

# MSC Tracer

## User Manual





---

<b>Usage - - - - -</b>	<b>5</b>
Overview .....	5
Launching the MSC Tracer .....	5
Connection .....	5
<b>Graphical User Interface - - - - -</b>	<b>6</b>
Main window .....	6
Starting MSC Trace .....	7
Options .....	8
MSC editor .....	9
<b>Command List - - - - -</b>	<b>22</b>
Common options and arguments .....	22
Task creation .....	23
Task deletion .....	24
Messages .....	24
Semaphore creation .....	26
Semaphore deletion .....	26
Semaphore take attempt .....	27
Semaphore take succeeded .....	27
Semaphore take timed out .....	27
Semaphore give .....	28
Timer start .....	28
Timer cancellation .....	28
Timer timed out .....	29
Task state changed .....	29
Action symbol .....	29
Start a new MSC trace .....	29
Pause MSC trace .....	30
Resume MSC trace .....	30
Close MSC trace .....	30
Exit the MSC tracer .....	30
Set directory .....	31
Acknowledgment .....	31
<b>Example - - - - -</b>	<b>32</b>



# 1 - Usage

## 1.1 - Overview

The *MSC Tracer* application can be launched in graphical mode or in command line mode. In graphical mode the trace is being displayed in real time while receiving the commands. In command line mode the trace is saved to a file; that is ideal when generating traces in batch mode. To generate an MSC trace the commands are sent through a socket in textual mode. The format of the commands are described in “Command List” on page 22.

You can find detailed explanations about the MSC graphical representation in the SDL-RT specification document available on SDL-RT web site:

<http://www.sdl-rt.org>

## 1.2 - Launching the MSC Tracer

The executable name is `msctracer.exe` on Windows and `msctracer` on Unix.

Usage is:

```
msctracer [-p <portNumber>] [-f <fileName>] [-d <directory>] [--nw]
```

- `-p <portNumber>` : sets the port number to use when starting socket connection.
- `-f <fileName>` : the filename where to save the MSC trace; if provided the MSC trace will be saved with this name, else a name will be generated or asked to the user,
- `-d <directory>` : the default directory in which the trace will be saved, if no file name is provided or if the file name does not indicate the entire path of the file.
- `--nw` : to launch the *MSC Tracer* in command line mode.

## 1.3 - Connection

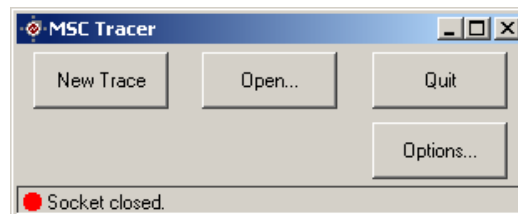
The connection is made by socket where the *MSC Tracer* runs as a server: the socket is initialized on a port number and then waits for a connection from a client. The port number can be set when starting the application (see above) or in the preferences (see “Options” on page 8). If no port number has been provided when starting the connection, the *MSC Tracer* uses the default value 50000. This port number must be the same for the client application which will connect to the *MSC Tracer*. If the *MSC Tracer* is in no window mode, the socket is opened as soon as the *MSC Tracer* is started.

Please note several clients can connect to the same *MSC Tracer* and contribute to the same MSC trace. As the socket can accept several connections, it might get tricky to synchronize several clients. For example if 2 client applications are executed in a row in a shell script, the beginning of the trace of the second client application might get mixed with the end of the trace of the first application. To avoid this behavior, it is possible to ask for an acknowledgment to the MSC Tracer (see “Acknowledgment” on page 31). Waiting for an acknowledgment after the last command in the client program guarantees that the next client program will start when all commands of the previous program have been treated.

## 2 - Graphical User Interface

### 2.1 - Main window

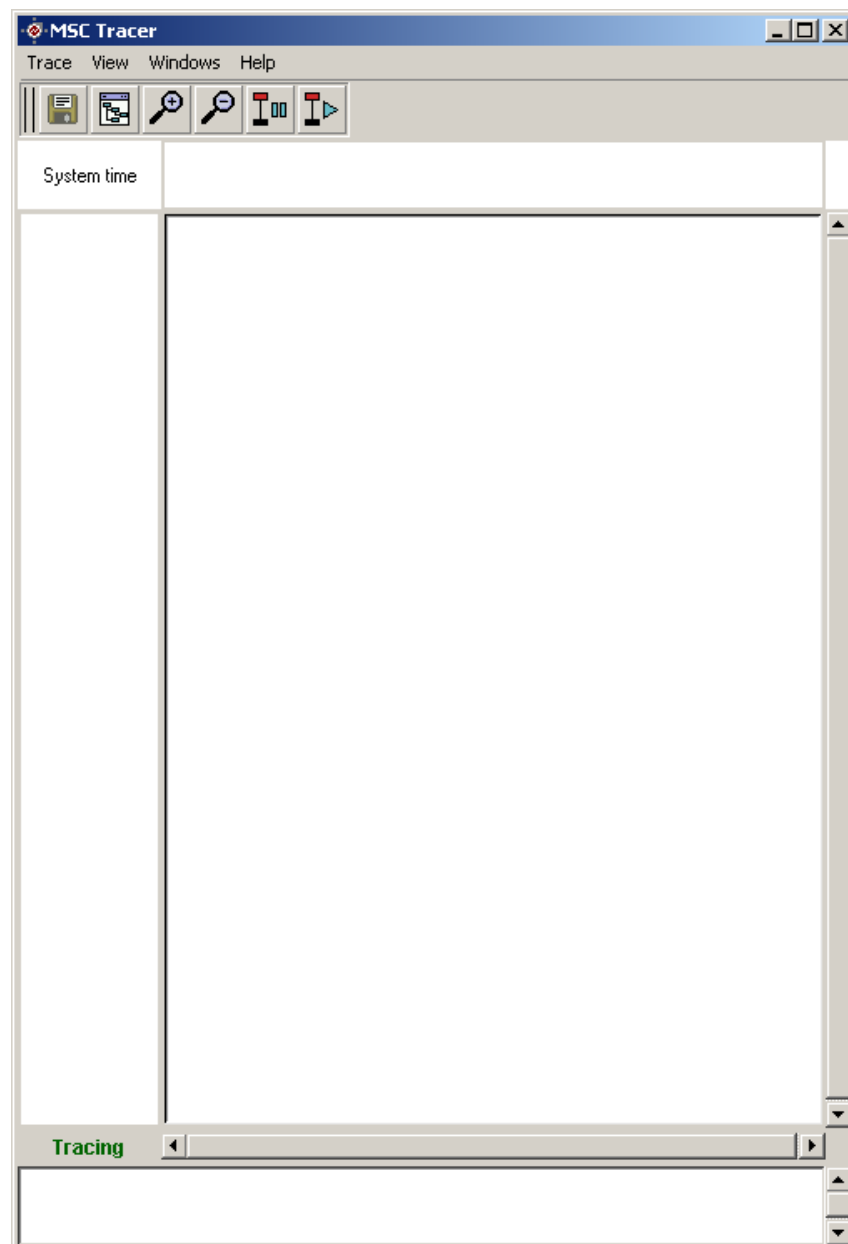
The main window is the main interface between the user and the *MSC Tracer*. It is the first that opens when starting the *MSC Tracer* application and closing this window closes the application.



*MSC tracer main window*

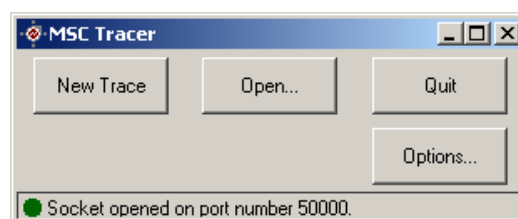
## 2.2 - Starting MSC Trace

A click on "Start" initiates the socket connection and a trace window pops up:



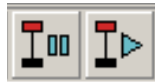
*The MSC trace will appear in this window.*

The main window will display the socket status:



*Socket opened*

The trace can be paused or resumed by using :



By default the trace is running. When the trace is paused all received commands are ignored. Note that, by pausing the trace, the socket is not released so the client application can continue to send commands without generating an error.

The MSC trace can be zoomed in or zoomed out with :



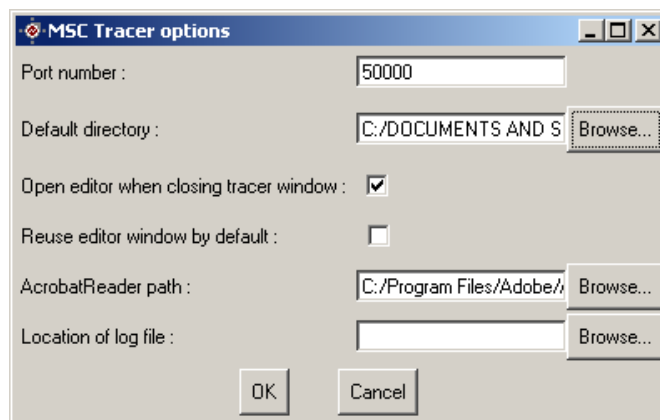
The trace can be saved at any moment with the button :



If no file name was provided when starting the application (with -f option as described in “Launching the MSC Tracer” on page 5), a new name will be asked. When closing this window, the connection and the trace stop.

