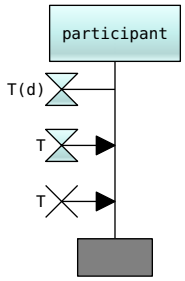
#### 6.2.3.1 Lifeline

A lifeline represents an interacting entity in a MSC or PSC diagram. PragmaDev Tracer allows the instance name appearing in the lifeline head to have the following format:
<instance name>[:<class name>][(<instance identifier>)]
Lifelines can appear in all kinds of diagrams: MSC trace or specification diagrams, as well as PSC diagrams.

#### 6.2.3.2 Lifeline components

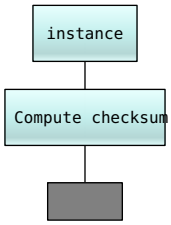
Lifeline components are events impacting a single lifeline. They appear as symbols attached to the lifeline.

**Timer events** An instance can start timers, that will time-out in a given amount of time. A timer can also be canceled by the instance that created it. The symbols for timers are the following ones:



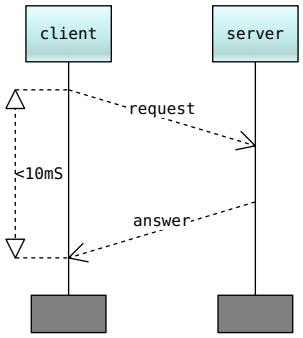
The first symbol means the timer named T starts for a duration of d. The second symbol represents a time out for the timer named T. The last symbol represents the cancellation of the timer named T.

**Action symbol** Action symbols describe actions performed by the lifeline. In the current version of PragmaDev Process, they represent activities or tasks. For example:




The sequence of tasks or activities can also be tested in PSC diagrams by using the same prefixes as for messages (cf. "PSC-specific normal, required and failed message syntax" on page 81). See the example "ApproveReject" for an example of a PSC property testing the sequence of tasks.

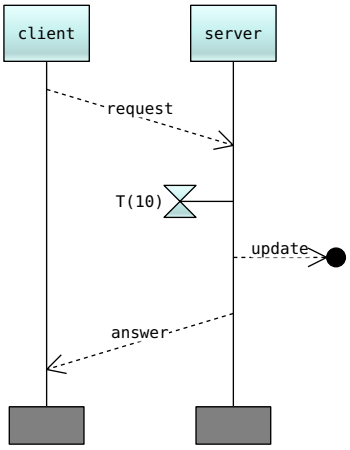
**Relative time constraints** A relative time constraint appearing in a specification MSC diagram or a PSC diagram indicates that the sequence of events it encloses must happen within a given time. For example:




This specifies there must be less than 10 ms between the time when Client sends the request message and the time when it receives the answer message.

During conformance checking, relative time constraints are compared to absolute times in the compared diagram (see 6.3.7 and 6.2.3.5). Please note that units are not yet supported: relative time constraints can only contain a valid comparison operator (<, >, <=, >=, ...) followed by a real number.

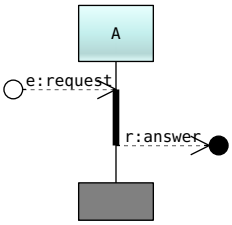
**Co-regions** A co-region on a lifeline specifies that all events happening on this lifeline can happen in any order, and not only the order specified graphically. For example:




The co-region, indicated by the dotted line on the server lifeline, indicates that the timer and the outputs of messages update and answer can happen in any order.

Note that co-regions are not supported in the conformance checking feature of PragmaDev Process (6.3.7). The same semantics can usually be specified by using inline expressions; see 6.2.3.4 for details.

**PSC strict operator** Events specified on lifeline in PSC diagrams are supposed to be loosely ordered by default. This means that if anything happens between two of these events, the property is matched anyway. It is however possible to specify a strict ordering for a set of events, meaning that these events must happen in this order without anything in between. This is done with the strict operator, that looks like follows:




This means that a request message received by A must be immediately followed by the output of an answer message, without anything in between (see 6.2.2.2 for the PSC-specific link text syntax).

**PSC constraints: wanted and unwanted messages & chains** PSC diagrams allow to specify on a message a set of messages that must or must not appear before or after it for the property to match. Unlike other messages, the messages in these constraint appear in what PSC calls the intra-message format, i.e as a text formatted like: <sender instance name>.<message name>.<receiver instance name>.

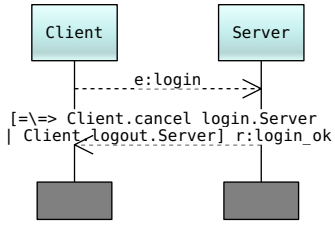
In the PSC specification, the constraint itself is represented by a symbol appearing under one end of the message link:

- If it appears under the link start (message output on sender), it is a past constraint, meaning it must be satisfied before the message is sent for the property to match;
- If it appears under the link end (message input on receiver), it is a future constraint, meaning it must be satisfied after the message has been received for the property to match.

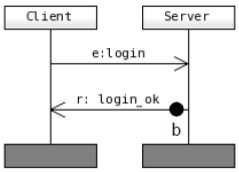
In PragmaDev Process, the constraint is actually specified directly in the text of the link. So a past constraint will appear in square brackets before the text for the message itself: [constraint] message_name(parameters...) and a future constraint will appear after the text for the message: message_name(parameters...) [constraint]

Constraints can have several forms:

- An unwanted message constraint specifies a set of messages that should not appear. If any of the specified messages appear, the constraint is not satisfied and the property does not match. In PragmaDev Process, this kind of constraint is represented as follows:

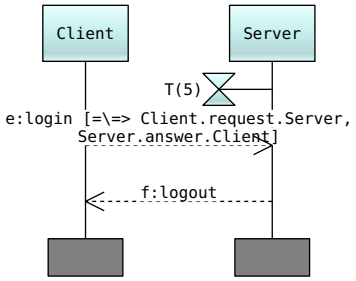



The brackets isolate the constraint from the message itself, the "=\=>" is the standard prefix for an unwanted constraint in PragmaDev Process, and the messages that should not appear before the login_ok message are separated by a "|", meaning that if Client sends a login message to Server, Server must answer by sending back a login_ok message, unless either the message cancel_login has been sent from Client to Server before, or the message logout has been sent by Client to Server before.
Note that is standard PSC, the representation would be something like:

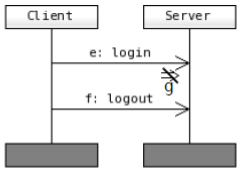



```
b = {Client.cancel_login.Server, Client.logout.Server}
```

- An unwanted chain constraint specifies a sequence of events that should not appear. If all messages in the constraint appear in the order specified in the constraint, then the property does not match. This kind of constraint is represented in PragmaDev Process as follows:

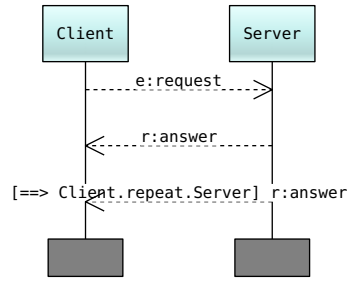



The brackets and prefix are the same as in the unwanted message constraint above, but the separator between the messages in the constraint is now a ",", denoting a sequence. The constraint also appears after the message text, so this is a future constraint. So this means that if Client sends a login message to Server, it is a property failure if it sends a logout message after it, unless it has sent the request message and Server has sent back the answer message in-between.
Note that in standard PSC, the representation would be something like:

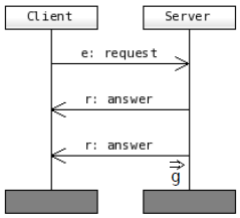



```
g = (Client.request.Server, Server.answer.Client)
```

- A wanted chain constraint specifies a sequence of events that must appear. If any of the messages in the constraint does not appear, or the messages appear in a different order than the one specified in the constraint, then the property does not match. This kind of constraint is represented in PragmaDev Process as follows:




The constraint appears before the message name, so it's a past constraint. The prefix "==>" is the standard one for all wanted constraints in PragmaDev Process. So this specifies that if Client sends a request message to Server, Server must send back an answer message, and then another one if Client sends the repeat message after the first answer. Note that the standard PSC representation would be something like:




```
g = (Client.repeat.Server)
```

Note: PragmaDev Process actually supports more general types of constraints called wanted and unwanted alternative chain constraint. These merge the message and chain constraints described above. The general syntax for these constraints is:

[<constraint type prefix> I1.m1.J1,I2.m2.J2,... | In.mn.Jn,... | Im.mm.Jm,... ]

where <constraint type prefix> can be either ==> for a wanted constraint, or =\=> for an unwanted constraint.

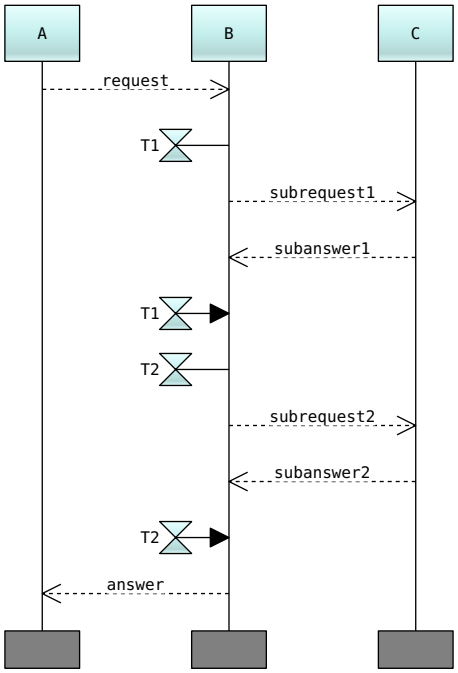
- If the constraint is unwanted, this specifies that neither the sequence I1.m1.J1, I2.m2.J2, ..., nor the sequence In.mn.Jn, ..., nor the sequence Im.mm.Jm, ... should appear for the property to match.
- If the constraint is wanted, this specifies that either the sequence I1.m1.J1, I2.m2.J2, ..., or the sequence In.mn.Jn, ..., or the sequence Im.mm.Jm, ... must appear for the property to match.

This allows to represent all the PSC constraint kinds:

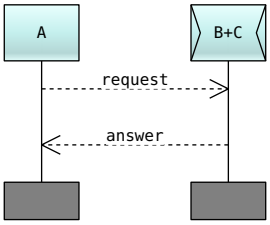
- An unwanted message constraint {I1.m1.J1, I2.m2.J2} will be represented as: [=\=> I1.m1.J1 | I2.m2.J2]
- An unwanted chain constraint (I1.m1.J1, I2.m2.J2) will be represented as: [=\=> I1.m1.J1, I2.m2.J2]
- A wanted chain constraint (I1.m1.J1, I2.m2.J2) will be represented as: [==> I1.m1.J1, I2.m2.J2]

#### 6.2.3.3 Collapsed lifelines

Collapsed lifelines are a PragmaDev extension and result from a 'collapse' operation. This allows to represent a set of lifelines as a single lifeline, events happening between the lifelines in the set being hidden. For example, after collapsing the instance B and C in the following diagram:



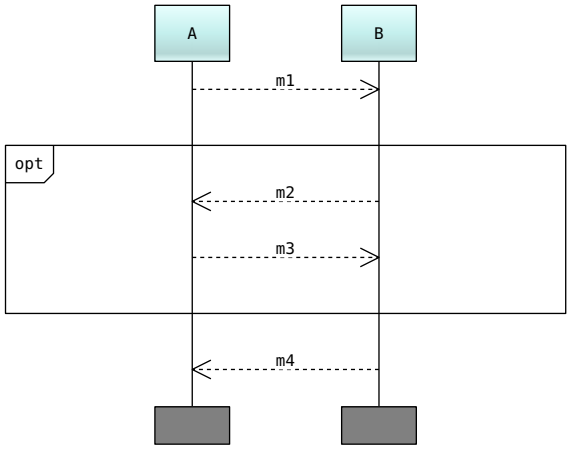

the diagram appears as follows:




#### 6.2.3.4 Inline expressions

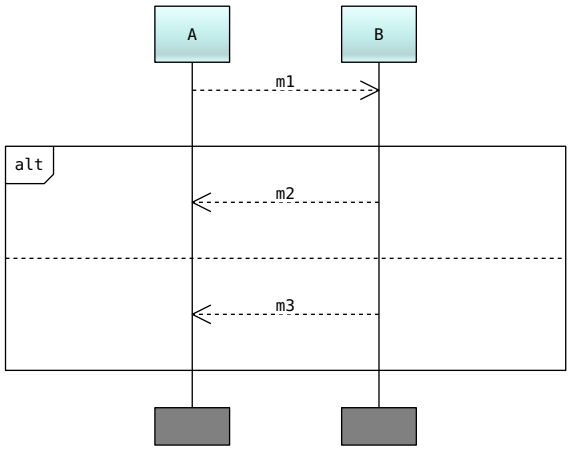
An inline expression in a specification or PSC diagram is a way to specify specific semantics for a group of events. The semantics depend on the kind of inline expression:

- An ‘opt’ inline expression specifies an optional set of events. For example:



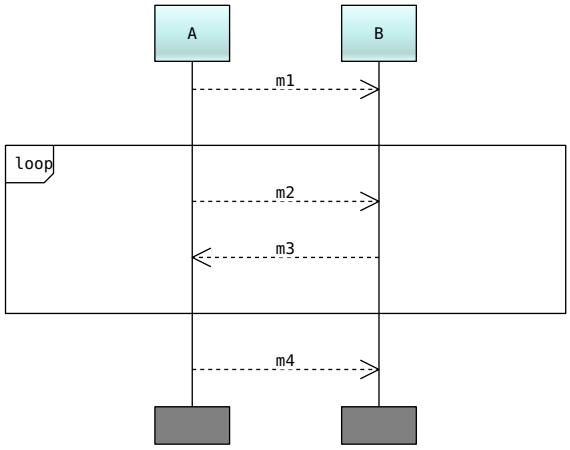

specifies that the message m1 is sent from A to B, then B may send m2 to A, which answers m3, then B sends m4 to A. So the sequences m1-m2-m3-m4, and m1-m4 are both valid.

- An 'alt' inline expression specifies a set of alternative behaviors. For example:



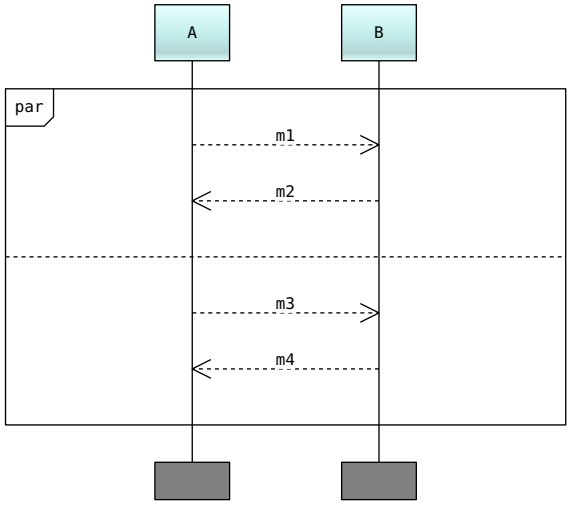

specifies that when A sends m1 to B, B may answer by sending back m2, or m3. So the sequences m1-m2 and m1-m3 are both valid. An 'alt' inline expression must have at least two compartments in it, and can have as many as needed.

- A 'loop' inline expression specifies a set of events that might be repeated several times. For example:
  specifies that after A has sent the message m1 to B, it may send any number of messages m2, to which B will answer by the message m3, until A sends the message m4 to B. So the sequences m1-m4, m1-m2-m3-m4, m1-m2-m3-m2-m3-m4, and so on, are all valid.



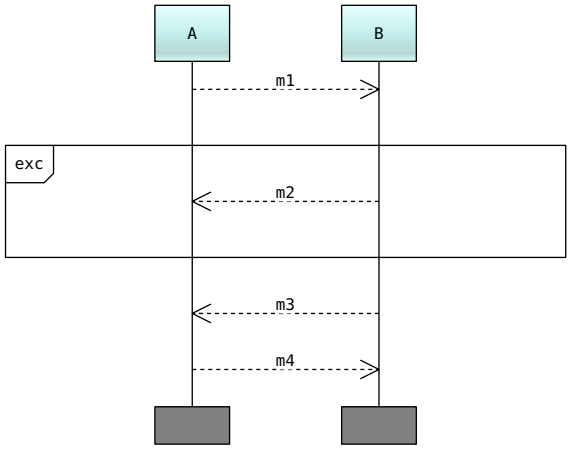

Note that the MSC standard allows to indicate minimum and maximum number of repeats in loop inline expressions. This feature is not yet available in PragmaDev Process.

- A 'par' inline expression specifies a set of event sequences that must all happen, but in any order. For example:



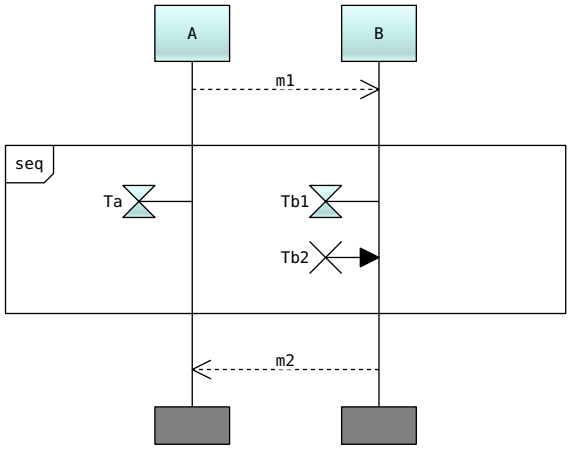

specifies that the two sequences A sending m1 to B and B answering m2, and A sending m3 to B and B answering m4 must both happen, but that the order is not significant between the sequences. So the global sequences m1-m2-m3-m4 and m3-m4-m1-m2 are both valid.
A 'par' inline expression must have at least 2 compartments, and can have as many as needed.

- An 'exc' inline expression represents an exception. This means the sequence of events in the inline expression is an error case and terminates the scenario. For example:




specifies that when A sends m1 to B and B answers m2, there is an error and the scenario should stop. So the sequence m1-m2 is valid, but is an error case, and the sequence m1-m3-m4 is valid and is a normal execution. Note that the MSC standard represents an 'exc' inline expression with a dotted bottom line. PragmaDev Process uses a solid line in the current version.

- A 'seq' inline expression represents a weak sequence. This means that within such an inline expression, the events on a specific lifeline must happen in the given order, but the general ordering can be anything. For example:

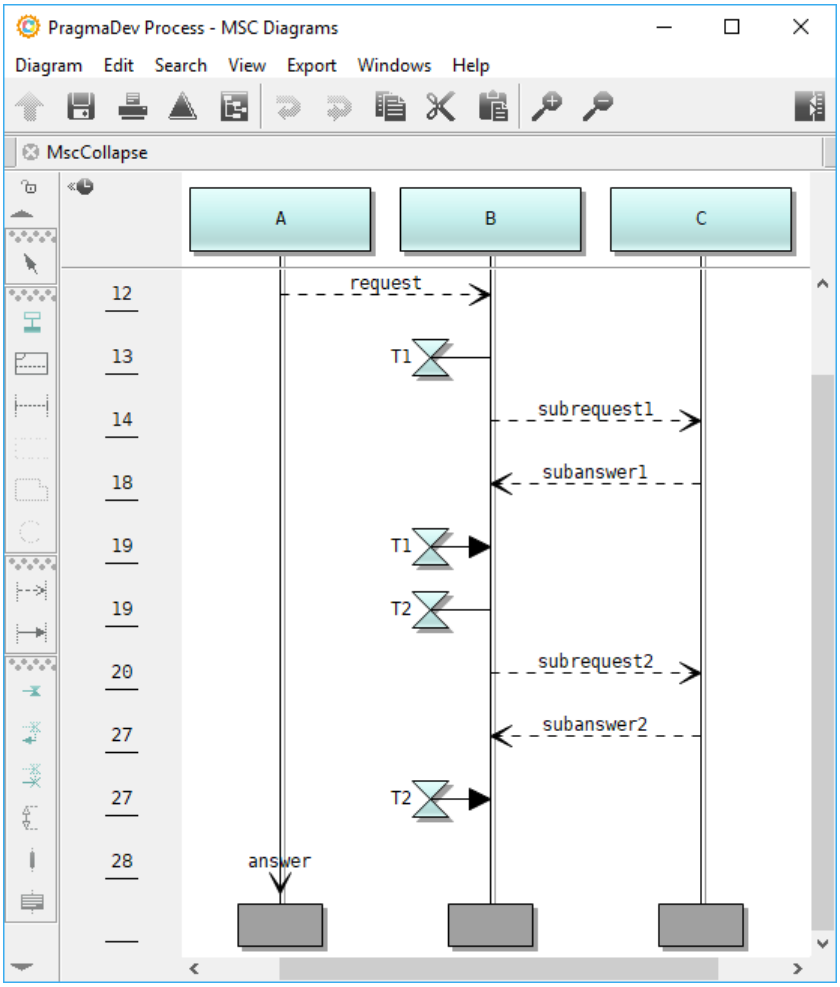



This means that on lifeline B, the starting of Tb1 has to happen before the canceling of Tb2, but that the starting of Ta by A can happen at anytime: before the starting of Tb1, after the canceling of Tb2 or between the two. Note that this kind of inline expression is not supported in conformance checking (6.3.7).

#### 6.2.3.5 Absolute times

In the MSC standard, absolute times can be associated to any event in the diagram by using a symbol consisting only in a dashed underline under the text for the time. PragmaDev Process supports absolute times, but only associated to complete 'event rows': the times are displayed in the left margin of the diagram and are associated to all events with the same y coordinate, instead of any event. To keep the same representation as

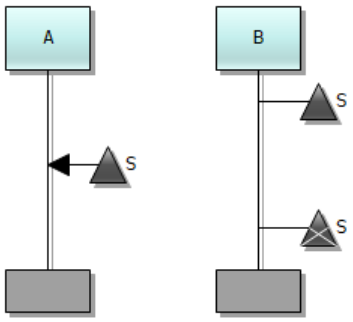
in the MSC standard, each absolute time is displayed with a dashed underline:




These absolute times are the reference when verifying relative time constraints during conformance checking (see 6.2.3.2 and 6.3.7). Please note that units are not yet supported: absolute time constraints must be written as a real number only.

### 6.2.3.6 BPMN signals throws & catches (PragmaDev extension)

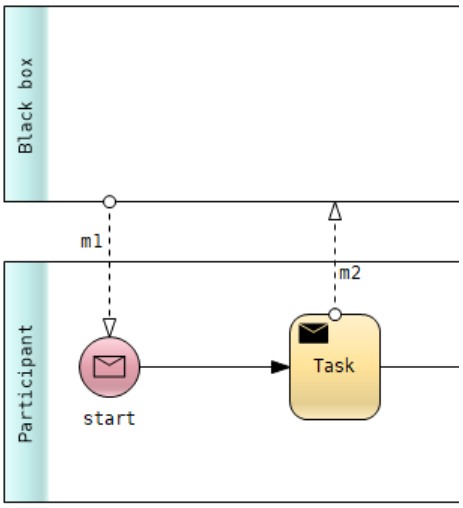
PragmaDev Process allows to attach to a lifeline symbols that represent the beginning of a BPMN signal throw, the end of a BPMN signal throw, and a signal catch. These allow to trace signals from a BPMN model. These symbols look like follows:




This means that the lifeline B starts to throw a signal, that is caught by lifeline A before the throw is ended by lifeline B.

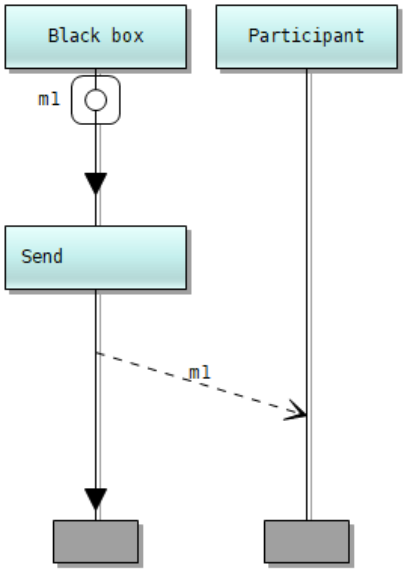
#### 6.2.3.7 Enabled and disabled gates (PragmaDev extension)

PragmaDev Process allows to attach to lifelines symbols representing gates for outgoing or incoming messages on a lifeline, also indicating that the gate has been enabled or disabled. This allows to trace the action performed during execution for the following kind of BPMN model:

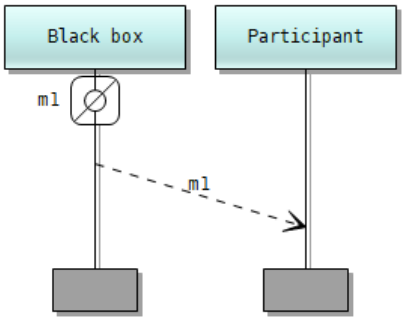



As explained in Explicit resolution on page 54, it is possible to either "open" the definition of the "Black box" partiticpant by enabling one of the gates for m1 or m2, or to decide to ignore it and handle the sending & receiving of the messages "manually" by disabling the gates.

The gates symbols on a lifeline allow to trace this action:




Here, the gate is enabled; the contents of "Black box" is loaded and the task "Send" sending the message m1 is traced.




Here, the gate is disabled; the contents of "Black box" is not loaded and the message m1 has been sent "manually".

#### 6.2.3.8 Comments

A comment symbol just contains a documentation text for the item it is attached to. Comment symbols are not yet supported in PragmaDev Process.

#### 6.2.3.9 Texts

A text symbol contains informal text usually describing global items in the diagram. Text symbols are not supported yet in PragmaDev Process.

## 6.3 MSC editor

This kind of editor is used for Message Sequence Charts. A MSC diagram describes a sequence of events happening in a system, with a set of "lifelines" represented as vertical lines, with symbols representing events attached to them.

There are 3 main kinds of MSC diagrams, which are all recognized by PragmaDev Process:

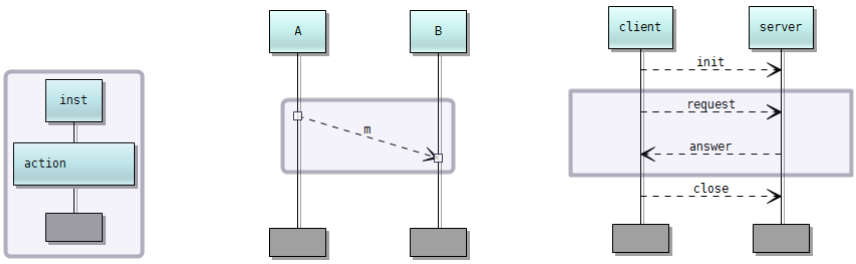
- Basic MSCs represent a sequence of events that have actually happened during a system execution. They will contain a lifeline for each participant or lane, and events will be sequence flows, message exchanges, and timeouts. They are typically obtained by using the MSC tracing functionality in the executor.
- Specification MSCs will contain the same kind of events, but can group them within other symbols with attached semantics. For example, a sequence of events can be isolated in another MSC diagram that will be referenced via a "MSC reference" symbol. Or a sequence of events can be enclosed in an "inline expression", allowing to specify this sequence is optional, or can be repeated several times.
- Property Sequence Charts are another kind of specification MSCs that are used to describe "if/then" conditions: if a given sequence of events appear in a diagram, then another sequence must appear behind it, or must not appear behind it.

The whole format for MSCs - basic & specification - and PSCs is described in 6.2. Note that there is no specific editor for each kind of diagrams: all symbols are available in the editor, and the kind of diagram is recognized automatically from what it contains.

The features of the MSC editor and their availability are described in the following paragraphs.

### 6.3.1 Specific tools

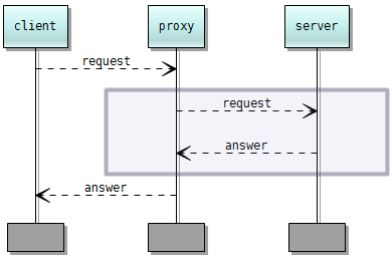
The selection tool in MSC diagrams allows to select symbols and links, just as in other editors. It also allows to select rectangular zones, that can be copied and pasted and exported as image files:



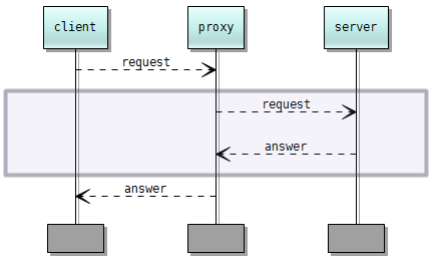

*Selected lifeline* *Selected message link* *Rectangular selection*

Note that copying and pasting rectangular zones will work only for full "horizontal slices" of the diagram: if a rectangular zone is selected, but does not span the full diagram width, it will be automatically extended when copied or cut.

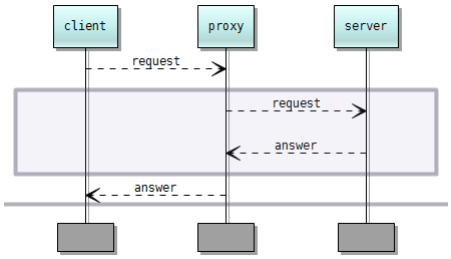
For example, if this zone is selected:



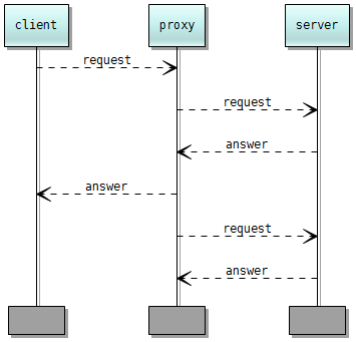

copying it will automatically extend the zone to the full width of the diagram and display a warning:




When pasting a rectangular zone, a horizontal insertion line will be displayed:




Clicking in the diagram while the insertion line is displayed will paste the copied zone at this position:

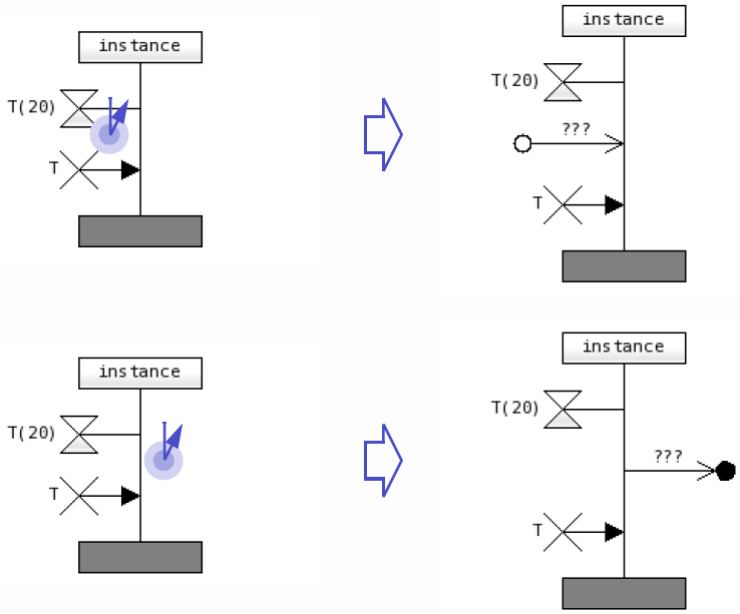



Note that the copy will fail if any object has an end within the slice but the other end outside it, such as a lifeline starting before the slice and ending in it. The paste will fail if one of the copied lifelines does not exist at the paste position.

