A LaTeX macro package for Message Sequence Charts

User Manual

Victor Bos
Turku Centre for Computer Science
Software Construction Laboratory
Lemminkäisenkatu 14
FIN–20520 Turku
Finland
vbos@abo.fi

Sjouke Mauw
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
NL–5600 MB Eindhoven
The Netherlands
sjouke@win.tue.nl

Version 1.13, last update July 1, 2002
Describing MSC macro package version 1.13

Abstract

The MSC macro package facilitates the LaTeX user to easily include Message Sequence Charts in his texts. This document describes the design and use of the MSC macro package.

Contents

1 New

Version 1.13 has a reimplementation 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 6.5) are changed from gray into black. The reason for this is that the gray lines became invisible after converting the document to HTML.

Version 1.12 is a non-public version. It features a preliminary implementation of message label position control.

Version 1.11 is a mainly a bugfix of version 1.10a, see [1]. The next list shows the new features of version 1.10a compared to version 1.4.

- Support for method replies (dashed message arrows):`\mess*` (Section 6.4)

- Fat instances, i.e., double line instances (Section 6.3)
- Comments (Section 6.5)
- Coregions are now special cases of regions (which includes also activation and suspension regions). The `\coregionstart` and `\coregionend` commands are obsolete and the `\coregionbarwidth` is replaced by `\regionbarwidth`.
- Gates (Section 6.16)
- High-level MSC's (Section 6.17)
- `\inlinestart` has additional optional argument to allow for different left and right overlap (Section 6.15)
- MSCdocs (Section 6.18)
- Reference manual [2]
- `\referencestart` has additional optional argument to allow for different left and right overlap (Section 6.14)
- Regions: generalization of coregions supporting activations, suspension, and coregions (Section 6.12)
- Time measurements (Section 6.8)

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 [4]. An introductory text on MSC can be found in [6]. 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 LaTeX 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 [2], 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 LaTeX 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 LaTeX macros which support the drawing of MSCs. In this way, an MSC can be represented in 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://www.win.tue.nl/~sjouke/mscpackage.html`.

The MSC macro package has a *LaTeX Project Public License* (LPPL), see `http://www.latex-project.org/lppl.txt`:

```
%% MSC Macro Package
%% msc.sty
%% June 27, 2002
%%
%% Copyright 2002 V. Bos and S. Mauw
%
% This program may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.2
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.2 or later is part of all distributions of LaTeX
% version 1999/12/01 or later.
%
% This program consists of the files
%   msc.sty
%   manual.tex
%   manual.ps
%   refman.tex
%   refman.ps
%   biblio.bib
%   maintenance.tex
%   maintenance.ps
%   README
%   COPYRIGHT
```

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 LaTeX 2ε. It has been tested with LaTeX 2ε version dated 1998/06/01 using TeX version 3.14159. The following additional packages are required: pstricks, calc, ifthen, and color. These packages are in general part of the standard LaTeX 2$_\varepsilon$ distribution. These additional packages can be obtained from the *ctan* database for LaTeX e.g. via the following URL: `http://www.tex.ac.uk/`. The pstricks package is described in [3, Chapter 4]. The generated output can only be previewed with recent previewing software (e.g. xdvi version 20c). It may be needed to update all 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 LaTeX for style files. The set of directories actually searched depends on the TeX 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 TeX 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.

```
\documentclass{article}
\usepackage{msc}
\begin{document}

\begin{msc}{Example}

\declinst{usr}{User}{}
\declinst{m1}{Machine 1}{control}
\declinst{m2}{Machine 2}{drill}
\declinst{m3}{Machine 3}{test}

\mess{startm1}{usr}{m1}
\nextlevel
\mess{startm2}{m1}{m2}
\nextlevel
\mess{continue}{m2}{m3}
\mess{log}{m1}{envleft}
\nextlevel
\mess{output}{m3}{usr}[2]
\nextlevel
\mess{free}{m1}{usr}
\nextlevel

\end{msc}

\end{document}
```

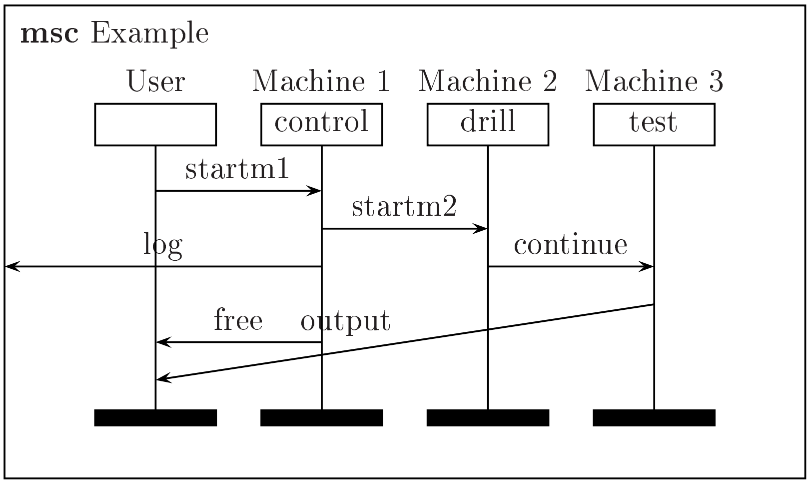


Figure 1: The generated MSC

The MSC macro package is activated by the clause `\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{startm1}{usr}{m1}` defines a message with name `startm1`, which goes from instance `usr` to instance `m1`.