## 2.3 - Options

Options and preferences can be set up by clicking on the *Options...* button. It opens a dialog :



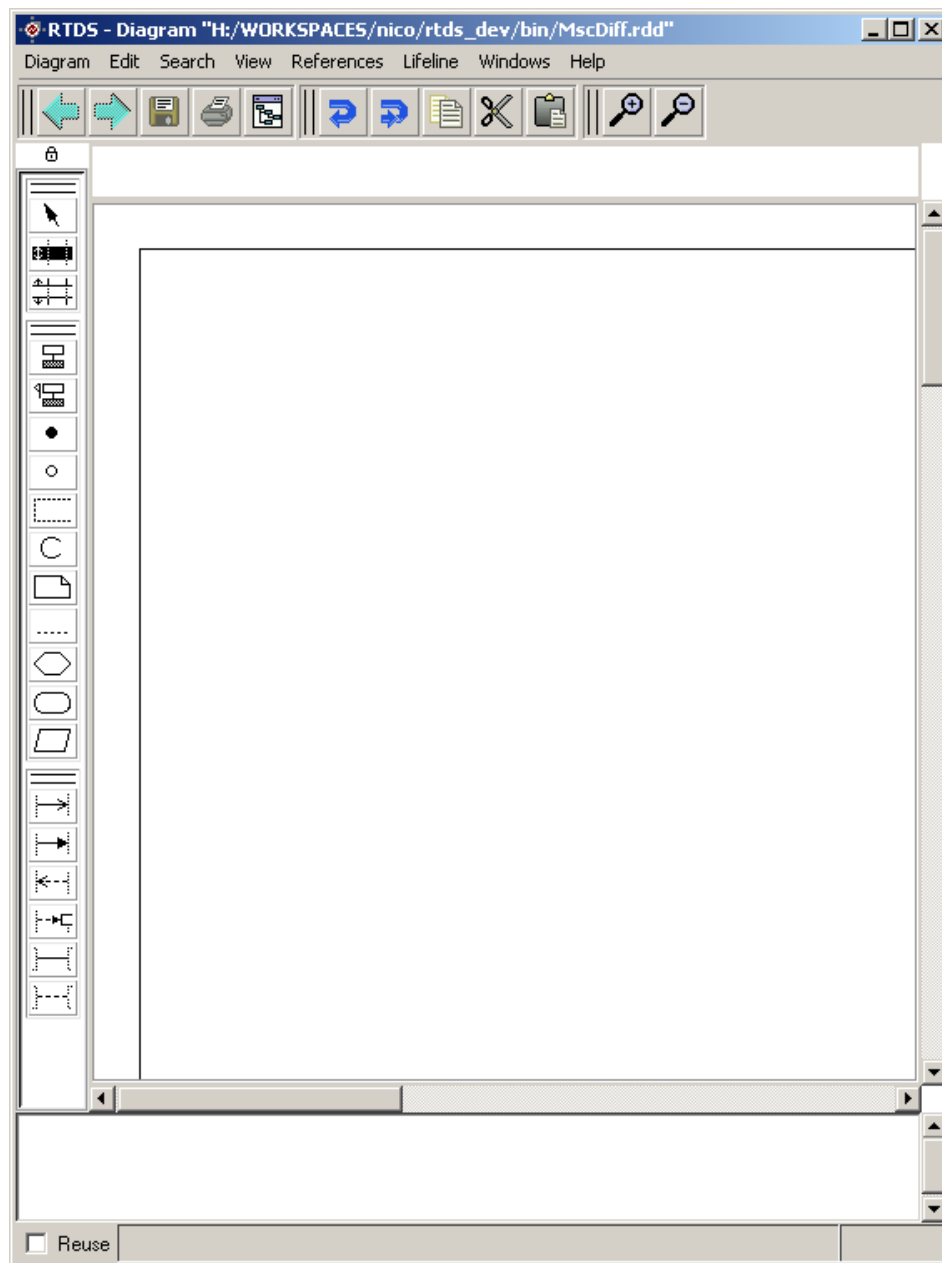
The user can indicate :

- the default port number to use for next connections,
- the default directory where the MSC traces will be saved if a file name is provided,
- if the saved trace will be automatically opened in a editor window when the user closes the tracer window,
- the default reuse option when opening MSC editors as described in “History” on page 14,
- the *Acrobat Reader* path to open the manuals,
- the location of the log file used only in no window mode.



## 2.4 - MSC editor

Diagrams representing MSC traces generated by the *MSC Tracer* can be open by clicking on the "Open" button of the main window. After choosing the name of the file to open, this will display a diagram window, on which a trace can be edited, exported to a publication (see "Publications" on page 14) or compared with another diagram (see "MSC Diff" on page 16).



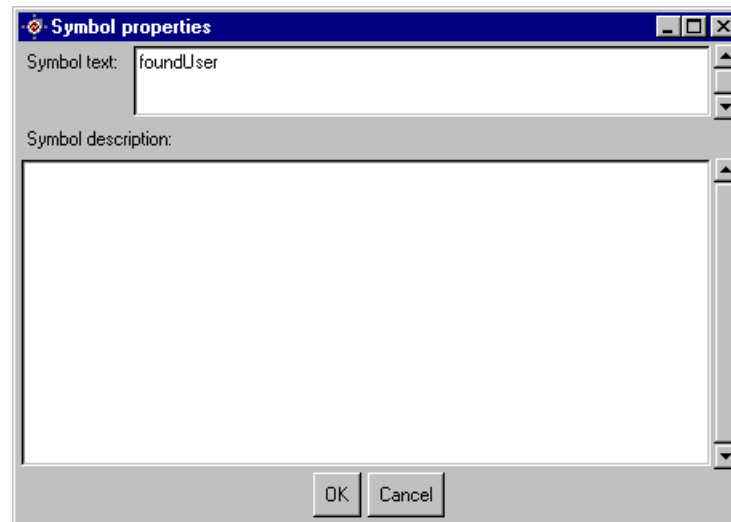
### 2.4.1 Frame concept

All diagrams have a surrounding frame, containing all the symbols in the diagram. You can not put symbols outside that frame.

## 2.4.2 Symbol properties

Each symbol has a property sheet allowing to enter all its features in a guided way. The actual properties depend on the type of the symbol. It may be opened by selecting a symbol and choose "Properties..." in the contextual menu.

The basic property sheet allows only to enter the *Symbol text* and description:






The *Symbol description* is only used for documentation.

## 2.4.3 Button and tool bars


The MSC editor window has several button bars and tool bars to give quick access to all usual operations.

The tool bars on the left of the window are separated in 3 parts:

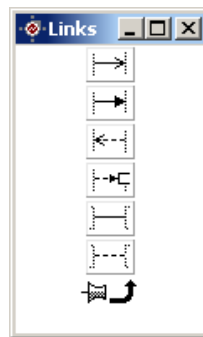
- The top tool bar gives access to general operations on contents of the diagram:
  -  is the default selection tool.
  -  allows to select a time range within the diagram: all events occurring to all life-lines between a given start time and a given end time may be selected, copied, cut and/or pasted somewhere else in the diagram. While this tool is selected, horizontal guidelines follow the mouse cursor to indicate precisely what will be selected. To select a time range, press the mouse button at the desired start time and drag to the desired end time. Once a time range has been copied or cut, pasting is done by clicking at the desired insertion time. The paste operation also displays a horizontal guideline, and may be cancelled by hitting the "Esc" key or by selecting the regular selection tool. Please note that it is not possible to select a time range in a diagram and to paste it in another.
  -  allows to insert empty space in the diagram. To do so press the mouse button at the desired insertion position and drag. When releasing the button, and if possible, a space having the length between the start and end position will be inserted in the diagram.
- The middle tool bar is the symbols tool bar. It allows to insert new symbols in the diagram. This insertion is made by clicking on the button corresponding to the symbol you want to insert, then clicking within the diagram frame where you want to insert the symbol.

- The bottom tool bar is the links tool bar. It allows to insert new links between symbols within the diagram. There is a button for each link type allowed in the diagram. Make sure you select the right type for the symbols you want to link, or the insertion may be refused.

The link insertion is made by clicking on the button corresponding to the link type you want to insert, then pressing the mouse button over the first symbol and dragging to the other symbol.




Above the tool-bars is a tiny lock icon: . When this icon is not active, the button inserts a single symbol or link. Once a symbol or a link has been inserted, the editor returns to the default selection tool. Clicking once on the icon makes it active. After that, any insertion button will be "sticky": any number of symbols or links may be inserted in a row. To go back to "non-sticky" mode, press the lock icon again or select the selection tool.

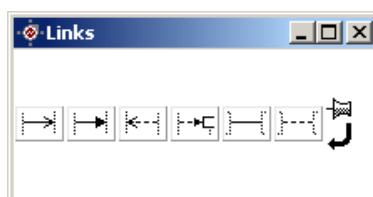
Double clicking on the double separator will detach the bar from its parent window and displays it in its own window.



Link tool bar

The two additional buttons are the following:

-  puts window on top of all displayed windows, preventing the tool bar from disappearing behind another window. When set, this button appears as follows:   
*Note:* this feature may not be available on some window managers such as the Solaris Common Desktop Environment (CDE).
-  changes the windows orientation. For the link tool bar, pressing this button will make the window horizontal:

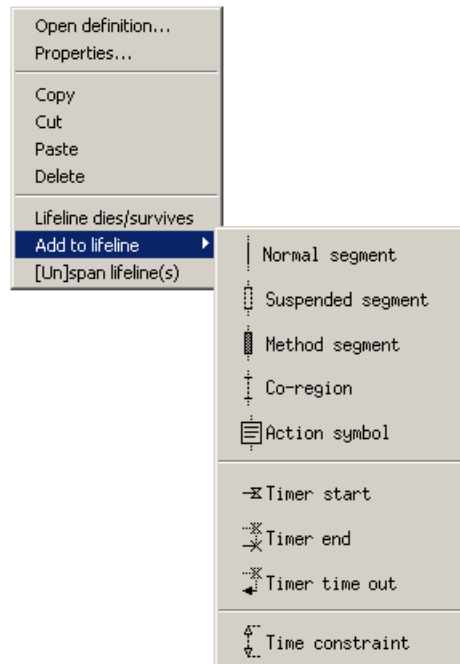


To put the tool bar back in its parent window, just close the tool bar window.