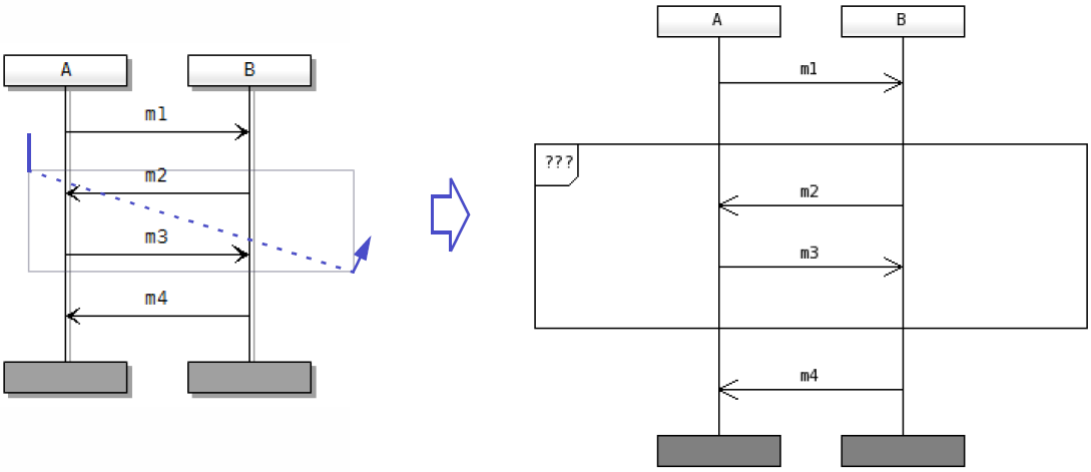
### 6.3.2 Symbol creation

Creating symbols in MSC diagrams is pretty straight forward. But some of the tools have a special behavior, mostly because of the nature of the MSC diagrams, which describes mostly a sequence of events, and not individual symbols:

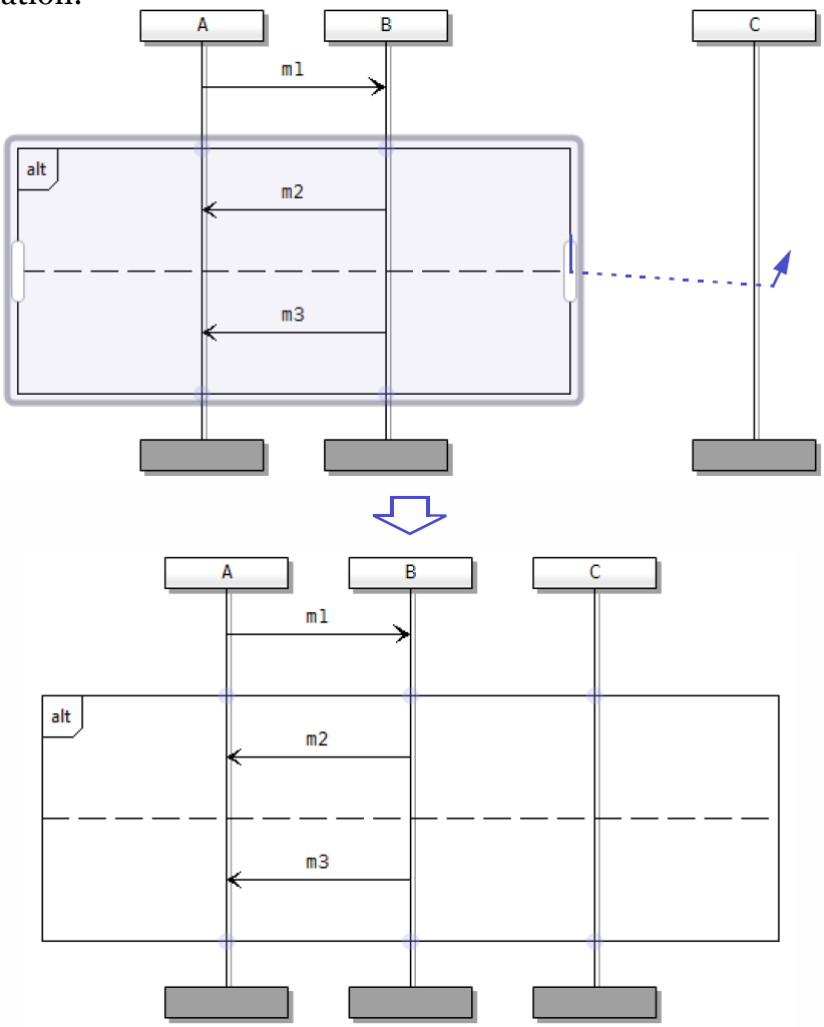
- When creating lifelines, only the horizontal position will be considered: lifelines are always created starting from the top of the diagram and going to the bottom.
- The creation of a lost (resp. found) message is done by selecting the message creation tool and clicking on the right side (resp. left side) of the lifeline sending it (resp. receiving it). For example:



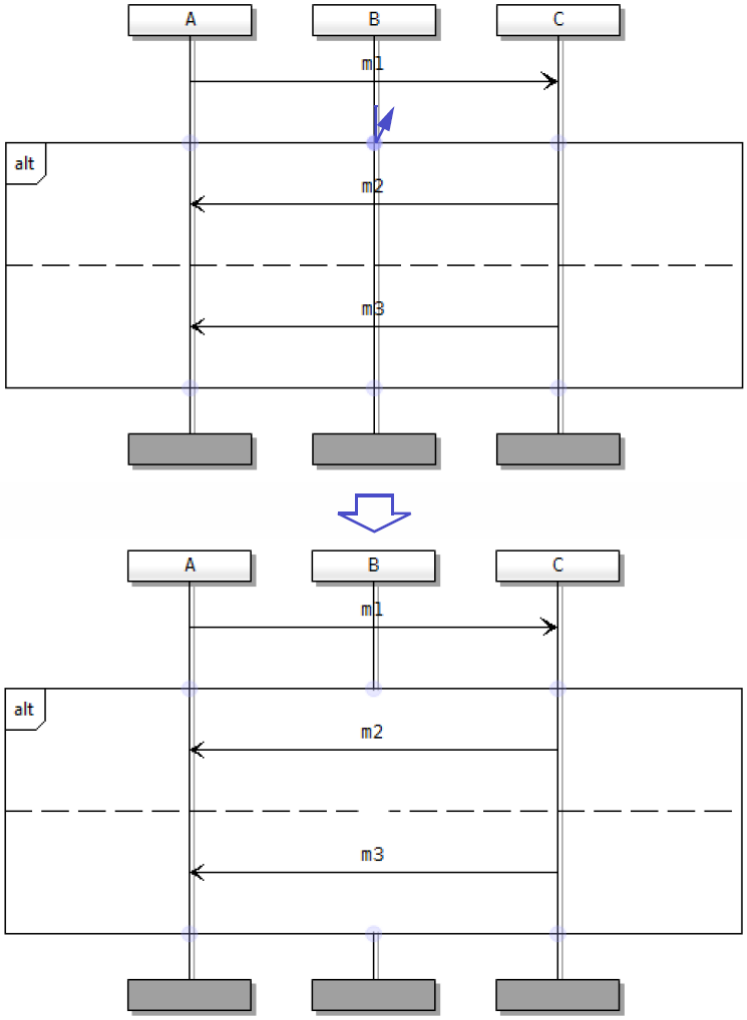

  Note that in legacy diagrams, lost and found messages have their own specific symbol that must be created the usual way, and a message link has to be created between the lost (resp. found) message symbol and its sender (resp. receiver) lifeline.
- For conditions, MSC references and inline expressions, they must be created over the lifelines they impact. This is done simply by making them span these lifelines, and optionally all the events that must be included in them:



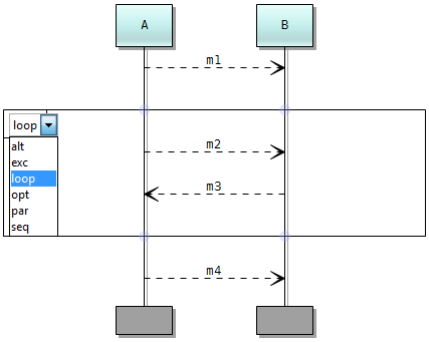

Once created, this symbols can be moved up or down by dragging them, and resized horizontally via the handles appearing on their sides when they are selected, which is the way to make them impact other lifelines than the ones setup at their creation:




Excluding from the symbol a lifeline included in it can be done via the circular handles appearing at the connection points between the symbols and the lifelines:




For inline expressions, their kind can then be changed by selecting it in the box appearing when the mouse cursor is over its text:




### 6.3.3 Manipulating components in lifelines

To the difference of all other symbols, lifeline are composite symbols: they may include several components like segments, timers or time constraints. They may also die before the end of the diagram or survive it.

In normal diagrams these features are managed via the toolbar:

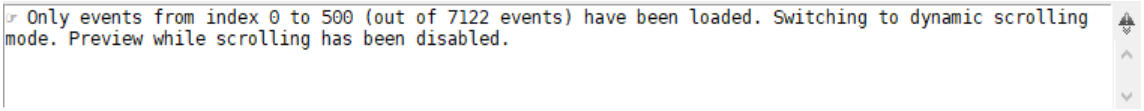
From top to bottom, the buttons add to a lifeline:

- a timer start;
- a timer time-out;
- a timer cancel;
- the start of a signal throw;
- the end of a signal throw;
- a signal catch;
- an enabled gate;
- a disabled gate;
- a time constraint;
- a PSC strict operator;
- an action symbol.

After selecting an item in the toolbar, press the mouse button at the desired position in the lifeline, and drag to its end position (if applicable). To cancel the insertion, hit the Esc key or select the selection tool.

### 6.3.4 Big diagrams handling

MSC diagrams can become quite big, especially if they are traces of models that can run for a very long time. To avoid slow-downs during editing and memory usage issues, the MSC editor switches to a special mode when it detects the opened diagram is big. In such a case, a message will be displayed under the diagram zone:



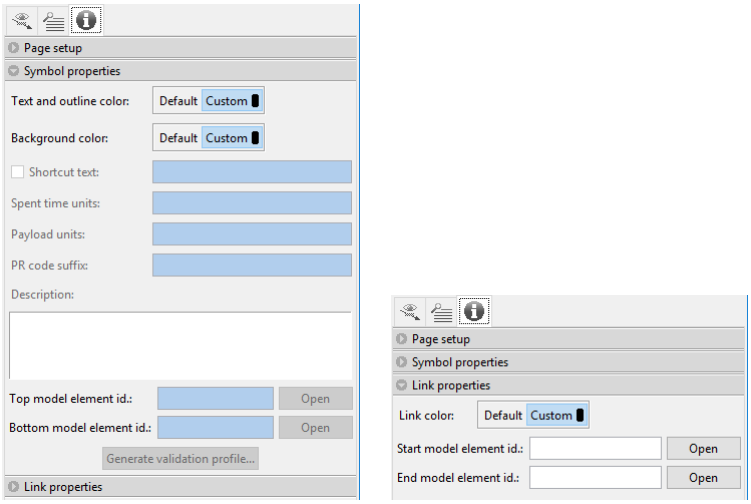
In this mode, most operations on MSC diagrams will remain available, but only a range of events will be displayed in the window, and not the full diagram. This causes a few limitations:

- The preview of the diagram while scrolling with the scroll-bar will be disabled. The diagram will appear faded out while the scroll-bar is used, and the view will update itself only when the scroll is over (mouse button released in the scroll-bar);
- Scrolling with the mouse wheel will occasionally trigger an update of the display, when the displayed part of the diagram gets out of the actually displayed event range;
- When selecting a rectangular region in the diagram, if the top or bottom border of the region gets too close to the limits of the displayed event range, it will become impossible to extend the region further;
- Events added or removed in the diagram will eventually trigger an update of the displayed event range to prevent it from becoming too big or too small.

Other operations will behave as in the normal mode.

### 6.3.5 MSC symbol and link properties

Symbols and links in MSC diagrams display internal information on the right panel. Please note some of the information may not be relevant to BPMN execution traces yet:




The information might be recorded automatically if the MSC diagram is a trace from an execution. They can also be specified “manually” via the symbol or link properties.

PragmaDev Process will open itself the model elements that it has recorded in traces.

### 6.3.6 Message parameters display

Please note this feature is not used in the current version of PragmaDev Process.

A specific sub-menu in the “View” menu controls the message parameter visibility:

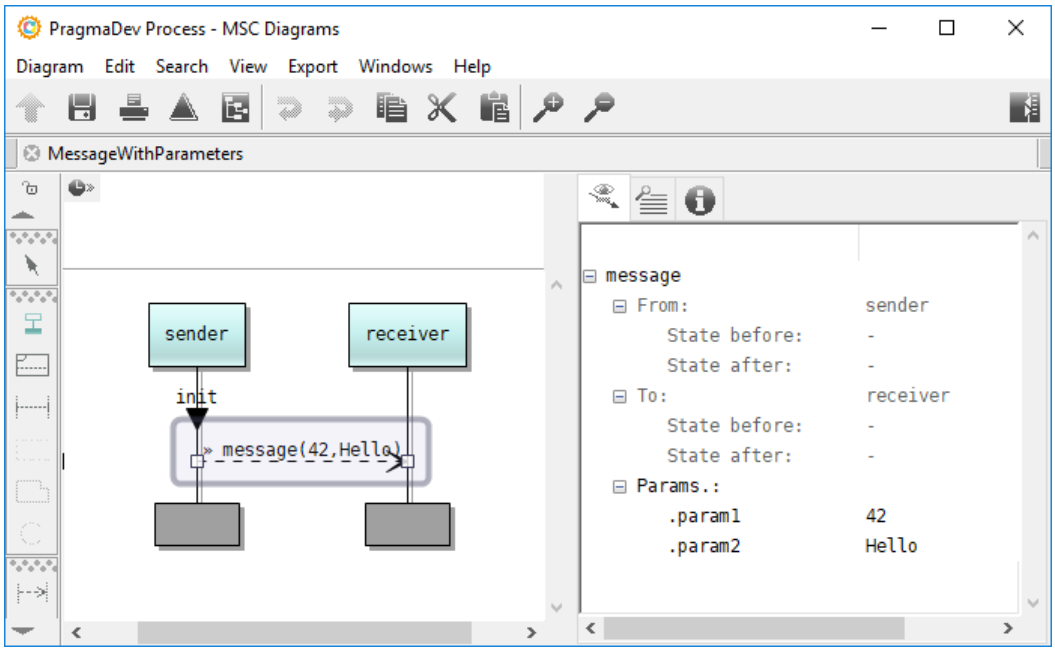
- A visibility set to “Full” displays the full text for the message parameters as it is recorded in the diagram file. The parameters for structured messages are then displayed in a flat textual format which can be quite difficult to read as this format is quite complex;
- A visibility set to “Abbreviated” still displays completely parameters for non-structured messages, but only displays the first level of parameter values in structured parameters. An example of this visibility can be seen below;
- A visibility set to “None” hides all message parameters.

This visibility setting is stored with the diagram. Please note it is only possible to modify the text for the message parameters if the visibility is set to “Full”.

When the visibility is set to “None” or “Abbreviated”, structured messages are indicated by a “»” before their name. Their parameters may be displayed by clicking on the message link: a panel then appears in the right part of the editor window displaying the parameters as a tree. For example, for a message with the full text:

mParams(|{param1|=42|,param2|=Hello|})

the display with parameter visibility set to “Abbreviated” and the link selected is:




Other information is also displayed in the panel:

- The sender and receiver process;
- The states of the sender and receiver processes before and after they sent or received the message (not relevant with current version);

### 6.3.7 Conformance checking: diagram diff & property match

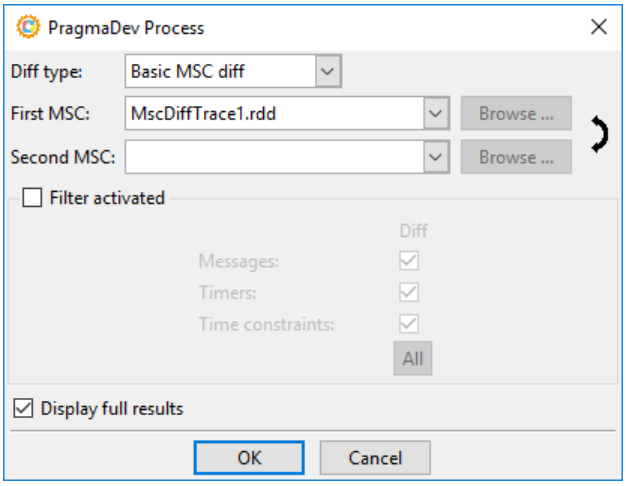
PragmaDev Studio offers 3 levels of conformance checking:

- A MSC trace can be compared to another MSC trace, used as a reference. This can typically be used for regression testing, the reference trace giving the wanted behavior, and being compared to a newly obtained trace. In this kind of comparison, all events in both diagrams are compared one by one without any interpretation of any kind. This is mainly intended for trace comparisons, but it also works on other diagram kinds, as items normally only present in specification or PSC diagrams are taken into account too, e.g inline expressions or relative time constraints.
- A MSC trace can be compared to a specification diagram. For this comparison, the semantics in the specification is taken into account. For example, if there is an 'opt' inline expression in the specification containing a sequence of message exchanges, the comparison will interpret it, and consider that the diagrams are matching if the sequence is there, or if it is not there at all. This allows to describe expected scenarios in a powerful way via specification MSC diagrams and match the execution traces against them later.
- Occurrences of a property described in a PSC diagram can be found in a MSC trace. In this case, the semantics are considered in the PSC diagram, as well as the PSC specific elements. Note that this is different from a specification vs. trace comparison, as properties describe a small part of a scenario that can actually match several times in a trace. MSC specification diagrams describe a whole scenario, and will be matched entirely on the trace. Properties are a good and powerful way to specify wanted and unwanted behavior in the designed system.

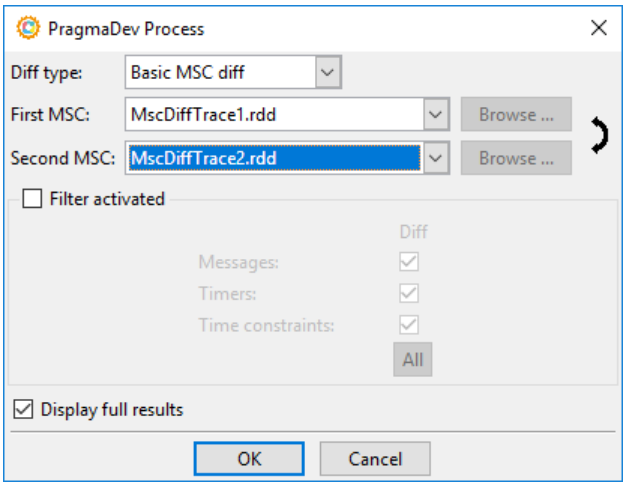
Important note: the current implementation of the algorithm used for specification vs. trace comparisons and property matches is limited in the number of events it can handle after a matched one and before the next one. The current limitation is 200 events in the trace, so if an event matches and the next event that should match is more than 200 events away from it, it won't be found. This limitation will be removed in a future version.

#### 6.3.7.1 Basic MSC diff: trace vs. trace, spec. vs. spec., ...

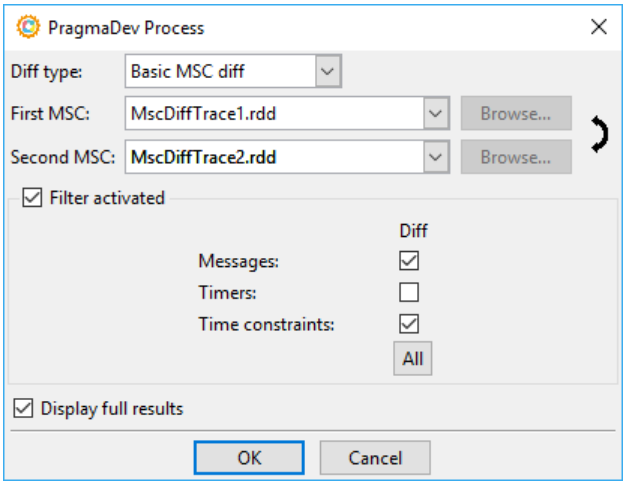
The basic MSC diff just compare two diagrams events by events and reports the found differences. This kind of comparison is launched by selecting 'Compare diagrams...' in the 'Diagram' menu, or by clicking the button in the toolbar. The following dialog then appears:




Selecting the basic MSC diff is done by selecting the corresponding value in the 'Diff type' field. The name for first MSC will be automatically set to the name of the currently displayed diagram. For the MSC to compare, it can be either selected in the list attached to the 'Second MSC' field, or loaded from a file via its 'Browse...' button. Once selected, the arrow in the right part of the dialog allows to exchange the two MSCs if the comparison must be done the opposite way.



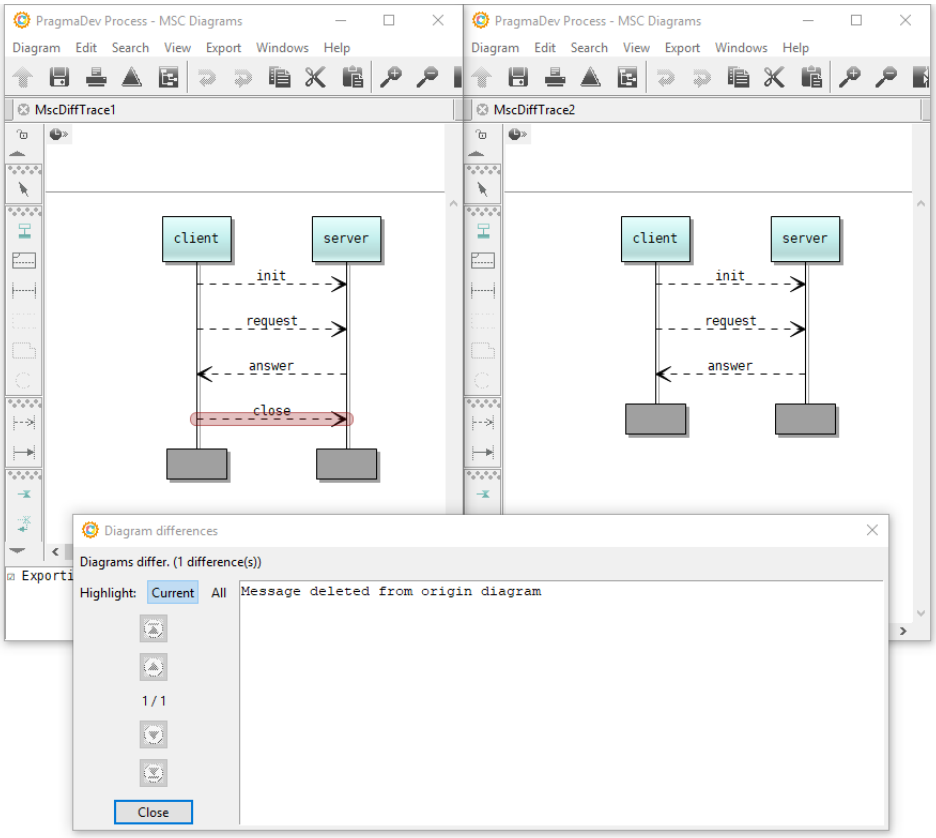

PragmaDev Studio allows to exclude some elements from the comparison based on their type. This is done by checking the 'Filter activated' option:




All the shown element types can be included or excluded from the comparison. The 'All' button will check all the boxes if any of them is unchecked, and uncheck them if all are checked.

The option 'Display full results' at the bottom of the dialog allows to display only a summary of the comparison results instead of the full set of differences. To display the summary, just uncheck the box.

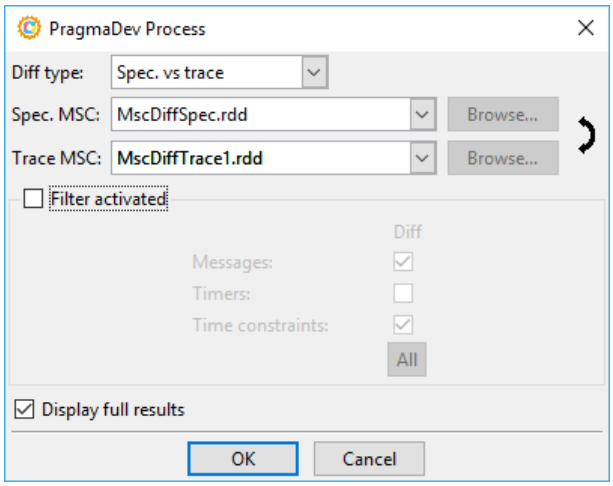
If this option is checked and after validating the dialog, PragmaDev Studio puts each diagram in its own window and displays them side by side. A dialog also appears at the bottom of the screen, allowing to browse through the found differences:



A summary of the differences is displayed at the top. Each difference will be highlighted in red in the diagram displayed on the left, and in blue in the diagram displayed on the right. The text in the dialog gives a short description of the identified difference. The arrows allow to browse through the differences. The option 'Highlight' allows to highlight all differences in both diagrams to get a quick view of what differs without having to browse through all the differences.

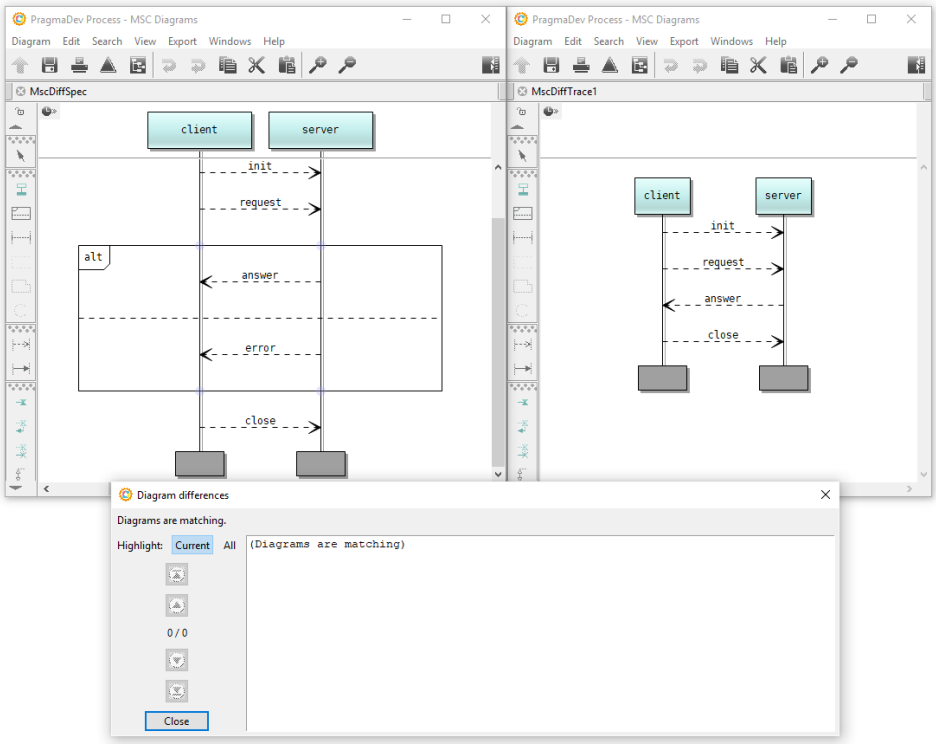
#### 6.3.7.2 Spec vs. trace comparison

Comparing a specification diagram to an actual trace is done the same way as for a basic MSC diff, except the diff type has to be set to 'Spec. vs. trace' in the dialog:



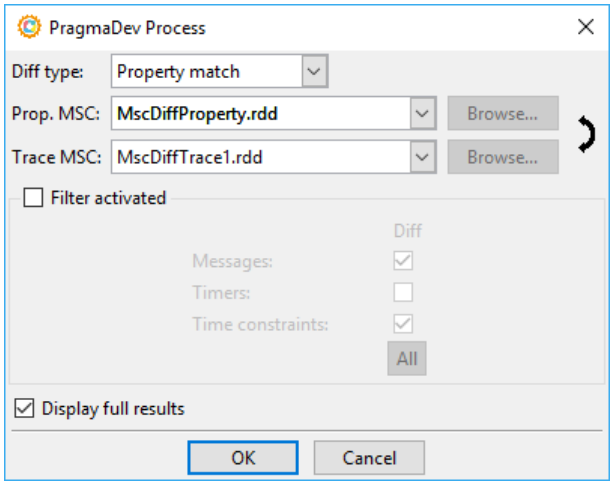

Note also that the specification diagram must be the one specified in the field 'Spec MSC' in the dialog, which is always the first one. If needed, the diagrams can be swapped by using the arrow button on the dialog's right side. The same filters are provided as for a basic MSC diff.

Once validated, the found differences are displayed in the same way as for a basic MSC diff; only the way to perform the comparison changes, as semantics in the specification is taken into account where it would not be in a basic MSC diff:




### 6.3.7.3 Property match

Matching a PSC diagram against a MSC trace is done the same way as for the other kinds of comparisons, except the diff type has to be set to 'Property match':

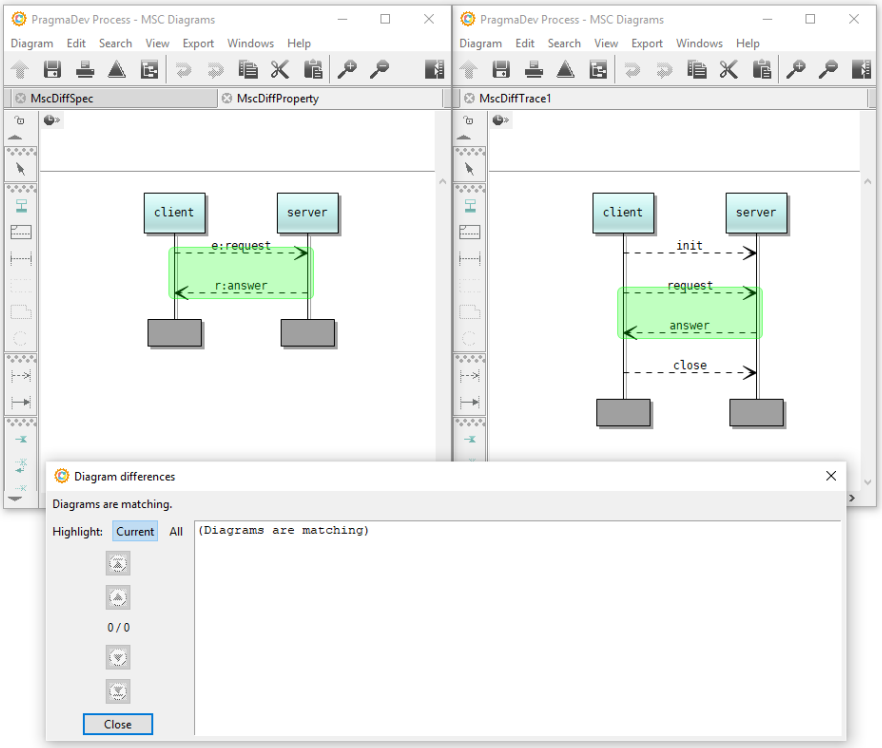



Note that the PSC diagram has to be the one specified in the 'Prop. MSC' field, which is always the first one. If needed, the diagrams can be swapped with the arrow button on the right side of the dialog. The same comparison options are provided as for basic MSC and specification vs. trace comparisons, but they are less significant here, as a property diagram is always partial.

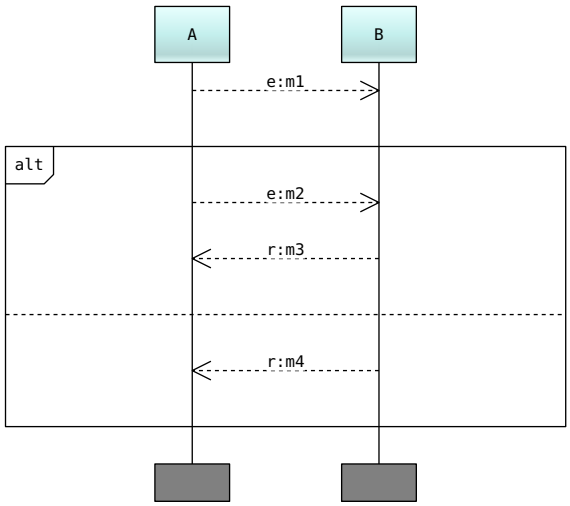
Once validated, the property matches and violations are displayed in a similar way to the display of differences in the other kinds of comparisons. Mostly the colors and the dif-

ference descriptions differ: each matched element in the property or the MSC diagram will be displayed as green, and each unmatched one as red. The difference description will be:

- 'Property match' if the property matches:




- 'Violated property!' if the property does not match.
- 'Possibly violated property' in some very specific cases where it is impossible to tell if the property is matched or not. A typical example where this case happens is the following:




If the trace contains a message m1 from A to B, followed neither by a message m2 from A to B, nor by a message m4 from B to A, there's no way to know which part of the alternative should have matched. But if it was the first part, the message

m2 is not there, so the property does not apply, and if it was the second one, the required message m4 is not there either, so the property is violated. In this case, a possible property violation will be reported. Note that a property is not necessarily violated if something does not match in it. Typically, an unmatched fail message means the property is matched.

