

# A L<sup>A</sup>T<sub>E</sub>X macro package for Message Sequence Charts

## User Manual

Victor Bos    Ton van Deursen    Piotr Kordy    Sjouke Mauw  
Universit  du Luxembourg    Universit  du Luxembourg    Universit  du Luxembourg  
`victor.bos@ssf.fi`    `ton.vandeursen@uni.lu`    `piotr.kordy@uni.lu`    `sjouke.mauw@uni.lu`

Version A2.0, last update June 3, 2014  
Describing MSC macro package version 2.0

### Abstract

The MSC macro package facilitates the L<sup>A</sup>T<sub>E</sub>X user to easily include Message Sequence Charts in his texts. This document describes the design and use of the MSC macro package.

## Contents

<b>1</b>	<b>New</b>	<b>2</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
<b>3</b>	<b>Background and motivation</b>	<b>3</b>
<b>4</b>	<b>Installation, copyright and system requirements</b>	<b>4</b>
<b>5</b>	<b>Quick start</b>	<b>5</b>
<b>6</b>	<b>Use of the MSC macro package</b>	<b>6</b>
6.1	Options . . . . .	6
6.2	The MSC frame . . . . .	6
6.3	Levels . . . . .	7
6.4	Instances . . . . .	7
6.5	Messages . . . . .	9
6.6	Comments . . . . .	10
6.7	Actions . . . . .	12
6.8	Timers . . . . .	13
6.9	Time measurements . . . . .	14
6.10	Lost and found messages . . . . .	15
6.11	Conditions . . . . .	15
6.12	Generalized ordering . . . . .	16
6.13	Instance regions . . . . .	17
6.14	Instance creation and instance stop . . . . .	18
6.15	MSC references . . . . .	19

6.16	Inline expressions . . . . .	20
6.17	Gates . . . . .	21
6.18	High-level MSCs . . . . .	22
6.19	MSC documents . . . . .	25
<b>7</b>	<b>Style parameters</b>	<b>26</b>
<b>8</b>	<b>Example</b>	<b>26</b>
<b>9</b>	<b>Acknowledgments</b>	<b>27</b>

# 1 New

**Version 2.00** is the result of a complete rewrite of the package. The main difference is that the package is now implemented in TikZ instead of PSTricks, making it compatible with pdfLaTeX and BEAMER.

In addition the syntax has been slightly extended. It now supports the specification of all options as a comma separated list in square brackets, while remaining backward compatible. All lengths are now stored using PGF keys. As a result it is possible to change sizes of shapes individually. Support for colors is added.

We are much indebted to Adrian Farmadin (Masaryk University, Czech Republic) for initiating and contributing to this software rehaul.

**Version 1.17** slightly modifies the handling of the `offset` argument of the `\mess`, `\setstoptimer`, and `\settimeout` commands. All commands with an `offset` argument can now properly handle real numbers as argument. Furthermore, the `\topheaddist`, `\firstlevelheight`, and `\instheadheight` lengths are now taken into account when computing the height of the instance and the `\gatesymbolradius` length is now properly implemented.

The `\condition*` macro is now extended to dynamically resize a condition symbol that spans multiple instances. The `\condition*` and `\action*` macros, introduced in the previous release, no longer shrink to a size smaller than their non-starred counterparts. The `\condition` and `\action` have crossed-out counterparts `\ncondition` and `\naction` respectively.

A variation to the `\stop` command is implemented. Rather than drawing a cross to denote that an instance has stopped, the `\stop*` command draws an instance footer.

Finally, the `\create` command no longer requires the `creator` parameter to be specified. If it is not specified, no arrow is drawn from the creator to the dynamically created instance.

**Version 1.16** solves a bug that was due to a change in syntax of the `scalebox` macro in the PSTricks package.

The `\action` and `\condition` macros are extended with a starred option. The starred version of the macros automatically adjusts the size of the rectangle and hexagon based on the size of the contents.

**Version 1.13** has a reimplementaion of message commands in MSC diagrams. The affected commands are: `\create`, `\found`, `\lost`, `\mess`, and `\order`. The new implementation provides more control over the placement of message labels.

The command `\selfmesslabelpos` has been removed.

The bounding box bug has been partly solved. Now, a white `\fbox` is drawn around every (msc, mscdoc, hmsc) diagram. This makes it possible for `dvips -E` to compute the correct bounding box for the diagrams. Due to the `\fbox`, each diagram is extended with 0.3pt on each side (left, top, right, and bottom). This bugfix fails if the background is not white. The xdvi program shows the white `\fbox` in black with the result that diagrams have two visible frames. This seems to be a bug of xdvi.

The lines around comments (`\msccomment`, Section 6.6) are changed from gray into black. The reason for this is that the gray lines became invisible after converting the document to HTML.

## 2 Introduction

The MSC language is a visual language for the description of the interaction between different components of a system. This language is standardized by the ITU (International Telecommunication Union) in Recommendation Z.120 [2]. An introductory text on MSC can be found in [5]. MSCs have a wide application domain, ranging from requirements specification to testing and documentation. An example of a Message Sequence Chart is in Figure 1.

In order to support easy drawing of MSCs in L<sup>A</sup>T<sub>E</sub>X documents, we have developed the MSC macro package. The current version of the MSC macro package supports the following MSC constructs: MSC frame, instances (both single line width and double line width), messages (including self-messages and messages to the environment), actions, singular and combined timer events (set, timeout, reset, set-timeout, set-stop), lost and found messages, generalized order, conditions, coregions, activation regions, suspension regions, gates, instance creation, instance stop, time measurements, references, and inline expressions. In addition, there is support for HMSC's (high-level MSCs) and MSCdocs.

In this manual we explain the design and the use of the MSC macro package. For a complete overview of all features, we refer to the reference manual [1], which is included in the distribution under the name `refman.ps`. Another way to learn how to use the MSC macro package is to have a look at the L<sup>A</sup>T<sub>E</sub>X source code of the manual and the source code of the reference manual. They are included in the distribution under the names `manual.tex` and `refman.tex`, respectively. The MSC constructs are simply introduced as syntactic constructs. This paper is not meant to describe their use or meaning.

We list the backgrounds of the package and some design decisions in Section 3. Section 4 contains notes on installing the package. Section 5 contains an example of using the package. It allows the impatient reader to quickly start using the package. The details of using the package are explained in Section 6. In Section 7 the parameters are explained which determine detailed layout of the various symbols. A large but meaningless example is given in Section 8.

## 3 Background and motivation

Several commercial and non-commercial tools are available, which support drawing or generating Message Sequence Charts. However, these tools are in general not freely available and often not flexible enough to satisfy all user's wishes with respect to the layout and graphical appearance of an MSC. Therefore, people often use general drawing tools,

such as *xfig* to draw MSCs. However flexible this approach is, it takes quite some effort to produce nice MSC drawings in a tool which is not dedicated to MSCs. Furthermore, when drawing a number of MSCs it requires some preciseness in order to obtain a consistent set of MSCs.

For these reasons, we have started the design of a set of  $\text{\LaTeX}$  macros which support the drawing of MSCs. In this way, an MSC can be represented in  $\text{\LaTeX}$  in a textual format and compiled into e.g. PostScript.

We aimed at satisfying the following requirements and design decisions.

- The package should follow the ITU standard with respect to shape and placement of the symbols. (The current version supports the MSC2000 standard.)
- Static and dynamic semantics are not considered. The user is allowed to violate all semantical restrictions and draw inconsistent MSCs. The package only supports elementary syntactical requirements.
- The package should offer functionality at the right level of abstraction. Rather than supplying coordinates of pixels, the user should be able to express the placement of symbols in terms of *levels*. Nevertheless, the textual representation of MSCs as defined by the ITU standard has a level of abstraction which is too high for our purposes. It lacks information about the actual positioning of the MSC symbols, while we think that in our package this should be under user control.
- There should be only minimal automatic restructuring and layout of the MSC (e.g. the relative positioning of two messages should be as defined by the user, even if the messages are not causally ordered).
- The user can customize the appearance of the MSCs by manipulating an appropriate set of parameters.

## 4 Installation, copyright and system requirements

The MSC macro package is still under development. The authors appreciate any comments and suggestions for improvements. The most recent version of the package can be downloaded from <http://satoss.uni.lu/mscpackage/>.

The MSC macro package has a *LaTeX Project Public License* (LPPL), see <http://www.latex-project.org/lppl.txt>: As such, it is free of charge and can be freely distributed. Furthermore, it is allowed to make modifications to the package, provided that modified versions get different names. The authors accept no liability with respect to the functioning of the package.

The MSC macro package runs with  $\text{\LaTeX} 2_{\epsilon}$ . It has been tested with  $\text{\LaTeX} 2_{\epsilon}$  version dated 1998/06/01 using  $\text{\TeX}$  version 3.14159. The following additional packages are required: `tikz`, `calc`, and `xstring`. These packages are in general part of the standard  $\text{\LaTeX} 2_{\epsilon}$  distribution. These additional packages can be obtained from the *ctan* database for  $\text{\LaTeX}$  e.g. via the following URL: <http://www.tex.ac.uk/>. The `tikz` package is described shortly in [4] or in the *PGFmanual* [6]. The generated output can be previewed with pdf/ps previewing software. It may be needed to update all  $\text{\LaTeX}$  related software to a more recent version in order to smoothly work with the MSC macro package.

The MSC macro package can be installed easily. Just put the file `msc.sty` in a directory which is searched by  $\text{\LaTeX}$  for style files. The set of directories actually searched depends

on the  $\text{T}_{\text{E}}\text{X}$  installation, but often the *current directory* is included. UNIX users may have to set the environment variable  $\$TEXINPUTS$  to an appropriate value. For more details on this topic consult documentation of your  $\text{T}_{\text{E}}\text{X}$  installation.