## 2.4.4 Manipulating components in lifelines

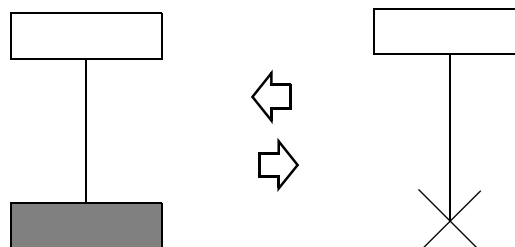
Unlike all other symbols, lifelines are composite symbols: they may include several components like segments, timers or time constraints. They may also end before the end of the diagram or con-

tinue after the end of the diagram. All these features are managed via the contextual menu in the MSC editor activated with right mouse button:



When a lifeline is selected, the items in this menu have the following action:

- *Lifeline dies/survives* toggles between the “instance tail” and the “instance stop” ending for the lifeline:



- *Add to lifeline* adds an item to the lifeline a segment. The item is selected from a sub-menu list:
  - action symbol,
  - timer,
  - time constraint.

After selecting an item in the sub-menu, press the mouse button at the desired start position of the segment, action symbol, timer or time constraint in the lifeline, and drag to its end position (if applicable). To cancel the insertion, hit the "Esc" key or select the default selection tool.

- *[Un]span lifeline(s)* attaches or detaches a spanning symbol from a set of lifelines. Spanning symbols are *conditions* or *MSC references*. They are attached to one or several lifelines so that when the lifelines are moved or resized the lifelines are always spanned by the symbol.

This menu item only works when a spanning symbol and one or several lifelines are selected:

- If the selected lifelines are not spanned by the symbol, they are added to the set of spanned lifelines,

- If the selected lifelines are already spanned by the symbol, they are removed from the set of spanned lifelines.

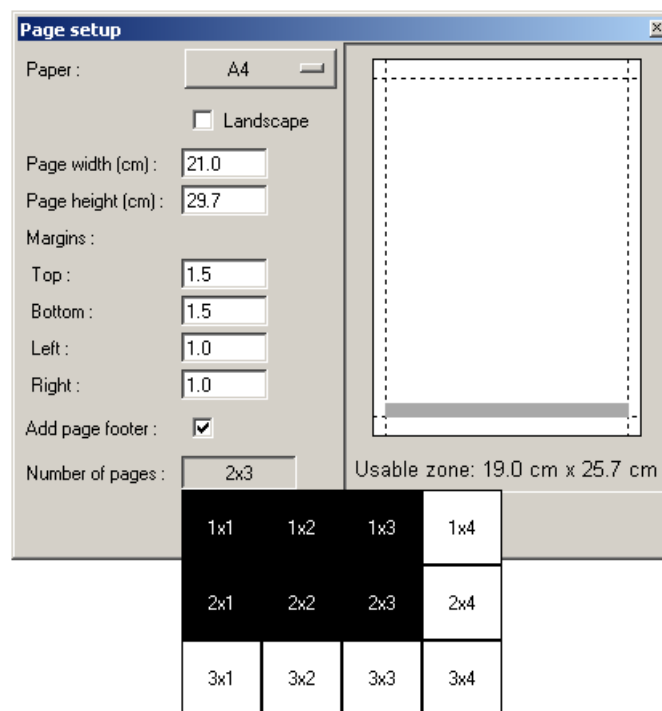
A dialog will inform you about the performed operation.

*Please note:* there is no visual information on the diagram about lifelines spanned by symbols. The lifelines are always behind spanning symbols, even if they are not spanned.

Clicking on a *segment*, *action symbol*, *timer* or *time constraint* element in a *lifeline* selects it as indicated by black handles around the selected item. Note the lifeline itself and all other items related to the lifeline will have white handles. That means some operations will act on the selected item, and others on the whole lifeline. For example, delete will only delete the item but a vertical move will move the whole lifeline. This is because the element is part of the lifeline.

## 2.4.5 Page setup

To ensure a WYSIWYG behavior, diagrams include a page setup that will be the one used when printing it. This page setup may be edited via the "Page setup..." item in the "Diagram" menu in diagram editors. The following dialog appears:



The fields "*Paper*", "*Landscape*", "*Page width (cm)*" and "*Page height (cm)*" are used to specify the paper size. The "*Margins*" are automatically removed from the usable area for the diagram and when printing. If the "*Add page footer*" option is checked, a footer will be displayed on each printed page, containing the name of the printed file and the page number. The height of this footer is also removed from the usable area for the diagram. This usable area is displayed with its dimensions in the right part of the dialog.

The field "*Number of pages*" is used to specify the number of rows and columns of pages in the diagram. The above example sets a width of 3 pages and a height of 2 pages for the diagram, which will have 6 pages. The usable page size is  $21 - 1 - 1 = 19$  centimeters wide and  $29.7 - 1.5 - 1.5 - 1$  (for the footer) = 25.7 centimeters high. So the diagram will be  $3 * 19 = 57$ cm wide and  $2 * 25.7 = 51.4$ cm high.

### 2.4.6 History

The MSC editor window may be configured to be reused when you open another diagram from the main window. If a window is reusable, it keeps track of all opened diagrams and allows you to browse this history.

To make a window reusable, just check the "Reuse" check-box in the bottom-left corner:



You may also make all windows reusable by default by checking the corresponding option in the preferences (see "Options" on page 8).

To browse the history of a reusable window, you can use the two arrow buttons in the button bar:



These buttons act like "Back" and "Forward" buttons in Web browsers. You may also browse the history via the "History" sub-menu in the "Diagram" menu.

Note that navigating through the history of previously opened diagrams will close the currently opened one, asking whether the current modifications should be saved or not. So there is actually only one diagram opened at a time for each editor.

### 2.4.7 Publications

It is possible to attach a set of publications to any diagram. A publication is a set of symbols that will be exported as an external image file. These publications can be re-exported at any time, and the user is asked each the diagram is saved.

These publications can be used by importing them in any word processing software that has an "insert with link" or "import by reference" function. This function allows to insert a file into the current document, but keeps a reference to the inserted file so that any modification to the inserted file will be reflected automatically in the document.

There are currently two types of publications:

- "Diagram" publications, that will export the whole diagram
- "Symbol" publications, that will export a subset of the diagram's symbols

These types are described in the paragraphs below.

*Notes:*

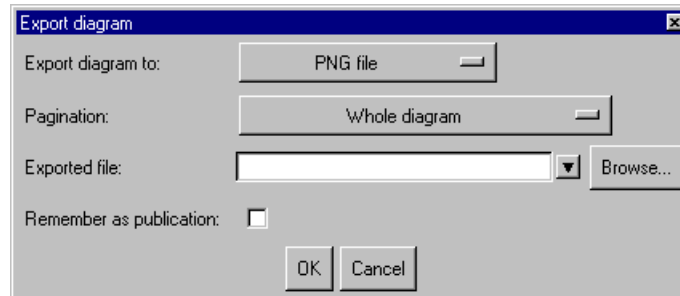
- The publications may also be used in documents written using a markup language like SGML, XML or HTML. For example, with HTML, images may simply be imported via the tag `<img src="">` without giving any dimension for the image so that they will always be read from the image file.
- Some word processors (e.g. FrameMaker) remember the size you gave to any imported graphics, even if these were imported by reference. RTDS publications still work with these, but you may end up with distorted graphics in the final document if the size of a publications changes.

#### 2.4.7.1 Diagram publications

These publications are managed via the two items "Export diagram..." and "Manage diagram publications..." in the "Diagram" menu in each editor. These publications are always exported explicitly: even if the corresponding option has been set in the preferences, they won't be exported automatically when the diagram is saved. The reason for this is simply that exporting the

whole diagram may take some time, and having all diagram publications exported each time the diagram is saved may make the save really long.

Selecting "Diagram" / "Export diagram..." opens the following dialog:



The field "*Export diagram to*" allows to select the type of image files that will be generated. This type may either be PNG or JPEG (*NB*: PNG usually gives better results).

On Windows, a special type "Windows clipboard" allows to copy the entire diagram as a bitmap in the clipboard. With this type, the following fields will be disabled.

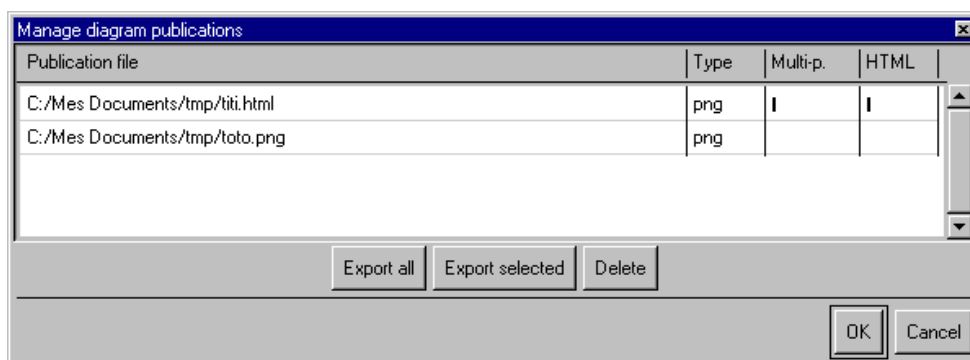
The field "*Pagination*" selects the pagination mode for the export:

- "Whole diagram" will export the whole diagram in a single image file
- "Multi-page" will export each diagram page in a file. The file will be named after the contents of the "Exported file" field, followed by a suffix giving the page row and column numbers in the diagram
- "Multi-page w. HTML wrapper" will do the same thing than the "Multi-page", but will also generate a HTML file including all generated image files. Importing the HTML file in a word processing software allows to import the whole set of pages in one operation, and to keep it up to date even if the diagram has more or less pages afterwards.