# 7 Explorer

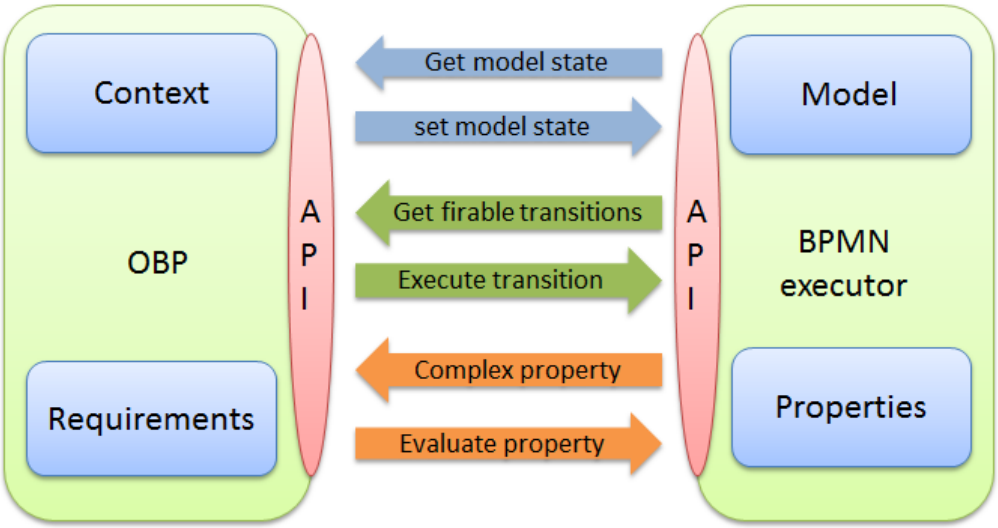
The Explorer automatically explores all possible execution paths of the model. The possible paths and their intermediate states build what is called a state space. There are two interesting results for this feature:

- The size of the state space, that is basically the number of possible execution steps in the model. If it is too high that means the model is complex and might not be doing what is expected.
- Verify a property in order to make sure some scenario actually can not happen whatever the scenario.

Please note OBP requires Java to be installed on the computer.

## 7.1 Architecture

The state space exploration is done with OBP (Observer Based Prover) tool developed by ENSTA Bretagne research lab. To do so OBP relies on PragmaDev Executor. At each step OBP asks the executor what are the possible paths of execution (what we call transitions). OBP will then try all possible paths and the executor will provide the resulting state for each one. For each resulting state OBP will ask for the possible transitions and so on. The key aspect here is that OBP does not execute the model, it relies on the executor. This is quite unique as usually verification tools have their own semantic and executor.




While exploring OBP can verify user defined properties defined with a PSC (Property Sequence Chart) diagram. Internally the PSC is translated to a Büchi automaton and

sent to OBP. The Büchi automaton is based on what we call atomic properties of the model. At each step of execution OBP asks the BPMN executor to evaluate the atomic properties to evaluate the overall Büchi automaton. While atomic properties are static and usually boolean, the Büchi automaton can express quite complex sequence of events.

## 7.2 Properties

The properties are defined using the PSC format. For the time being only the following features are supported in the PSC:

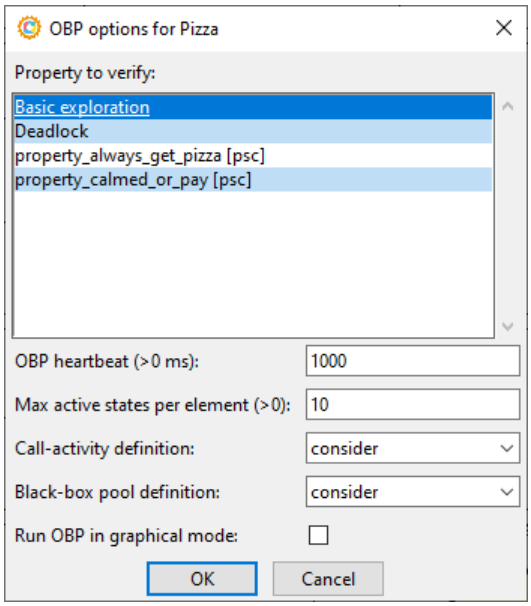
- e (regular message or task),
- r (required message or task),
- f (failed message or task),
- Alt in-line expression,
- Strict operator.

Please refer to "PSC-specific normal, required and failed message syntax" on page 81 and "Main symbols" on page 82 for more information.

## 7.3 Launch an exploration

The Explorer is launched from the editor windows with the following button:

The following window will then open:




The *Basic exploration* enables full state space exploration. *Deadlock* is a built-in property that allows checking for any blocking state. A blocking state during exploration

means there are no further actions to be performed (i.e., enabling and disabling) even though the process is not finished (terminated). The lines after *Deadlock* are all the PSCs found in the project. A PSC defines a single property. OBP can only explore one property at a time.

The following can be configured:

- *OBP heartbeat*

  An exploration might take a substantial amount of time. For that reason, every heartbeat, some status information is displayed in the exploration window. It is possible to change the refresh value (in milliseconds) of the status information.

- *Max active states per element*

  This is is the the upper limit for *active* states allowed for a given BPMN element during exploration. For sequence flows, the limit is applied to *active* and *ready* states together. This option is also used to address the issue of infinite configuration that may be caused by complex looping paths for example.

- *Call-activity definition*

  Exploration can be limited to certain scenarios by defining how a call-activity definition is handled:

  - *consider* : always step into call-activity (if possible) during exploration,
  - *do not consider* : always step over call-activity during exploration.

- *Black-box pool definition*

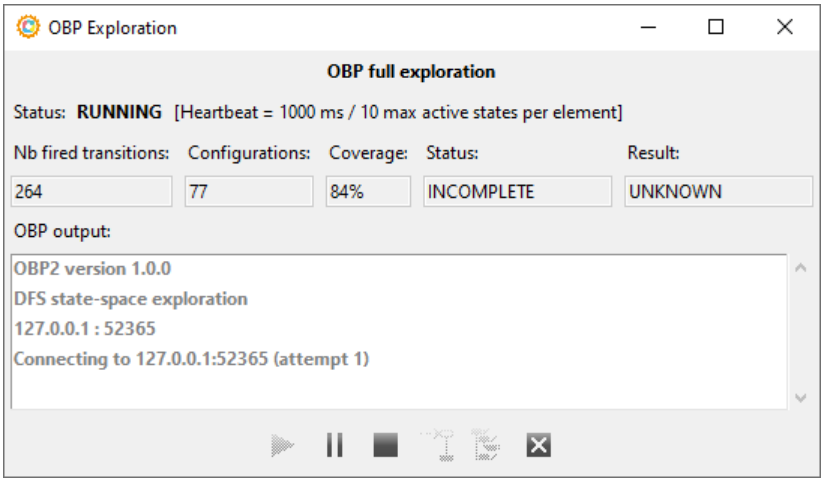
  Exploration can be limited to certain scenarios by defining how a black-box pool definition is handled:

  - *consider* : always load (if possible) black-box pool definition during exploration,
  - *do not consider* : never load black-box pool definition during exploration.

- *Run OBP in graphical mode*

  This is an advanced feature that allows more in depth analysis of the exploration state-space via the OBP GUI tool. More information on the tool ca be found at `http://www.obpcdl.org`. The Java SE Runtime Environment 8 (`https://www.oracle.com/java/technologies/javase-jre8-downloads.html`) must be installed to run OBP in graphical mode.

Once the exploration is launched, the following window will show what the status is:

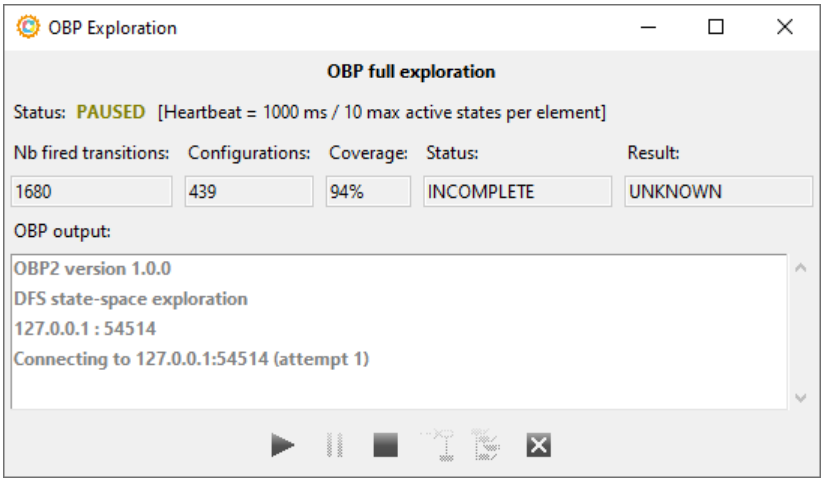



The dialog shows the progress of the exploration:

- *Nb fired transitions* is the number of executed transitions in the model since the beginning of the exploration;
- *Configurations* is the number of distinct system states the exploration has seen so far;
- *Coverage* is the percentage of symbols executed at least one during the exploration compared to the total number of symbols in the explored model(s);
- *Status* indicates whether the exploration is running and still incomplete or if it's over;
- *Result* is the result for the exploration when applicable.

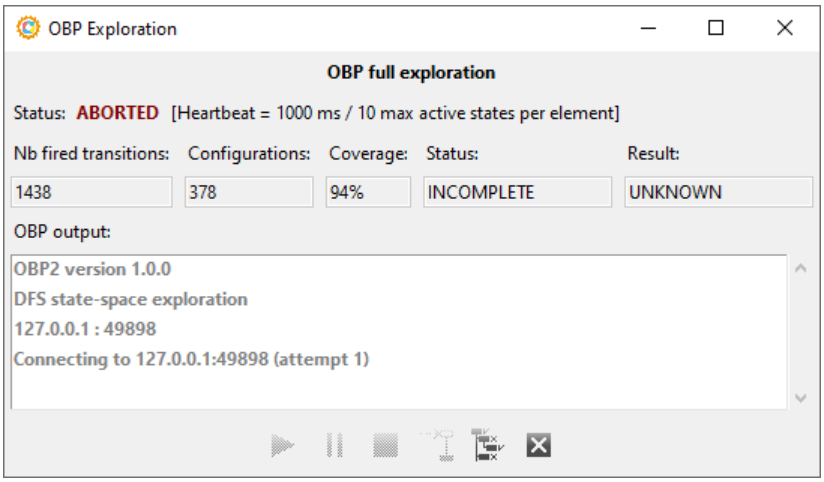
Since exploration can take quite some CPU and memory, it is possible to pause the exploration with the *Pause* button:

The exploration can then be resumed or paused again:



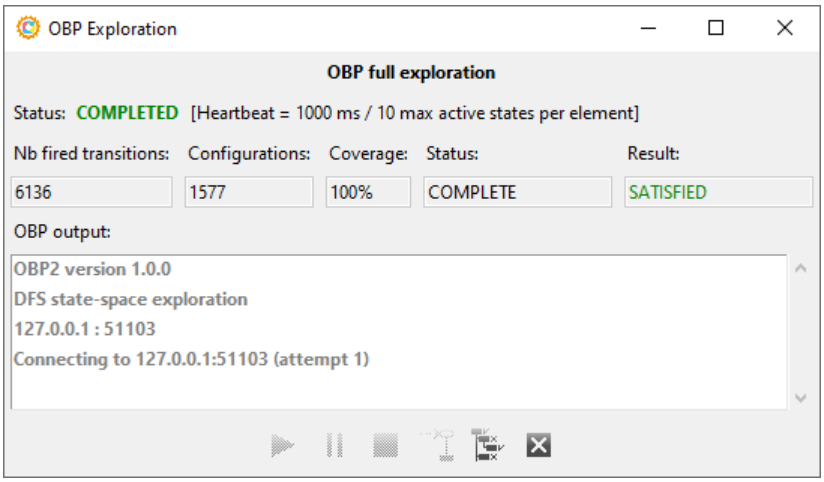

It is also possible to abort the exploration, for example in the case where it starts taking too much time or resources for the explored model, which probably means there

is something wrong with the model itself. This is done via the *Stop* button: . The progress window will then show the status "*ABORTED*":




Note that the exploration cannot be resumed. However, it is possible to extract the model coverage to see how far the exploration went.

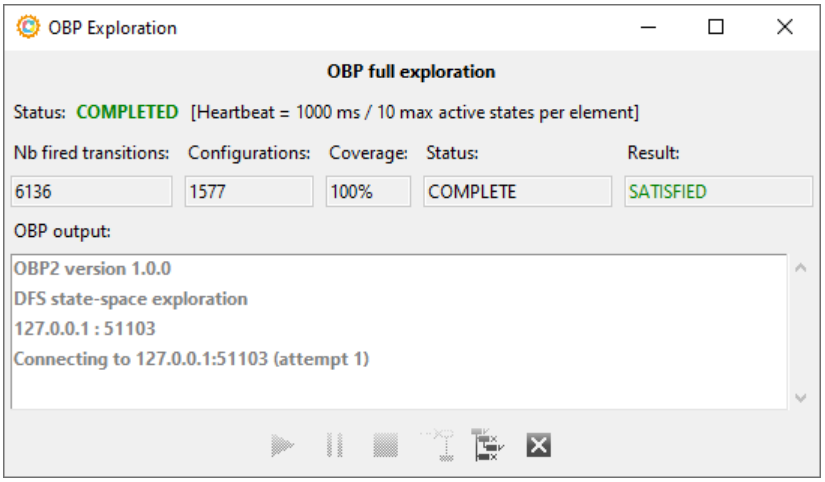
A completion status will be indicated at the end of the exploration in the "OBP exploration status" field:




## 7.4 Result analysis

### 7.4.1 Full state space exploration

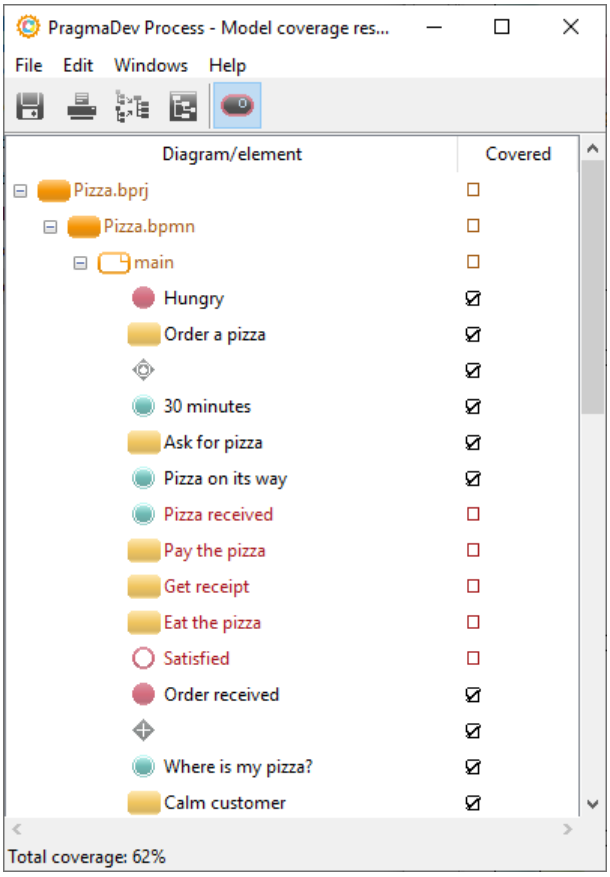
The full state space exploration does not verify any property, it is only a full exploration of all the possible configurations of the model. At the end of the exploration, the only valuable information is the number of configurations that have been explored. In the example below:




1577 different configurations have been explored. This has to be compared to the model complexity to find out if that result is too high or normal. A simple plain sequence is usually less than a hundred configurations. If it is too high, there is probably a misconstruct in the model. But if there are messages exchanged in loops in the model the number of configurations can go way up a thousand.

### 7.4.2 Uncovered elements

The *Show uncovered elements* button may be used to obtain coverage information when exploration is complete: all symbols in the model will be presented in a tree with the associated coverage information. Since the number of times the symbol was hit is not relevant for an exploration, only a covered / uncovered indicator will be displayed for each symbol:

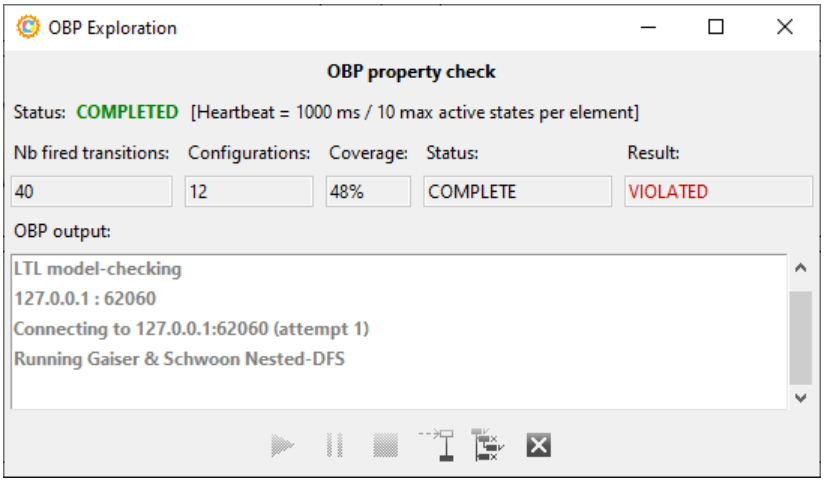



All non covered elements will be automatically marked in the editor as described in Highlight non-covered symbols.

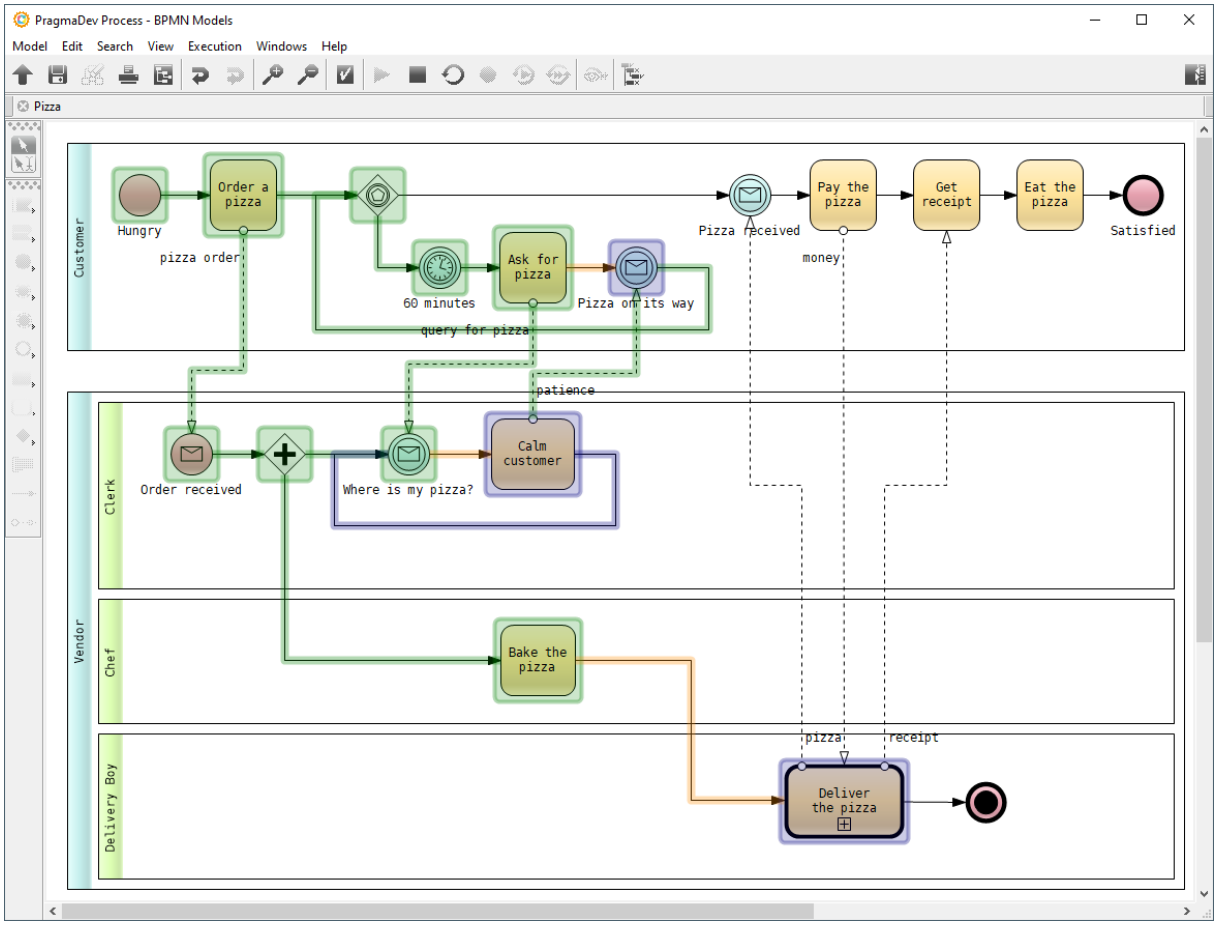
The *Quit* button will stop the exploration and close the window.

### 7.4.3 Property verification

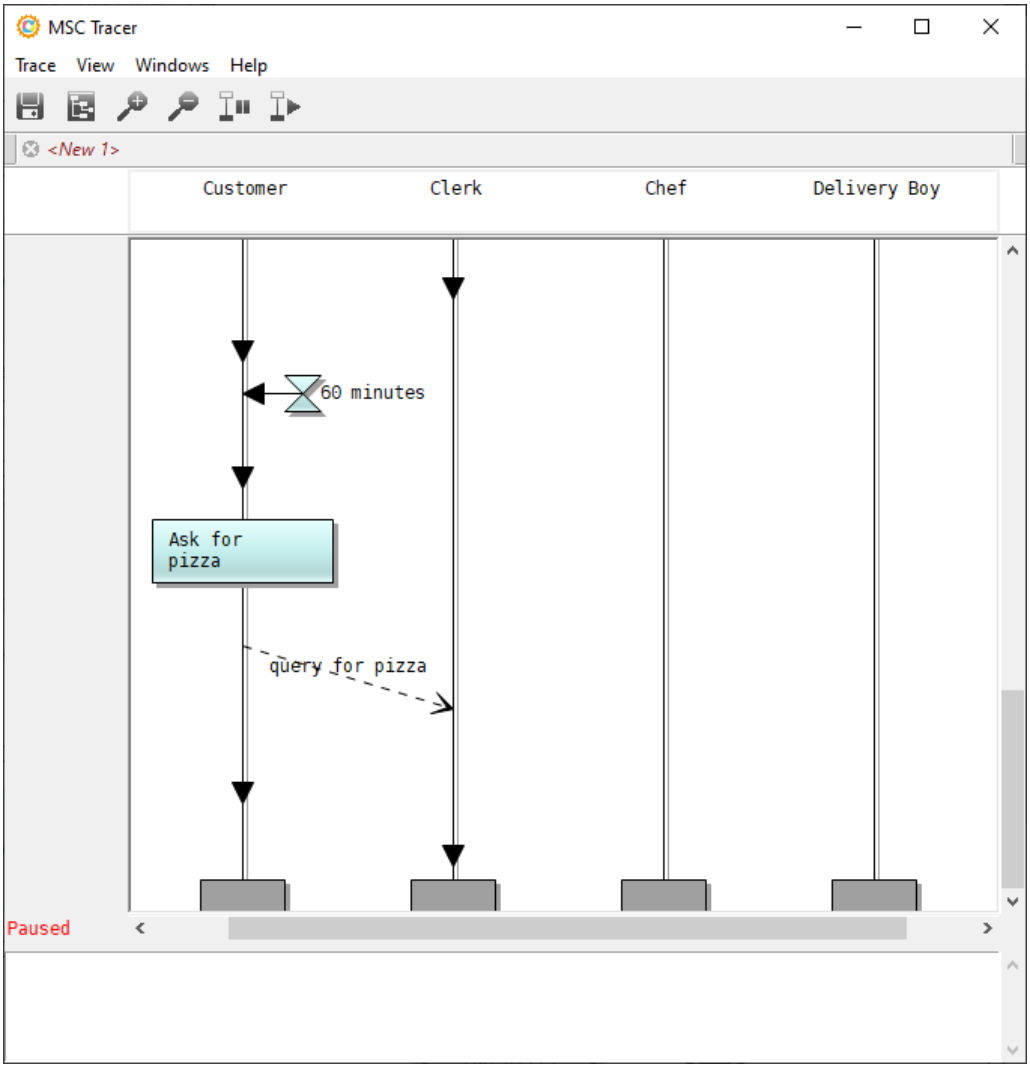
At the end of the exploration, the window will indicate if the property (*Deadlock* or user defined with PSC) has been violated or not in the "Result" field. In the example it has been violated quickly after starting the execution:




Pressing the *Display counter example* button will display the scenario that violated the property in the editor:




An execution trace is also generated with all the execution steps:




The problem can then be fully analyzed and the model corrected.

# 8 Simulator

## 8.1 Principles

The objective of a simulation of BPMN models is to evaluate the time needed to perform a given process, as well as its cost. For that, individual times and/or costs are defined on the elements of the model, and the simulation will evaluate these times and cost in the order of the process and compute the totals. To allow the simulation to be completely automatic, BPMN elements that require a choice such as exclusive or inclusive gateways can also define probabilities for each of their branches, so that the simulation can pick one up as it goes. Times and costs of BPMN elements can be defined as fixed values (e.g a task lasts 2 days, or costs 300$) or as variant values following a distribution (e.g a task last in average 30 minutes, following a normal distribution with a standard deviation of 10 minutes). In addition to time and cost information, elements of the model can also use resources that are shared between all processes. An element will not start until it can get the resources it needs to run.

The simulation feature of PragmaDev Process relies on the BPSim standard[1]. This standard adds attributes to the elements in a BPMN model for their time and cost. It also defines a way of setting the kind of results that must be produced in the end. BPSim also defines the concept of resources, but PragmaDev Process extends this concept with unique features, that are described in "Resources Editor" on page 157.

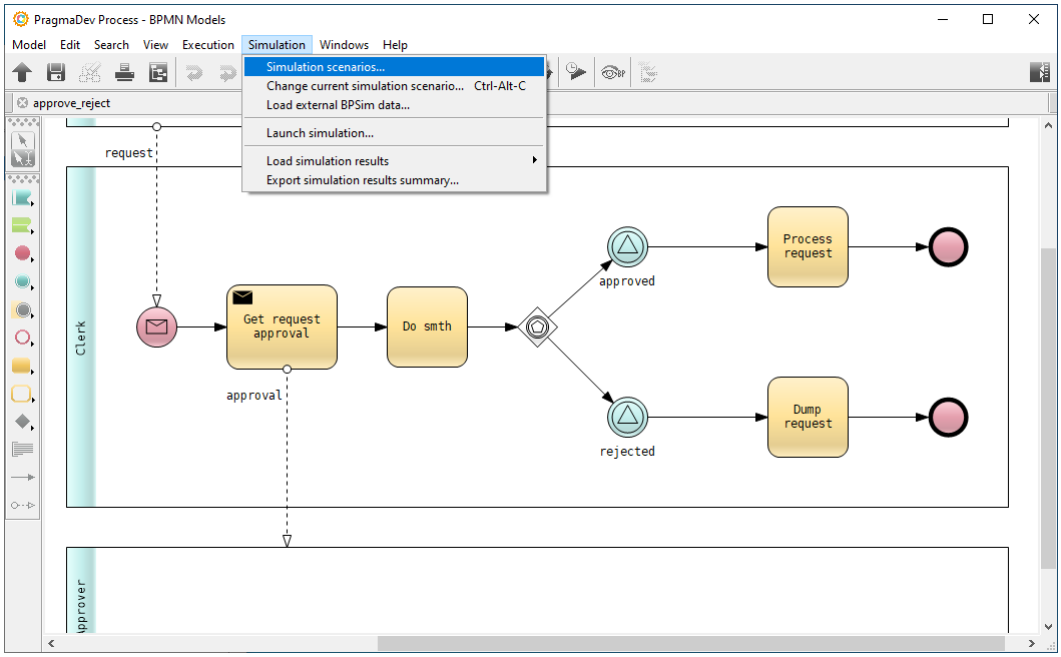
PragmaDev Process also adds the possibility to gather logs on individual executions for simulation, that are not present in the BPSim standard. These allow to get precise details on each individual execution to be able to generate a MSC trace if needed, for example for executions that cause the total time to be bigger than expected.

BPSim relies on the definition of simulation scenarios, that include simulation attributes for each element in the BPMN model. A simulation will then execute one or several of these scenarios.

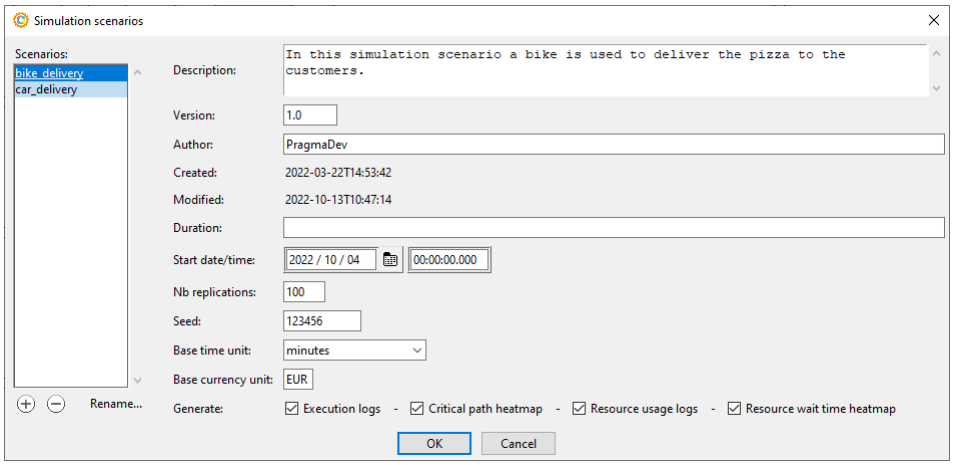
## 8.2 Simulation scenarios

Simulation scenarios can be added to a BPMN model from within the BPMN editor window by selecting "Simulation scenarios..." in the "Simulation" menu:

[1] https://bpsim.org




The following dialog is then displayed:




The list on the left side of the dialog shows the already defined simulation scenarios, which are identified by their name. The name appearing in orange in the list is the "active" scenario, which is the one for which the simulation attributes will be defined for the elements in the model, as described below. The buttons under the list allow to create a new scenario, optionally by copying another one, to delete the selected scenario or to rename it.

The zone of the right side of the dialog lists the attributes of the scenario itself:

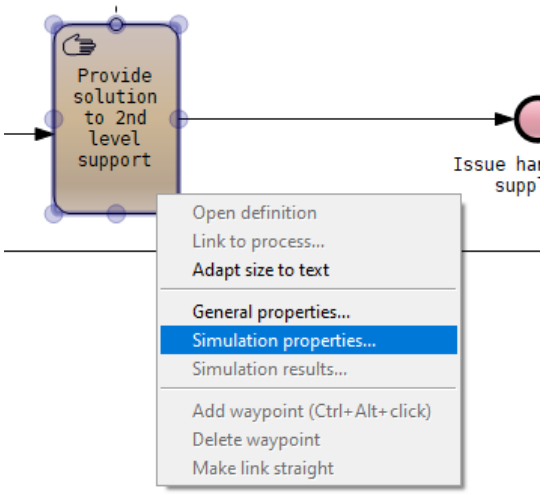
- *Description* is an informal textual description of the scenario.
- *Version* is an informal version number for the scenario.
- *Author* is the name of the scenario author.
- *Created* is the creation date for the scenario. It is set automatically and cannot be modified.