## 5 Quick start

The MSC macro package is easy to use. Below is an example of the use of the package and Figure 1 shows the generated MSC. The MSC macro package is activated by the clause

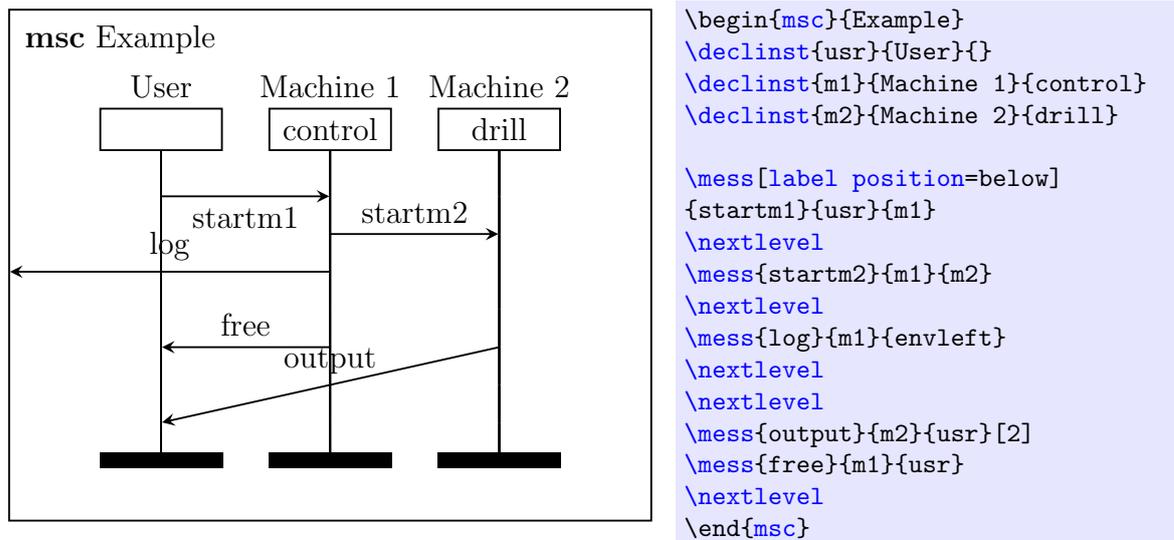


Figure 1: The generated MSC

`\usepackage{msc}`. This package contains, among others, a definition of the environment `msc`. This environment is used to draw MSCs. The MSC definition is surrounded by the clauses `\begin{msc}{Example}` and `\end{msc}`. The name of the MSC, `Example`, is displayed in the upper-left corner of the MSC.

The next four lines define the *instances*: `\declinst{m1}{Machine 1}{control}` defines an instance with *nickname* `m1` and a description consisting of two parts, namely, `Machine 1` and `control`. The nickname is used in all subsequent references to this instance. The first part of the description is drawn above the rectangular instance head symbol, and the second part of the description is drawn inside the instance head symbol.

The following lines contain the definitions of the messages. Every message has a source and a destination instance. The clause `\mess[label position=below]{startm1}{usr}{m1}` defines a message with name `startm1`, which goes from instance `usr` to instance `m1`. Between square brackets we can put optional list of options. Each option is comma separated from another option. In this example we specify the placement of the label to be below the message arrow.

In order to control the vertical placement of the messages, the MSC is divided into levels. At every level, any number of messages may start. The vertical position of the end point of a message is determined by the optional fourth argument of the message definition, as in the clause `\mess{output}{m2}{usr}[2]`. This argument is the vertical offset (in number of levels) between the start point of the message (i.e., the current level) and its end point. If the value is 0 the message is drawn horizontally. A negative offset means that the arrow

has an upward slope. Equivalently we could specify this value using option: `\mess[level shift=2]{output}{m2}{user}`.

The clause `\nextlevel` is used to advance to the next level.

## 6 Use of the MSC macro package

### 6.1 Options

Each command supported by the MSC macro package can be provided with a list of options. A list of options is surrounded by square brackets and options are separated by a comma. In fact options are PGF keys and the parsing of options is done internally by the TikZ package. All keys defined by the MSC macro package are prefixed by the `/msc` keyword. When a key is not fully specified (i.e., it is specified without the `/msc`, `/tikz` or some other prefix) the package will search first for the keys in the `/msc` namespace and if no key is found it will search the default `/tikz` name space.

In the description below, optional arguments are marked in green.

### 6.2 The MSC frame

The `msc` environment is used for making MSC definitions. Thus, such a definition looks as follows.

```
\begin{msc}[\green{<options>}]{<msc name>}
  <environment contents>
\end{msc}
```

This declares the `msc` environment. The frame and the header of the MSC are drawn. The argument `<msc name>` is the name of the MSC. The header of an MSC is formed from the keyword `msc`, followed by the `<msc name>`.

The size of the MSC frame is determined vertically by the number of levels occurring in the MSC (see Section 6.3) and horizontally by the number of instances (see Section 6.4).

`/msc/title position=<position>` (no default, initially `left`)

Controls the position of the header of the MSC. Possible values of `<position>` are: `left`, `right`, and `center`.

`/msc/msc keyword=<keyword>` (no default, initially `msc`)

Specifies the `<keyword>` in the header.

`/msc/title distance=<distance>` (no default, initially `0.2cm`)

Specifies the `<distance>` between left/right side of the frame and the MSC header.

`/msc/title top distance=<distance>` (no default, initially `0.2cm`)

Specifies the `<distance>` between top of the frame and the (top of the) MSC header.

`/msc/left environment distance=<distance>` (no default, initially `2cm`)

Specifies the `<distance>` between the left-most instance and the left side of the frame.

`/msc/right environment distance=<distance>` (no default, initially `2cm`)

Specifies the `<distance>` between the right-most instance and the right side of the frame.

- `/msc/environment distance` (no value)  
 Sets the keys `right environment distance` and `left environment distance` to the provided value.
- `/msc/instance distance=<distance>` (no default, initially 0.6cm)  
 Sets the `<distance>` between the (vertical) instance lines.
- `/msc/draw frame=<colour>` (default , initially )  
 Sets the colour of the frame. If you do not want to draw a frame, set `<colour>` to `none`. When `<colour>` is empty, then the last colour is used, usually black.
- `/msc/draw grid=<grid type>` (default grid, initially none)  
 Specifies if we want to draw the help grid in msc. Possible values for `<grid type>` are `grid`, `color grid` and `none`.

### 6.3 Levels

An MSC is vertically divided in *levels*. All events in an MSC are attached to a certain level, or stretch out over several levels. Any number of events can be drawn at a certain level. An event will always be drawn (or started) at the current level, unless a level offset is specified (see e.g. the `\mess` command in Section 6.5). The `<level offset>` is a positive integer number, which denotes at which level, relative to the current level, an event should be drawn. Drawing starts at level 0. The following command is used to advance to the next level.

`\nextlevel[<level offset>]`

The `<level offset>` is a positive integer value which denotes the number of levels to advance. By default, the value of `<level offset>` is 1, which means drawing continues at the next level.

The following keys influence the `\nextlevel` command:

- `/msc/first level height=<distance>` (no default, initially 0.6cm)  
 Specifies the `<distance>` between the instance start symbol and the first level.
- `/msc/level height=<distance>` (no default, initially 0.5cm)  
 Specifies the `<distance>` between two consecutive levels.
- `/msc/last level height=<distance>` (no default, initially 0.4cm)  
 Specifies the `<distance>` between the last level and the instance end symbol.

Figure ?? on page ?? shows all lengths of the MSC macro package.

### 6.4 Instances

All instances have to be declared before they can be used. An instance consists of an instance head symbol with an associated name, an instance axis and an instance end symbol. Normal instances have a single line axis. Fat instances have a double line axis. The order of the instance declarations determines the order in which the instances occur in the drawing.

An instance is declared with the following command.

`\declinst*{<nickname>}{<instance name above>}{<instance name within>}`

The starred version produces a fat instance. The `<nickname>` is used for referring to this instance in the rest of the MSC definition. The `<instance name above>` is put above the instance head symbol. The `<instance name within>` is put inside the instance head symbol. MSC definition.

The keys associated with the command:

`/msc/head height=<height>` (no default, initially 0.55cm)  
Specifies the minimum *<height>* of the instance head symbol.

`/msc/instance width=<width>` (no default, initially 1.6cm)  
Specifies the minimum *<width>* of the instance head symbol.

`/msc/head top distance=<distance>` (no default, initially 1.3cm)  
Specifies the *<distance>* between top of the head symbols and frame.

`/msc/foot height=<height>` (no default, initially 0.2cm)  
Specifies the *<height>* of the instance foot symbol.

`/msc/foot distance=<distance>` (no default, initially 0.7cm)  
Specifies the *<distance>* between the instance foot symbol and the bottom of the MSC frame.

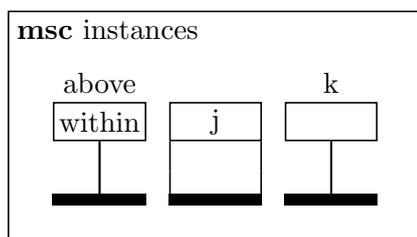
`/msc/environment distance=<distance>` (no default, initially )  
Specifies the *<distance>* between the edge of the MSC and the first/last instance axis.

`/msc/instance width=<width>` (no default, initially 1.6cm)  
Specifies the *<width>* of the instance head and foot symbols.

`/msc/label distance=<distance>` (no default, initially 1ex)  
Specifies the *<distance>* between the instance head symbol and the part of the instance name drawn above the head symbol.

`/msc/draw head=<colour>` (no default, initially )  
Sets the colour of the instance head. If you do not want to draw an instance head, set *<colour>* to **none**. When *<colour>* is empty, then the last colour is used, usually black.