The field "*Exported file*" gives the name for the generated file. Note that with the "Multi-page" pagination, suffixes may be added to the name you give here. A drop-down list allows you to choose a file already associated to a publication.

The check-box "*Remember as publication*" controls whether the export will only be made one time or actually remembered as a diagram publication that can be re-exported on demand.

Selecting "Diagram" / "Manage diagram publications..." opens the following dialog:

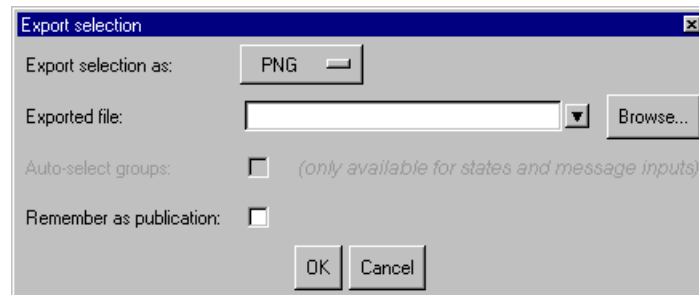


The table lists all current diagram publications with their features. All or any subset of these publications can be exported or deleted using the buttons under the list.

### 2.4.7.2 Symbol publications

These publications are managed via the two items *Export selection...* and *Manage symbol publications...* in the *Edit* menu in each editor. When saving, the user is asked if the publications are to be exported.

Selecting *Edit / Export selection...* opens the following dialog:

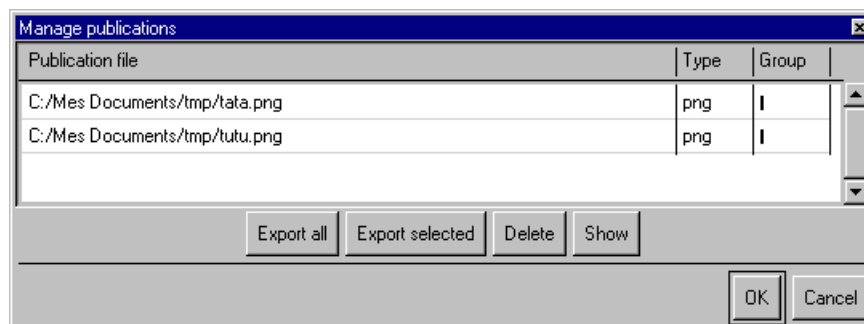


The field *Export selection as* controls the type of the image file into which the selection is exported. The field *Exported file* contains the actual name of the image file.

The field *Auto-select groups* is always inactive in the current version.

The check-box *Remember as publication* controls whether the export will only be made one time or actually remembered as a symbol publication.

Selecting *Edit / Manage symbol publications...* opens the following dialog:



The table lists all current symbol publications with their features. All or any subset of these publications can be exported or deleted using the buttons under the list. Clicking on the *Show* button will show the exported symbols in the editor.

When a diagram having symbol publications is saved, the user is asked for update confirmation.

*Note:* on Windows, the selected symbols may also be directly copied to the Windows clipboard by using the item "Copy as bitmap" in the "Edit" menu (shortcut: Shift-Ctrl-C). This feature is not available on Unix platforms.

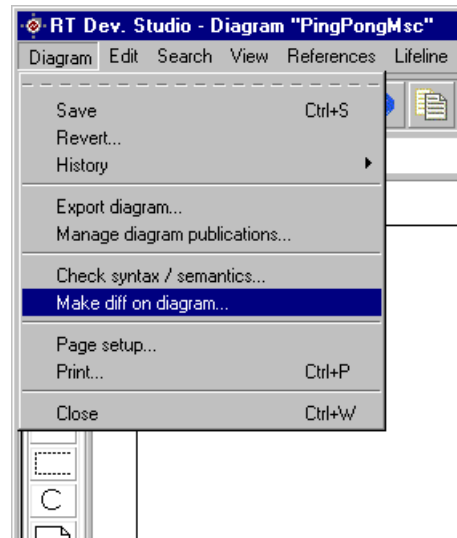
### 2.4.8 MSC Diff

This feature shows the differences between two MSC diagrams.

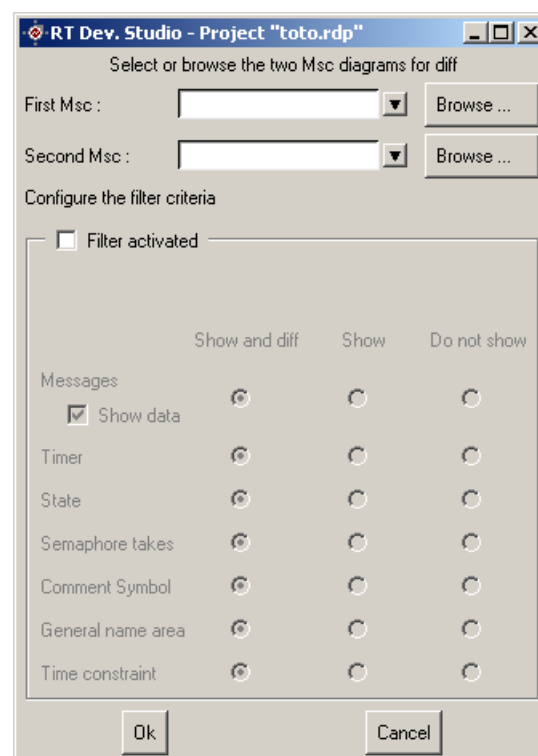


### 2.4.8.1 Running the diff

The *MSC diff* launched from the *MSC editor* through the *Diagram* menu.



A dialog window pops up:



The two MSCs to compare are selected from the current project or from a file. Filters can be applied on the result of the *MSC Diff*. Symbol types can be:

- removed from the resulting diagram, or
- viewed but not checked in the diff, or
- viewed and checked in the diff.

In this case a symbol specific to a diagram will be colored.

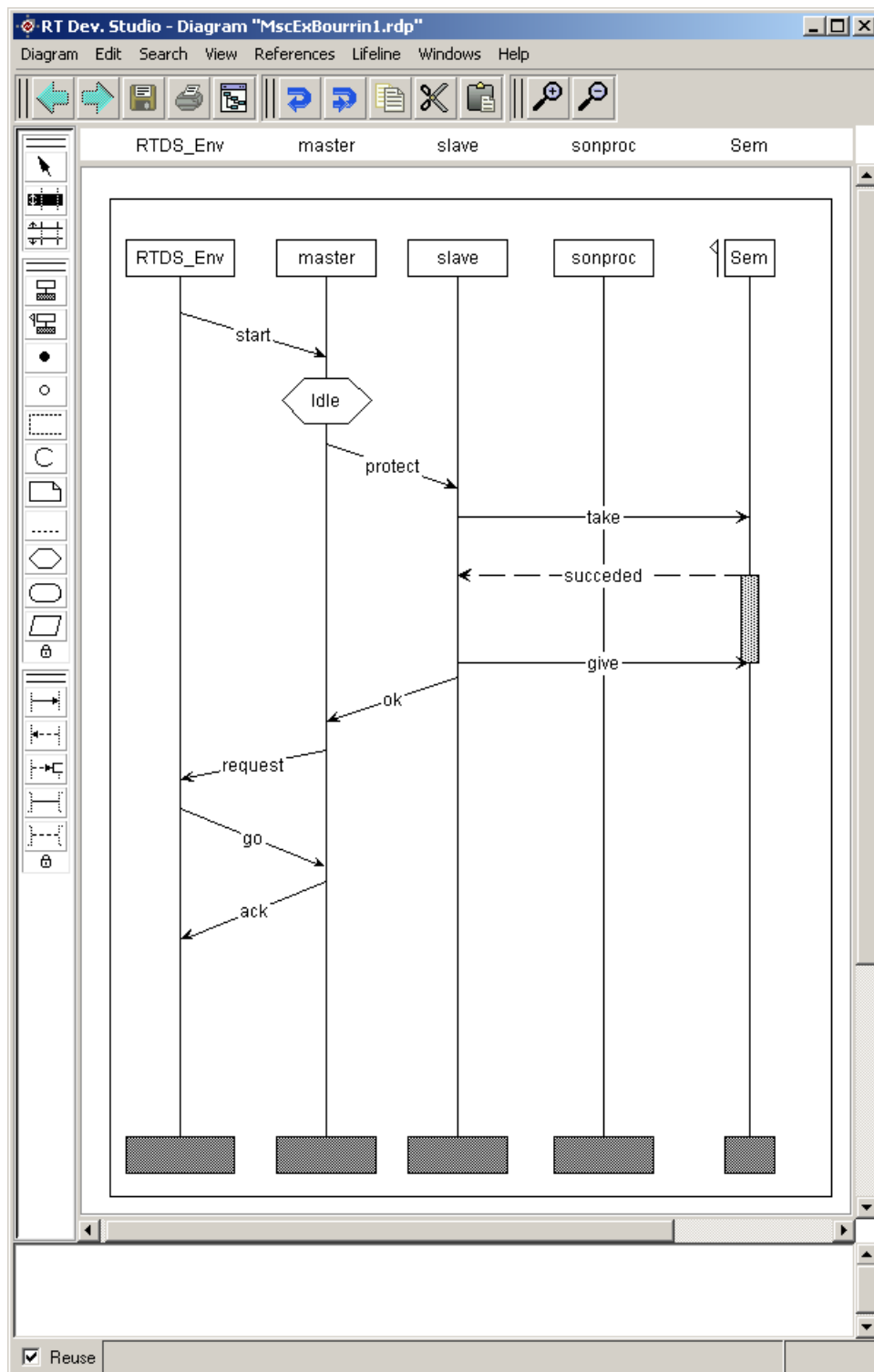
#### 2.4.8.2 Result analysis

The resulting MSC identifies the lifelines between the two diagrams by their name. If there are several processes with the same name it will use the sequence of events. If two lifelines in the two diagrams represent the same process but do not have the same name, they will not match.