- *Modified* is the date of last modification of the scenario. It is also set automatically and cannot be modified.
- *Duration* is the duration of the scenario. After this duration, the scenario will be ended and all processes that have not finished will be aborted.
- *Start date/time* is the starting date and time of the simulation.
- *Nb replications* is the number of replications for the scenario. For example, if this number is set to 10, the scenario will be repeated 10 times with the same parameters. Replications are also called *instances* of the scenario.
- *Seed* is the seed for the random number generator. This allows to create scenarios that will always give the same results, even if the attributes rely on random parameters based on distributions. Note that the random number generator will be restarted with this seed each time the scenario is started, but *not* between replications.
- *Base time unit* is the time unit to use for the timing attributes for the BPMN elements. Note that this unit applies only if the time is specified using a plain number (integer or real). Times can also be specified via a duration that includes its own time unit.
- *Base currency unit* is the currency unit to use for all costs defined in the simulation parameters for BPMN elements.
- *Generate / Execution logs* indicates that the individual logs for each execution must be generated with the final simulation results.See "Logs" on page 145 for more details.
- *Generate / Critical path heat map* indicates the critical path will calculated and stored for each replication. At the end of the simulation the resulting statistical information can be displayed as a heat map. See "Critical path" on page 155 for more details.
- *Generate / Resource usage logs* indicates that the logs for resource usage for each activity must be generated with the simulation results. A log is generated for each scenario replication and lists all the takes and releases of resources for each activity in the model. See "Resource logs" on page 151 for more details.
- *Generate / Resource wait time heatmap* indicates that a heatmap will be generated during simulation showing which activity has waited for the longest time in average for its resources for the scenario. See "Resource wait time heatmap" on page 153 for more details.

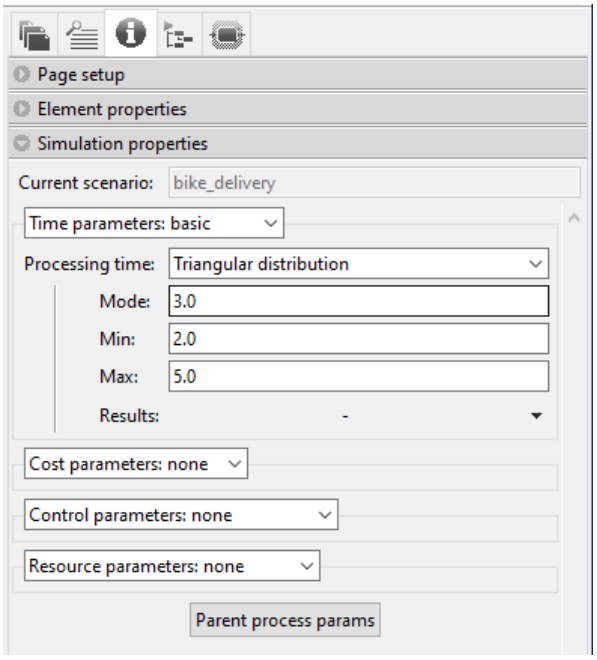
## 8.3 Simulation attributes

Simulation attributes define the parameters for a simulation for individual model elements. They are always associated to a scenario, and two scenarios will have a different set of parameters for the model elements.

The simulation attributes can be displayed within the BPMN editor by right-clicking on any model element and selecting *Simulation properties...* in the contextual menu:




This will display the simulation properties in the panel on the right side of the editor window, listing all the simulation attributes:




The simulation attributes are called parameters, and are displayed in 4 groups:

- The *time parameters* allow to define the time spent in the element. Times are typically associated to tasks. There are several times defined by the BPSim standard, that are described below in "Time parameters", on the following page.
- The *cost parameters* allow to define the cost for the element. Costs are typically associated to tasks. There are also different kind of costs defined in the BPSim standard, that are described below in "Cost parameters", on page 125.
- The *control parameters* allow to define how things are triggered within the model. They define the number of process instances that will run within one replication

of a scenario, and the probabilities for gateway branches when they are needed. Control parameters are described in "Control parameters", on page 126

- The *resource parameters* allow to specify which resources the selected element needs to be able to run. Resources can only be associated to tasks. Resource selection is described in "Resource parameters" on page 130.

The values for all the time, cost & control parameters may be defined by one of the following:

- A fixed value of a type compatible with the parameter. Available types are booleans, integers, floating point numbers (reals), strings, and special types named Date-Time and Duration, which allow to specify dates & durations in ISO-8601 format.
- A value defined by a distribution, that will be computed randomly during the simulation. Available distribution types are: beta, binomial, Erlang, gamma, log normal, negative exponential, normal, Poisson, triangular, truncated normal, uniform and Weibull. Each distribution type has its own set of parameters.

If a parameter is defined via an integer, a real, or a distribution, its unit is the unit defined in the scenario: base time unit for times, and base currency unit for costs.

Time, cost & control parameters also have associated *result requests*, which define which kind of results will be included in the results file produced by the simulation. They are described in detail in "Result requests" on page 129.

### 8.3.1 Time parameters

Time parameters are usually associated to tasks, and define the time spent in them. BPSim defines 7 kinds of times:

- The *transfer time* is the time spent between the previous processing step and the task;
- The *queue time* is the time between the moment when the token enters the task and the moment when the task is ready;
- The *wait time* is the time between the moment when the task is ready and the moment when it actually starts;
- The *setup time* is the time needed by the task to setup everything before actually starting its work;
- The *processing time* is the time actually spent doing the task's work;
- The *validation time* is the time spent checking the task's work;
- The *rework time* is the time spent correcting or redoing the task's work.

All of these times can be defined either via a fixed value, which can be an integer, a real number or a duration, or via a distribution of values. Note that only duration values include their own time units and are not based on the scenario's base time unit.

All of these parameters do not have to be defined. If only the definition of the time spent in the task needs to be defined, the time to use is the processing time.

In addition to the kinds above, BPSim also defines the following times, but they support only result requests and cannot have an associated value:

- The *lag time* is the time spent before the task's work actually starts. It is the sum of the transfer time, the queue time & the wait time.
- The *duration* is the time spent actually doing the task's work. It is the sum of the setup time, the processing time, the validation time & the rework time.
- The *elapsed time* is the total time spent in the task. It is the sum of the lag time and the duration, i.e the sum of all the times defined for the task.

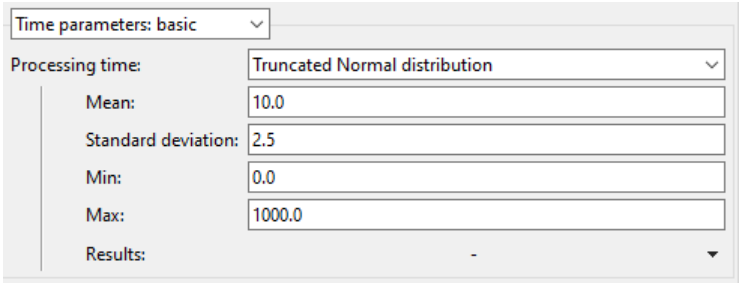
Note that PragmaDev Process supports only the elapsed time.

There are 3 possibles forms for the time parameters displayed in a model element's simulation properties.

The first form is the following:

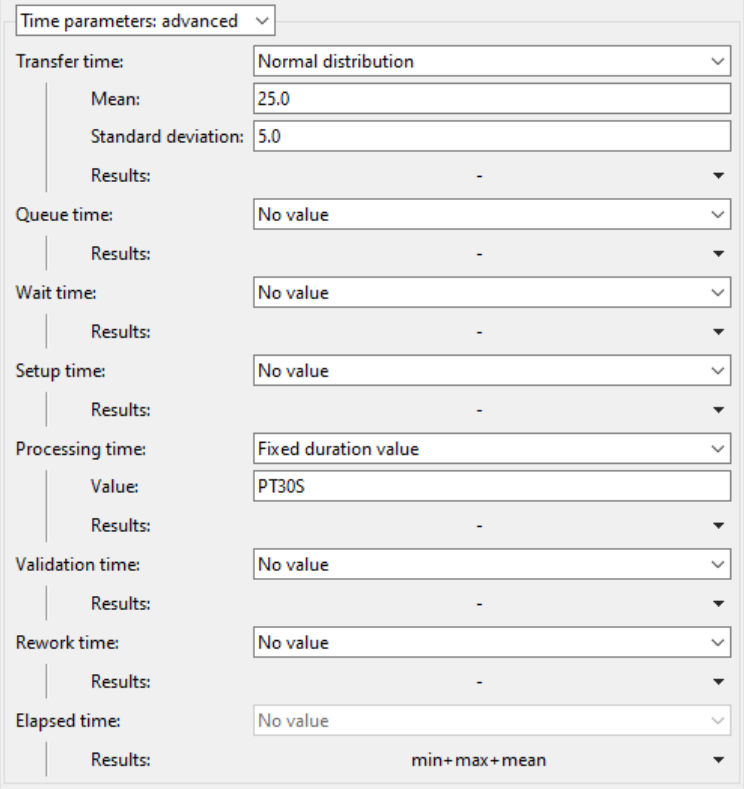
This form is used when no time parameters are defined for the element.

The second form is the following:




This form is used when only the processing time has a value. Here, its value is based on a truncated normal distribution and it has no associated results requests.

The third form is the following:




This form must be used when any other time than the processing time has a value or a results request.

Note that PragmaDev Process allows to define time parameters on call activities, which is not allowed in the BPSim standard. When such a parameter is defined, the call activity can be considered just as a task, i.e that its definition will be ignored. In this case, the time(s) used for the call activity for the scenario will be the time(s) defined in the parameters, and not the times used in the process it calls. See the semantics of the load definition option in "Control parameters" on the next page.

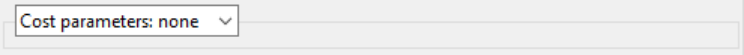
### 8.3.2 Cost parameters

Cost parameters are usually associated to tasks, and define the needed amount of money for them. BPSim defines 2 kinds of costs:

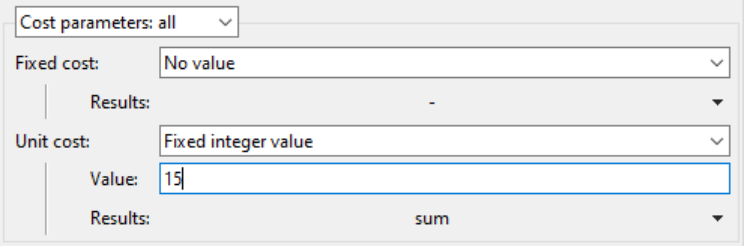
- The *fixed cost* is the fixed amount of money needed for the task;
- The *unit cost* is the amount of money needed for the task for each time unit spent in it.

Both costs use the base currency unit defined in the scenario. Any task can have one of these times defined, or both. The valid values for these parameters are integers, floating point numbers or distribution-based. Both kinds support result requests, on individual model elements or on a process as a whole.

The two forms for the cost parameters displayed in the simulation properties are:




when no cost parameter is defined for the element, and:




when one or both of the cost parameters are defined, or have an associated result request.

Note that just as for time parameters, PragmaDev Process allows to define cost parameters on call activities, which is not allowed by the BPSim standard. Just as for time parameters, this means that the actual process called by the call activity will be ignored and the cost associated to the call activity in the results will be the one defined in its parameters.

### 8.3.3 Control parameters

Control parameters are associated either to events, to sequence flows attached to a gateway or to "black box" participants. The following kinds of control parameters are available:

- The *inter-trigger timer* can only be set on events; it defines the amount of time between the event is triggered. Its value can be an integer, a floating point number, a duration, or distribution-based. Note that if this parameter does not have a value, this means that the event is never triggered.
- The *trigger count* can only be set for events; it defines the maximum number of times the event will be triggered during the simulation. Its value must be an integer. If this parameter does not have a value, it means that there is no limit and that the event will be triggered until the scenario ends.
- The *probability* can only be defined on sequence flows attached to a gateway. It defines the probability that this sequence flow will be selected. Its value must be an integer, a floating point number, or (rarely) distribution based. Note that PragmaDev Process only supports probabilities expressed as reals between 0 and 1, or percentages between 0 and 100. Anything else will be considered as an error.

  For exclusive gateways, not specifying any probability on outgoing sequence flows means that each one of them has an equal chance of being selected. For inclusive gateways, all probabilities must be specified. Note that since inclusive gateways must have at least one outgoing sequence flow selected during simulation, the computation of probabilities will be redone until at least one of the branches is selected, unless the *condition* parameter is set (see below).

- The *condition* can only be defined on sequence flows attached to a gateway. Its value is a boolean. Alone, it specifies that this sequence flow will either be always selected if the value is true, or never if the value is false. If they are associated with a probability in the context of an inclusive gateway, they are used as a "fallback" if the probabilities computed during the simulation end up not selecting any of the sequence flows. In this case, those triggered are the ones which have their condition set to true.
- The *load definition* parameter is a PragmaDev Process extension[2]. It can only be defined on:
  - "black box" participants, i.e participant symbols having no symbol in them;
  - call activities.

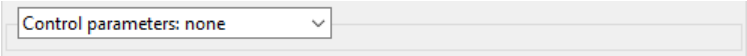
  It is a boolean that specifies if the definition of the participant or call activity must be loaded if it exists, or if it shouldn't. In the case of call activities, it has a meaning only if the call activity has time and/or cost parameters associated to it. In this case, if *load definition* is false, the call activity will be "stepped over", and the times & costs used in the simulation will be those defined on the call activity itself; if *load definition* is true, the process called by the call activity will actually be executed, and the time & cost parameters defined on the call activity will be ignored.
- The *message delay* parameter is another PragmaDev Process extension[3]. It can only be defined on message flows originating from "black box" partiticipants.

  By default, i.e when this parameter is not set or 0, when the definition of a black-box participant is not loaded and there is a message flow from this black box to a task, the message is assumed to be sent as soon as the receiving task is executed.

  The parameter introduces a delay between the time when the task becomes active and the time the message is sent / received. It is usually a number specifying the numer of time units for the delay, but it can also be a duration, or even a distribution-based value.

All these parameters except *load definition* and *message delay* support result requests, but in practice, only a *count* result request on the trigger count actually has any meaning.

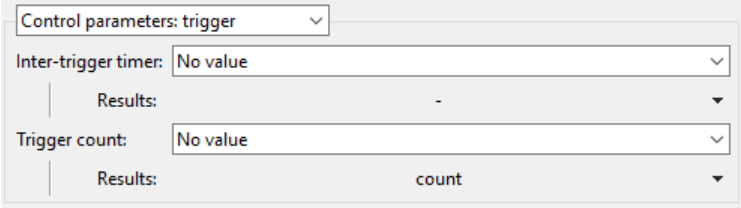
Control parameters will appear differently depending on which are defined and which model element is selected:



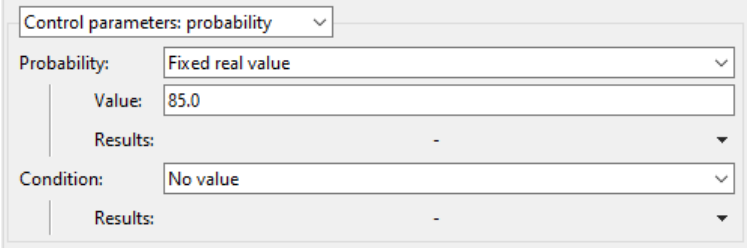

This form is used when no control parameters are defined on the selected element.

[2]Internally, this parameter uses the *condition* parameter, which can be set neither on participants nor on call activities according to the BPSim standard.

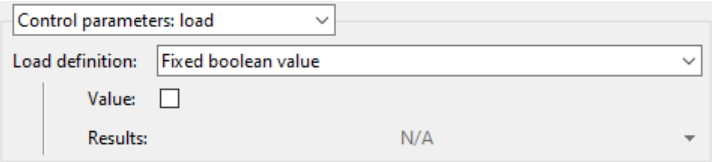
[3]Internally, this parameter uses the *inter-trigger timer* parameter, which cannot be set on message flows according to the BPSim standard.



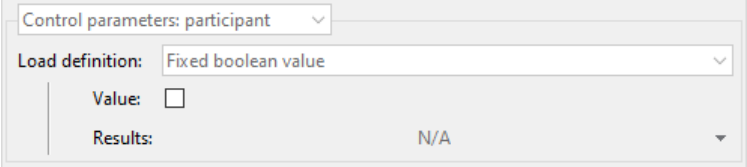

This form is used when the inter-trigger timer and/or the trigger count is defined on the element, or when any of them has an associated result request. This will typically be the case for events.



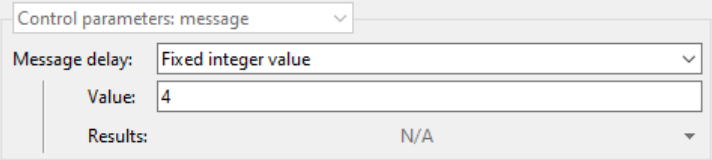

This form is used when the probability and/or the condition is defined on the element, or when any of them has an associated result request. This will typically be the case for outgoing sequence flows for a gateway.




This form is only used for call activities to define their *load definition* parameter. Note that it is slightly different from the form used for participants, since call activities can have other control parameters, such as *inter-trigger timer* and *trigger count*, since they can start a process.




This form is only used for black box participants when its *load definition* parameter is defined.




This form is used only for message flows originating from a black-box participant to allow to define their *message delay* parameter.

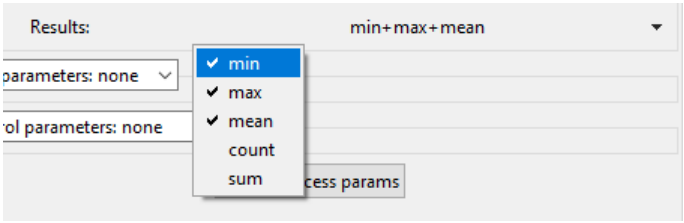
### 8.3.4 Result requests

Result requests define which kind of results will be included in the results file produced by the simulation. They can include one or several of the following types:

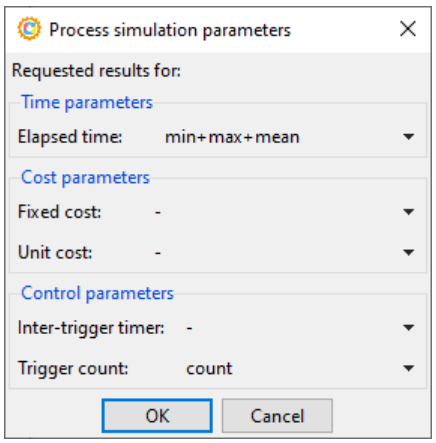
- *min* will include in the final results file the minimum value for the parameter; this is typically used for times or costs;
- *max* will include in the final results file the maximum value for the parameter; this is typically used for times or costs;
- *mean* will include in the final results file the average value for the parameter; this is typically used for times or costs;
- *sum* will include in the final results file the sum of all values for the parameter; this is typically used for costs;
- *count* will include the number of times the parameter has been evaluated; this is typically used for control parameters such as the trigger count.

Note that a parameter does not need to have a value defined to be able to request a result for it. For example, a *count* result request can be put on the trigger count parameter for a start event to include in the results the number of times its parent process has been started, even if the trigger count parameter has no defined value.

Result requests are set on parameters by using the "Results" field that has an associated menu with check options:




Result requests can also be associated to a process as a whole. In this case, they will include in the results the requested result types for the parameter value for the whole process. Defining result requests on a process is done by selecting one symbol in the process, displaying its simulation properties, and then press the "Parent process params" button in the panel on the right side of the window. This opens the following dialog:



The fields use the same kind of menus as those for parameters on symbols.

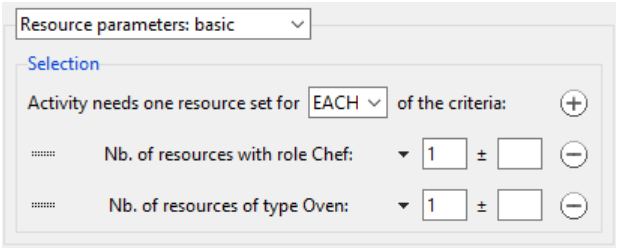
Note that for processes, only the elapsed time is supported since any other kind of time doesn't have any meaning when several branches execute in parallel. The elapsed time included in the results will be the total time spent in the process, considering parallelism: if 2 branches execute in parallel, the time for them in the process will be the time spent in the longest branch, not the sum of the 2 times.

## 8.3.5 Resource parameters

Resource parameters are associated to tasks, and define the resources needed by the task to be able to run. There is actually only one resource parameter, which is a selection expression in the XPATH language (actually a very limited subset of this language). PragmaDev Process supports specifying the resource selection via an XPATH expression, but also offers a simpler way of specifying which resources are needed for a task.

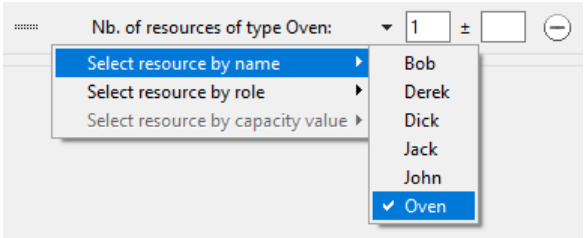
### 8.3.5.1 Basic resource selection

The needs in resources for a task can be expressed via a set of criteria by using the "basic" option in the "Resource parameters" in the simulation parameters for the task:



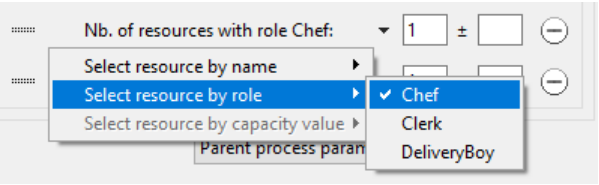

Several lines defining a needed resource can be inserted with the "+" button:

- A specific resource can be selected via its name by using the "Select resource by name" option:



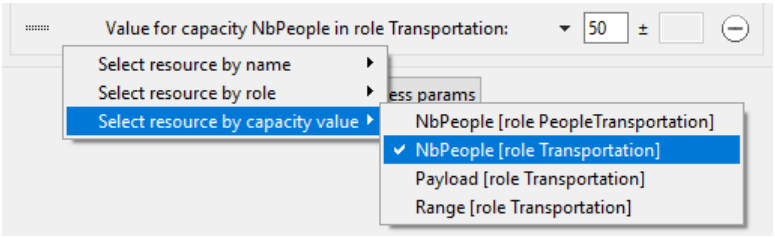

  Once the resource name has been selected the quantity of resources to select can be entered, with an optional deviation. For example, a quantity set to 5 with a deviation of 2 means that the task can run with 3, 4, 5, 6 or 7 of these resources. Note that the actual number of selected resources will have an impact on the running time for the task.

- A resource can be selected by one of its roles. For the definition of roles for resources, see "Resources Editor" on page 157. To do so, use the "Select resource by role" option:




Once the role has been selected, the quantity of resources with this role can be selected the same way as when selecting resources by name.

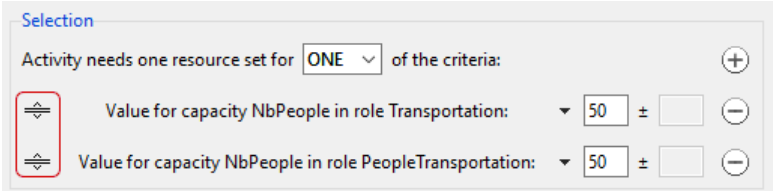
- A resource can be selected by the value for one of its capacities in a given role. For the definition of capacities in roles, see "Resources Editor" on page 157. To do so, use the "Select resource by capacity value" option:




Once the role & capacity have been selected, the value for the capacity can be entered. It is not possible to specify a deviation in this case.

Once lines are entered, they can be removed by using the "-" button on the line. The selector above the list of lines allows to specify how the lines are interpreted:

- If the selector states "Activity needs one resource set for EACH of the criteria", it means that a set of resources matching each of the criteria specified in the lines must be available for the activity to start. For example, if the lines specify one resource with the role "Chef" and one resource with the type "Oven", the activity needs one chef *and* one oven to run.
- If the selector states "Activity needs one resource set for ONE of the criteria", it means that a set of resources matching one of the criteria specified in the line must be available for the activity to start. In this case, the order of the line matters: if a resource set is found that matches the criteria on the first line, the activity will start with these resources; if not, and if a resource set is found that matches the criteria on the second line, the activity will start with these resources. And so on, until the last line is analyzed. Since the order matters, the lines can be reordered by using the handles on the right side of the line:




### 8.3.5.2 Advanced resource selection (XPATH expression)

The selection of resources can also be expressed via a selection expression using a subset of the XPATH language, with specific functions for actual resource selection. The

standard BPSim functions are supported, as well as extended versions of these functions and specific functions allowing criteria for resource selection that do not exist in BPSim.

**Resource selection functions** In a resource selection expression, the criteria for resources are expressed via functions. There are 2 basic functions that are standardized in BPSim, and 3 PragmaDev extensions:

- `bpsim:getResource` allows to select a resource by its name, and specifies the quantity of this resource the activity needs. For example:

  ```
  bpsim:getResource('Foo', 1)
  ```

  specifies that the activity needs exactly one `Foo` to be able to run.
- `bpsim:getResourceByRoles` allows to select a resource by one or several role(s) the resource has (see "Resources Editor" on page 157 for more details on resource roles). For example:

  ```
  bpsim:getResourceByRoles(['R1', 'R2'], 3)
  ```

  specifies that 3 resources that have the role `R1` and the role `R2` are needed for the activity to be able to run.
- `pragmadev:getResource` is the same function as `bpsim:getResource`, but accepts a thirs parameter that is a possible deviation for the number of selected resources. For example:

  ```
  pragmadev:getResource('Foo', 5, 2)
  ```

  specifies that the activity needs 5 resources of type `Foo` to be able to start, plus or minus 2. So the activity can start with 3, 4, 5, 6 or 7 resources of type `Foo`. This will have an impact on the execution time of the activity: the ones specified in the BPSim time parameters are those for the average number of resources (5 in the example). If the activity gets less than the average, it will take longer to execute; if it gets more, it will take shorter.
- `pragmadev:getResourceByRoles` is the same function as `bpsim:getResourceByRoles` with the same additional deviation parameter. The deviation has the same effect on the execution time as the one in `pragmadev:getResource`.
- `pragmadev:getResourceByCapacities` is a PragmaDev extension using the capacities defined for the roles and their values set for the resources with that role (see "Resources Editor" on page 157). This function takes as arguments a list of triplets (role name, capacity name, capacity value), each defining a constraint on the value of a capacity for the resources to select. The semantics of the constraint depends on whether the capacity is additive or not:
  - If the capacity is not additive, all select resources must have a value for this capacity that is greater than or equal to the value given in the constraints.
  - If the capacity is additive, the sum of the capacity values of all selected resources must be greater than or equal to the value given in the constraint.

  For example, with the roles `PeopleTransporation` with an additive capacity `nbPeople` and the role `Transportation` with a non-additive capacity `range` (expressed in

km), an activity needing to transport 50 people on 100 km could express its needs for resources with:

```
pragmadev:getResourceByCapacities([
    ( 'PeopleTransportation', 'nbPeople', 50 ),
    ( 'Transportation', 'range', 100 )
])
```

This means that the resources to select must have the sum of their value for `nbPeople` greater than or equal to 50, and each of them must have a value for the `range` capacity greater than or equal to 100.

**Composition of resource selection** Resource selections can be composed in two ways:

- To specify that two sets of resources must both be available, the standard "`union`" XPATH operator is used. For example:

  ```
  bpsim:getResource('Foo', 1) union bpsim:getResource('Bar', 2)
  ```

  specifies that the activity needs one `Foo` and two `Bar`'s to be able to run. The `union` operator can also be written as "`|`".
- To specify that only one set of resources among a list is needed for the activity to run, the standard BPSim function `orResource` can be used. For example:

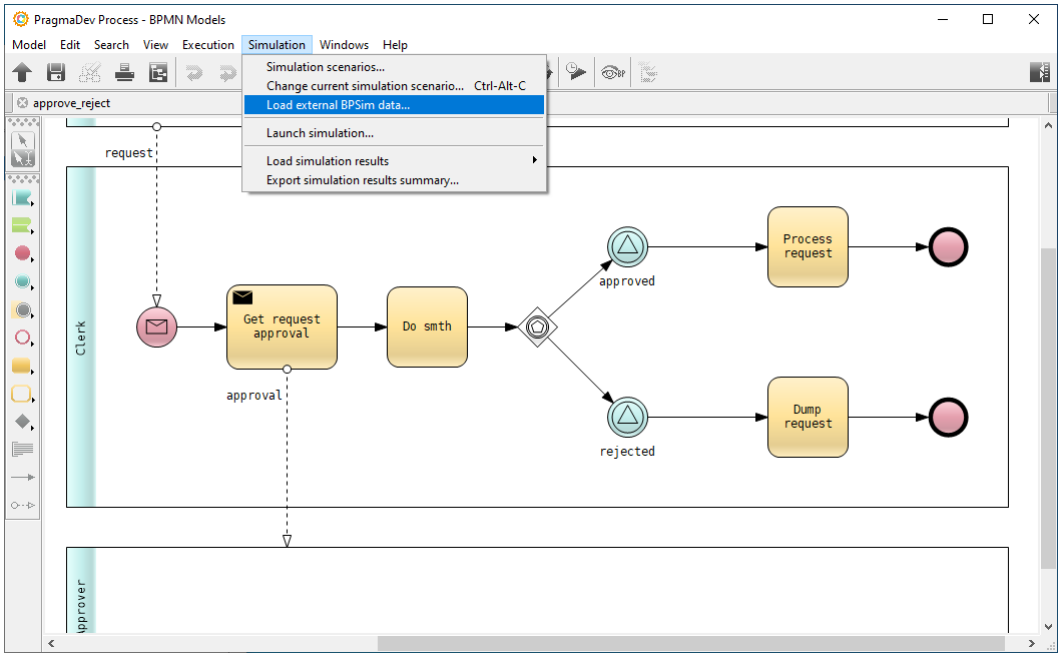
  ```
  bpsim:orResource([
      bpsim:getResource('Foo', 1),
      bpsim:getResource('Bar', 2)
  ])
  ```

  means that if one `Foo` is available, the activity can start with it; if it's not and two `Bar`'s are available, then the activity can start using the two `Bar`'s; otherwise, the activity waits until either one `Foo` or two `Bar`'s become available.

Compositions can be nested: an operand for the `union` or `|` operator can be a call to `bpsim:orResource`, and a parameter of `bpsim:orResource` can be a union.

## 8.4 External BPSim data

The usual way to define BPSim parameters associated to the elements in the BPMN model is via the editor, as explained above. However, there are cases where the BPSim parameters associated to the elements in the model is already defined in an external file containing BPSim data only. In such a case, it is possible to load this data by using the entry "Load external BPSim data..." in the "Simulation" menu:



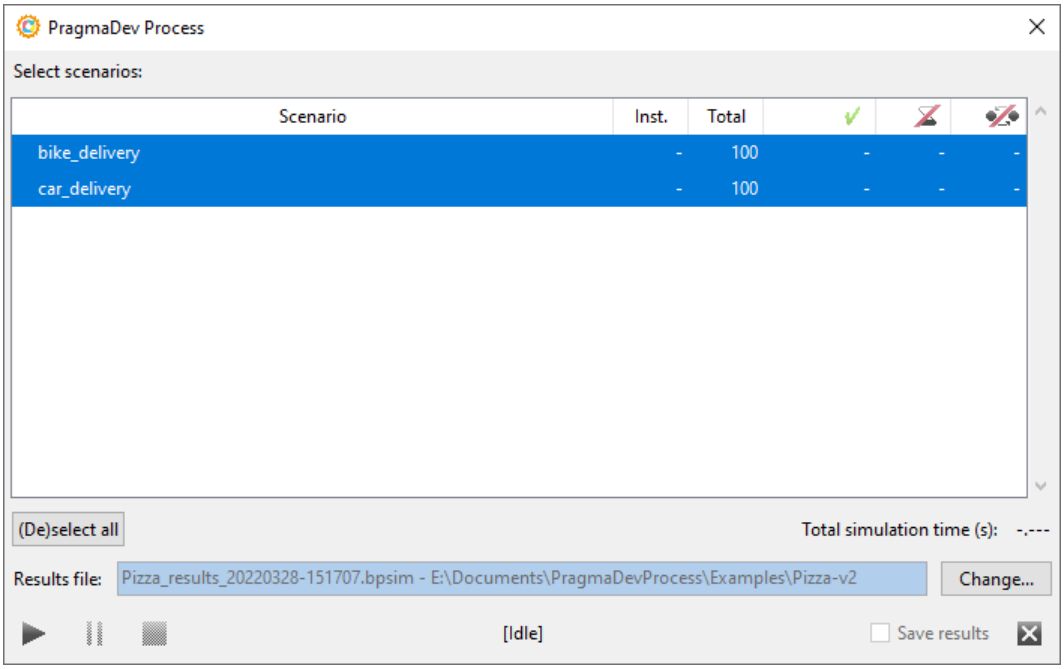

This will display a standard "open file" dialog allowing to select the BPSim data to load. The default extension for the file is ".bpsim".