The following MSC shows the declaration of an MSC with three instances. The first and the last are normal instances (one line axis), whereas the second is a fat instance (double line axis). The optional parameter **values small**, indicates that the small drawing style should be used (see Section 7).



```
\begin{msc}[small values]{instances}
\declinst{i}{above}{within}
\declinst*{j}{}{j}
\declinst{k}{k}{}
\end{msc}
```

## 6.5 Messages

A message is denoted by an arrow from the sending instance to the receiving instance. The instances are referred to by their nicknames. A message is defined with the following command.

```
\mess*[\langle options \rangle]{\langle name \rangle}{\langle sender \rangle}{\langle receiver \rangle}[\langle leveloffset \rangle]
```

The  $\langle name \rangle$  of the message may be any string. The MSC macro package processes the  $\langle name \rangle$  argument in LR-mode, see [3, page 36]. This means that the string will consist of one line. To generate multi-line message names, use the standard `parbox` command, see the *Tricks* section in the reference manual [1]. By default, the name of a message label is drawn above the center of the arrow, but the optional keys `label position`, `side`, and `pos` influence the actual location, as described below. The arrow starts at the current level at the sending instance. The arrow ends at the current level plus the level  $\langle leveloffset \rangle$ , at the receiving instance.  $\langle leveloffset \rangle$  can be also modified by specifying keys `level shift` and `offset`. The  $\langle sender \rangle$  and  $\langle receiver \rangle$  should be the nicknames of declared instances. Messages to or from the environment (i.e., the left or the right side of the MSC frame) can be specified by setting the sender or the receiver argument to one of the values `envleft` or `envright`. (Note: Since instances and environments are treated equally in the implementation, at every position where the nickname of an instance is required, also `envleft` and `envright` are allowed.)

In case the sending and the receiving instance are the same, the message is a *self message*. A self message is drawn as a polyline connecting the instance axis to itself.

The starred version of the command, `\mess*`, produces the same result as `\mess`, except that the arrow is drawn with a dashed line. This can be used to draw a *method reply* (see [2]).

The following keys influence the messages:

```
/msc/level shift=\langle integer \rangle (no default, initially 0)
```

For non-self message,  $\langle integer \rangle$  determines at which level message should arrive relative to the starting level of the message.

```
/msc/offset=\langle integer \rangle (no default, initially 2)
```

For self message,  $\langle integer \rangle$  determines how many levels the message arrow should advance.

```
/msc/side=\langle side \rangle (no default, initially left)
```

Determine the position of the arrow with respect to the instance axis. Valid values for  $\langle side \rangle$  are `left` and `right`. In case of a non-self message the key is ignored.

```
/msc/label position=\langle position \rangle (no default, initially above)
```

Determines the position of the label. In case of non-self message, valid values for  $\langle position \rangle$  are `above`, `below`, `above left`, `above right`, `below left` and `below right`. In case of self message valid values are `left` and `right` and the default value is taken from the value of the key `side`. As the implementation uses corresponding `/tikz` keys the `label position=` part can be skipped.

```
/msc/pos=\langle real \rangle (no default, initially 0.5)
```

Determines the relative distance of the message label to the beginning of the message. Valid values for  $\langle real \rangle$  are real numbers in the closed interval  $[0, 1]$ . The key works in similar way as `/tikz/pos` key.

`/msc/self message width=<width>` (no default, initially 0.6cm)  
 Controls the  $\langle width \rangle$  of the polyline used for drawing self-message.

`/msc/label distance=<distance>` (no default, initially 1ex)  
 Specifies the  $\langle distance \rangle$  between the label of a message and the message arrow. It has the same effect as specifying the key `/tikz/inner sep`.

`/msc/arrow scale=<scale>` (no default, initially 1.5)  
 The  $\langle scale \rangle$  specifies by which factor should the message arrow be scaled. Influences all arrows drawn in `msc` environment.

In Figure 2 we see MSC that shows an example of the use of messages. In this sample MSC and the following MSCs in this section we will not list the complete textual representations of the MSCs. For brevity we omit the environment call and the declarations of the instances. Note the final `\nextlevel` command which is needed to make the instance axis long enough to receive message a.

Figure 3 shows different options that can be used to position label. We use the keys `pos`, `label position` and `side` to place the labels. Additionally we use `/tikz/anchor` key, which is an alternative way to specify the placement of the label (see PGF Manual [6]).

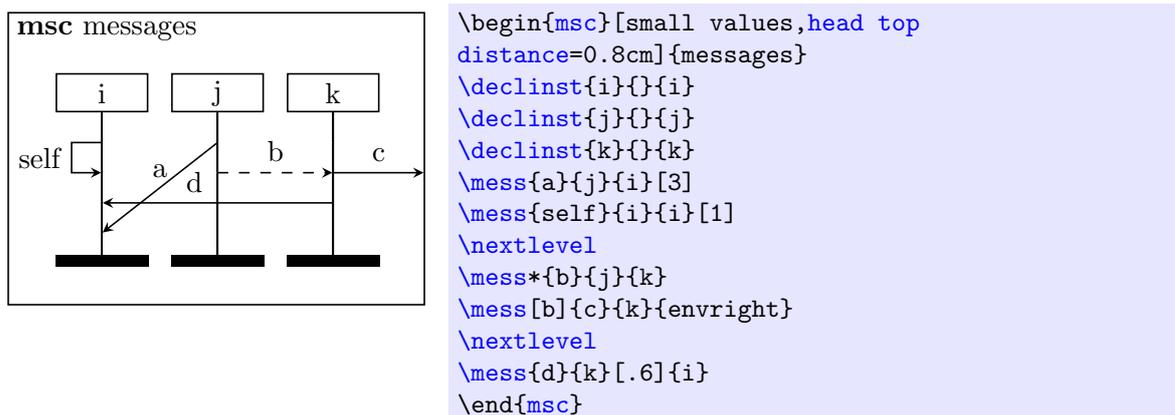


Figure 2: Example showing msc messages

## 6.6 Comments

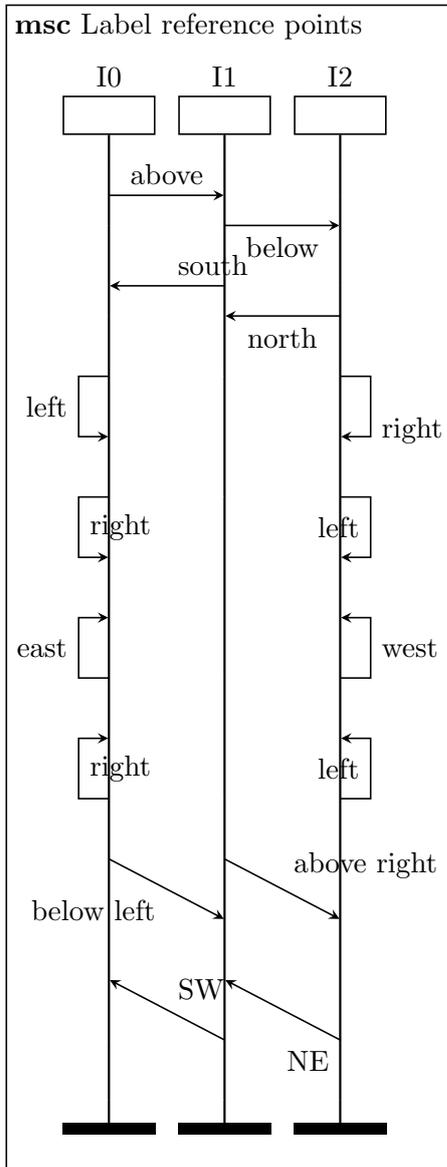
Comments are additional texts to clarify (events on) an instance. The following command can be used to add comments to an MSC diagram.

`\msccomment [<options>]{<text>}{<instance>}`

The  $\langle instance \rangle$  parameter defines the instance to which the comment is attached. The text of the comment is specified by the  $\langle text \rangle$  parameter and is processed in LR-mode. The key `side` affect the horizontal position relative to its instance.

The following key affects the drawing of the comment:

`/msc/msccomment distance=<distance>` (no default, initially 1.1cm)  
 Specifies the  $\langle distance \rangle$  at which the comment will be placed from the  $\langle instance \rangle$  instance.



```

\begin{msc}[small values]{Label reference points}
\declinst{m0}{I0}{}
\declinst{m1}{I1}{}
\declinst{m2}{I2}{}
\nextlevel

\mess[label position=above]{above}{m0}{m1}
\nextlevel
\mess[label position=below]{below}{m1}{m2}
\nextlevel[2]

\mess[anchor=south,pos=0.1]{south}{m1}{m0}
\nextlevel
\mess[anchor=north]{north}{m2}{m1}
\nextlevel[2]

\mess[label position=left]{left}{m0}{m0}[2]
\mess[side=right,pos=0.9]{right}{m2}{m2}[2]
\nextlevel[4]
\mess[label position=right]{right}{m0}{m0}[2]
\mess[left,side=right]{left}{m2}{m2}[2]
\nextlevel[6]

\mess[anchor=east]{east}{m0}{m0}[-2]
\mess[anchor=west,side=right]{west}{m2}{m2}[-2]
\nextlevel[4]
\mess[right]{right}{m0}{m0}[-2]
\mess[side=right,left]{left}{m2}{m2}[-2]
\nextlevel[2]

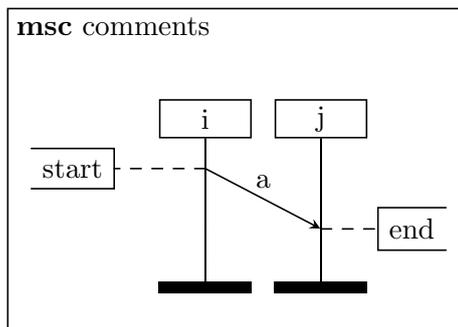
\mess[below left]{below left}{m0}{m1}[2]
\mess[above right]{above right}{m1}{m2}[2]
\nextlevel[6]

\mess[anchor=south west]{SW}{m1}{m0}[-2]
\mess[anchor=north east,pos=0]{NE}{m2}{m1}[-2]
\nextlevel[2]
\end{msc}

```

Figure 3: Different placing the message labels

The following diagram shows how to use comments. In this diagram we modify the distance between the frame and the instances in order to fit the comments inside the frame.



```
\begin{msc}[small values,
  left environment distance=2.6cm,
  right environment distance=1.9cm]{comments}
\declinst{i}{}{i}
\declinst{j}{}{j}
\mess{a}{i}{j}[2]
\msccomment[msccomment
distance=1.2cm]{start}{i}
\nextlevel[2]
\msccomment[side=right]{end}{j}
\nextlevel
\end{msc}
```

## 6.7 Actions

An instance can perform an action, which is denoted by a rectangle. There are two commands to draw actions:

`\action*[\langle options \rangle]{\langle name \rangle}{\langle instance \rangle}`

The action is attached at the current level to the  $\langle instance \rangle$ . The  $\langle name \rangle$  is centered inside the action symbol and is processed in LR-mode. The height and width of the action symbol are adjusted so that the  $\langle name \rangle$  fits inside a rectangle.