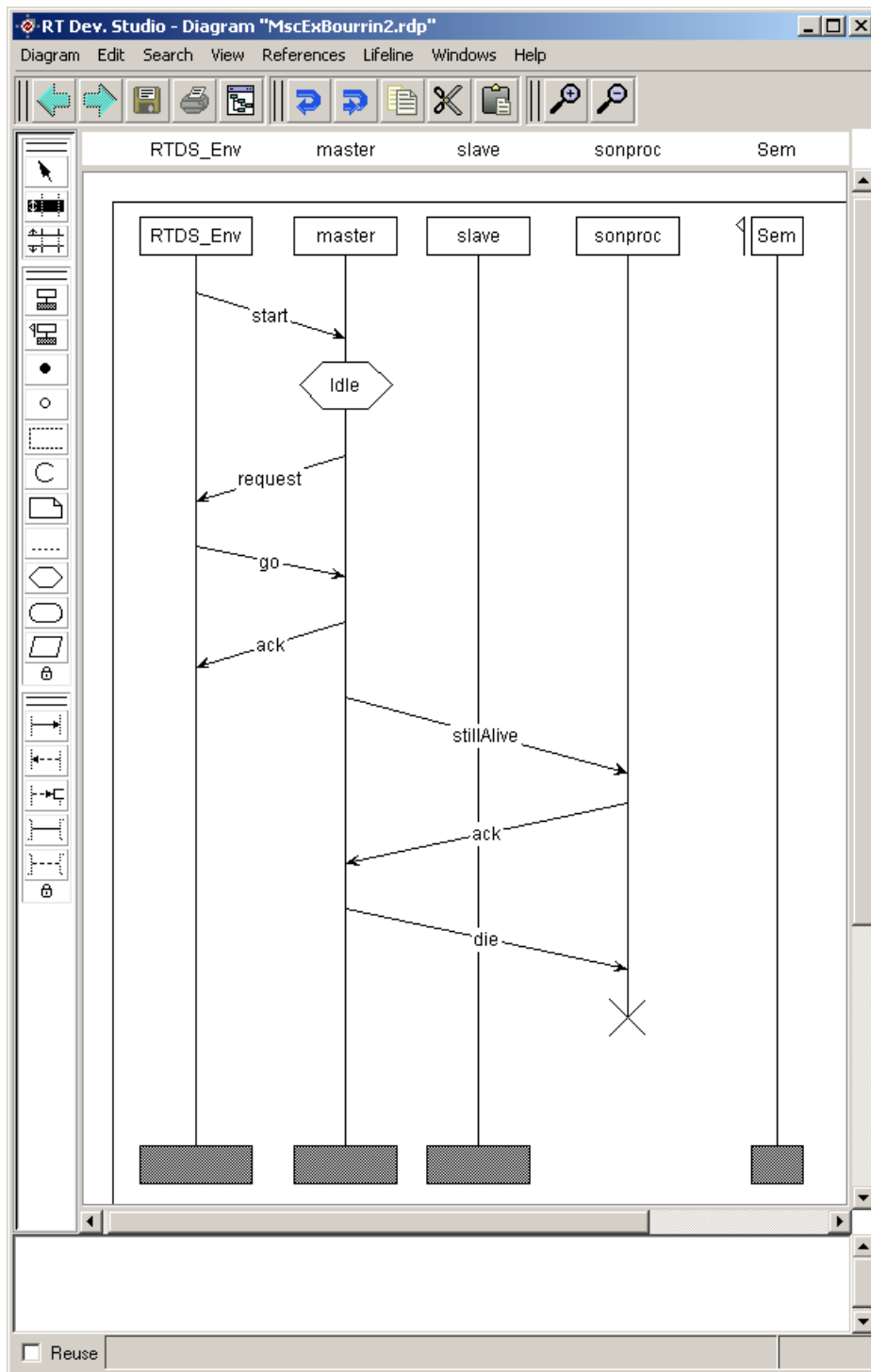
Note the *MSC Diff* searches for the difference in the sequence of events on the lifelines. The elapsed time between two events on the resulting diagram is not significant.

### 2.4.8.3 Example

Considering the following two MSC diagrams:



*First MSC diagram in diff*

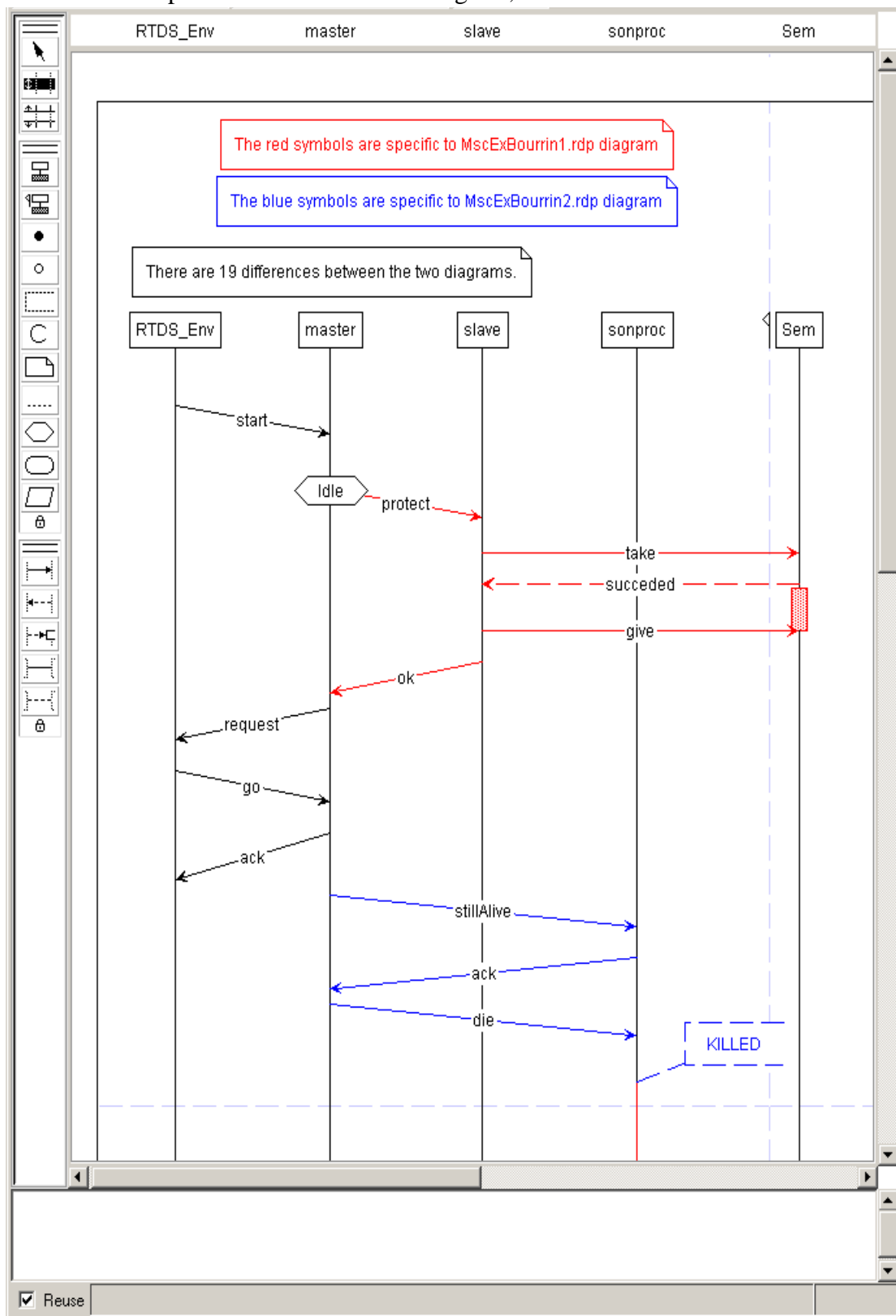


### *Second MSC diagram in diff*

The *MSC diff* shows the following differences:

- The "slave" process takes the semaphore "Sem" in the first diagram, but not in the second.
- The processes "master" and "sonproc" exchange messages "stillAlive", "ack" and "die" in the second diagram, but not in the first.

- Process "sonproc" dies in the second diagram, but not in the first.



The differences are identified by colors: in the diff diagram, the events specific to the first diagram are represented in red, and the events specific to the second diagram are represented in blue.

## 3 - Command List

The *MSC Tracer* accepts several commands to generate the MSC trace. They represent a specific real-time event illustrated by a symbol in the MSC diagram.