Note that the verification of the consistency between the loaded BPSim data and the elements in the model is minimal, so make sure that the BPSim parameters described in the external file were actually created for the current BPMN model.

## 8.5 Running a simulation

### 8.5.1 Via the graphical user interface

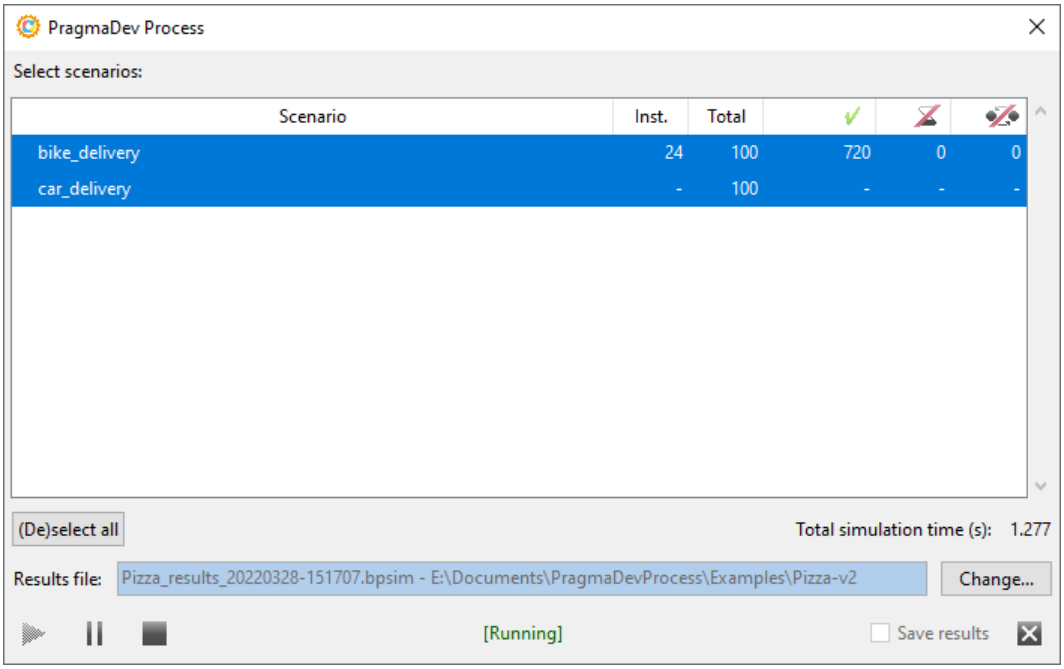
Within PragmaDev Process, running a simulation is done by selecting "Launch simulation..." in the "Simulation" menu of the BPMN model editor, or simply by clicking on the button. This opens the simulation dialog:




The table lists the available simulation scenarios with their total number of replications. By default, all scenarios are selected; if only some of the scenarios must be considered for the simulation, they can be selected or deselected by control-clicking on them, and ranges can be selected with shift-click. The "(De)select all" button can also be used to select all scenarios or none of them.

The "Results file" field contains the name for the results file that will be produced. A default value is computed automatically, but it can be changed via the "Change..." button.

Once everything is setup, the simulation can be run by clicking the ▶ button. While the simulation runs, the current instance number as well as the number of terminated processes are updated in the dialog:



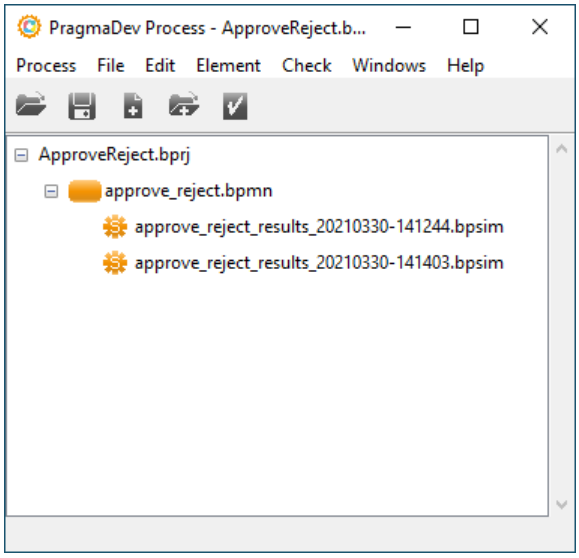

The number of terminated processes are displayed in the 3 last columns:

- The one with the green check-mark displays the number of processes that terminated normally (i.e when they reach all their end symbols);
- The next one displays the number of processes that had to be aborted because the simulation reached its end, as specified in the *duration* scenario attribute;
- The last one displays the number of processes that had to be aborted because of a deadlock: they didn't reach all their end symbols, but at some point, nothing could be done to continue their execution.

Note that only the processes that terminated normally are taken into account for the final results; the other ones are considered as not significant since they never reached their normal end.

While running, the simulation may be paused with the ⏸ button, or aborted with the ■ button; note that results are not saved for an aborted simulation. Once the simulation is over, the "Save results" checkbox near the close button is automatically checked and enabled. Closing the dialog with this checkbox checked saves the results; if the results should be discarded, the checkbox can be unchecked before closing.

The results will appear in the project manager, as children of the BPMN model:




### 8.5.2 Via the command line interface

Simulating a BPMN model can also be done via a command line. The command line interface for PragmaDev Process uses a single command called `pragmaprocesscommand`, which accepts a sub-command, then the options and arguments for this sub-command. The sub-command for a simulation is `simulate`, so simulating a model via the command line interface is done via:

```
pragmaprocesscommand simulate \
        [ -k ] \
        [ -i ] \
        [ -c <nr or cores> ] \
        [ --itt-samples=<nr or samples> ] \
        [ -s <name>,<name>,... ] \
        [ -o <BPSim results file name> ] \
        [ -p <project file name> ] \
        <BPMN model file name> [ <BPSim data file name> ]
```

The command arguments & options are the following:

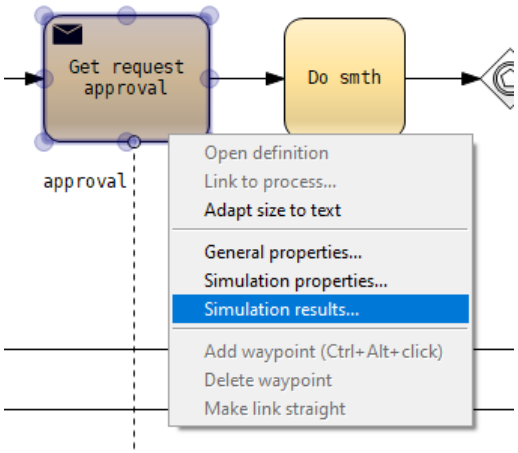
- `<BPMN model file name>` is the name of the model to simulate. It must be present in the command.
- `<BPSim data file name>` is an optional file name for the BPSim data to load along with the BPMN model. If it is not specified, the BPSim data is supposed to be present in the BPMN model file itself; if it is present and there is BPSim data stored in the BPMN model, the BPSim data stored in the model is ignored and the data specified in the external BPSim file is used.
- If the `-s` option is present, it must be followed by a list of BPSim scenario names separated by commas. In this case, only the scenarios with these names will be included in the simulation. If the option is not specified, all scenarios are included.

- If the `-o` option is present, it must be followed by the name of the file where the simulation results must be stored. The preferred extension for such a file is `.bpsim`, and it should be included in the name. If this option is not present, a default results file name will be used, built from the name of the model, the current date and the current time. Note that in all cases, a directory with the same name as the results file without its extension and followed by "`-LOGS`" will also be created if any scenario specifies that logs should be created. This directory will contain the CSV files for the simulation logs.
- If the `-k` option is present, activities will use the same set of resources during their entire lifetime, i.e, from start to finish including suspensions and resumes (see Executor preferences).
- If the `-i` option is present, the simulation paramaters on call-activities will be ingored, hence the simulator will always step-into defined call-activities (see Executor preferences).
- If the `-c` option is present, it must be followed by the number of CPU cores to be used for parallel simulation (see Executor preferences).
- If the `--itt-samples` is present, the value assigned to it will be used to set the number of samples to check for non-zero inter-trigger-timer (see Executor preferences). The default value is 100.

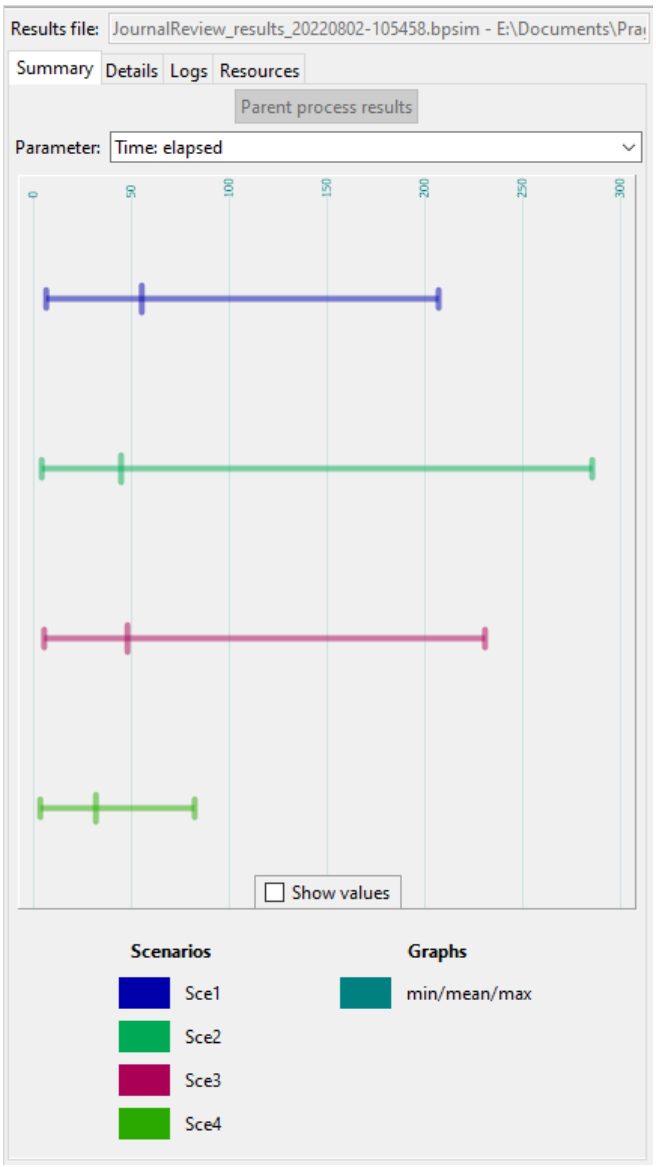
## 8.6 Simulation results

After running a simulation and storing its results in the project, results can be viewed within the BPMN model editor. To load a set of simulation results, either open it from the project manager, or open its parent model and select it in the "Load simulation results" sub-menu of the "Simulation" menu. The BPMN model will be opened, and the selected set of results loaded with it.

To see the results associated to an element, right-click on it and select "Simulation results..." in the contextual menu:



The simulation results will appear in the panel on the right side of the window:




At the top is displayed the name of the loaded simulation results file for reference. Under it are 4 tabs:

- "Summary" displays a summary of the results as a graph;
- "Details" displays the individual results for each instance of all simulated scenarios;
- "Logs" displays the individual logs for each run of each process in all simulated scenarios;
- "Resources" displays the usage of resources for each scenario replication.

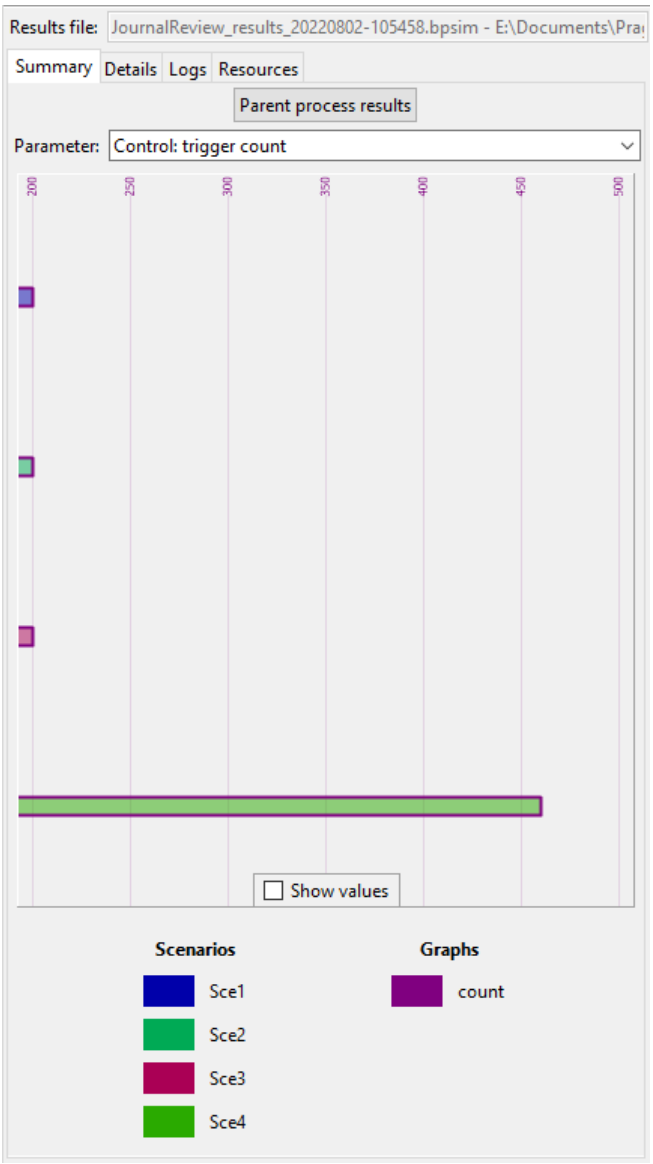
## 8.6.1 Summary

### 8.6.1.1 Graphs

The "Summary" tab displays a summary of the results as a graph. This graph displays the results for all executions of the process within a scenario. For example, the max for the processing time displayed above is the maximum value of the processing time for the selected task among all the times it has been executed in the context of the scenario. The displayed parameter can be selected using the menu for the "Parameter" field.

The graph can be a min/average/max graph as displayed above, which are used when results for some of the parameters include a min, a mean and/or a max.

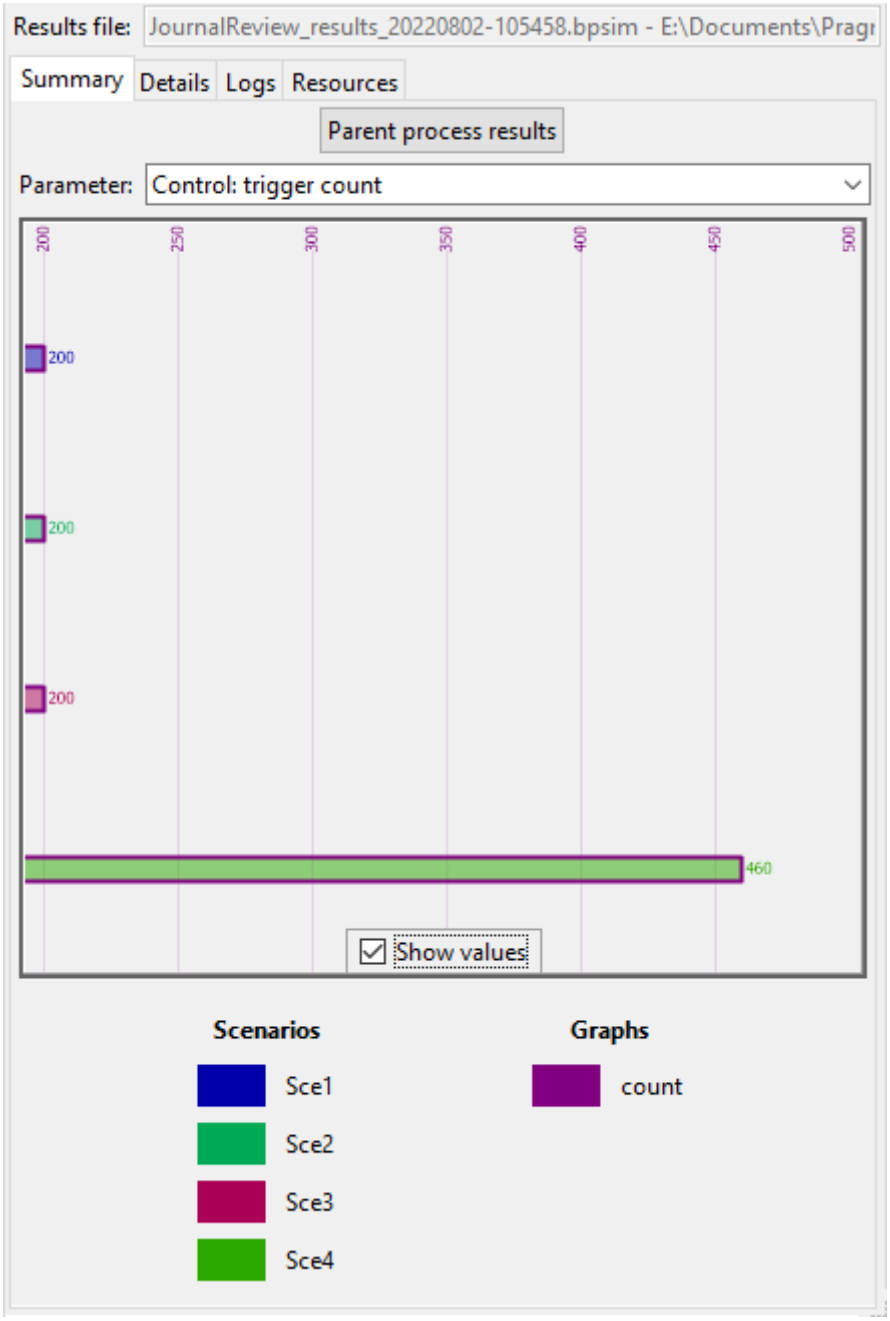
The graph can also be a simple bar graph:




This one is used for simple values such as those for results of type count or sum. Note that for this kind of results, all the results for the scenario are added and the graph

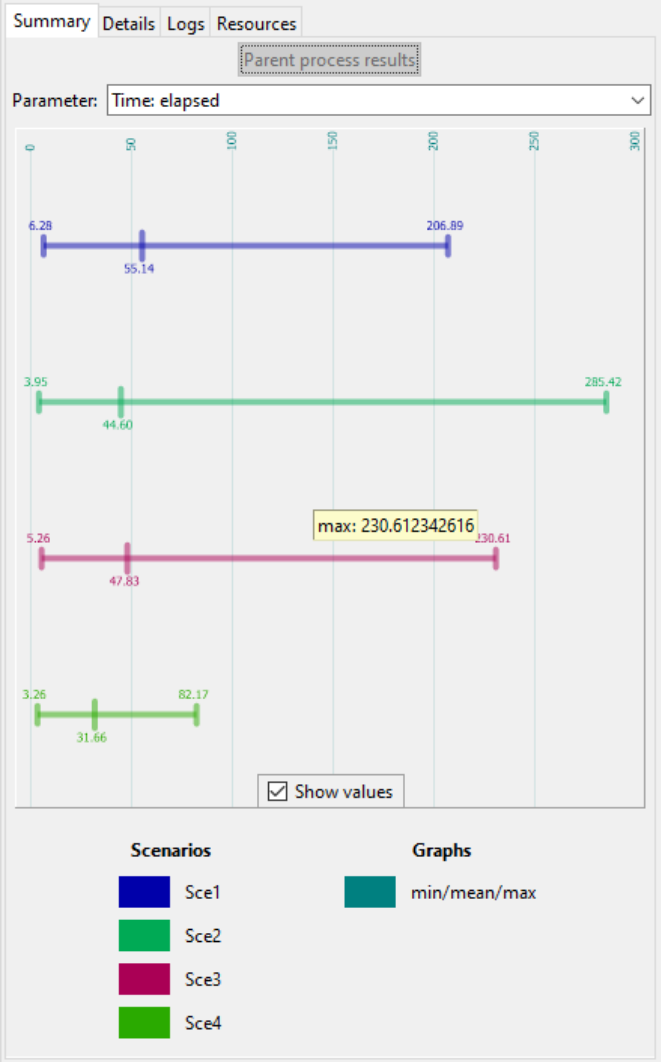
displays the sum.

The checkbox 'Show values' at the bottom of the graph allows to display the values in the graph:




The displayed values may be approximate; the actual values can always be displayed as a tool tip when hovering over the graph:

The button "Parent process results" above the graph allows to switch to the results requested for the selected element's parent process. To make clear that the displayed graph is no more associated to the currently selected element, but to its parent process, all symbols in the process are selected in the editor and the displayed graph is updated:




The results are displayed here with the values shown, and a tool tip showing the exact value when hovering over an end of the graph. Note that the "Parent process results" button is now grayed out. The displayed results are now those requested for the process.

Note that if both the fixed cost and the unit cost are included in the results, a "dummy" parameter named "Cost: TOTAL" will be included in the available parameters to display that will be the sum of the 2 costs.

#### 8.6.1.2 Exporting

The results summary for all elements in the model can be exported to a CSV file, allowing to open it with a spreadsheet application. This is done by selecting the entry "Export simulation results summary..." in the "Simulation" menu while a set of simulation re-

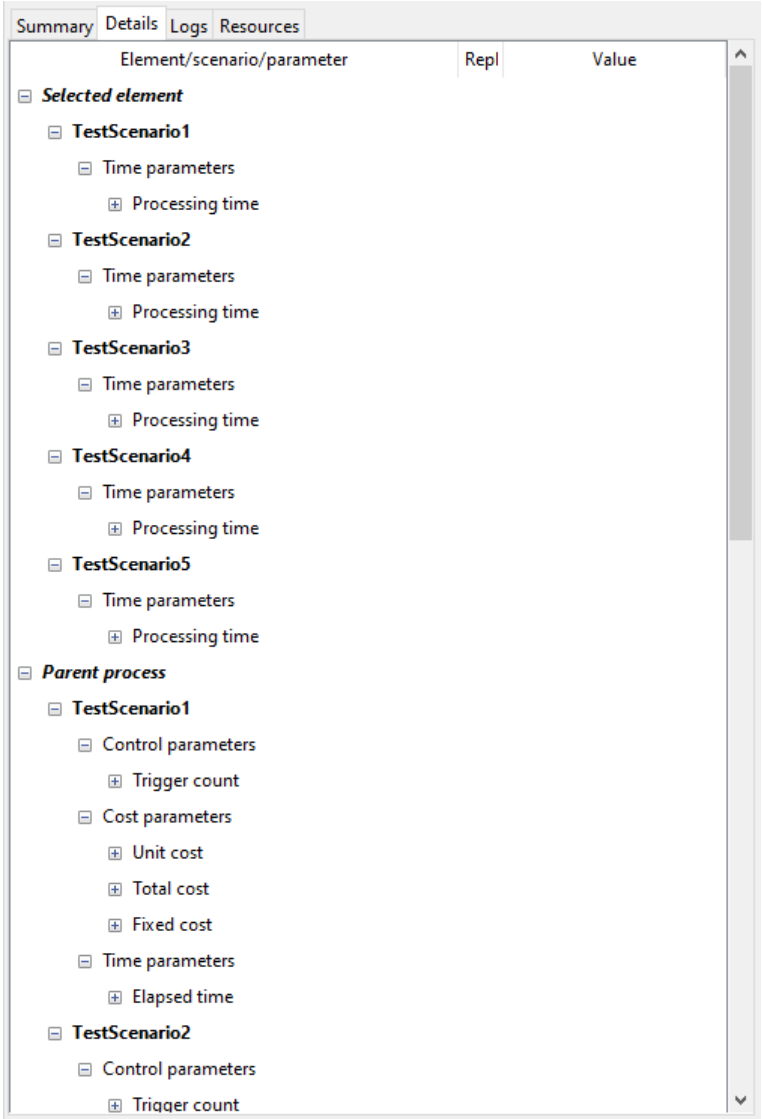
sults is loaded. A dialog will then ask for the CSV file to save to, then export the results summary.

The columns exported in the CSV file are:

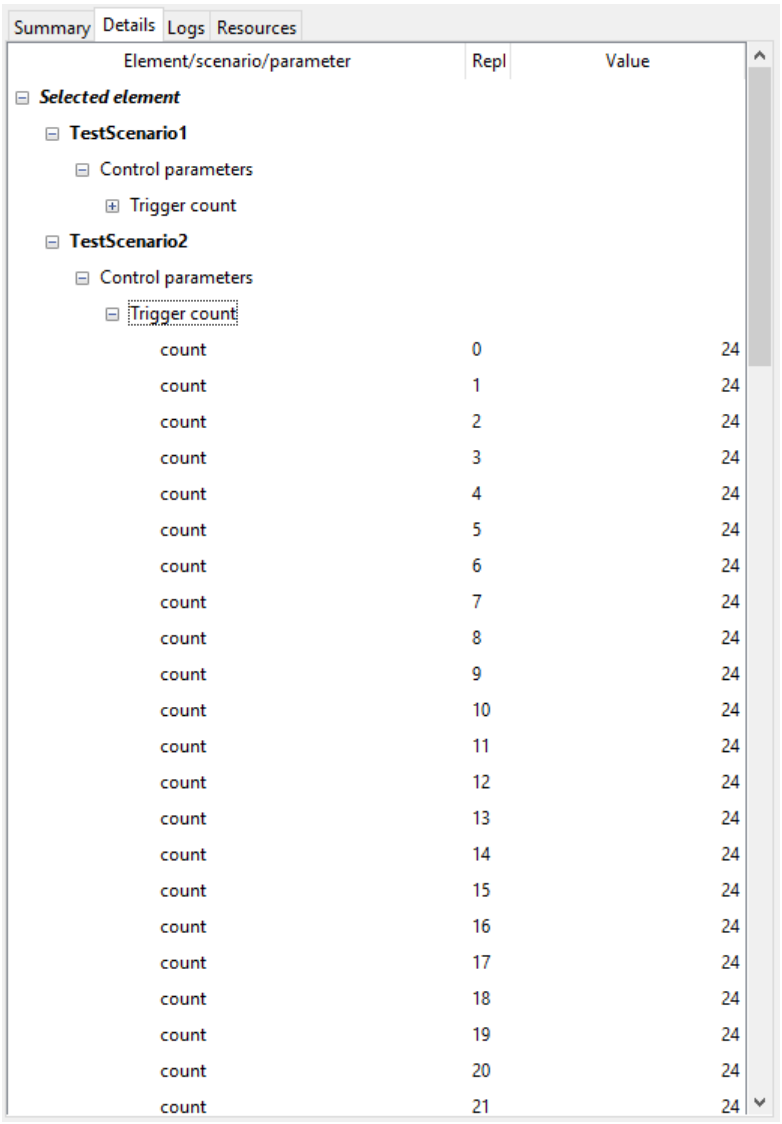
- The internal identifier for the simulation scenario;
- The scenario name;
- The internal identifier for the element having the result;
- The name of the parameter for the result;
- And one column per result kind: minimum value, maximum value, mean (average) value, count (number of times the symbol was hit) & sum of values.

### 8.6.2 Details

The "Details" tab list all results for each instance of all simulated scenarios as a tree:



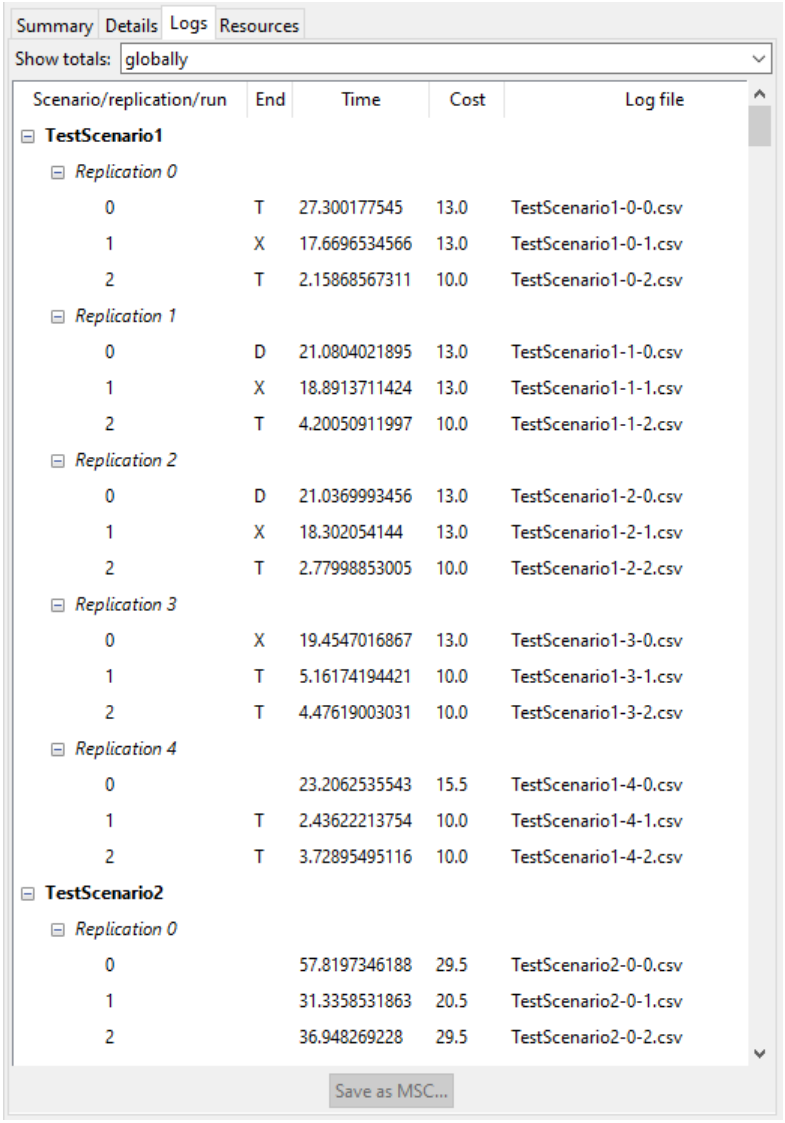
The results are sorted by element first: selected element, and its parent process. Under it are listed the simulated scenarios, then the parameter set (time, cost and/or control parameters), then the parameter. If the parameter level is expanded, all requested results will be displayed:




Here, a result request of type "count" has been set on the "trigger count" parameter for the selected element. The individual counts for each instance of "TestScenario1" are displayed, with the instance number and their value.

### 8.6.3 Logs

The "Logs" tab displays the information about all executions of processes in all instances of all simulated scenarios, with their time and cost information if they were recorded:




There are two modes for the display of the logs: the global mode, and the selection mode. The mode can be changed with the selector at the top of the tab:

- "Show totals: globally" will select the global mode;
- "Show totals: for selected element's parent process" will select the selection mode.