The starred version of the command, `\action*`, produces the same result as `\action`, except that the width of the action is not adjusted.

The following keys determine the detailed drawing of the action symbol:

`/msc/action width=\langle width \rangle` (no default, initially 1.25cm)

Specifies the  $\langle width \rangle$  of the action symbol.

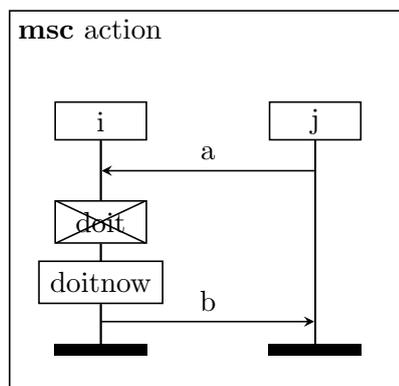
`/msc/action height=\langle height \rangle` (no default, initially 0.6cm)

Specifies the  $\langle height \rangle$  of the action symbol.

`\naction*[\langle options \rangle]{\langle name \rangle}{\langle instance \rangle}`

The `\naction` command produces an action box with a diagonal cross in it. It is otherwise identical to the `\action` command.

The following example shows that after an action often a multiple level increment is needed to obtain nice pictures.



```
\begin{msc}[small values, instance
distance=1.6cm]{action}
\declinst{i}{}{i}
\declinst{j}{}{j}
\mess{a}{j}{i}
\nextlevel
\naction{doit}{i}
\nextlevel[2]
\action*{doitnow}{i}
\nextlevel[2]
\mess{b}{i}{j}
\end{msc}
```

## 6.8 Timers

There are five commands to draw timer events.

`\settimer` [*options*] {*name*} {*instance*}

Setting of a timer is drawn as a line connecting the *instance* to the *hour glass* symbol. The *name* is put near this symbol. The key `side` is determines on which side of the instance axis the timer is drawn. The key `self message width` determines the length of the arm between the symbol and the instance axis. The key `label distance` determines the distance between the name and the timer symbol. The key `arrow scale` determines the scaling of the arrow. *hour glass* symbol to the *instance*.

`\timeout` [*options*] {*name*} {*instance*}

A time-out is represented by an arrow from an *hour glass* symbol to the *instance*.

`\stoptimer` [*options*] {*name*} {*instance*}

Stopping a timer is drawn as a line connecting the *instance* with the timer stop symbol (denoted by a cross).

`\settimeout` [*options*] {*name*} {*instance*} [*offset*]

The command `\settimeout` is a combination of the setting of a timer and a time out. The *offset* denotes the number of levels between the two events.

`\setstoptimer` [*options*] {*name*} {*instance*} [*offset*]

Likewise, `\setstoptimer` is a combination of the setting of a timer and stopping a timer.

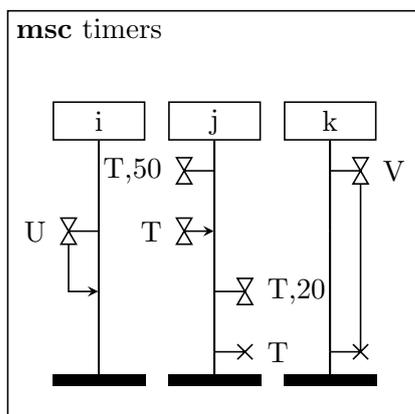
`/msc/offset`=*offset* (no default, initially 2)

The key holds the value of *offset*. It denotes the number of levels between the two events.

`/msc/timer width`=*width* (no default, initially 0.3cm)

The key determines the *width* of the hour glass and the time out symbols.

The various timer symbols are shown in the following example.



```
\begin{msc}[small values]{timers}
\declinst{i}{i}
\declinst{j}{j}
\declinst{k}{k}
\settimer{T,50}{j}
\setstoptimer[r]{V}{k}[6]
\nextlevel[2]
\timeout{T}{j}
\settimeout{U}{i}
\nextlevel[2]
\settimer[r]{T,20}{j}
\nextlevel[2]
\stoptimer[r]{T}{j}
\end{msc}
```

## 6.9 Time measurements

There are several commands to add time measurements to an MSC.

`\mscmark` [*options*] {*name*} {*instance*}

This command attach an absolute time stamp to an event on an *instance*. The *name* is the text attached to the mark symbol, which is a dashed polyline. The position of the mark is determined by the keys `position` (above, below) and `side` (left,right).

`/msc/position`=*position* (no default, initially above)

The key determines the position of the mark symbol relative to the marked event. The allowed values for *position* are `above`, `mid` and `below`.

`\measure`\* [*options*] {*name*} {*instance1*} {*instance2*} [*offset*]

This command connects two events from *instance1* and *instance2*. The first event is at the current level. The second event is *offset* levels lower than the first event. Not that instead giving optional argument *offset* we can specify the key `offset`. The *name* is attached to the measure symbol. The measure symbol can be placed at the left or right of *instance1*. This is determined by the value of the key `side`. In the starred version of the command arrow heads are at the outside of the measured interval.

In case the two events are far apart, the measure may be split in two parts. We use two commands to draw this parts:

`\measurestart`\* [*options*] {*name*} {*instance*} {*gate*}

Draws the first part of the measure. The points where these two parts should be connected are drawn by a small circle, to which the text *gate* is attached. In the starred version of the command arrow heads are at the outside of the measured interval.

`\measureend`\* [*options*] {*name*} {*instance*} {*gate*}

Draws the first part of the measure in similar way as `\measurestart` command.

Several keys can be used to control the detailed layout of the time measurement symbols.

The `label distance` key specify the distance between the label of a measurement and the measurement symbol.

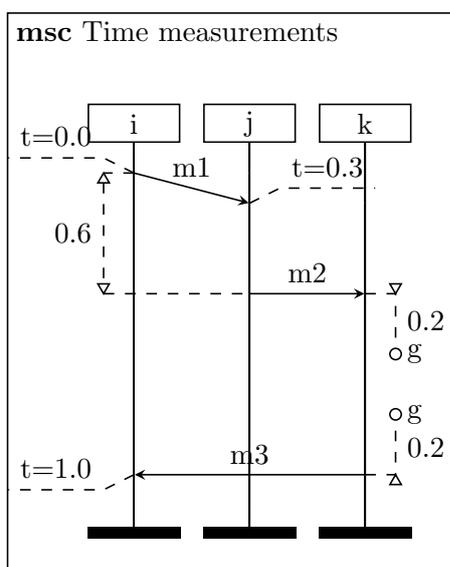
`/msc/measure distance`=*distance* (no default, initially 0.6cm)

This key specifies at which *distance* the measurement symbols are drawn from the *instance* line.

`/msc/measure symbol width`=*width* (no default, initially 0.2cm)

Specifies the *width* of the measurement triangle symbol.

The following example illustrates marks and measurements in an MSC..



```
\begin{msc}[small values]{Time measurements}
\declinst{i}{i}
\declinst{j}{j}
\declinst{k}{k}
\mess{m1}{i}{j}[1]
\mscmark{t=0.0}{i}
\measure{0.6}{i}{j}[4]
\nextlevel
\mscmark[position=above,side=right]{t=0.3}{j}
\nextlevel[3]
\mess{m2}{j}{k}
\measurestart*[side=right]{0.2}{k}{g}
\nextlevel[6]
\mess{m3}{k}{i}
\mscmark[position=below,side=left]{t=1.0}{i}
\measureend*[side=right]{0.2}{k}{g}
\nextlevel
\end{msc}
```

## 6.10 Lost and found messages

`\lost`[*options*]{*name*}{*gate*}{*instance*}

A lost message is denoted by an arrow starting at an instance and ending at a filled circle. The argument *instance* determines the instance to which the arrow is attached. The *name* of the message is put above the message arrow. The *gate* is a text associated to the circle. The keys `side`, `label position`, `pos`, `label distance` and `self message width` function in similar way as in the `\mess` command (Section 6.5). That is, `side` controls the placement of the lost or found message with respect to the instance axis. The `label position` and `pos` controls the placement of the *name* with respect to the message arrow. The key `self message width` determines the length of the arrow and `label distance` specifies the distance between the name and the arrow.

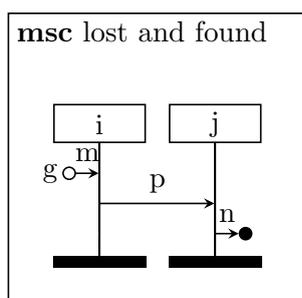
`\found`[*options*]{*name*}{*gate*}{*instance*}

A found message is denoted by an arrow starting at an open circle and ending at an instance. Otherwise the command is identical to the `\lost` command.

`/msc/lost symbol radius`=*radius* (no default, initially 0.24cm)

Determines the *radius* of the circle for lost and found messages.