The field separator in a command is:

'| ' (pipe followed by a space)

The command is ended by:

'|\n' (pipe followed by the line feed character i.e. 0x13)

To insert a printable pipe in the command, another pipe must be put up front:

'||' => printable '|'

For example, the command for a task creation named pPing with pid 1 will be:

```
taskCreated| -npPing| 0x01|\n
```

The generic syntax of a command is:

```
<command_name or alias>| [<options>]| <parameters>| <optional_arguments>|
```

If a command name or alias is not recognized, the whole command is ignored.

If one of the parameters is not provided, the whole command is ignored.

If a provided option does not exist, the whole command is ignored.

If the number of optional arguments provided is greater than the expected number, the command interpreter will ignore the additional arguments.

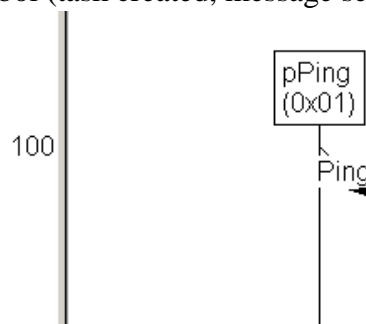
Each command has an alias; e.g. : *ms* has the same meaning as *MessageSent*.

### 3.1 - Common options and arguments

The following options and arguments may appear in several commands:

- -t<time>

Represents the time when an event occur; if provided, a symbol containing the value of time of the associated event will appear at the left of the MSC trace at the same height that the event symbol (task created, message sent...) involved.



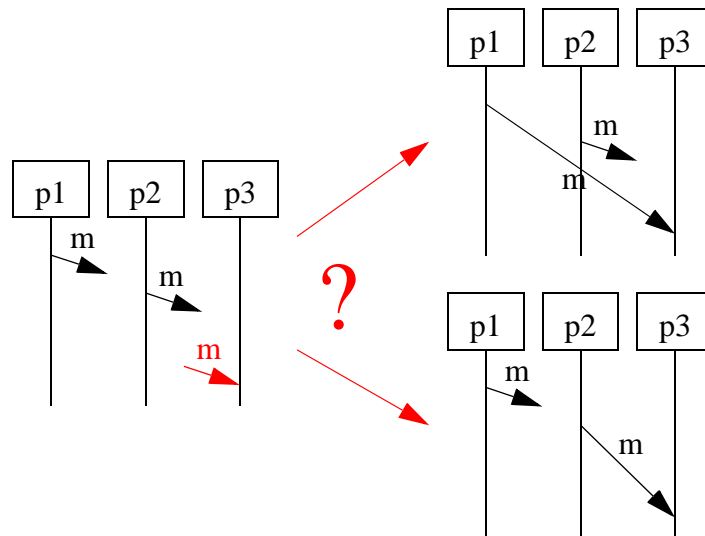
Here the sending of message ping from process pPing occurs at time 100 after the beginning of system execution.

- -i<mId>

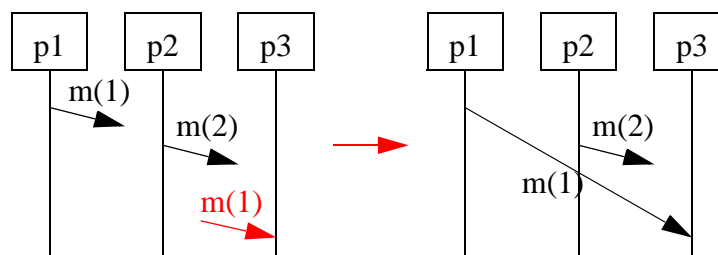
Represents a unique identifier for a given message. It comes with the commands *messageSend*, *messageReceive*, and *messageSave*. This option provides a unique identifier for the sending of a message, allowing to match a message receive to the corresponding message send. If this option is not used, a message receive will always suppose that the received message is the last message sent with the same name. This may

result in errors in the trace if several messages with the same name exist at the same moment.

For example:



*Without message unique ids*



*With message unique ids (after message name)*

- **<sigNum>**  
Numerical identifier for the name of a message. Note both the name and this identifier may have to be provided.
- **<pId> OR <semId>**  
Identifier of a process or semaphore; it is an unique identifier. `pId` and `semId` represents the id of the lifeline, so it is not possible to have a semaphore and a process with the same id.
- **<tId>**  
unique timer identifier; it must be provided, and may be used as the name of the timer if no option `-T` is provided.

## 3.2 - Task creation

To trace a task creation, the command syntax is:

```
taskCreated| [-t<time>]| [-c<creatorId>]| [-n<pName>]| [-N<creatorName>]| pId|\n
or
pc| [-t<time>]| [-c<creatorId>]| [-n<pName>]| [-N<creatorName>]| pId|\n
```

options:

- -t : time of event, See “Common options and arguments” on page 22.
- -c : identifier of creator process,
- -N : name of creator process,
- -n : name of created process; if this option is not present, the process name on the MSC trace will be its id.

parameters:

- pId : unique process identifier.

### 3.3 - Task deletion

To trace a task deletion, the command syntax is:

```
taskDeleted| [-t<time>]| [-n<pName>]| pId|\n
or
pd| [-t<time>]| [-n<pName>]| pId|\n
```

options:

- -t : time of event, See “Common options and arguments” on page 22.
- -n : process name,

parameters:

- <pId> : unique process identifier.

### 3.4 - Messages

#### 3.4.1 Message sending

To trace a message sending from a process, the command syntax is:

```
messageSent| [-t<time>]| [-d<msgData>]| [-n<pName>]| [-i<mId>]| pId| sigNum| msgName|\n
or
ms| [-t<time>]| [-d<msgData>]| [-n<pName>]| [-i<mId>]| pId| sigNum| msgName|\n
options:
```

- -t : time of event. See “Common options and arguments” on page 22.
- -d : data of the message. Spaces are allowed and the data format is free except for the ‘|’ character that should be doubled: ‘||’. Detailed format is described in “Message parameter format” on page 25.
- -n : name of the process sending the message.
- -i : unique identifier for message, See “Common options and arguments” on page 22.

parameters:

- <pId> : unique process identifier.
- <sigNum> : signal number of the message, See “Common options and arguments” on page 22.
- <msgName> : message name.

#### 3.4.2 Message reception

To trace a message reception, the command syntax is:

```
messageReceived| [-t<time>]| [-d<msgData>]| [-n<pName>]| [-i<mId>]| pId| sigNum| msgName|\n
or
mr| [-t<time>]| [-d<msgData>]| [-n<pName>]| [-i<mId>]| pId| sigNum| msgName|\n
```



options:

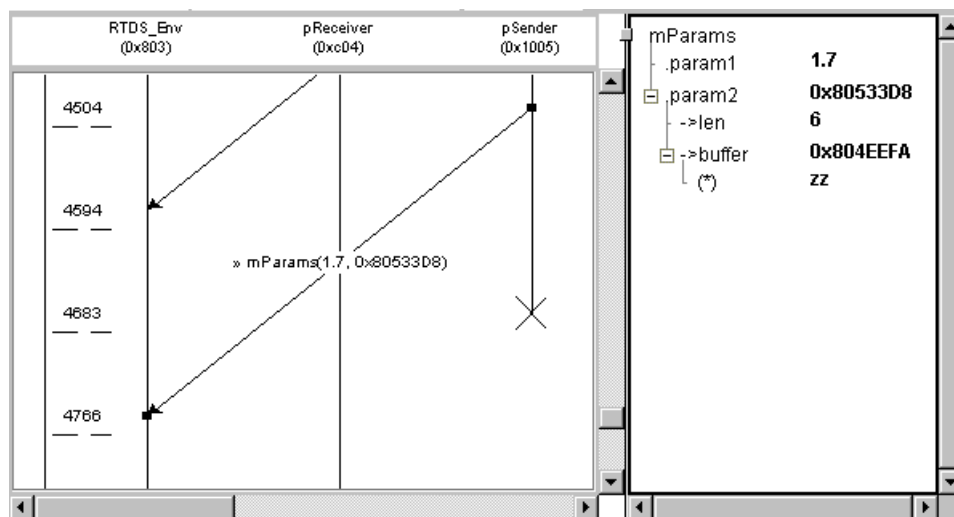
- -t : time of event, See “Common options and arguments” on page 22.
- -d : data of the message. Spaces are allowed and the data format is free except for the ‘|’ character that should be doubled: ‘||’. Detailed format is described in “Message parameter format” on page 25.
- -n : name of the process receiving the message.
- -i : unique message identifier, See “Common options and arguments” on page 22.

parameters:

- <pId> : unique process identifier.
- <sigNum> : signal number of the message, See “Common options and arguments” on page 22.
- <msgName> : message name.

### 3.4.3 Message parameter format

The format described below applies to structured parameters in messages sent or received. Sending structured messages with the following format allows to display parameters in a tree in the MSC editor as shown below.



The format for this text or argument depends on whether the message is structured or not. Structured parameters are fully described in RTDS Reference Manual. In short, a message is structured if and only if it is declared with several parameters or with one parameter that is a pointer to a struct or a union.

- For a non-structured message, the text for the parameter must be a sequence of bytes written in hexadecimal format, exactly as they will appear in the target program memory.
- For a structured message, the text for the parameter must be written as follows:
  - The values for base types are written as in C: for example 12 or 871 are valid values for an int, x is a valid value for a char, and so on...
  - The values for pointers are written in hexadecimal, optionally prefixed by 0x, and followed by | : and the pointed value. For example, for an int\*:
    - 804A51FE | : 67 will define the pointer to be 0x804A51FE and 67 will be the pointed value;
    - | : 123 will only describe the pointed value to be 123;

There is a special case for `char*` pointers: the value can be a full string instead of just a single `char`. Please note all `'|'` characters must be doubled in this string.