#### 8.6.3.1 Global mode

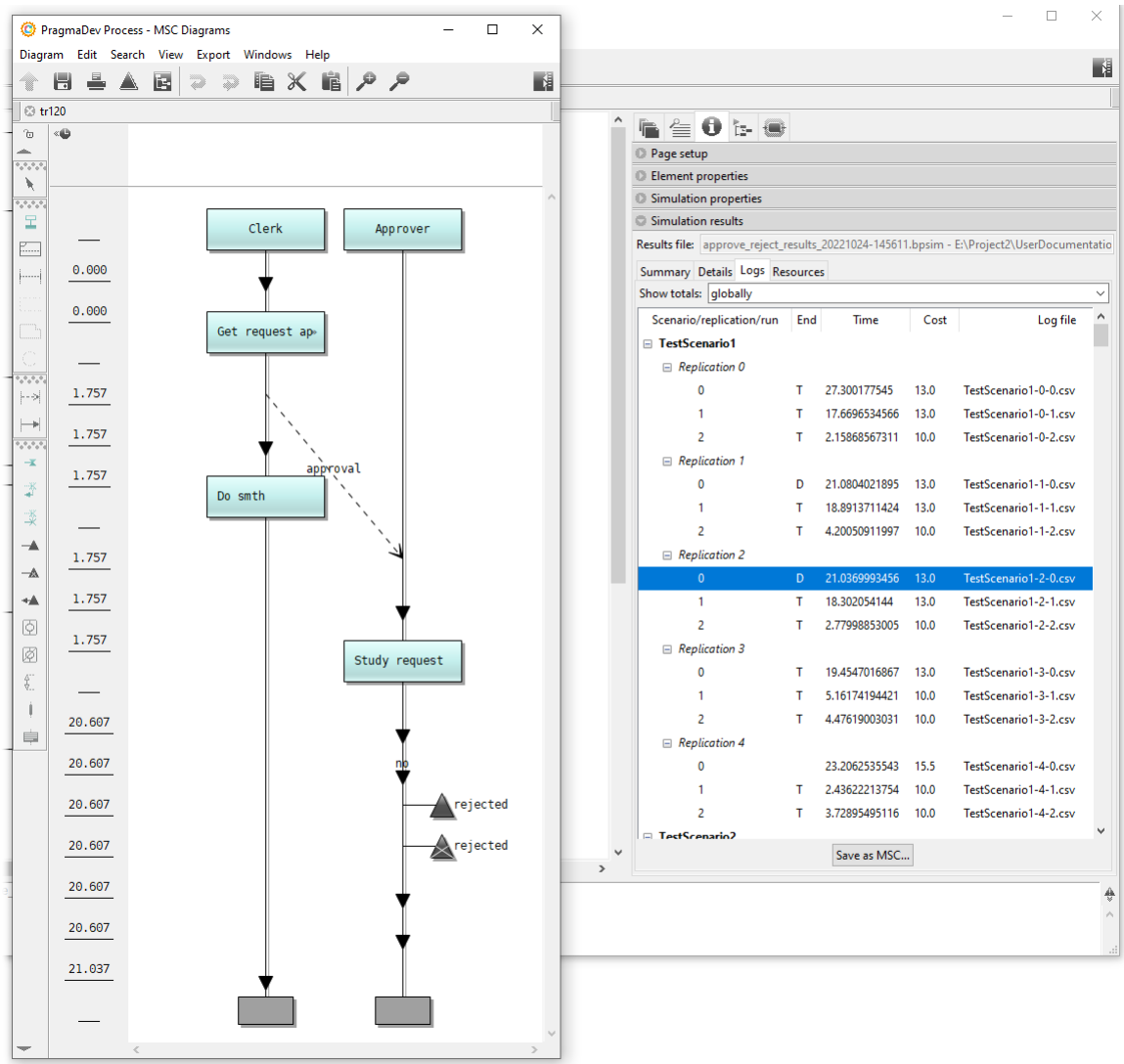
In this mode, the first column gives the scenario name, the instance/replication number, and an index for the run of the processes within the model. The number of runs will depend on the inter-trigger timer and the trigger counts defined on the start symbols, and on the scenario duration if it is defined.

The second column indicates how the processes have terminated:

- If the column is empty, the process has terminated normally;
- If the column contains a T, the processes had to be aborted because of a time-out, typically because the duration of the scenario had been reached;
- If the column contains a D, the processes had to be aborted because one of them ended up in a deadlock.

The third column contains the total time for all the processes, and the fourth the total cost for all the processes.

The fifth & last column contains the actual log file for the process execution. This file allows to generate a MSC trace for this particular execution, either by selecting the line in the tree and clicking the "Generate MSC trace" button, or simply by double-clicking on it. This will ask for a file to save the MSC trace, which will then be opened automatically:

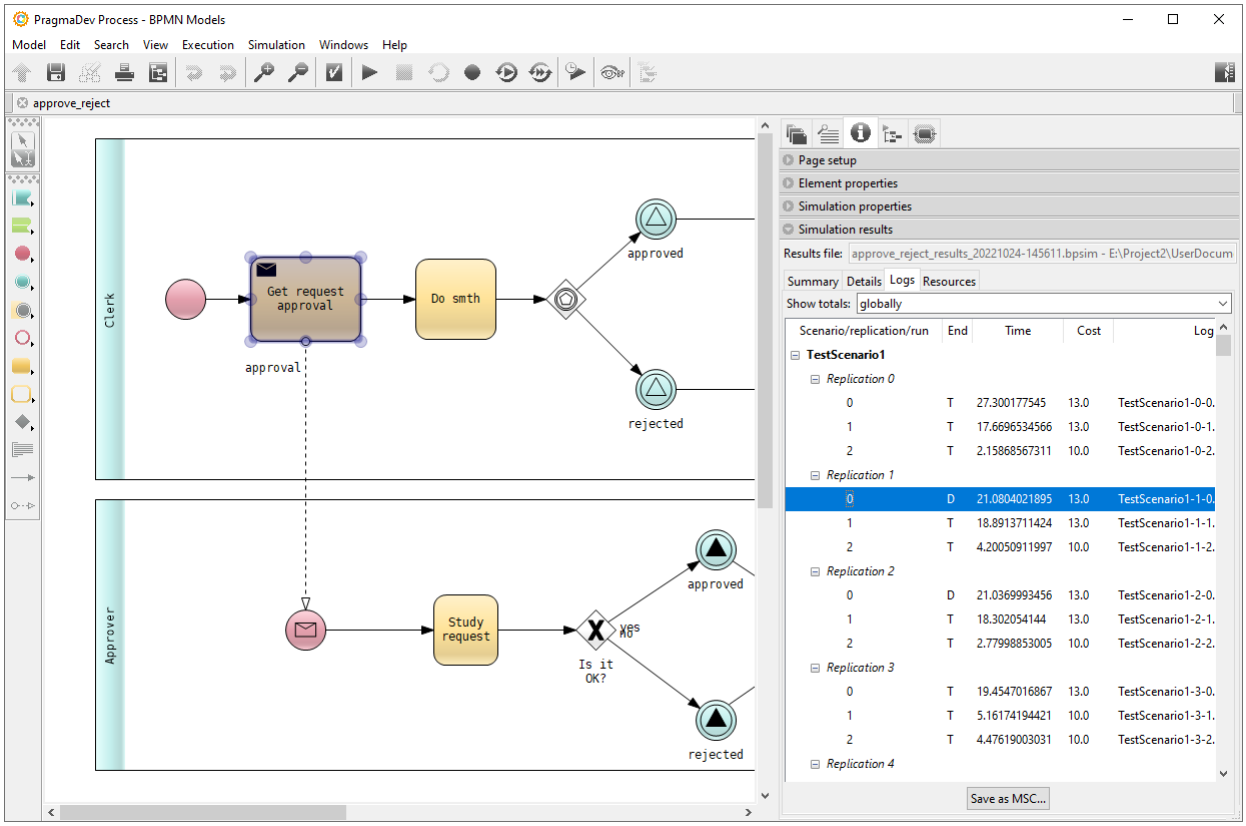



Here, generating the trace for an execution where a deadlock happened actually shows in the trace the deadlock: the "Clerk" process got stuck and didn't go to completion.

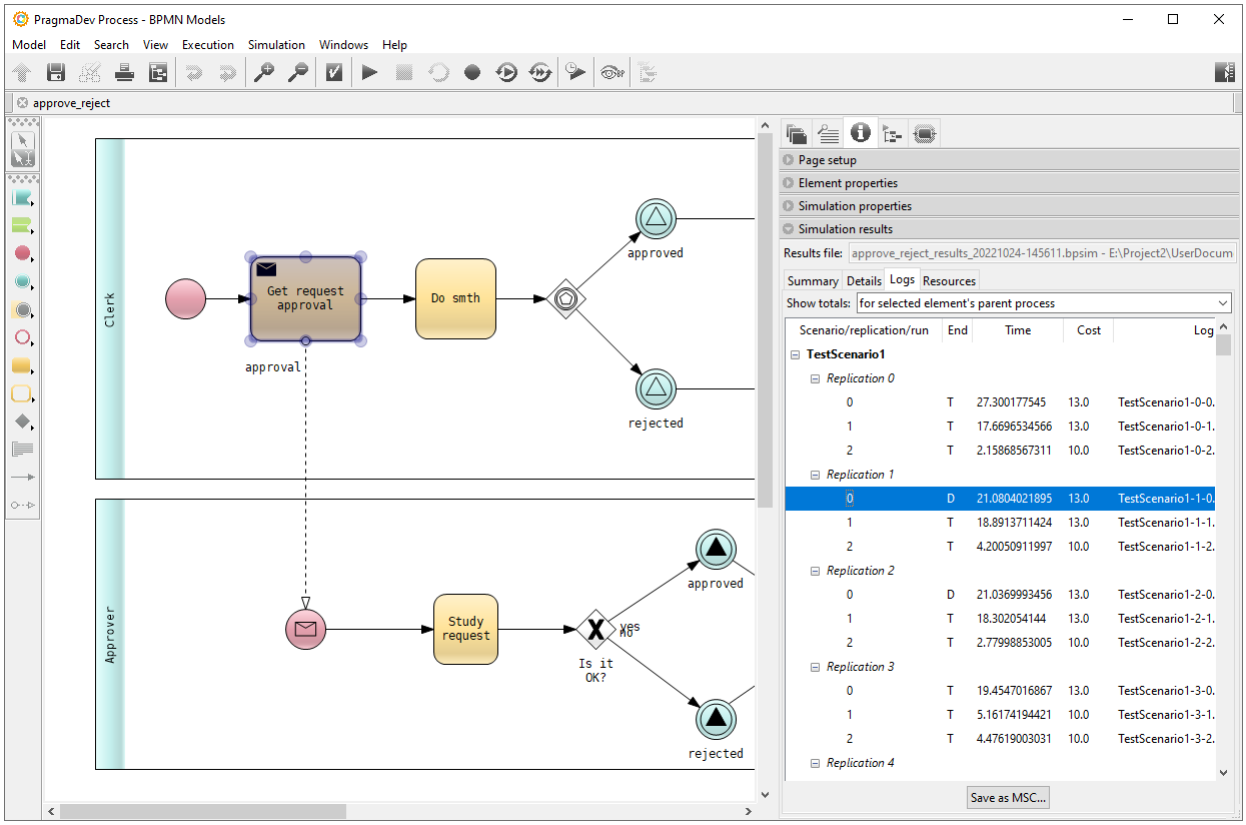
### 8.6.3.2 Selection mode

This mode is mostly identical to the global mode, except the shown end cause, total time and total cost are for the parent process of the selected element only. Note that the times and costs will be available only if they were recorded in the simulation results via a result request on the process.

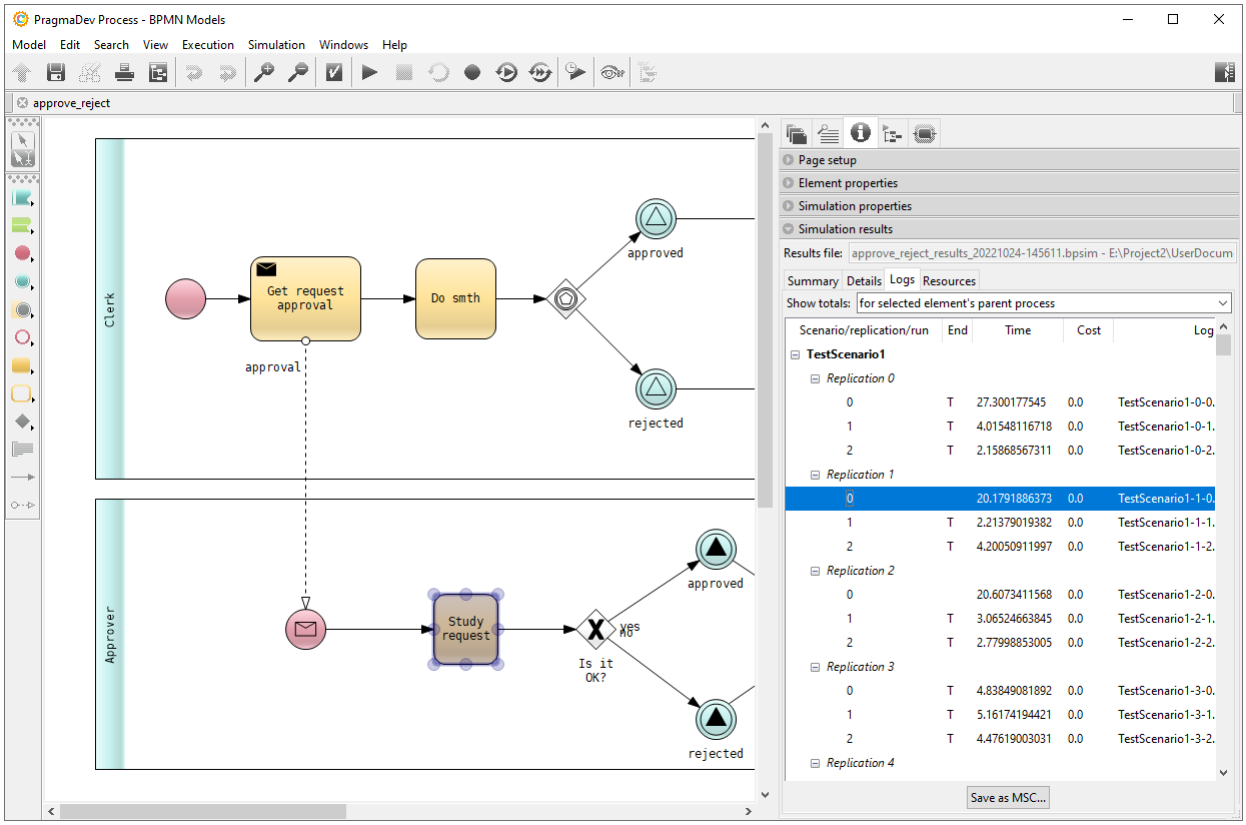
For example, if this log is shown in global mode:




switching to selection mode will update the end cause to the one for the parent process for the "Get request approval" task:




and selecting the "Study request" task will change the displayed information to the one for its parent process:




Note that all information has changed: the end cause is now T, the process time is shorter and its cost 0.

### 8.6.3.3 Log files

When logs are requested for a simulation scenario, all files for the logs will be created in a directory named "<results file name>-LOGS", where <results file name> is the name for the simulation results file without the .bpsim extension, and in the same parent directory as the results file itself.

Two kinds of files are generated, both in CSV format:

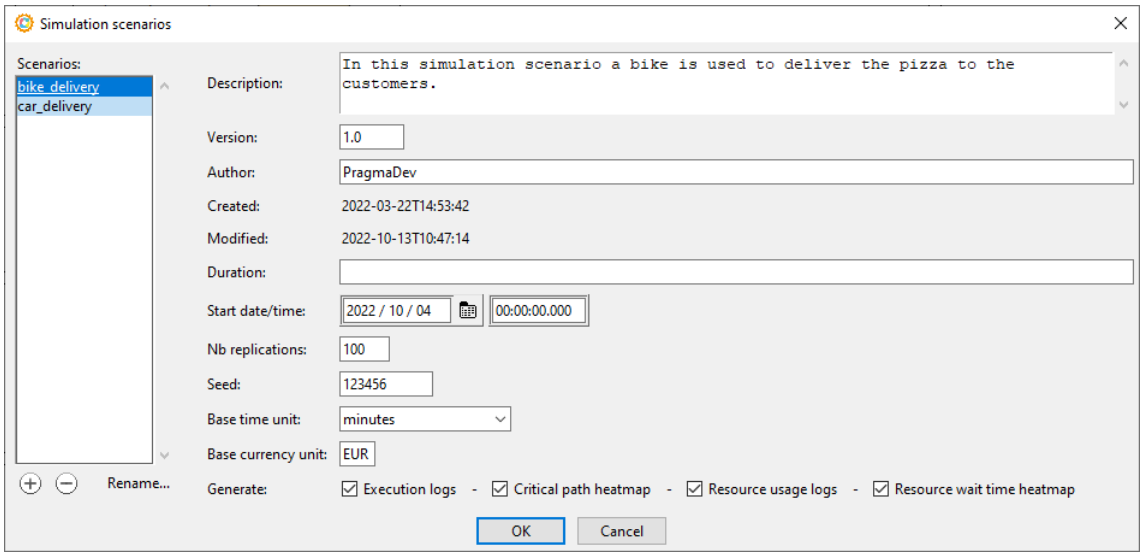
- The logs summary file is always named `logs.csv`. Its columns are, in order:
  - The name for the simulation scenario;
  - The instance/replication number for the scenario;
  - The index for the execution when the inter-trigger timer and trigger count settings on processes indicate that several executions must be made;
  - A space separated list of process identifiers for the end cause indicators;
  - The end cause indicators for the processes in the previous column, in the same order, space-separated too;
  - A space separated list of process identifiers for the process times;
  - The times for the processes in the previous column, in the same order, space-separated too; note that a process will only appear if a result request is set on its elapsed time in the simulation parameters;

  - A space separated list of process identifiers for the process costs;
  - The costs for the processes in the previous column, in the same order, space-separated too; note that a process will only appear if a result request is set on its fixed cost or its unit cost in the simulation parameters;
  - The total time for all processes for this execution; this is the execution time of the longest process. This information is always included, regardless of the results requests defined on the processes;
  - The total cost for all processes for this execution; this is the sum of the costs of all processes. This information is always included, regardless of the results requests defined on the processes;
  - The name of the log file for this execution.
- The log files for individual executions. There are as many of those as there are lines in the summary file. Each line in the file is an event that occurred during the simulation. The columns in such a file are, in order:
  - The timestamp for the event, in ISO-8601 format, including the date & time, with milliseconds;
  - An instance identifier for the diagram; this is to handle the case where the same diagram is executed several times in parallel, typically via a call activity;
  - The identifier for the symbol or link for the event;
  - The identifier for the parent process for the symbol or link; note that this will always be a top-level process: if the symbol or link is in a call activity, the process will be the calling one, not the process called by the call activity;
  - The lag time for the event if any; if this has a non-zero value, the line represents an activity that got suspended because the resources it needed were not available;
  - The duration for the event; if this has a non-zero value, the line represents an activity that was actually running;
  - The cost for the event;
  - The new state for the symbol or link if any.

  Note that all symbols are included, even those on which no state change can occur. For example, a task that changes its state automatically depending on its incoming sequence or message flows will appear in the log anyway, but without a new state.

## 8.6.4 Resource logs

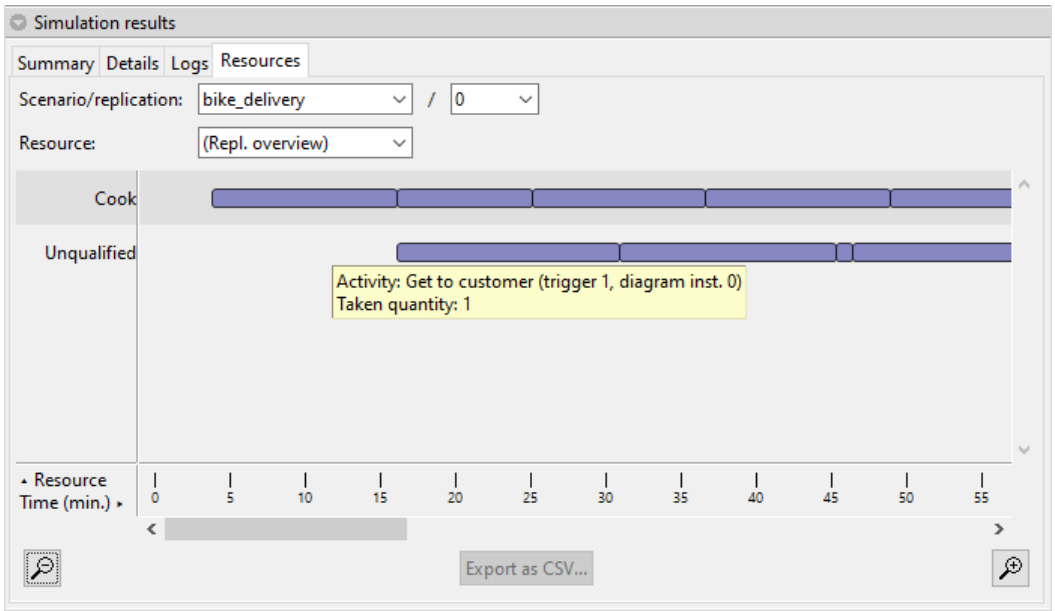
The resource logs contain entries for each time an activity takes or releases resources when it actually starts running, or when it ends. To activate the extraction of these logs, the "Generate" / "Resource usage logs" check box must be checked in the scenario:




When showing the simulation results, these logs are analyzed and displayed in the 4th tab of the "Simulation results" panel. For any replication can be displayed:

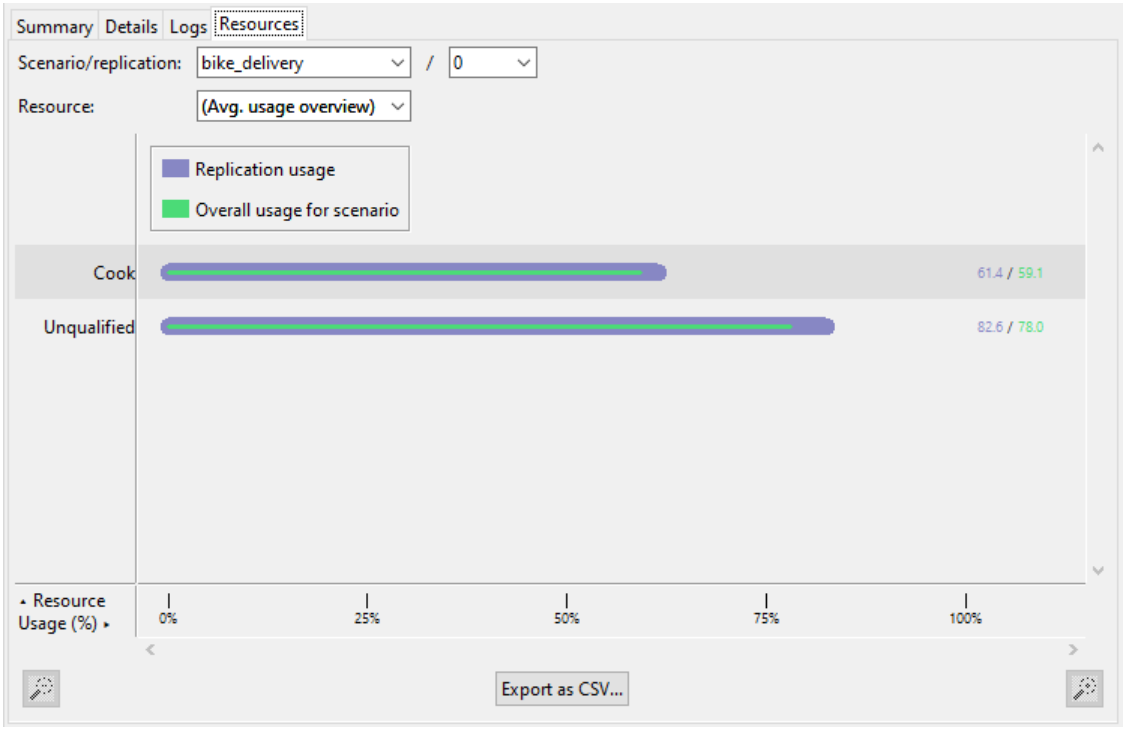
- An overview of the replication in terms of takes of releases of all resources;
- An average usage overview of each resource for the replication and for the scenario as a whole;
- A detailed usage of each resource during the replication.

The replication overview looks like follows:



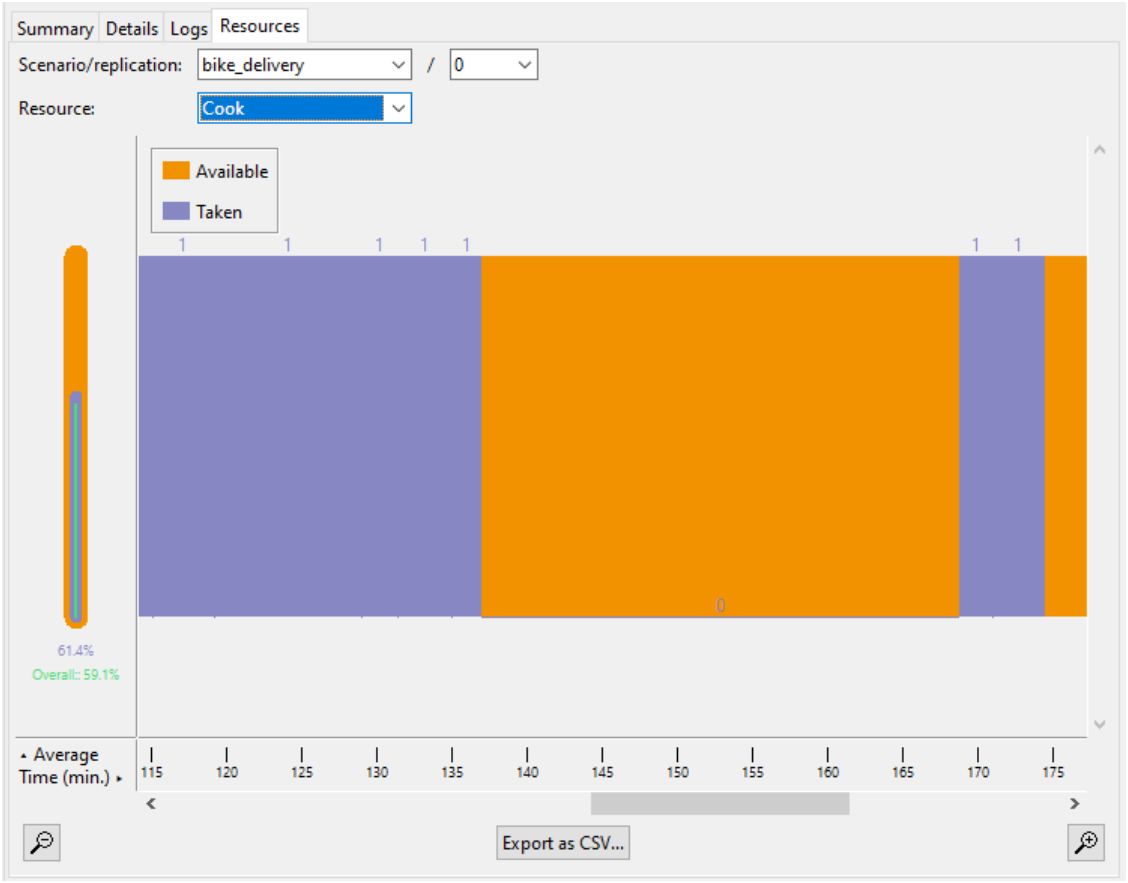
In this graph, the time flows horizontally. For each resource is displayed a bar which describes when the resource is taken and for how long. Hovering with the mouse pointer over a bar opens a tooltip displaying which activity took the resource, for which run and diagram instance, and what quantity was taken.

The average usage overview looks like follows:




For each resource is displayed a bar showing its average usage during the replication (blue bar), and a bar showing its average usage during the whole scenario (green bar). These averages are computed using the number of taken resources, the time during which they were taken and the number of available resources during the duration of the replication or scenario.

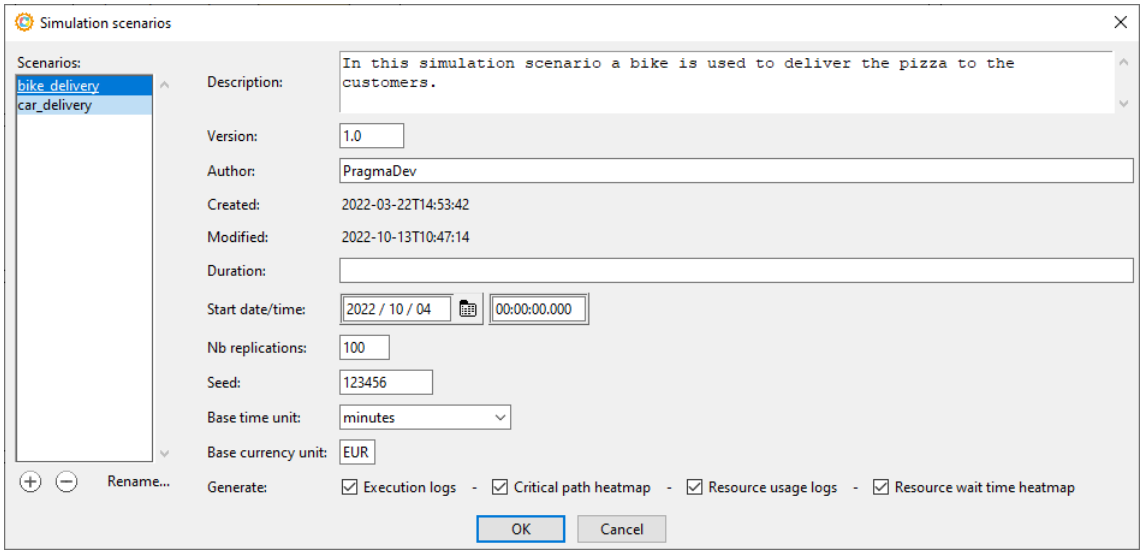
The single resource usage graph looks like follows:



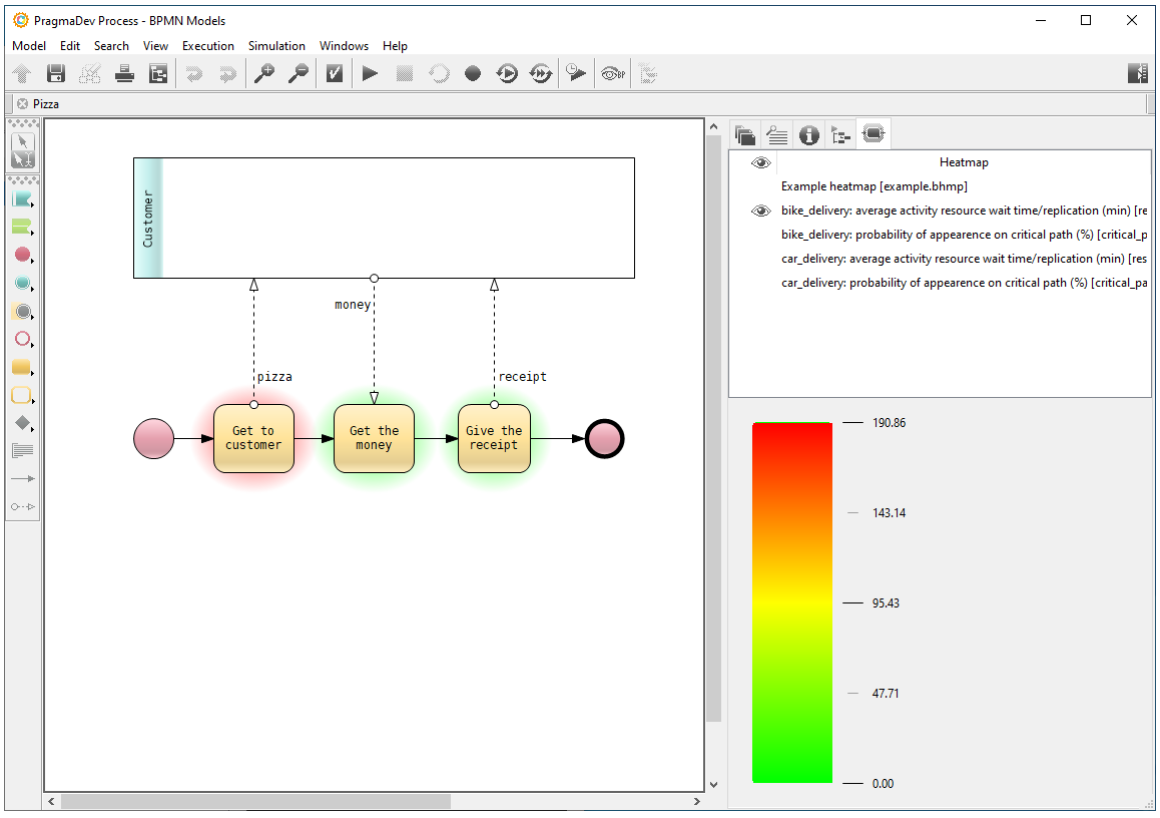

The orange bars show the number of available resources of this kind all along the duration of the replication. The blue bars show the number of taken resources of this kind. Time flows hrizontally. On the left side of the graph is displayed the overall usage for the resource for the replication and for the scenario as a whole. This bar is the same as the one appearing for the resource in the averag usage overview.