The following example shows a found and a lost message.



```
\begin{msc}[small values]{lost and found}
\declinst{i}{}{i}
\declinst{j}{}{j}
\found{m}{g}{i}
\nextlevel
\mess{p}{i}{j}
\nextlevel
\lost[side=right]{n}{}{j}
\end{msc}
```

## 6.11 Conditions

There are two commands to draw condition boxes:

`\condition`\*[*options*]{*text*}{*instance list*}

A condition is denoted by a hexagon. It is used to express that the system has entered a certain state. A condition relates to a number of instances. All conditions which take part in the condition are covered by the condition symbol. The other instances are drawn through the condition symbol.

The *text* is placed in the center of the condition. The *instance list* expresses which instances take part in the condition. It is a list of nicknames of instances, separated by commas. Take care not to add extra white space around the nicknames, since this is considered part of the nickname in L<sup>A</sup>T<sub>E</sub>X. The order in which the instances are listed is immaterial.

The starred version of the command `\condition`\*, produces the same result as the command `\condition`, except that the width of the condition symbol is not adjusted to fit the contents of the hexagon.

`\ncondition`\*[*options*]{*text*}{*instance list*}

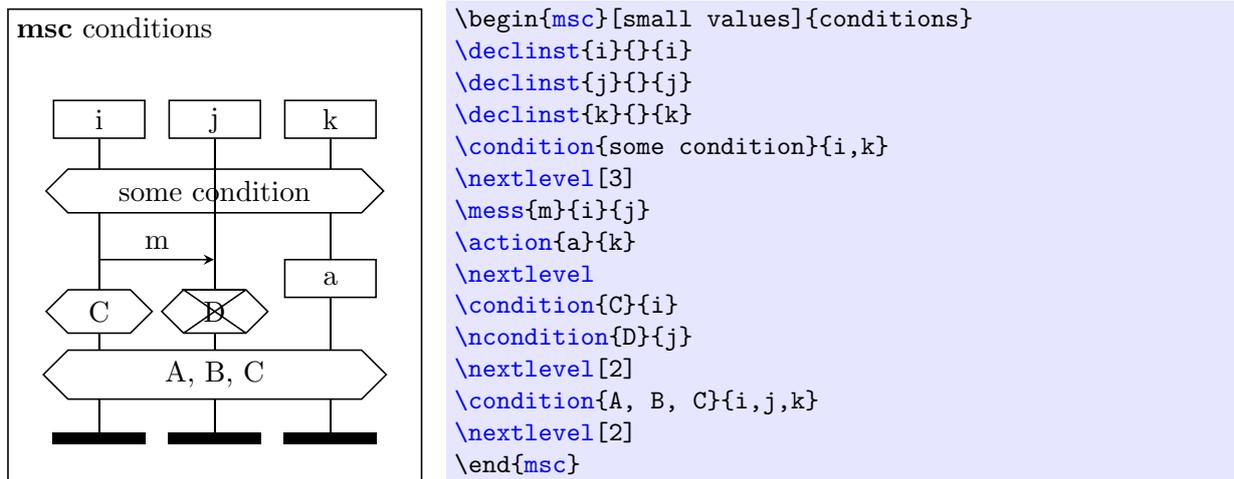
The `\ncondition` command produces a condition with a diagonal cross in it. It is otherwise identical to the `\condition` command.

There are several keys that controls the shape of the condition symbol:

`/msc/condition height=<height>` (no default, initially 0.6cm)  
 Determines the *<height>* of the condition symbol.

`/msc/condition overlap=<distance>` (no default, initially 0.5cm)  
 The *<distance>* determines the width of the part of the condition symbol which extends over the rightmost/leftmost contained instance axis.

The following example contains some conditions.



## 6.12 Generalized ordering

A generalized order is treated much like a regular message (see Section 6.5). There are three differences: a generalized order is drawn with a dotted line, it has no label, and the arrow head is in the middle of the line. A generalized order is defined with the following command.

`\order [<options>] {<sender>} {<receiver>} [<level offset>]`

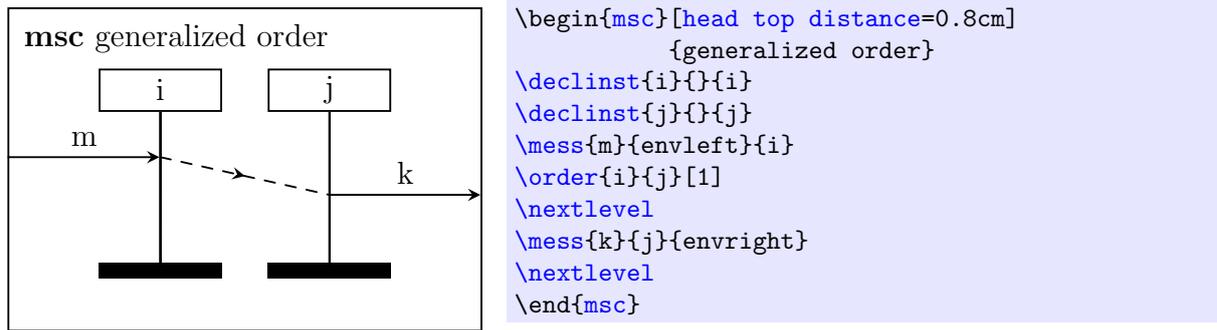
The *<sender>* and *<receiver>* are the nicknames of the instances which are connected by the generalized ordering symbol. At the *<receiver>* instance, the generalized ordering symbol ends at the current level plus the *<level offset>*. The *<level offset>* is an optional integer value, with default 0.

In case *<sender>* and *<receiver>* denote the same instance, the order is a *self order*. The placement of the order arrow of a self order is controlled by the key `side`. It can have values `left` (meaning that the ordering symbol is drawn left of the instance axis) and `right` (meaning that the ordering symbol is drawn right of the instance axis). By default it is drawn at the left side of the instance.

The position of the arrow is controlled with the `pos` key. The key `selfmessage width` specify the width of the polyline used for drawing orderings on a single instance axis. The key `arrow scale` controls the size of the arrow.

Orderings to or from the environment (i.e., the left or the right side of the MSC frame) can be specified by setting the sender or the receiver argument to the value `envleft` or `envright`.

An example of a generalized order is given in the following diagram.



## 6.13 Instance regions

A part of the instance axis can be drawn in a different style. Such a part is called an *instance region*. The following regions are supported: *coregion* (the instance axis is dashed, which means that the order of the attached instances is immaterial), *suspension region* (a small rectangle with dashed left and right sides, which denotes that the instance is suspended), *activation region* (a small filled rectangle, which denotes that the instance has control).

The following commands are used to draw an instance region.

`\regionstart` [*options*] {*type*} {*instance name*}

After this command the instance axis of *instance name* is drawn in the shape determined by *type* starting from the current level. The *type* can have values *coregion*, *suspension*, and *activation*.

`\regionend` {*instance name*}

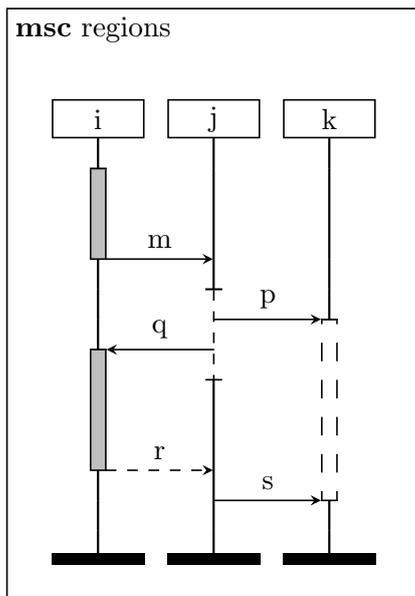
After this command the instance axis for the *instance name* is drawn in the usual way. In other words it ends current region.

The following keys control the shape of on instance region:

`/msc/region bar width`=*width* (no default, initially 0.15cm)  
Determines the *width* of the coregion start and end symbol.

`/msc/region width`= Determines the width of the activation and suspension rectangle. (no default, initially 0.4cm)

In the following example, several instance regions are demonstrated. Note that the relative order of `\mess`, `\regionstart`, and `\regionend` commands does not matter. Also notice that the second activation region on *i* ends with a method reply (which is produced by the command `\mess*`).



```

\begin{msc}[small values]{regions}
\declinst{i}{i}
\declinst{j}{j}
\declinst{k}{k}
\regionstart{activation}{i} \nextlevel[3]
\regionend{i}
\mess{m}{i}{j} \nextlevel
\regionstart{coregion}{j} \nextlevel
\regionstart{suspension}{k}
\mess{p}{j}{k} \nextlevel
\regionstart{activation}{i}
\mess{q}{j}{i} \nextlevel
\regionend{j} \nextlevel[3]
\regionend{i}
\mess*{r}{i}{j} \nextlevel
\mess{s}{j}{k}
\regionend{k} \nextlevel
\end{msc}

```

## 6.14 Instance creation and instance stop

The MSC language offers constructs for dynamic instance creation and instance destruction. An instance can dynamically create another instance by issuing a create command. An instance creation is drawn as a dashed message arrow. At the side of the arrow head, the instance head symbol for the created instance is drawn. An instance end symbol does not denote the end of the specified process, but merely the end of its current description. Therefore, a different symbol is needed which denotes that an instance stops before the end of the MSC in which it is contained. The instance stop symbol is a cross.

The following commands are used for instance creation and instance stop.

**\dummyinst**[*<options>*]{*<created instance>*}

This command reserve space for on instance which will be created dynamically. **\dummyinst** command is mixed with the declarations of normal instances (see the **\declinst** command, Section 6.4). The argument *<created instance>* is the nickname of the instance that will be created later.