- The values for structs or unions are coded as follows:

```
|{field1|=value|,field2|=value|,...|}
```

For example, for a struct defined as:

```
struct MyStruct { int i; char *s; };
```

a valid format is:

```
|{i|=4|,s|=|:abcd|}
```

In the struct, the field `i` will be set to 4 and the field `s` will point to the "abcd".

### 3.4.4 Message saving

To trace a saved message, the command syntax is:

```
messageSaved| [-t<time>]| [-n<pName>]| [-i<mId>]| pId| sigNum| msgName|\n
```

or

```
mv| [-t<time>]| [-n<pName>]| [-i<mId>]| pId| sigNum| msgName|\n
```

options:

- `-t` : time of event, See "Common options and arguments" on page 22.
- `-n` : name of process that saves the message,
- `-i` : unique message identifier, See "Common options and arguments" on page 22.

parameters:

- `<pId>` : unique process identifier,
- `<sigNum>` : signal number for the message, See "Common options and arguments" on page 22.
- `<msgName>` : saved message name

## 3.5 - Semaphore creation

To trace a semaphore creation, the command syntax is:

```
semaphoreCreated|sc| [-t<time>]| [-s<semName>]| [-c<creatorId>]| [-N<creator-Name>]| [-a<stillAvailable>]| pId|\n
```

options:

- `-t` : time of the event, See "Common options and arguments" on page 22.
- `-s` : name of semaphore; if not provided the name of the created semaphore in the MSC trace will be its internal identifier,
- `-c` : identifier of the creator process,
- `-N` : name of the creator process,
- `-a` : boolean indicating if semaphore is empty (value 0) or full (value 1),

parameters:

- `<pId>` : unique identifier of the created semaphore.

## 3.6 - Semaphore deletion

To trace a semaphore deletion, the command syntax is:

```
semaphoreDeleted| [-t<time>]| [-s<semName>]| [-c<destructorId>]| [-N<destructorName>]| pId|\n
```

or

```
sd| [-t<time>]| [-s<semName>]| [-c<destructorId>]| [-N<destructorName>]| pId|\n
```

options :

- `-t` : time of event, See "Common options and arguments" on page 22.

- -s : name of deleted semaphore,
- -c : identifier of the destructor process,
- -N : name of the destructor process,

parameters :

- <pId> : unique identifier of the deleted semaphore.

### 3.7 - Semaphore take attempt

To trace an attempt to take semaphore, the command syntax is:

```
takeAttempt| [-t<time>]| [-n<pName>]| [-s<semName>]| [-T<timeout>]| pId|
semId|\n
```

or

```
sa| [-t<time>]| [-n<pName>]| [-s<semName>]| [-T<timeout>]| pId| semId|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.
- -n : name of process taking the semaphore,
- -s : name of the taken semaphore,
- -T : timeout for the take; a value of -1 indicate that the process will wait until the take have succeeded,

parameters :

- <pId> : identifier of taker process,
- <semId> : identifier for taken semaphore.

### 3.8 - Semaphore take succeeded

To trace a semaphore has been successfully taken, the command syntax is:

```
takeSucceeded| [-t<time>]| [-n<pName>]| [-s<semName>]| [-a<stillAvailable>]|
<pId>| <semId>|\n
```

or

```
ss| [-t<time>]| [-n<pName>]| [-s<semName>]| [-a<stillAvailable>]| <pId>|
<semId>|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.
- -n : name of the process taking the semaphore,
- -s : name of taken semaphore,
- -a : boolean indicating if semaphore is empty (value 0) or full (value 1),

parameters :

- <pId> : identifier for taker process,
- <semId> : identifier for taken semaphore.

### 3.9 - Semaphore take timed out

To trace a semaphore take attempt timed out, the command syntax is:

```
takeTimedOut| [-t<time>]| [-n<pName>]| [-s<semName>]| <pId>| <semId>|\n
```

or

```
st| [-t<time>]| [-n<pName>]| [-s<semName>]| <pId>| <semId>|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.

- -n : name of taker process,
- -s : name of taken semaphore,

parameters :

- <pId> : identifier of taker process,
- <semId> : identifier of taken semaphore.

### 3.10 - Semaphore give

To trace a semaphore give, the command syntax is:

```
giveSem| [-t<time>]| [-n<pName>]| [-s<semName>]| pId| semId|\n
```

or

```
sg| [-t<time>]| [-n<pName>]| [-s<semName>]| pId| semId|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.
- -n : name of the process giving the semaphore,
- -s : name of given semaphore,

parameters :

- <pId> : identifier of giver process,
- <semId> : identifier of given semaphore.

### 3.11 - Timer start

To trace the start of a timer, the command syntax is:

```
ts| [-t<time>]| [-n<pName>]| [-T<timerName>]| pId| tId| [<timeLeft>]|\n
```

or

```
timerStarted| [-t<time>]| [-n<pName>]| [-T<timerName>]| pId| tId| [<timeLeft>]|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.
- -n : name of process starting the timer,
- -T : name of the timer started,

parameters :

- <pId> : identifier of starter process,
- <tId> : identifier of timer, See “Common options and arguments” on page 22.

optional argument :

- <timeLeft> : time before timer times out.

### 3.12 - Timer cancellation

To trace the cancellation of a timer, the command syntax is:

```
tc| [-t<time>]| [-n<pName>]| [-T<timerName>]| pId| tId|\n
```

or

```
timerCancelled| [-t<time>]| [-n<pName>]| [-T<timerName>]| pId| tId|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.
- -n : name of process having starting the timer,
- -T : name of the stopped timer,

parameters :

- <pId> : process identifier,
- <tId> : identifier of timer stopped, See “Common options and arguments” on page 22.

### 3.13 - Timer timed out

To trace a timed out timer, the command syntax is:

```
tt| [-t<time>]| [-n<pName>]| [-T<timerName>]| pId| tId|\n
```

or

```
timerTimedOut| [-t<time>]| [-n<pName>]| [-T<timerName>]| pId| tId|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.
- -n : name of process having starting the timer,
- -T : name of the timer that times out ,

parameters :

- <pId> : process identifier,
- <tId> : identifier of the timer that times out, See “Common options and arguments” on page 22.

### 3.14 - Task state changed

To trace a task state has changed, the command syntax is:

```
ps| [-t<time>]| [-n<pName>]| pId| stateName|\n
```

or

```
taskChangedState| [-t<time>]| [-n<pName>]| pId| stateName|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.
- -n : name of process involved,

parameters :

- <pId> : process identifier,
- <stateName> : name of the new state for process.

### 3.15 - Action symbol

To trace an action in a lifeline, the command syntax is:

```
in| [-t<time>]| [-n<pName>]| <pId>| <message>|\n
```

or

```
information| [-t<time>]| [-n<pName>]| <pId>| <stateName>|\n
```

options :

- -t : time of event, See “Common options and arguments” on page 22.
- -n : name of process involved,

parameters :

- <pId> : process identifier,
- <stateName> : information to be displayed in the action symbol.

### 3.16 - Start a new MSC trace

To start a new MSC trace, the command syntax is:

```
n| [-f<filename>] |\n
or
newTrace| [-f<filename>] |\n
options:
```

- `-f<filename>` : file name is the file name of the new trace.

In graphical mode, this command closes the current trace if it exists, saves it if needed and launches a new viewer window. In no window mode, the previous trace is stopped and saved and a new trace starts, ready to accept commands.

### 3.17 - Pause MSC trace

To pause the current MSC trace, the command syntax is:

```
p/\n
or
pause/\n
```

The trace is resumed with the *resume* command.

### 3.18 - Resume MSC trace

To resume the current MSC trace, the command syntax is:

```
r/\n
or
resume/\n
```

This command is used only after the trace has been paused via the graphical interface or with the *pause* command.

### 3.19 - Close MSC trace

To stop the MSC trace, save and close the file, the command syntax is:

```
c|\n
or
close|\n
```

Use to indicate that the current trace will be definitively stopped and then saved. If a file name has been provided when the current trace started, it will be used to save the diagram; otherwise *MSC Tracer* will generate a file name like "mScTracer.rdd" in no window mode or open a dialog asking to choose a file name.

### 3.20 - Exit the MSC tracer

Exit *MSC Tracer* application. If a trace was running, it will be stopped and saved as if a command `close` was sent.

```
e/\n
or
exit/\n
```

### 3.21 - Set directory

To set the default directory, the command syntax is:

*dr/* <dir>\n

or

*setDirectory/* <dir>\n

parameters:

- <dir> : default directory.

Used to save traces when no file name is provided or when a file name without an entire path is given.