### 8.6.5 Resource wait time heatmap

If the execution logs and the resource usage logs are extracted during a simulation, PragmaDev Process is also able to generate a heat map showing the activities that have waited the longest for the resources before they were able to actually start. This is enabled by checking the "Generate" / "Resource wait time heatmap" check box in the scenario:




When this option is checked and the results for the simulation are opened, an additional heatmap will appear in the heatmap browser in the model editor window, named "<*scenario name*>: average activity resource wait time / replication (<*time unit*>)":




The scale in the lower right side shows the colors for the average wait times for each activity, which expressed in the base time unit for the scenario. The "halos" around the activities allow to identify which are the ones that had to wait the longest before they could get the resources allowing them to start. Here, "Get to customer" had to wait the longest (around 190 minutes in average for one replication), while the other tasks hardly waited at all.

# 8.7 Critical path

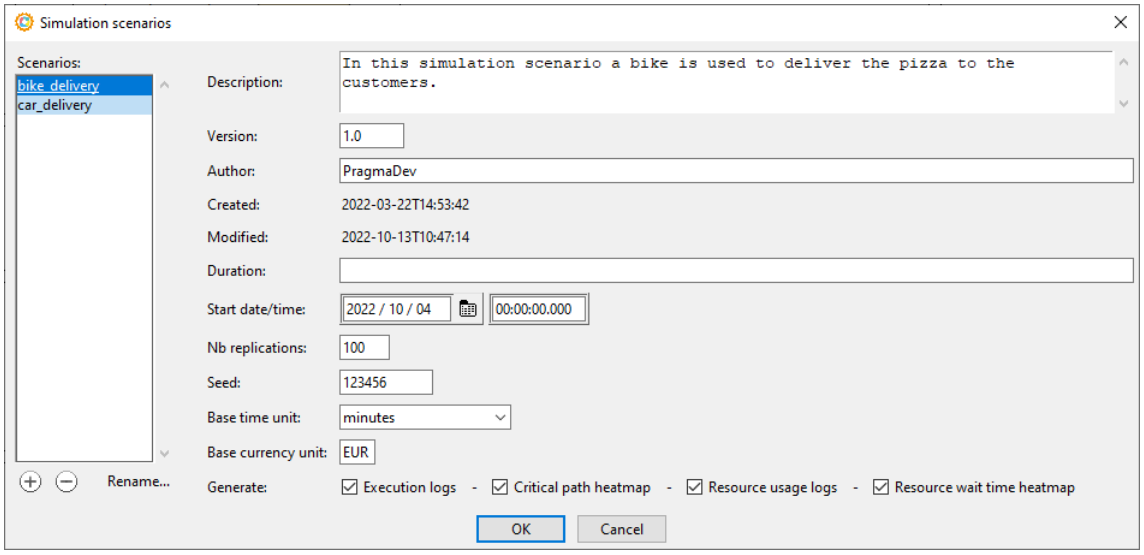
## 8.7.1 Principles

In a business process the critical path is the sequence of activities that led to the actual overall execution time with the least inactivity. The link from one activity to another can be a sequence flow, a message flow, a signal or a timer. The analysis of the critical path starts from the ending symbol. For each activity it finds the previous activity in terms of execution time.

Each replication of a simulation has a different critical path. At the end of the simulation each activity has an associated percentage representing how statistically critical it is.

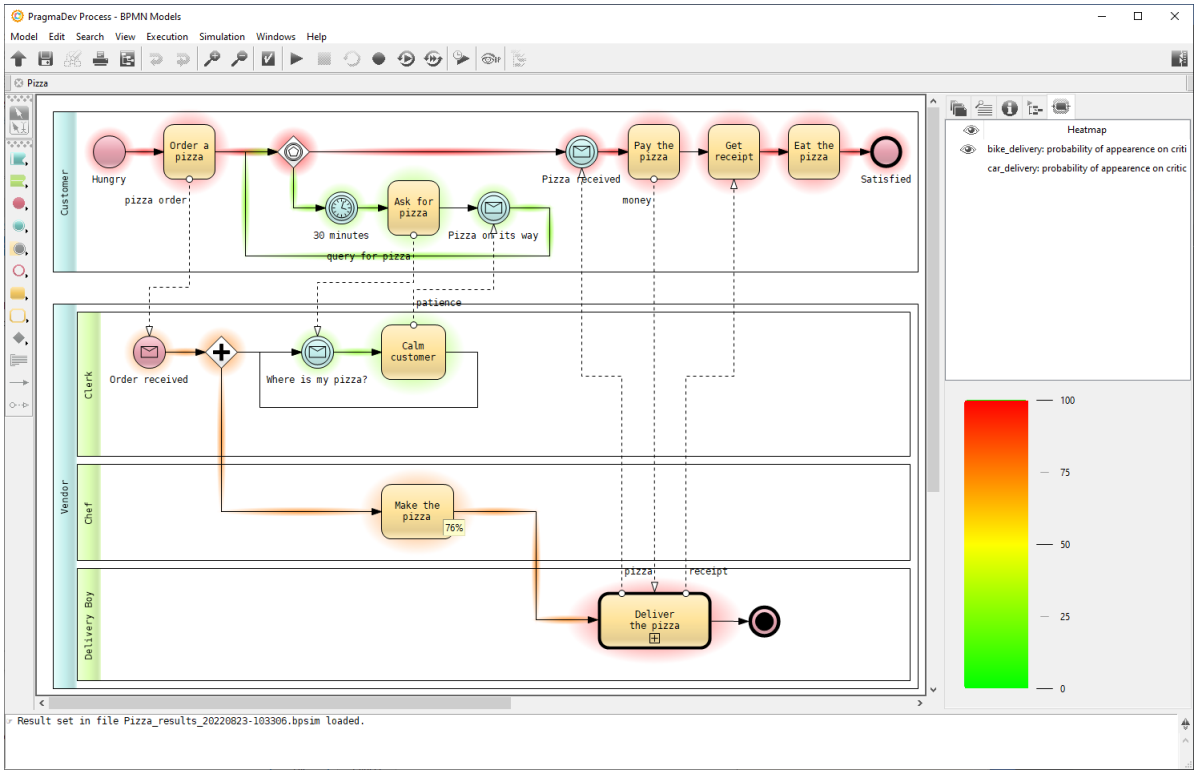
## 8.7.2 Configuration

In the Simulation scenarios window the "Generate" / "Critical path heatmap" checkbox will activate the critical path information during the simulation.




## 8.7.3 Heat map

To display the critical path graphically as a heat map, open the simulation results and a specific diagram. Click on the heat map tab on the right of the window and select the desired scenario.




The color scale is displayed at the bottom right of the window. Each symbol has a colored halo associated to its statistical value. A tool tip, when point to a halo, will show the exact value for the symbol.

# 9 Resources Editor

## 9.1 Overview

Resources in PragmaDev Process are based on the BPMN & BPSim concept of resources, but has been extended significantly. Resources do exist in BPMN but are fairly limited: they have no graphical representation, and their association to activities is very simple: an activity can be declared to need a given quantity of resources to be able to start, and that's all.

BPSim adds to that several features:

- It introduces the notion of role: resources can have roles, and activities can ask not only for a given resource, but also for resources having a given role, or even several roles.
- The link from an activity to its needed resources is much more flexible in BPSim, as it is described via a selection expression, which allows a lot more possibilities than what BPMN offers.
- BPSim allows activities to have several possible sets of resources associated to them, only one of which is needed for the activity to be able to start. This is done via a specific function in the selection expression.
- BPSim offers an advanced mechanism for the availability of the various resources: to each resource can be associated a set of quantities with associated calendars, defining how many of these resources are available at a given date and time.

PragmaDev Process supports everything BPSim introduces, but extends the concepts even further:

- In expression specifying that an activity needs a given quantity of resources, PragmaDev Process allows to specify also a deviation. So for example, if an activity specifies it needs 5 resources of a given type with a deviation of 2, it can actually start with any number of these resources between 3 and 7. This also has an impact on the time spent in the activity: the more resources it can get, the shorter it will be.
- The concept of role is extended by specifying capacities for them. Capacities are characteristics of the role which will have a specific value for each resource having this role. For example, a role PeopleTransportation could have the capacities NbTransportedPeople and Range. Each resource playing the role PeopleTransportation will have a value for these capacities, which will of course be different for each resource: a truck will be able able to transport more people than a helicopter for example, but might have a shorter range.

The selection expression for activities can select resources based on their capacity values. For example, an activity could ask for enough resources to transport a given number of people for a given distance, whatever these resources actually are.

- Resources in BPSim are stored with the BPMN model. This can be a problem, since within a given organization, the resources are likely to be fixed, and shared between all the processes that can occur in this organization. To solve this issue, PragmaDev Process stores the resources in a file separate from the model and the BPSim data, which allows the resource definitions to be shared among any number of models.

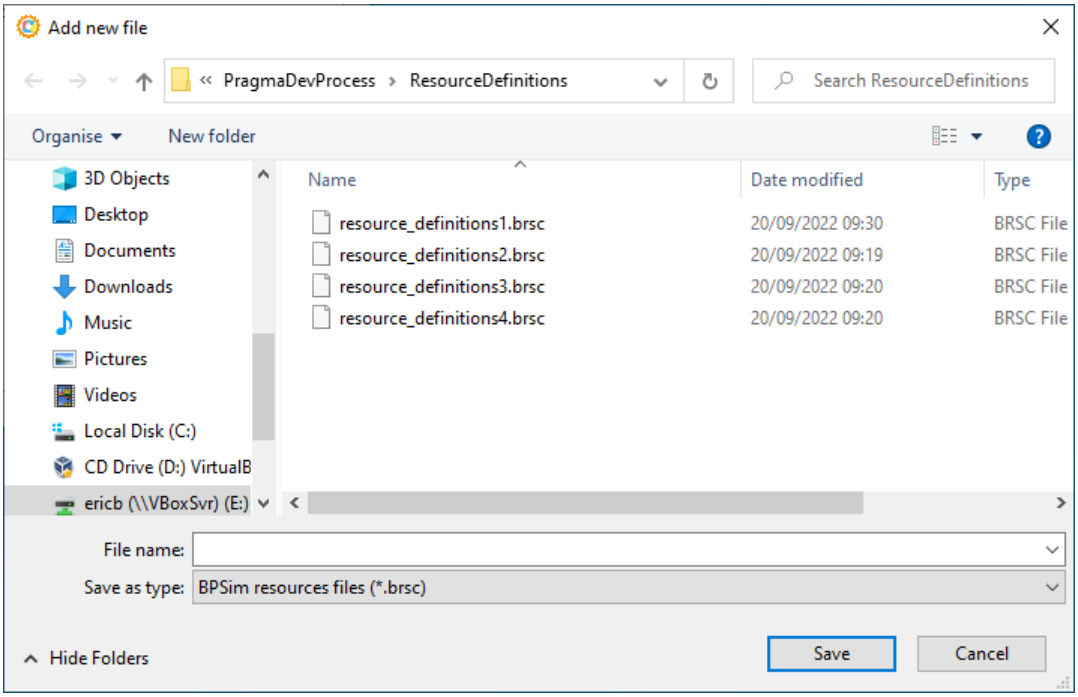
Resources with their roles, capacities and availability are defined in their own file and via a specific editor, that is described in the following sections.

Note that the level of granularity at which resources are described is not enforced by BPMN or BPSim. Some organizations will want to define resources very finely, for example with each person being a resource, and assign roles to them defining what an individual person can do. Other organizations will describe resource much less finely, with a resource being a pool of people who have a given ability. So for example, if we need resources that are software engineers:

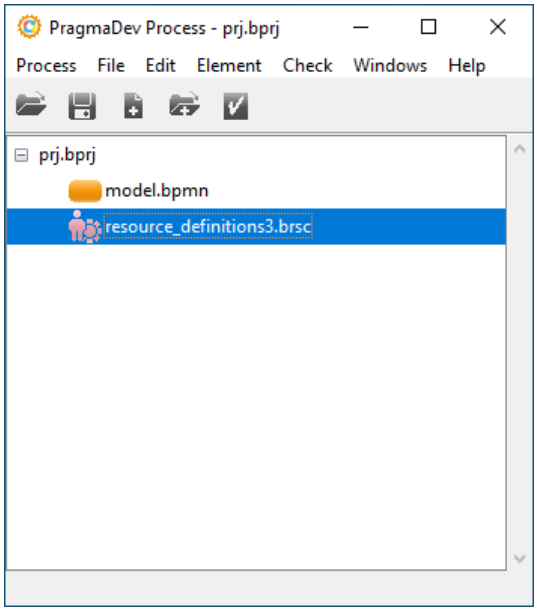
- In the first case, there would be a role named SoftwareEngineer, and there would be resources for individual people like Jack and John, each of them having the role SoftwareEngineer.
- In the second case however, SoftwareEngineer would be a resource, with an associated quantity which is the number of available software engineers in the organization (which can vary over time; see "Resource quantities" on page 162).

## 9.2 Resource definition files

A set of resource definitions can be added to a project the usual way, by clicking either on the button to add a new one, or on the button to add an existing file. A standard file dialog is displayed; the file type to select is "BPSim resource files", with the extension `.brsc`:



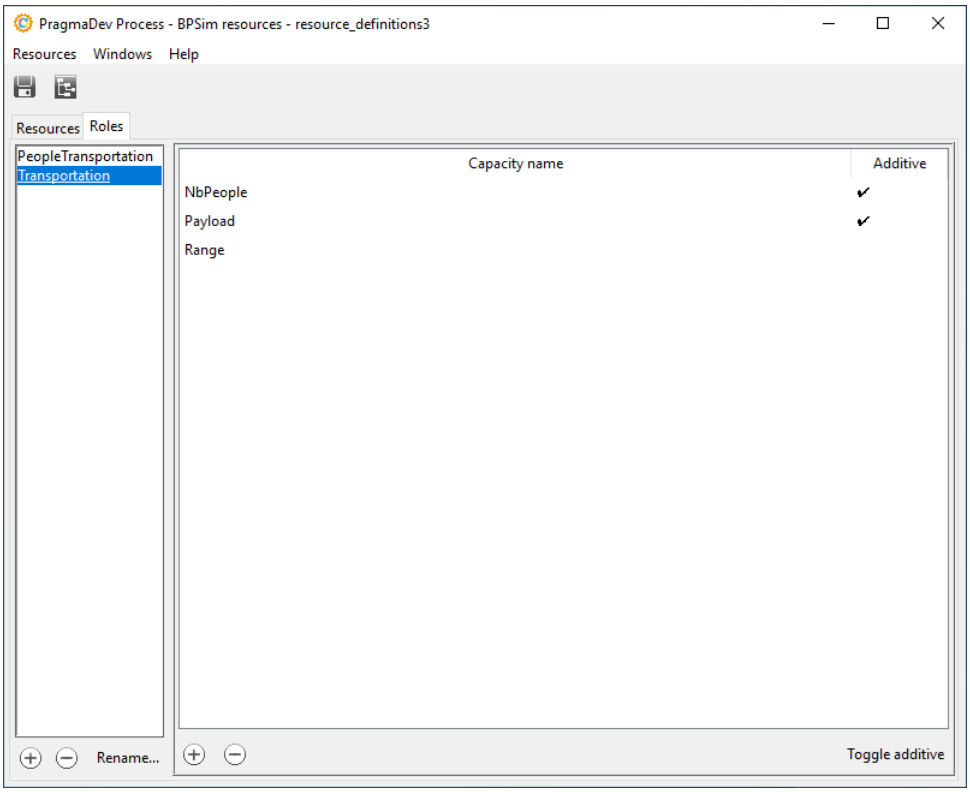

Once selected, the resource definitions file is added to the project:




Double-clicking on the resource definitions file opens the resources editor.

## 9.3 “Roles” tab

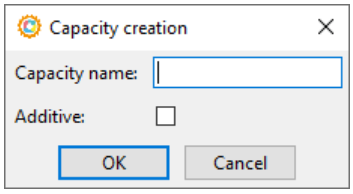
If roles need to be defined for resources, they will be in the second tab of the resource definitions editor:




Here, we have two roles, named "Transportation" and "PeopleTransportation". The list on the left side allow to add new roles via the "+" button, to remove the selected one via the "-" button, or to rename it via the "Rename..." button. Capacities are listed in the zone on the right side.

A capacity can be additive, or non-additive. In the example here, the capacity "NbPeople" is additive: if we get two resources that can transport 50 people, we are able to transport 100 people. But the capacity "Range" is not: if we get two "Transportation" resources with a range of 200 km, the range is still 200 km and not 400.

Capacities can be added via the "+" button under the list. It will display the following dialog:

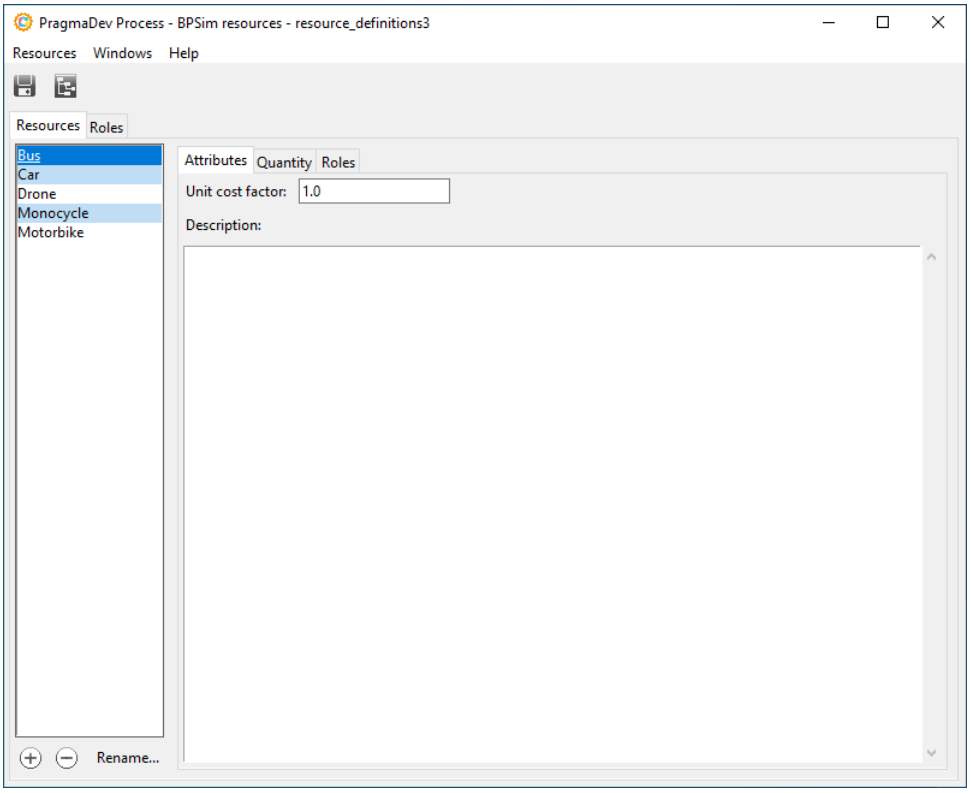



Note that we define only the capacity name here; the values for the capacity will be defined in resources having the corresponding role. Once a capacity has been defined,

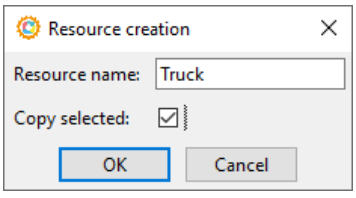
the "Toggle additive" button under the list allows it to be turned into an additive one if it's not, or vice-versa.

## 9.4 "Resources" tab

The first tab in the resources editor allows to define the resource themselves:




The list on the left is the list of resources, with the usual "+" & "-" buttons to add a new one or delete the selected one, and the "Rename..." button allowing to rename the selected resource. The "+" button displays a dialog that also offers to duplicate the selected resource if any:




If a resource is duplicated, all of its attributes, roles, capacity values and quantity rules are duplicated in the new one.

The resource detail zone on the right side has three tabs: attributes, quantity & roles.

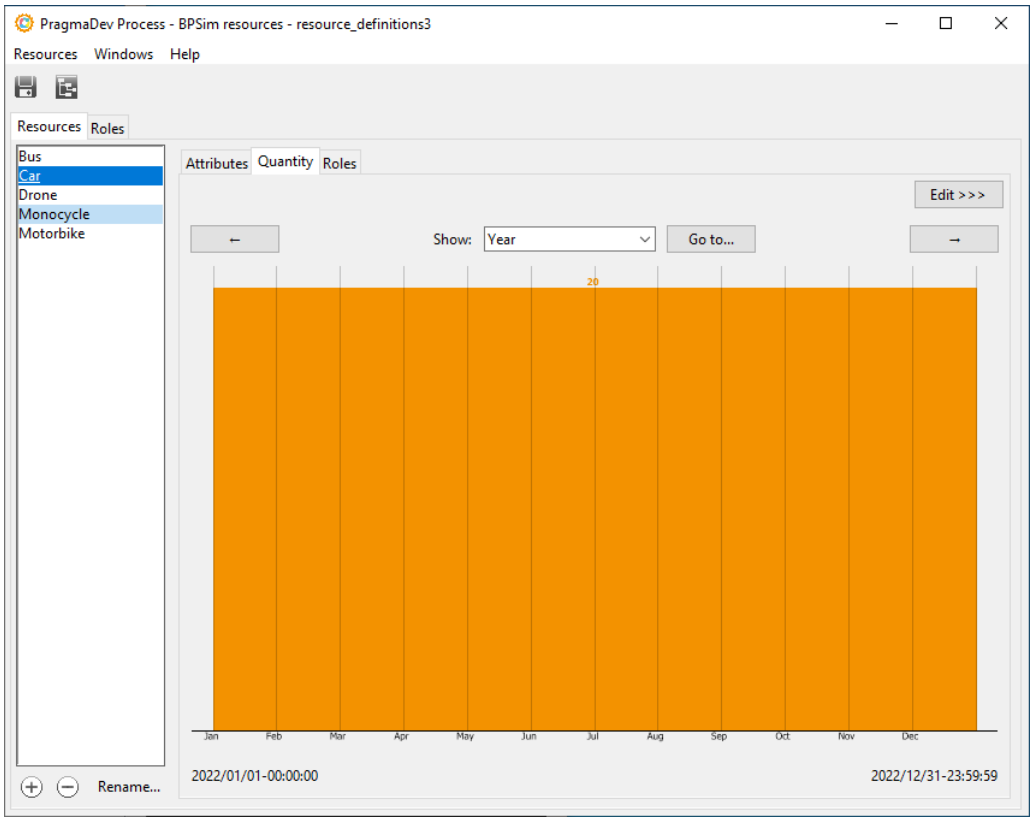
### 9.4.1 Resource attributes

There are two attributes for a resource, as displayed in the screenshot above:

- The "unit cost factor" is the factor to apply to the unit cost of the activity using the resource to get the actual cost. This is to represent the fact that some resource are more costly to use than others. If several resources are used by an activity, the unit cost factor is computed from all the resources the activity uses. So for example, if the activity uses one resource with a unit cost factor of 2, and three resources with a unit cost factor of 0.75, the actual unit cost factor for the activity will be $(1 \times 2 + 3 \times 0.75) / 4 = 1.0625$
- The "description" is just an informal textual description for the resource.

### 9.4.2 Resource quantities

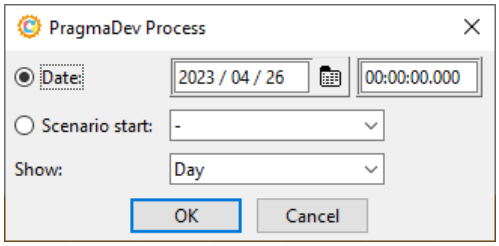
The second tab in the resource details pane allows to display and define the available quantity for the resource, which can vary over time:




The default view shows a graph with the available quantity over time, which flows horizontally from left to right. The extent of the range can be configured with the menu above the graph. Here, the graph displays a full year. To change the displayed range, the following controls are available:

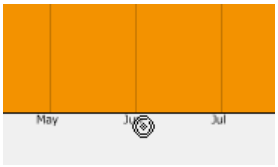
- The "<" and ">" buttons allow to move one range back or forward.

- The "Go to..." button allows to go directly to a date. It displays the following dialog:

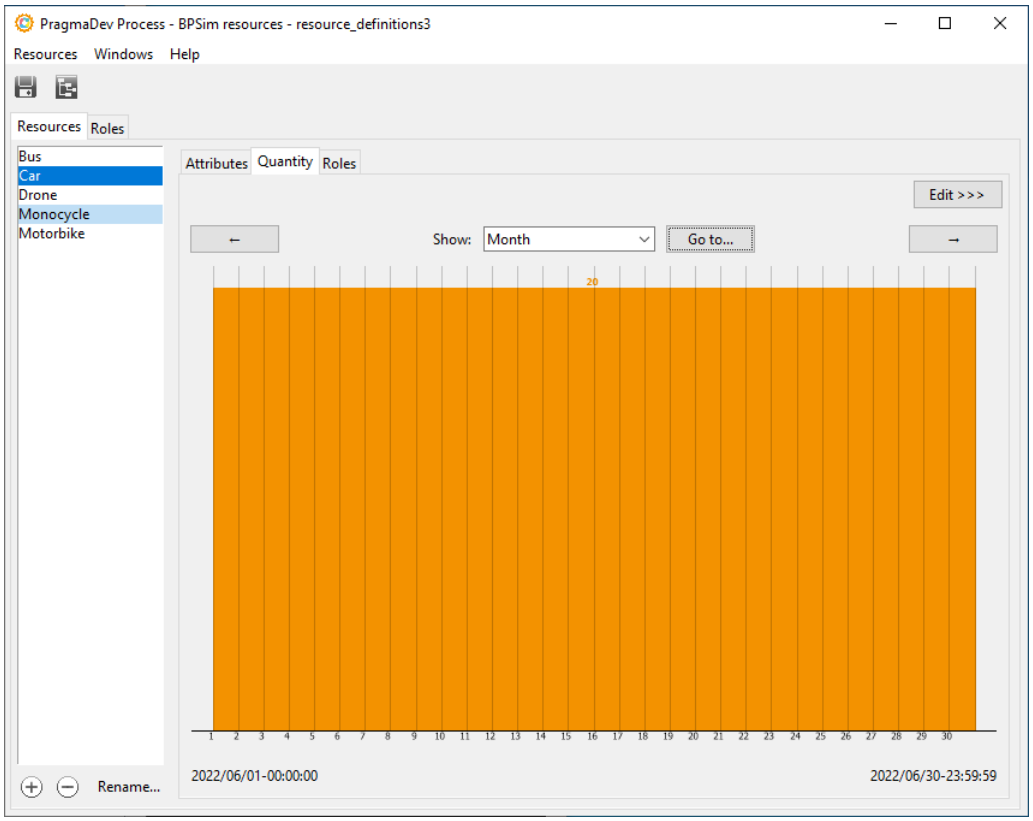



The dialog allows to directly select a date, or to select the start date of any scenario in any BPMN model in the current project via the 'Scenario start' entry. The extent of the range can also be configured here; it is automatically set to the base time unit of the scenario when a scenario start date is selected.

- Any subdivision displayed under the graph can also be "zoomed in": if the mouse pointer hovers over one of these subdivision, it will change to indicate zooming is available, and clicking will directly open the corresponding range. For example, in the graph above, it is possible to zoom on the month of June:

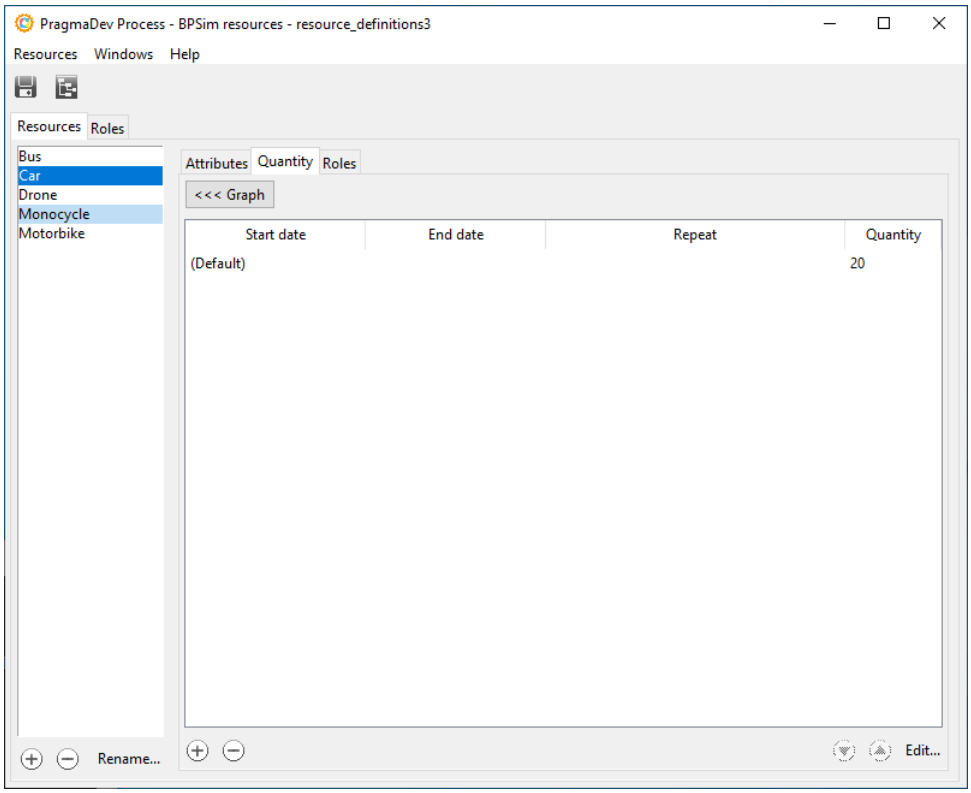



Clicking will "zoom" on the month of June of the displayed year:

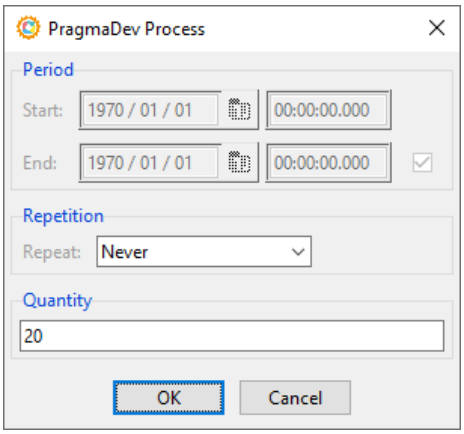



The start and end dates/times for the displayed range are shown under the graph.

Here, the quantity for the resource is 20 and does not vary over the year. To show the rules defining how the quantity evolves or to change them, the "Edit >>>" button switches to the rule list view:

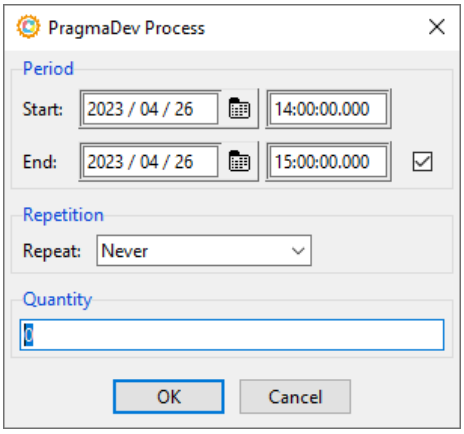



The first rule is always the default one and has no associated calendar. It gives the base quantity for the resource, which can be completed by other rules giving the quantities for various periods of time, defined via a calendar. To modify the base quantity, double-click on the rule, or select it and press the "Edit..." button under the list:



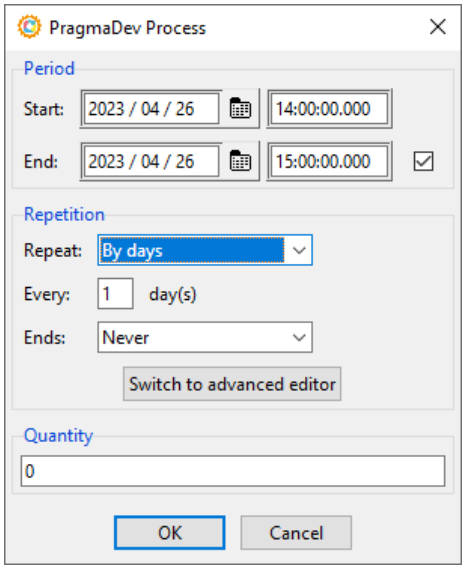

This is the editor used for all quantity rules, but since we're editing the default rule, only the quantity can be changed.