**\startinst**[*<options>*]{*<instance name>*}{*<text above>*}{*<text inside>*}

This command create an instance. The created instance must have been declared first with the **\dummyinst** command. The name of the created instance consists of two parts. The part called *<text above>* is placed above the instance head and the *<text inside>* is centered within the instance head.

**\create**[*<options>*]{*<name>*}{*<creator>*}{*<instance name>*}{*<text above>*}{*<text inside>*}

This command is identical to **\dummyinst** command except it allows to specify the creator of the instance. The command results in a horizontal message arrow labelled with *<name>*. The arrow starts at the current level at the instance with the nickname *<creator>* and ends at the current level at the instance head of the instance with nickname *<instance name>*. As with normal messages, placement of the message label is controlled by the keys **label position** and **pos**.

`\stop*[\langle options \rangle]{\langle instance name \rangle}`

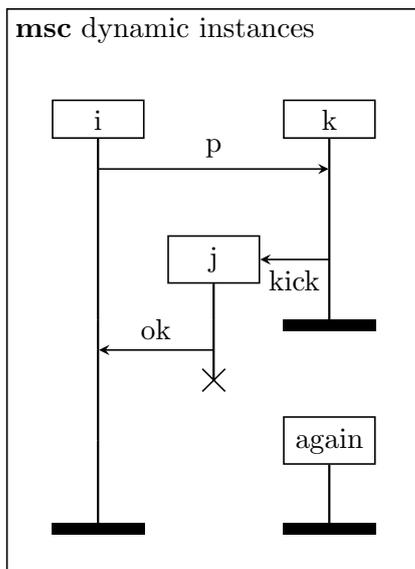
An instance is stopped with the `\stop` command. The  $\langle instance \rangle$  is the nickname of the stopped instance. The instance axis is not drawn any further below the level at which the `\stop` command is issued. Also, the instance foot symbol is not drawn.

If the `\stop*` command is used, then the instance foot symbol is drawn instead of the stop symbol.

`/msc/stop width=\langle width \rangle` (no default, initially 0.5cm)

The  $\langle width \rangle$  determines the size of the stop symbol.

Take care not to specify any events on an instance which has not yet been created or which has already been stopped. However, it is possible to create an instance after it has stopped, as showed in the next example.



```
\begin{msc}[small values]{dynamic instances}
\declinst{i}{}{i}
\dummyinst{j}
\declinst{k}{}{k}

\mess{p}{i}{k}
\nextlevel[3]
\create[label position=below]{kick}{k}{j}{}{j}
\nextlevel[2]
\stop*{k}
\nextlevel
\mess{ok}{j}{i}
\nextlevel
\stop{j}
\nextlevel[2]
\startinst{k}{}{again}
\nextlevel[2]
\end{msc}
```

## 6.15 MSC references

Within an MSC a reference to other MSCs can be included. Such a reference is drawn as a rectangle with rounded corners, covering part of the MSC. The following commands are used to draw MSC references.

`\referencestart[\langle options \rangle]{\langle nickname \rangle}{\langle text \rangle}{\langle left instance \rangle}{\langle right instance \rangle}`

The reference symbol starts at the level where the `\referencestart` command is used, and ends at the level where the corresponding `\referenceend` command occurs. These commands correspond if they have the same  $\langle nickname \rangle$ . The  $\langle text \rangle$  is placed in the center of the reference symbol. The reference covers all instances from  $\langle leftinstance \rangle$  to  $\langle rightinstance \rangle$ .

The left and right edge of the reference symbol are  $\langle nickname \rangle left$  and  $\langle nickname \rangle right$ , where  $\langle nickname \rangle$  is the nickname of the MSC reference as defined in the `\referencestart` command. These names can be used at every place where the nickname of an instance is required, e.g. as the sender or receiver of a message.

`\referenceend[\langle options \rangle]{\langle nickname \rangle}`

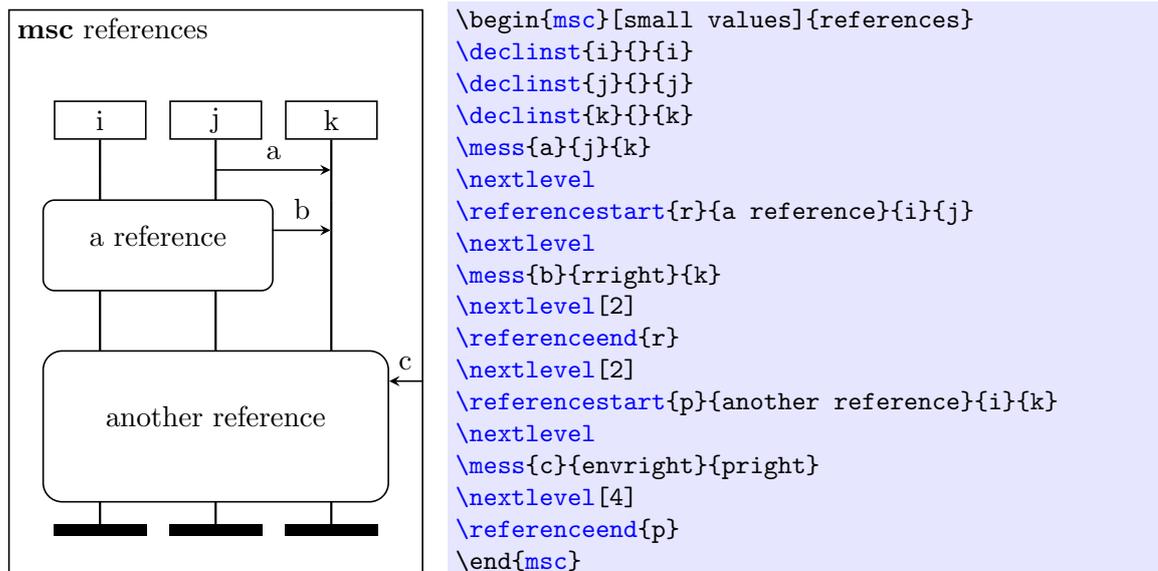
This command end the reference  $\langle nickname \rangle$  that has been started with the `\referencestart` command.

The following keys affect the drawing of reference symbol:

`/msc/left reference overlap=<distance>` (no default, initially 1cm)  
 Determines the *<distance>* between the left edge of the reference symbol and the leftmost covered instance axis.

`/msc/right reference overlap=<distance>` (no default, initially 1cm)  
 Determines the *<distance>* between the right edge of the reference symbol and the rightmost covered instance axis.

The following example shows the usage of references:



## 6.16 Inline expressions

An inline expression is a part of an MSC on which an operation is defined. A rectangle surrounds the part of the MSC containing the operands. The operands are separated by horizontal dashed lines. The operator is placed in the upper left corner of the inline expression symbol. The following commands are used to draw inline expressions.

`\inlinestart` [*<options>*] {*<nickname>*} {*<operator>*} {*<left instance>*} {*<right instance>*}

Starts the inline expression at the current level. The inline expression spans over the instances from *<left instance>* to *<right instance>*. The *<operand>* is placed in the upper left corner of the rectangle. The left and right edge of the inline expression symbol are named *<nickname>left* and *<nickname>right*, where *<nickname>* is the nickname of the inline expression as defined in the `\inlinestart` command. These names can be used at every place where the nickname of an instance is required, e.g. as the sender or receiver of a message.

`\inlineseparator` [*<options>*] {*<nickname>*}

Draws a dashed line at the current level for the inline expression with nickname *<nickname>*.

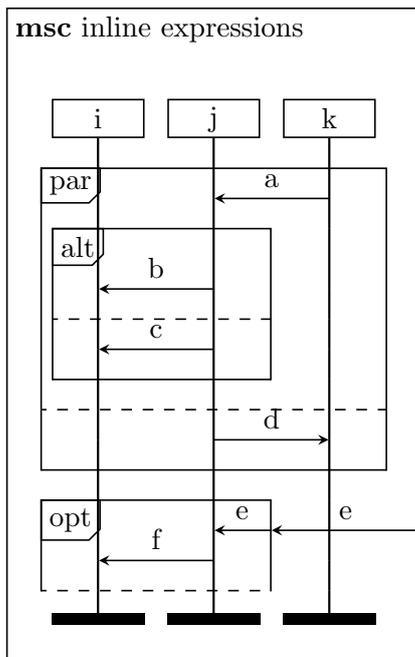
`\inlineend*` [*<options>*] {*<nickname>*}

Ends the inline expression with the nickname *<nickname>* at the current level. The command `\inlineend*` does the same as the command `\inlineend`, except that the bottom line of the rectangle is dashed. This is used to indicate that the operand is optional.

`/msc/left inline overlap=<distance>` (no default, initially 1cm)  
 Determines the *<distance>* which is a distance between the left edge of the inline expression symbol and the leftmost included instance axis.

`/msc/right inline overlap=<distance>` (no default, initially 1cm)  
 Determines the *<distance>* which is a distance between the right edge of the inline expression symbol and the rightmost included instance axis.

The following example shows how to draw inline expressions:



```
\begin{msc}[small values]{inline expressions}
\declinst{i}{}{i}
\declinst{j}{}{j}
\declinst{k}{}{k}
\inlinestart{exp1}{par}{i}{k} \nextlevel
\mess{a}{k}{j} \nextlevel
\inlinestart[left inline overlap=0.6cm]
{exp2}{alt}{i}{j} \nextlevel[2]
\mess{b}{j}{i} \nextlevel
\inlineseparator{exp2} \nextlevel
\mess{c}{j}{i} \nextlevel
\inlineend{exp2} \nextlevel
\inlineseparator{exp1} \nextlevel
\mess{d}{j}{k} \nextlevel
\inlineend{exp1} \nextlevel
\inlinestart{exp3}{opt}{i}{j} \nextlevel
\mess{e}{envright}{exp3right} \nextlevel
\mess{e}{exp3right}{j} \nextlevel
\mess{f}{j}{i} \nextlevel
\inlineend*{exp3}
\end{msc}
```

## 6.17 Gates

A gate determines a connection point for messages.