### 3.22 - Acknowledgment

In order to be sure all commands sent have been received, it is possible to ask for an acknowledgment from the MSC Tracer. The command syntax is:

*wa|* \n

or

*waitingForAck|* \n

The acknowledgment sent back after receiving the *waitingForAck* command is the string:

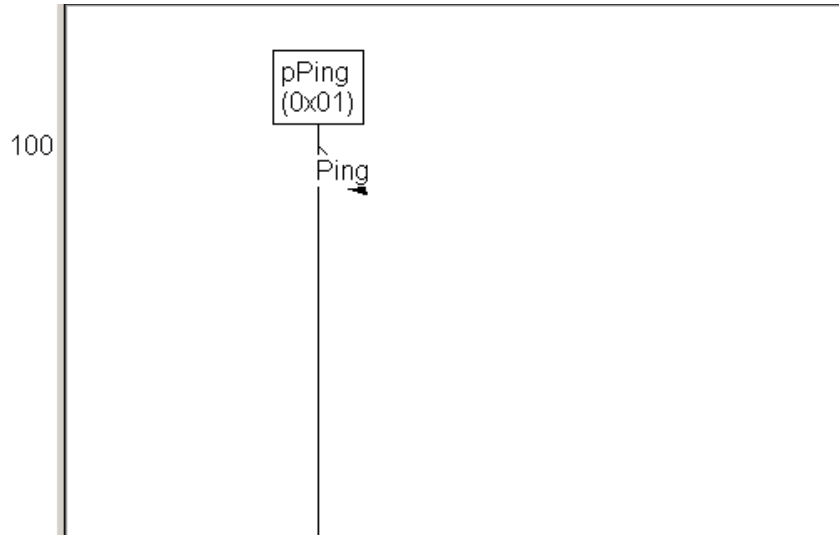
*ack*

## 4 - Example

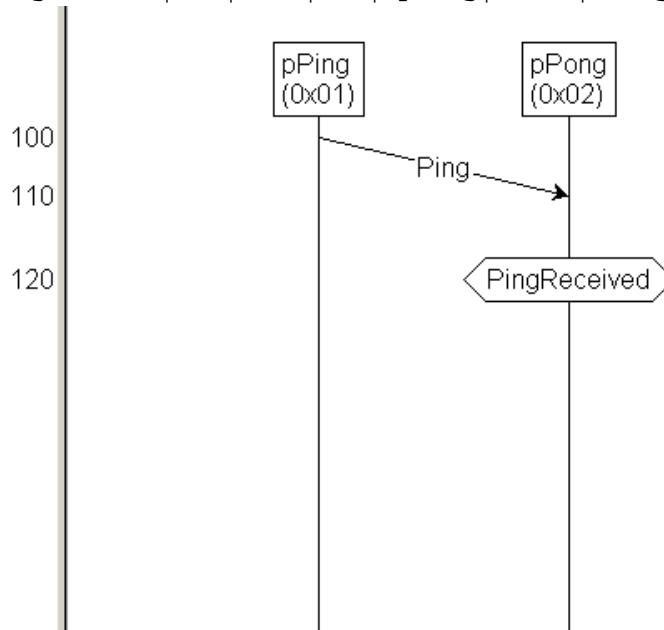
This example will illustrate the use of the *MSC Tracer*.

Commands sent :

- taskCreated| -n| pPing| 0x01|\n
- messageSent| -t| 100| -n| pPing| 0x01| 12| Ping|\n

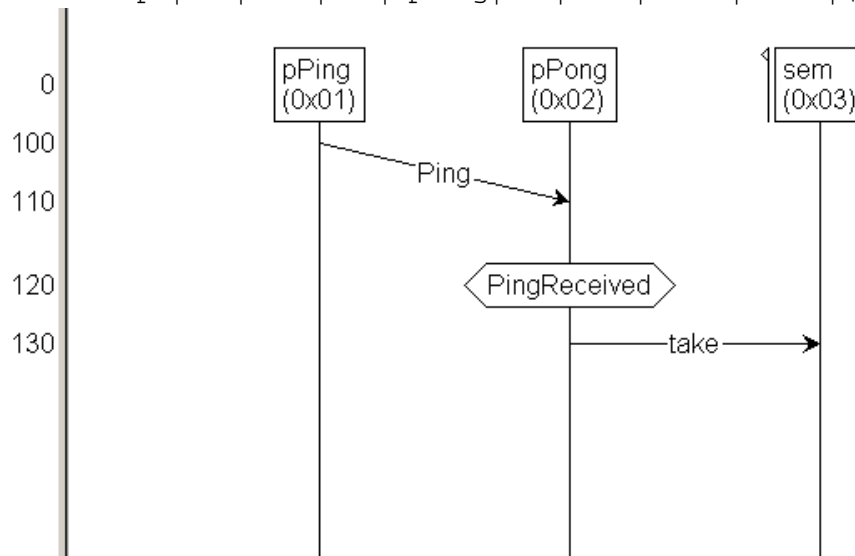


- messageReceived| -t| 110| -n| pPong| 0x02| 12| Ping|\n
- taskChangedState| -t| 120| -n| pPong| 0x02| PingReceived|\n

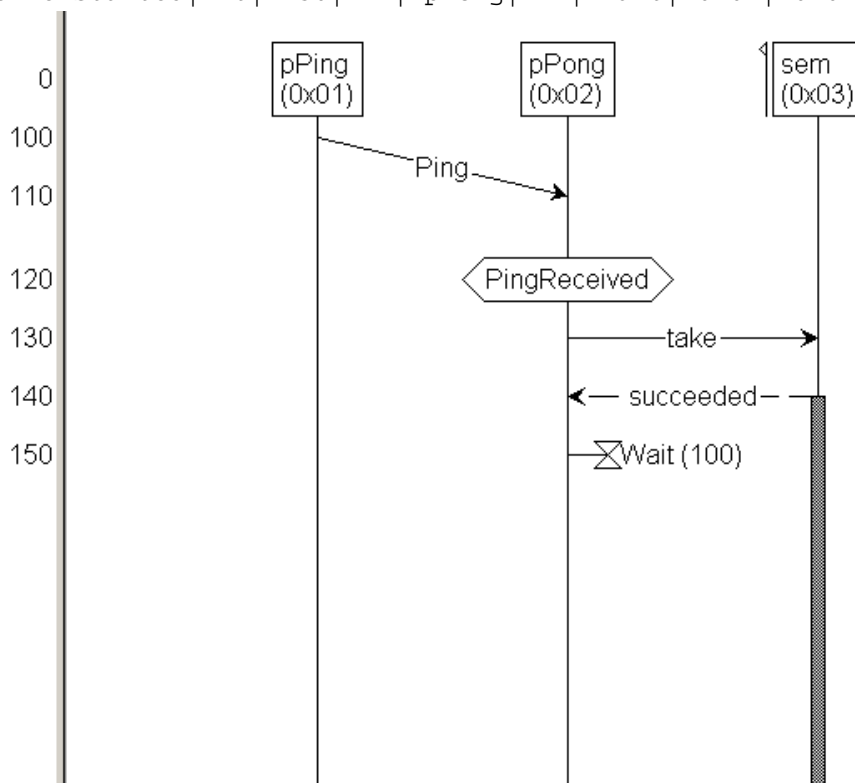




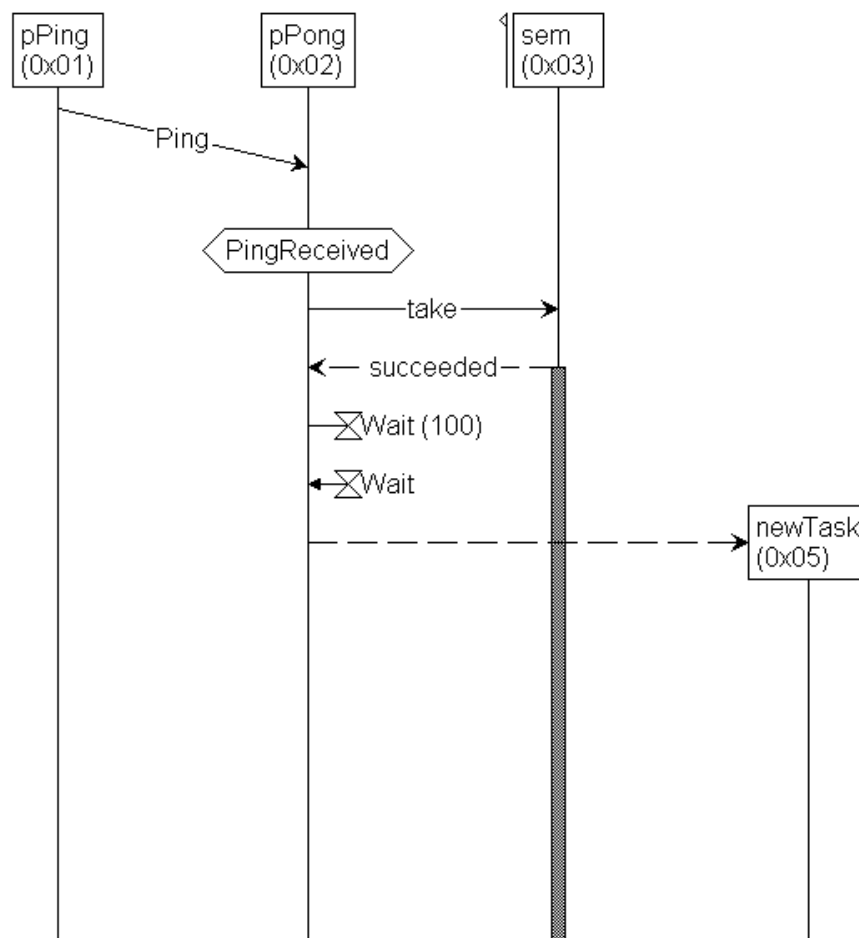
- semaphoreCreated| -s| sem| 0x03|\n
- takeAttempt| -t| 130| -n| pPong| -s| sem| 0x02| 0x03|\n



- takeSucceeded| -t| 140| -n| pPong| -s| sem| 0x02| 0x03|\n
- timerStarted| -t| 150| -n| pPong| -T| Wait| 0x02| 0x04| 100|\n



- timerTimedOut| -t| 250| -n| pPong| -T| Wait| 0x02| 0x04|\n
- taskCreated| -t| 260| -c| 0x02| -n| newTask| 0x05|\n



- giveSem| -t| 270| -n| pPong| -s| sem| 0x02| 0x03|\n
- taskDeleted| -t| 280| 0x04|\n