If a new rule is added via the "+" button under the list, the same dialog appear, but with a lot more options:



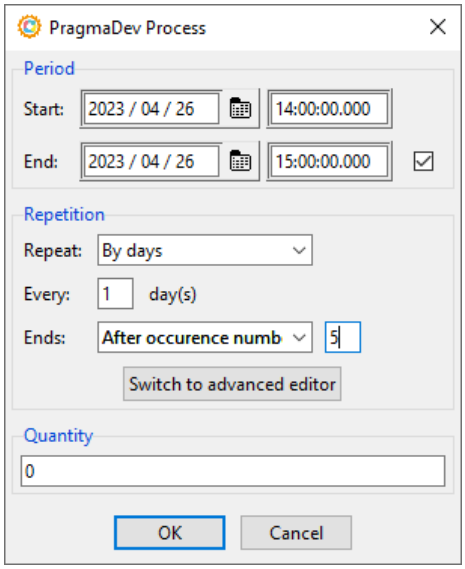

The quantity for the resource appears in the "Quantity" field at the bottom of the dialog. The other fields define when the resource will have this quantity. They are based on the normalized notion of calendar, as defined in RFC 5545 (iCalendar specification - https://icalendar.org/RFC-Specifications/iCalendar-RFC-5545).

- The top part defines the base period for the rule. It can either be a start date/time and an end date/time, or a start date/time only (uncheck the box near the end date), which means that the rule never ends.
- If the base period has a start date, it can be repeated at a given frequency, as specified in the "Repeat" field. The repetition can be by seconds, minutes, hours, days, weeks, months or years.
- The rest of the dialog depends on the chosen frequency:
  - For a repetition by seconds, minutes, hours and days, the dialog looks like follows:



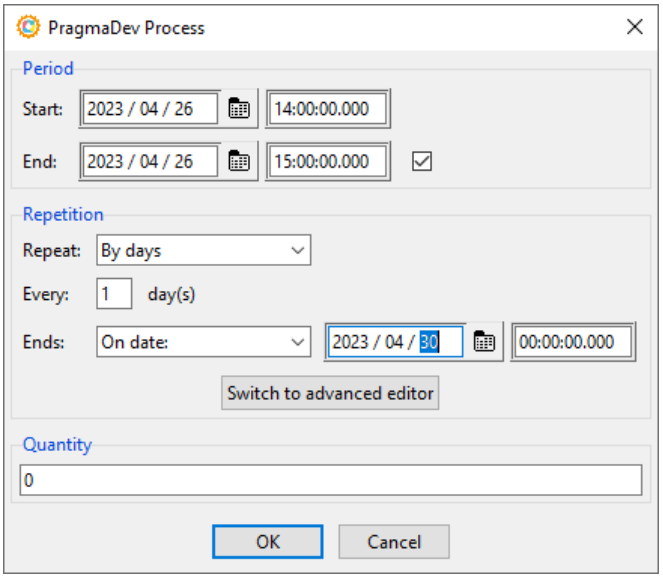

The "Every" field defines the number of time units for the repetition. So a repetition "By days" with "Every" set to 2 will repeat every 2 days. The

"Ends" field defines when the repetition ends. The choices are either "After occurence number", in which case a second field will appear:




This means that the rule will repeat 5 times, then stop repeating.

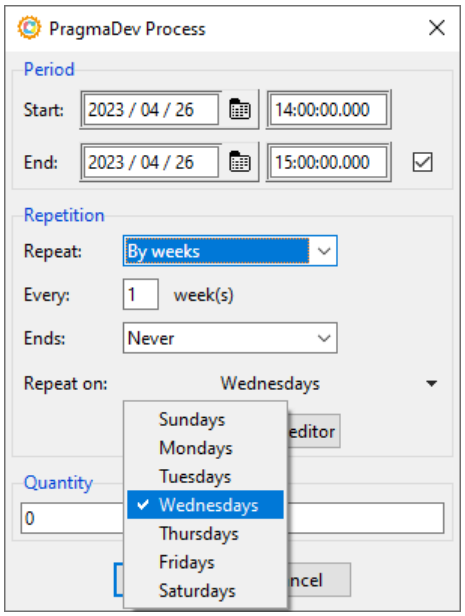
The other choice is "On date", in which case a date/time entry field will appear:




This means that the rule will repeat until it reaches the 31st of October, 2022 at midnight, then stop repeating.

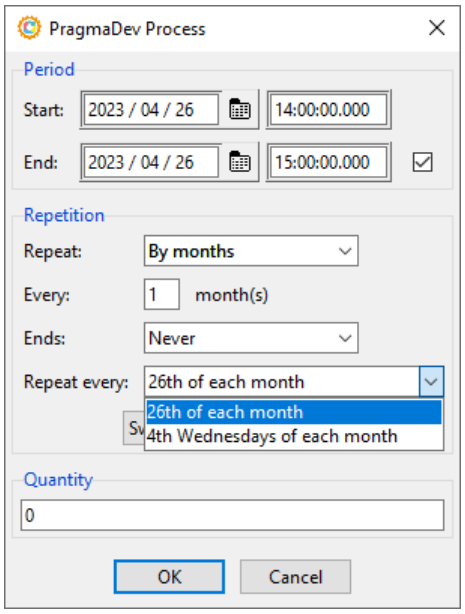
The "Switch to advanced editor" button is described further in this section.

- For a repetition by week, another field will be displayed:



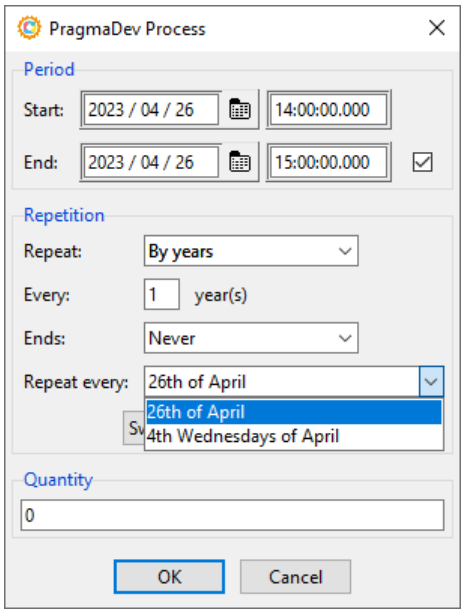

It allows to select on which days of the week the repetition will occur. The default is to repeat only on the week day defined by the start date for the period, but other days can be added.

- For a repetition by months, another field will be added:



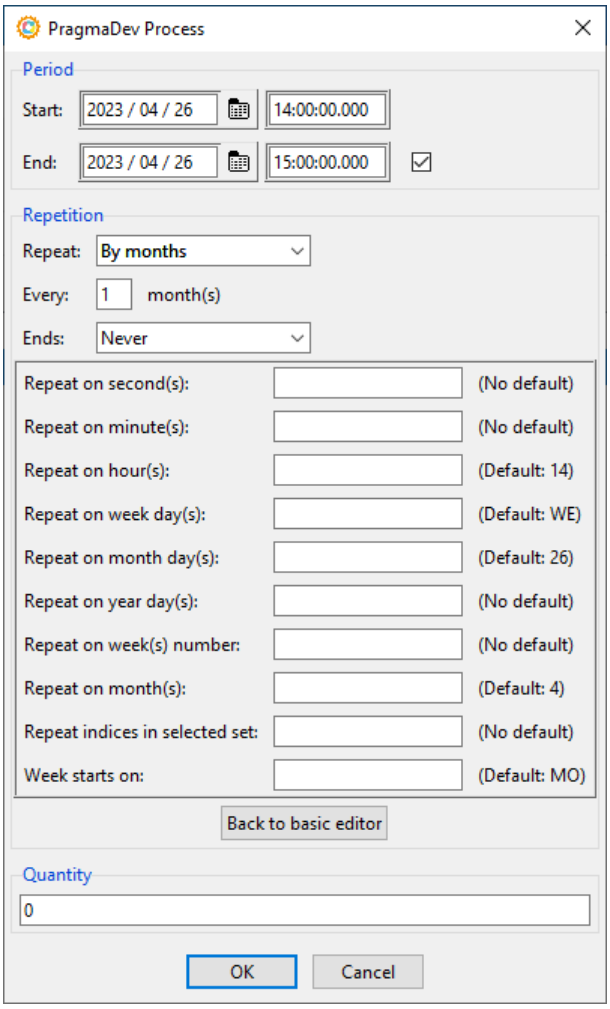

It allows to define if the repetition is based on the day in the month, or on the day of the week within the month.

- For repetition by years, another field will be added:



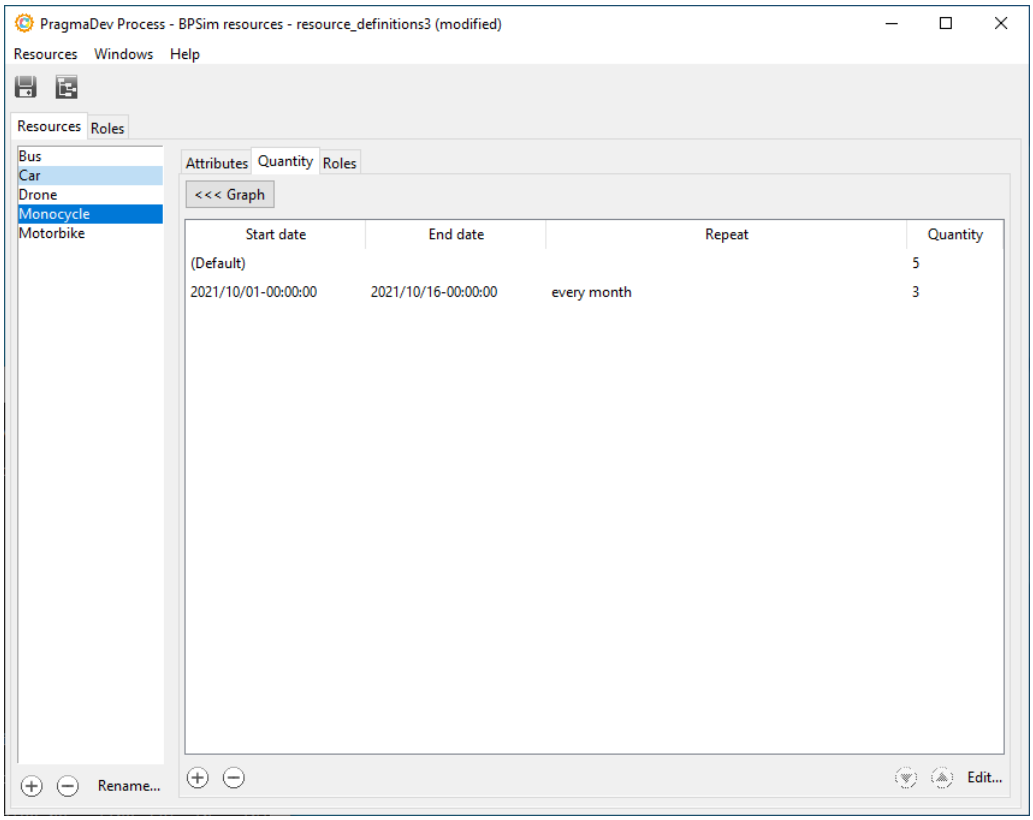

It allows to define if the repetition is based on the day in the month, or on the day in the week within the month.

Whatever the repetition frequency is, there will always be a button "Switch to advanced editor" that will allow to define more precisely how the repetitions will occur. Once clicked, the dialog will look like follows:

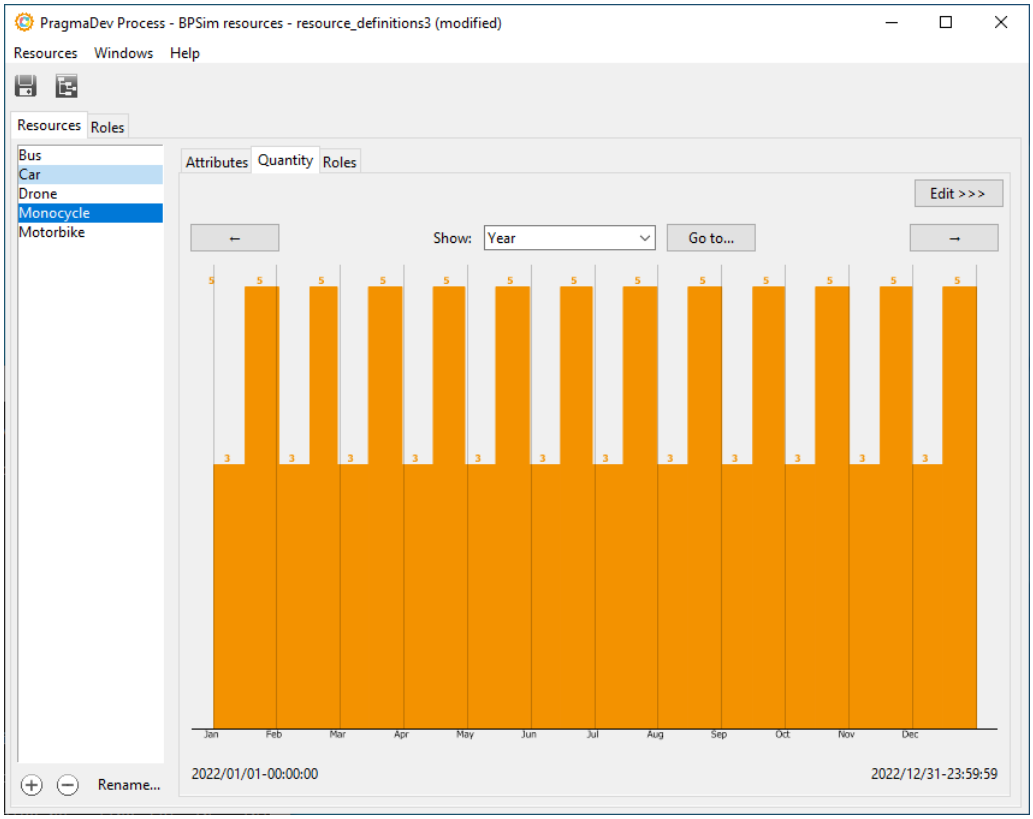



This allows to enter all fields in the repetition rule as defined in RFC 5545. The description of all these fields is out of scope for this manual; please refer to https://icalendar.org/iCalendar-RFC-5545/3-3-10-recurrence-rule.html for more details.

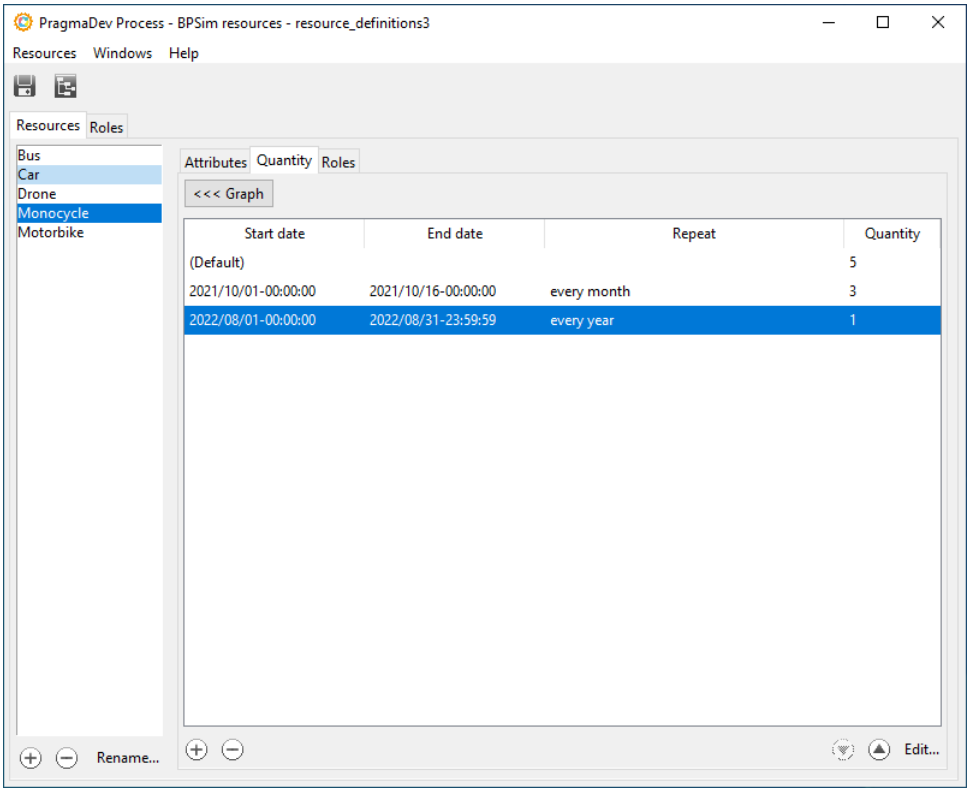
Once several rules are defined, the graph is updated. For example, for a set of rules like this:



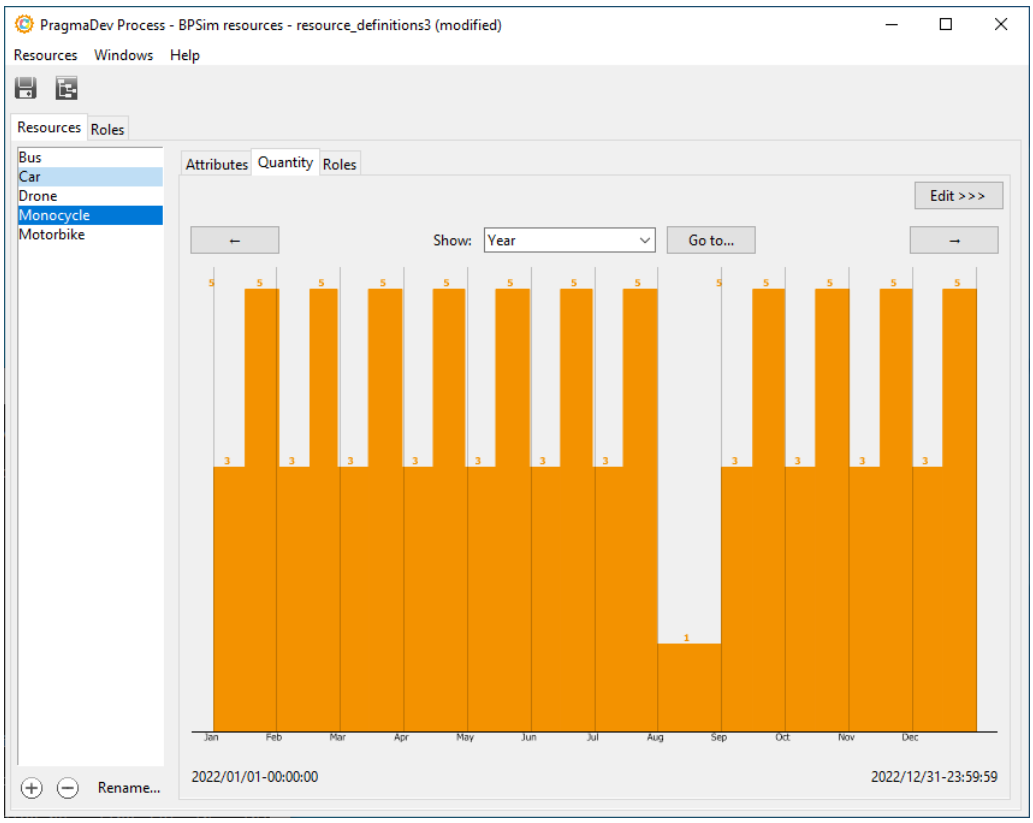

defining a basic quantity of 5 dropping to 3 from the 1st to the 15th of each month, the graph displayed for a whole year will look like this:



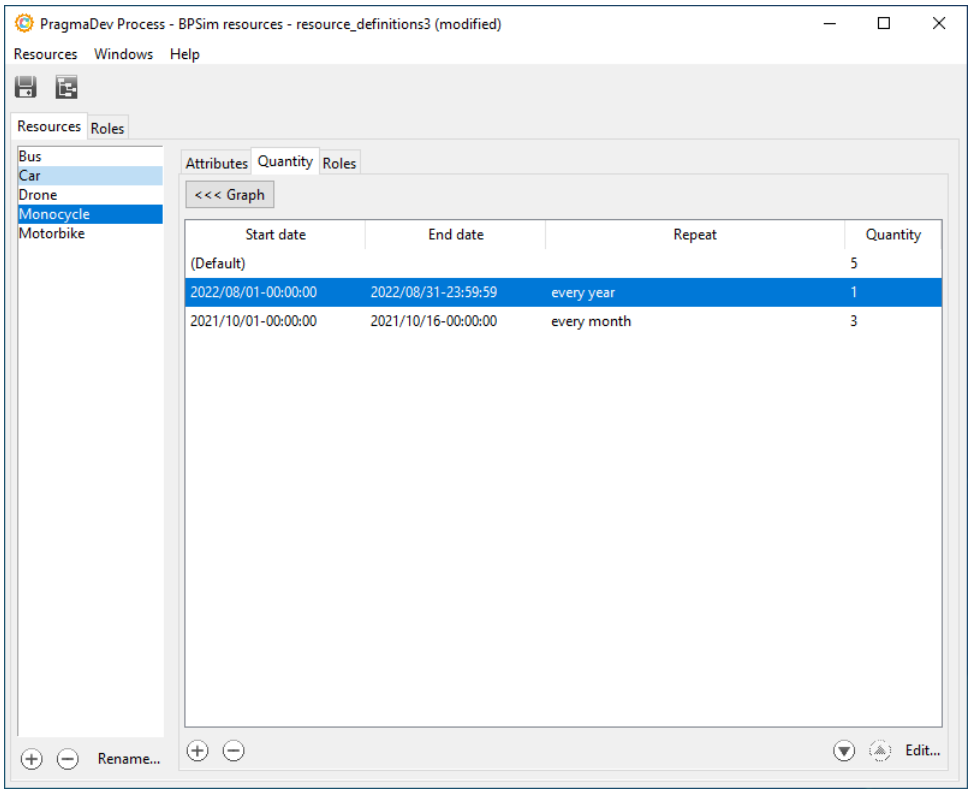

Note that if there are more than 1 rule in addition to the first default one, the ordering is important. For example, in this set of rules:



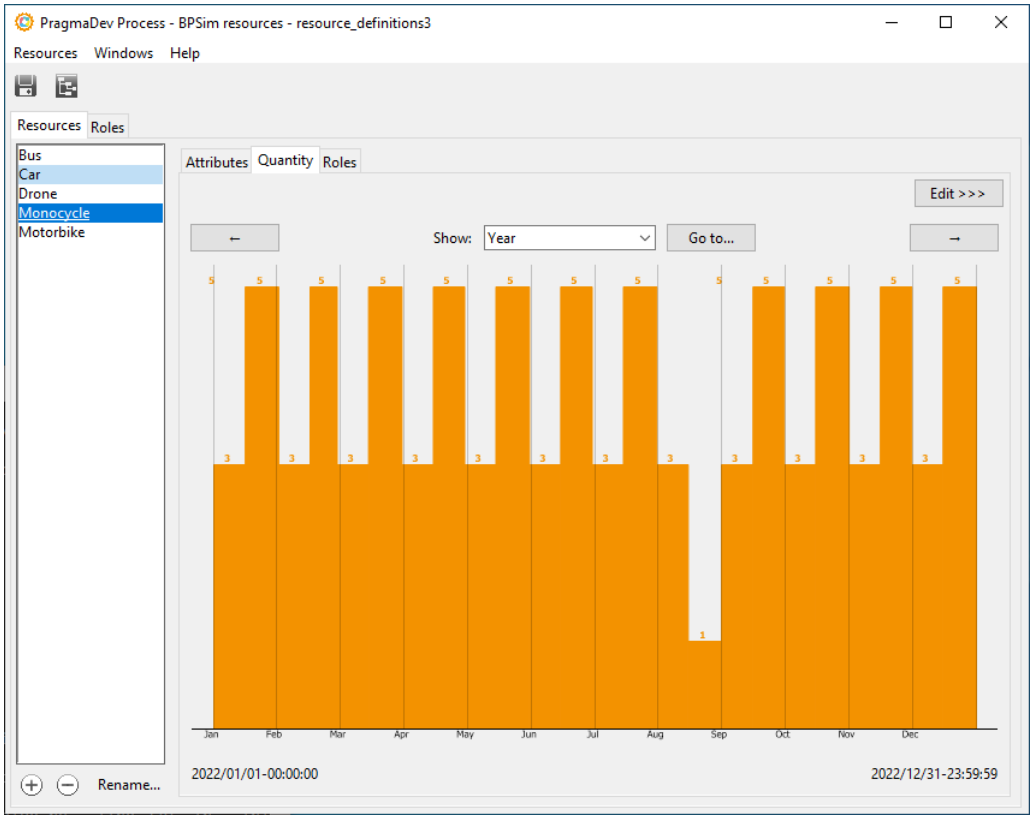
the 3rd rule has priority over the 2nd one. So during August of every year, the 2nd rule does not apply and the quantity is 1:



This is different for the following set of rules:



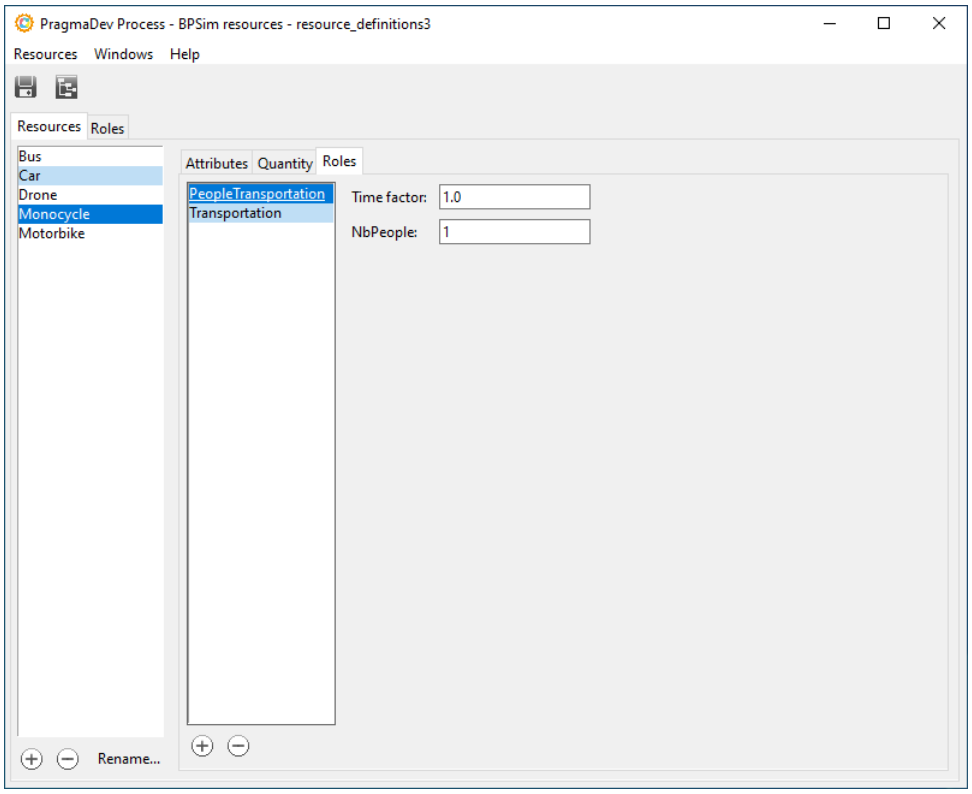

In this case, the rule repeating every month is after the rule repeating every year, so it has priority, and the quantity during August will be 1 (2nd rule), except from the 1st to the 15th where it will be 3 (3rd rule):



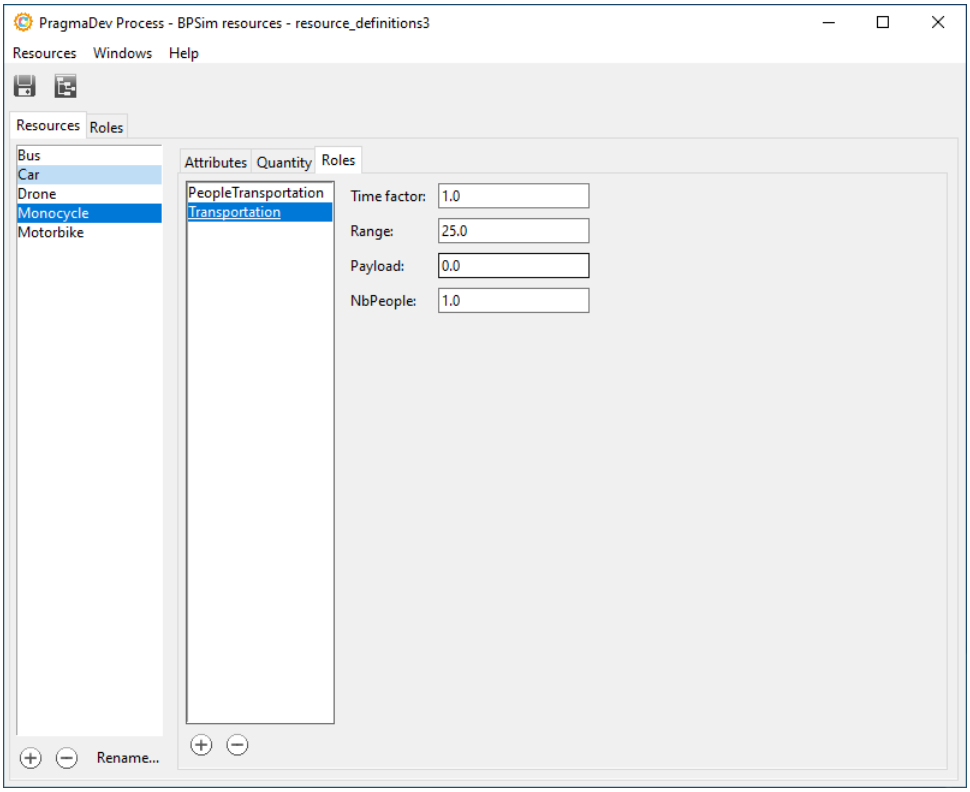

Since the order is important, the rules can be reordered with the up & down arrow buttons appearing under the rule list on the right side.

### 9.4.3 Resource roles

The third and last tab in the resource details pane allow to define which roles the resource has, and define the values for the capacities defined in the role:




Here, the resource "Monocycle" has the roles "PeopleTransportation" & "Transportation". Since the role "PeopleTransportation" defines a capacity named "NbPeople", we have to give a value for this capacity for the resource. Here, it is set to 1. The resource also has the role "Tranbsportation" which defines other capacities:




Here, the resource has values for the capacities defined by the "Transportation" role: "Range" is 25, "Payload" is 0 and "NbPeople" is 1.

These capacity values can be used to select resources, as described in "Resource parameters" on page 130.

For each role it has, a resource also defines a time factor. This time factor defines the efficiency of the resource when it's "playing" the role.

For example, we could have a role "Printer" that will be assigned to all resources representing individual printer. Now all printers are not the same in terms of printed pages per minute: some will be able to do something pretty average, like 15 pages per minute; older ones might only be able to print 5 pages per minute, and faster one could go up to 45 pages per minute.

The time factor defined for the role allows to take that into account. So an activity that would need a printer would define its running time with an average printer, so considering a number of pages per minute of 15. Then we could have the following resources:

- "AveragePrinter" can do 15 pages per minute. Since it is the value considered for the activities using a printer, its time factor for the "Printer" role would be 1.
- "OlderPrinter" can only do 5 pages per minute. So the time configured in the activity will actually be the third of the actual time it will actually need if it uses "OlderPrinter". So the time factor for "OlderPrinter" for the "Printer" role would be 3.
- "FastPrinter" can do 45 pages per minute. So an activity using this printer would actually be 3 times faster than the time configured for it. So the time factor for

"FastPrinter" for the "Printer" role would be 0.333.

Note that the case where an activity uses several resources with different time factors is complex: depending on the kind of activity and on the kind of resources it uses, the time actually spent in the activity could be the maximum time computed from any of the resources it uses, or the time computed with the average of the time factors of the resources it uses, or anything in between. The choice made today in this case in PragmaDev Process is to take the average of the time factors for all the resources an activity uses. This might require to adjust the time factors to get a result that is the closest to reality.

# 10 Glossary

| Acronym | Meaning |
|---|---|
| BPMN | Business Process Model & Notation |
| MSC | Message Sequence Chart |
| PSC | Property Sequence Chart |
| OBP | Observer Based Prover |
| BFS | Breadth First Search |
| DFS | Depth First Search |