The following command can be used to draw gates.

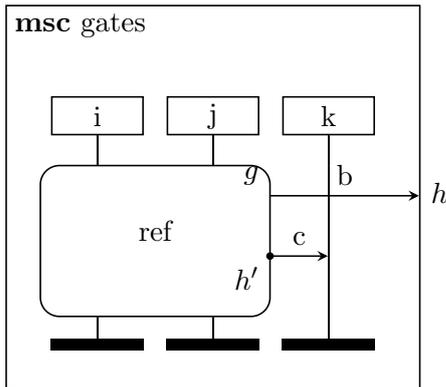
`\gate* [<options>]{<gate name>}{<instance name>}`

The unstarred version produces a normal (invisible) gate. The starred version produces a visible gate (a small dot). The gate is drawn at the current level at the instance *<instance name>* (which can also be the left and right edge of e.g. an MSC reference). The *<gate name>* is attached to the gate. The positioning of the *<gate name>* relative to the gate is determined by the keys `side` (left or right) and `position` (above, mid or below).

There are several keys to control the size and shape of the gate symbol:

`/msc/gate symbol radius=<radius>` (no default, initially 0.05cm)  
 The *<radius>* defines the radius of the gate symbol.

The next example shows a number of gates.



```

\begin{msc}[small values]{gates}
\declinst{i}{}{i}
\declinst{j}{}{j}
\declinst{k}{}{k}
\referencestart{r}{ref}{i}{j}
\nextlevel
\gate{g$}{rright}
\mess{b}{rright}{envright}
\gate[r][c]{$h$}{envright}
\nextlevel[2]
\mess{c}{rright}{k}
\gate*[1][b]{$h'$}{rright}
\nextlevel[2]
\referenceend{r}
\end{msc}

```

## 6.18 High-level MSCs

A High-level MSC (HMSC) is a drawing which defines the relation between a number of MSCs. It is composed of a start symbol (an upside down triangle), a number of end symbols (represented by triangles), a number of MSC references (these are rectangles with rounded corners), a number of conditions (hexagons) and possibly several connection points (circles). These symbols are connected by arrows.

The following commands can be used to draw HMSCs.

```

\begin{hmsc}[\langle options \rangle]{\langle hmsc name \rangle}(\langle llx \rangle,\langle lly \rangle)(\langle urx \rangle,\langle ury \rangle)
\langle environment contents \rangle
\end{hmsc}

```

In order to draw HMSCs, a new environment is defined, which is called `hmsc`. The command to begin this environment has several arguments.

The header of an HMSC is formed from the value of the key `hmsc keyword` (initially `hmsc`), followed by the `hmsc name`. The positioning of the header depends on the key `title position`. The possible values are `left`, `right` and `center`. The keys `title top distance` and `title distance` control the distances of the header from the frame. The size of the HMSC frame is calculated automatically unless the optional parameters `(llx, lly)` and `(urx, ury)` are specified. In such case the frame is determined by the coordinates of the lower-left corner, `(llx, lly)`, and the coordinates of the upper-right corner, `(urx, ury)`.

`/msc/hmsc keyword`=`keyword` (no default, initially `hmsc`)

The `keyword` will be a starting word in a header of HMSC.

`/msc/north hmsc margin`=`distance` (no default, initially 0.5cm)

The `distance` determines the north margin of hmsc environment

`/msc/south hmsc margin`=`distance` (no default, initially 0.5cm)

The `distance` determines the south margin of hmsc environment

`/msc/east hmsc margin`=`distance` (no default, initially 0.5cm)

The `distance` determines the east margin of hmsc environment

`/msc/west hmsc margin`=`distance` (no default, initially 0.5cm)

The `distance` determines the west margin of hmsc environment

`/msc/west hmsc margin`=`distance` (no default, initially 0.5cm)

The `distance` determines the west margin of hmsc environment

`/msc/hmsc margin=<distance>` (style, no default)  
Sets the `north hmsc margin`, `south hmsc margin`, `west hmsc margin`, `east hmsc margin` to *<distance>*.

In the HMSC we can draw number of symbols. Each symbol is in fact a node and *<options>* can be used to specify for example the placement relative to the other nodes. The following commands are used to draw the symbols:

`\hmscstartsymbol [<options>] {<nickname>} (<x>, <y>)`  
Draws an equilateral triangle pointed down representing the start symbol. The *<nickname>* can be used later as a reference to the drawn symbol. Optional *<x>* and *<y>* specify the coordinates at which the symbol should be drawn.

`/msc/start symbol` (style, no value)  
A style that is applied to draw a triangle node in the `\hmscstartsymbol` command.

`/msc/hmsc symbol width=<width>` (no default, initially 0.7cm)  
Sets the width of the triangle of the start and end symbol in hmsc to the *<width>*.

`\hmscendsymbol [<options>] {<nickname>} (<x>, <y>)`  
The command is identical to `\hmscstartsymbol` command but draws an equilateral triangle pointed up representing the start symbol.

`/msc/end symbol` (style, no value)  
A style that is applied to draw a triangle node in the `\hmscendsymbol` command.

`\hmscreference [<options>] {<nickname>} {<text>} (<x>, <y>)`  
Draws a *<text>* inside a frame with rounded corners. The *<nickname>* can be used later to connect it to other hmsc elements.

`/msc/reference` (style, no value)  
A style that is applied to draw a reference node in the `\hmscreference` command.

`/msc/reference height=<height>` (no default, initially 0.7cm)  
Sets the minimal height of the reference to *<height>*.

`/msc/reference width=<width>` (no default, initially 1.4cm)  
Sets the minimal width of the reference to *<width>*.

`\hmsccondition [<options>] {<nickname>} {<text>} (<x>, <y>)`  
Draws a *<text>* inside a rectangular frame with diamond shaped sides. The *<nickname>* can be used later to connect it to other hmsc elements.

`/msc/hmsc condition height=<height>` (no default, initially 0.7cm)  
Sets the minimal height of the condition to *<height>*.

`/msc/hmsc condition width=<width>` (no default, initially 1.4cm)  
Sets the minimal width of the condition to *<width>*.

`\hmscconnection [<options>] {<nickname>} (<x>, <y>)`  
Draws a small circle symbolizing connection at coordinates (*<x>*, *<y>*). Later we can refer to the connection using *<nickname>*.

`/msc/hmsc connection radius=<distance>` (no default, initially 0.05cm)  
Sets the radius of the connection symbol to the *<distance>*

The HMSC grid is not drawn, but used to control the positioning of the HMSC symbols (`startsymbol`, `endsymbol`, `reference`, `condition`, and `connection`). The center of each symbol is drawn on the grid point with coordinates  $(\langle x \rangle, \langle y \rangle)$ . Each symbol also has a  $\langle nickname \rangle$  for later reference.

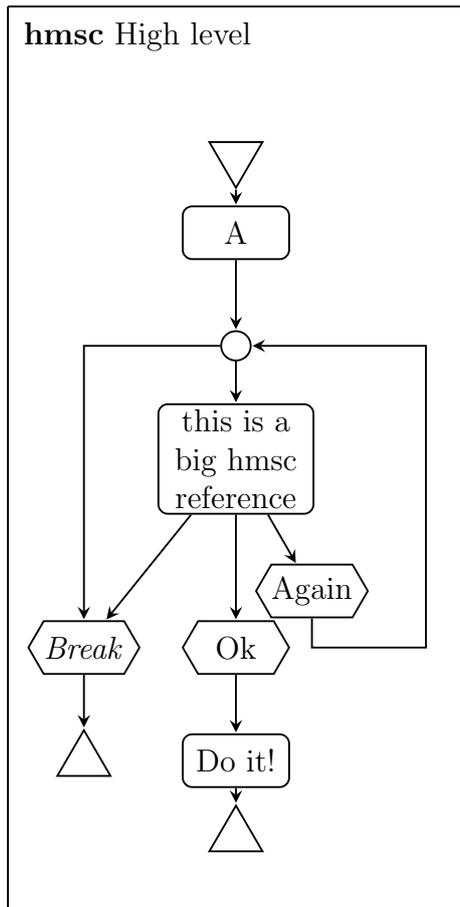
```
\hmscreference{nickname}{text}(x,y)
\hmscondition{nickname}{text}(x,y)
\hmsconnection{nickname}(x,y)
\arrow{from-nickname}[coord-list]{to-nickname}
```

HMSC symbols can be connected by means of the `arrow` command. This draws an arrow from the symbol with nickname `from-nickname` to the symbol with nickname `to-nickname`. The optional argument `coord-list` can be used if the line connecting the source and the destination should not be straight. The `coord-list` has the following syntax:  $(x_1, y_1) (x_2, y_2) \dots (x_k, y_k)$ . This means that the connecting line goes through the points with coordinates  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $\dots$ ,  $(x_k, y_k)$ .

Arrows always leave the start symbol at the bottom. They enter the end symbol at the top. Arrows start and end either at the middle of the top or at the middle of the bottom of a reference and condition symbol. The incoming (outgoing) direction of the arrow determines whether it will start (end) at the top or at the bottom.

There are several parameters to control the size and shape of the symbols (see Section 7). These are `\hmsconditionheight` (the height of the condition symbol), `\hmsconditionwidth` (the width of the condition symbol, excluding the left and right angular parts), `\hmsreferenceheight` (the height of the reference symbol), `\hmsreference width` (the width of the reference symbol), `\messarrowsscale{size}` (a command to set the size of the arrow head of a connection line); `setconnectiontype(type)` (set the shape of the polyline connection the symbols; `type` can be `straight`, `rounded`, and `curved`), `\startsymbolwidth` (the width of the start and end symbol), `\topnamedist` (sets the distance between the top of the HMSC frame and the HMSC header).

An example of an HMSC is in the following diagram. Notice that the width and height of reference symbols are changed locally (i.e., between `{` and `}` braces) just before the big reference `b` is defined.



```

\begin{hmsc}{High level}(-3,0)(3,12)
\hmscstartsymbol{s}(0,10)
\hmscreference{a}{A}(0,9)
\hmscconnection{c}(0,7.5)
{\setlength{\hmscreferencewidth}{2cm}
\setlength{\hmscreferenceheight}
{3\baselineskip}
\hmscreference{b}{\parbox{1.9cm}
{\centering this is a big hmsc reference}
}(0,6)}
\hmsccondition{t}{Again}(1,4.5)
\hmsccondition{ok}{Ok}(0,3.5)
\hmsccondition{q}{\textit{Break}}(-2,3.5)
\hmscendsymbol{e1}(-2,2)
\hmscreference{do}{Do it!}(0,2)
\hmscendsymbol{e2}(0,1)
\arrow{s}{a}
\arrow{a}{c}
\arrow{c}{b}
\arrow{c}[(-2,7.5)]{q}
\arrow{b}{q}
\arrow{q}{e1}
\arrow{b}{ok}
\arrow{ok}{do}
\arrow{do}{e2}
\arrow{b}{t}
\arrow{t}[(1,3.5)(2.5,3.5)(2.5,7.5)]{c}
\end{hmsc}

```

## 6.19 MSC documents

An MSCdoc is a drawing which contains various declarations of objects used in the MSC description. For drawing MSCdocs the following commands are provided.

```

\begin{mscdoc}[headerpos]{mscdocname}{text}(llx, lly)(urx, ury)
\end{mscdoc}
\reference{text}(x,y)
\separator{y}

```

As for MSC and HMSC, a new environment is defined, which is named `mscdoc`. The command to begin an MSCdoc has several arguments. The argument `headerpos` is optional. It controls positioning of the header of the MSCdoc. This argument can have values `l` (for a left aligned header), `c` (for a centered header) and `r` (for a right aligned header). The header of an MSCdoc is formed from the keyword `mscdocument`, followed by the `mscdocname`. The `text` is placed left-aligned below the MSCdoc header. The size of the MSCdoc frame is determined by coordinates of the lower-left corner, `(llx, lly)`, and the coordinates of the upper-right corner, `(urx, ury)`.

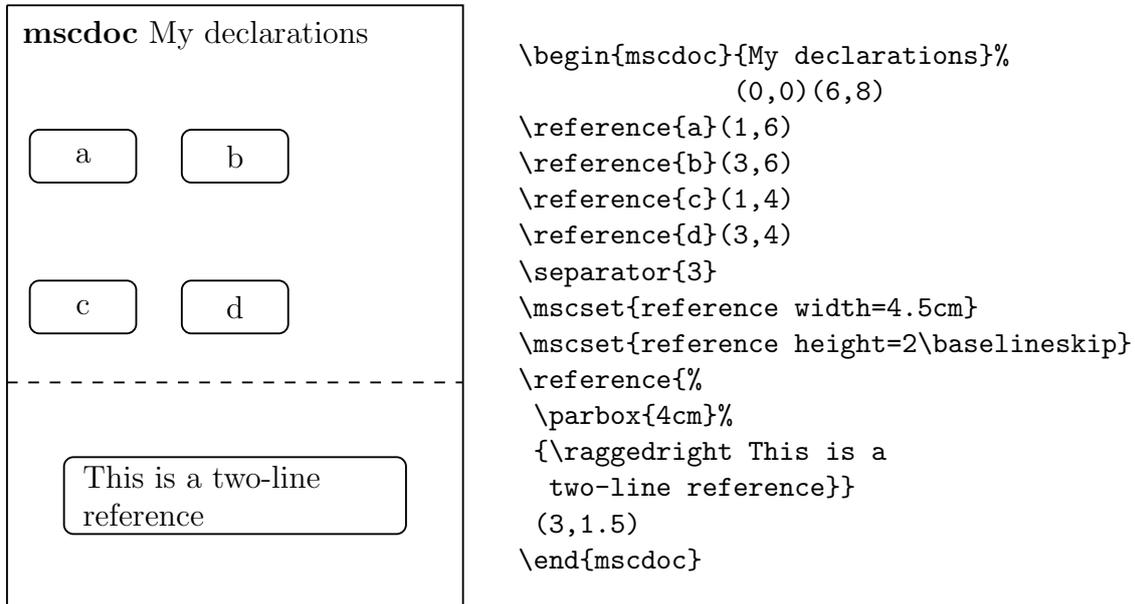
The MSCdoc grid is not drawn, but used to control the positioning of the MSC references. The center of such a reference is drawn on the grid point with coordinates `{x,y}`.

The `separator` command draws a dashed horizontal line. The MSC references above the separator are the exported, while the ones below the separator are local.

There are several parameters to control the size and shape of the symbols (see Section 7). `\mscdocreferenceheight` (the height of the reference symbol), `\mscdocreferencewidth`

(the width of the reference symbol), `\topnamedist` (sets the distance between the top of the MSCdoc frame and the MSCdoc header).

An example of an MSCdoc is in the following diagram. Notice that the size of references in an MSCdoc had to be changed for the last reference.



## 7 Style parameters

## 8 Example

Figure 4 on page 28 shows the MSC defined in the following L<sup>A</sup>T<sub>E</sub>X fragment.

```

\begin{msc}{Example}
\declinst{usr}{The user}{User}
\declinst{m1}{Control}{M1}
\dummyinst{m2}
\declinst{m3}{Another Machine}{M3}

\create{start}{m1}{m2}{Processing}{M2}
\mess{msg 0}{envleft}{usr}
\mess{msg 1}{envright}{m2}[1]
\nextlevel

\mess{msg 2}{usr}{m1}
\order{m1}{m2}[4]
\action{a}{m3}

\nextlevel
\found{msg x}{}{usr}
\nextlevel

\mess{msg 3}{usr}{m2}[-1]
\coregionstart{m1}
\settimeout{S}{m3}[2]
        
```

```

\nextlevel

\mess{msg 4}{m1}{usr}
\coregionstart{m2}
\settimer[r]{T}{m3}
\nextlevel

\mess[r]{msg 5}{m2}{m2}[3]
\mess{msg 6}{usr}{usr}[2]
\nextlevel

\mess{msg 7}{m2}{usr}
\timeout[r]{T}{m3}
\nextlevel

\coregionend{m2}
\nextlevel

\coregionend{m1}
\stoptimer[r]{T'}{m3}
\nextlevel

\lost[r]{msg y}{Mach 1}{usr}
\mess{msg 8}{m1}{envrigh}
\nextlevel

\condition{condition 1}{usr,m2}
\setstoptimer[r]{U}{m3}
\nextlevel[2]
\stop{usr}

\end{msc}

```

## 9 Acknowledgments

Thanks are due to the following people for providing us with useful input: Cas Cremers, Hugo Jonker, Peter Peters, Saša Radomirović, Michel Reniers, and Pim Vullers.

## References

- [1] V. Bos, T. van Deursen, and S. Mauw. *A L<sup>A</sup>T<sub>E</sub>X macro package for Message Sequence Charts—Reference Manual—Describing MSC macro package version 2.0*, December 2010. Included in MSC macro package distribution.
- [2] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). Geneva, 2011.
- [3] L. Lamport. *L<sup>A</sup>T<sub>E</sub>X—A Document Preparation System—User’s Guide and Reference Manual*. Adsison-Wesley, 2nd edition, 1994. Updated for L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>.

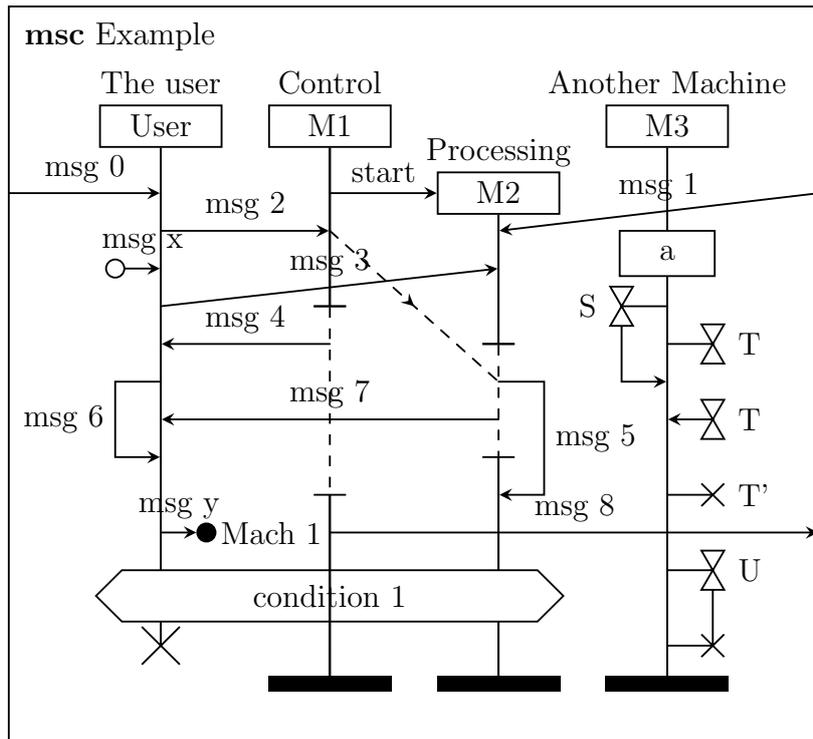


Figure 4: A menagerie of MSC symbols

- [4] Andrew Mertz and William Slough. Graphics with tikz. *The PracTEX Journal*, (1), 2007.
- [5] E. Rudolph, P.Graubmann, and J. Grabowski. Tutorial on message sequence charts (MSC'96). In *FORTE*, 1996.
- [6] T Tantau. Tikz and pgf: Manual for version 2.10, 2010.