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}{user}[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.

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

6 Use of the MSC macro package

6.1 The MSC frame

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

```
\begin{msc}[headerpos]{mscname}
...definition of the MSC...
\end{msc}
```

This draws the frame and the header of the MSC. The argument `mscname` is the name of the MSC. The header of an MSC is formed from the keyword `msc`, followed by the `mscname`. Positioning of this header is controlled by the optional argument `headerpos`. This argument can have values `l` (for a left aligned header), `c` (for a centered header) and `r` (for a right aligned header). The default value of `headerpos` is `l`.

The size of the MSC frame is determined vertically by the number of levels occurring in the MSC (see Section 6.2) and horizontally by the number of instances (see Section 6.3).

The parameter `\topnamedist` controls the distance between the top of the MSC frame and the header (see Section 7). The parameter `\leftnamedist` controls the distance between the left of the MSC frame and the header if the `headerpos` is `l` and it controls the distance between the right of the MSC frame and the header if `headerpos` is `r` (see Section 7).

6.2 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 6.4). The level offset is an 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[leveloffset]
```

The `leveloffset` is an integer value which denotes the number of levels to advance. By default, the value of `leveloffset` is 1, which means drawing continues at the next level. Setting `leveloffset` to a negative value may result in unexpected drawings, however, see the *Tricks* section in the reference manual [2].

There are three parameters which control the size of the levels (see Section 7). These are `\firstlevelheight` (the distance between the instance start symbol and the first level), `\levelheight` (the distance between two consecutive levels), and `\lastlevelheight` (the distance between the last level and the instance end symbol). Figure 4 on page 34 shows all lengths of the MSC macro package.

6.3 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.

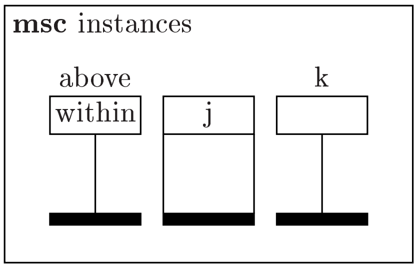
An instance is declared with the following command.

```
\declinst(*){nickname}{instancenameabove}{instancenamewithin}
```

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

Several parameters allow the user to customize the shape and positioning of instances (see Section 7). These are `\topheaddist` (the distance between the top of the MSC and the instance head symbol), `\instheadheight` (the height of the instance head symbol), `\instfootheight` (the height of the instance foot symbol), `\bottomfootdist` (the distance between the instance foot symbol and the bottom of the MSC frame), `\instwidth` (the width of the instance head and foot symbols), `\instdist` (the distance between two instance axes), `\envinstdist` (the distance between the edge of the MSC and the first/last instance axis), and `\labeldist` (the distance between the instance head symbol and the part of the instance name drawn above the head symbol). The command `\setfootcolor{color}` sets the fill color of the footer symbol. Valid color values are, e.g., `black` (default), `white`, and `gray`.

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 second line, `\setmscvalues{small}`, indicates that the small drawing style should be used (see Section 7).




```
\begin{msc}{instances}
\setmscvalues{small}

\declinst{i}{above}{within}
\declinst*{j}{}{j}
\declinst{k}{k}{}

\end{msc}
```

6.4 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(*)[pos]{name}[labelpos]{sender}[placement]{receiver}[leveloffset]
```

The `name` of the message may be any string. The MSC macro package processes the `name` argument in LR-mode, see [5, 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 [2]. By default, the name of a message label is drawn above the center of the arrow, but the optional parameters `pos`, `labelpos`, and `placement` 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 `leveloffset`, at the receiving instance. The `leveloffset` is an optional integer argument with default value 0. The `sender` and `receiver` should be the nicknames of declared instances.

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 [4]).

As mentioned above, placement of the message and its label is controlled by the optional parameters `pos`, `labelpos`, and `placement`. In case of a self message, `pos` denotes the position of the arrow with respect to the instance axis. Valid values are `l` (left) and `r` (right). The default value is `l`. In case of a non-self message, the `pos` parameter is ignored.

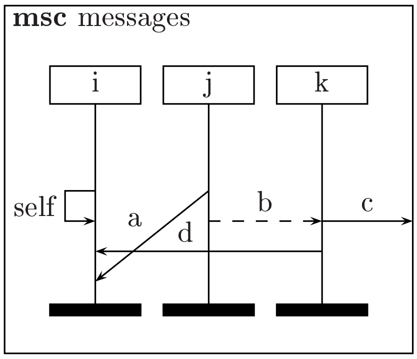
Whereas the `pos` parameter defines the position of the arrow symbol with respect to the instance axis, the `labelpos` parameter defines the position of the message label with respect to the message arrow. In case of a self message, valid values for `labelpos` are `l` (left) and `r` (right). The default value is equal to the, possibly user defined, value of the `pos` parameter. In case of a non-self message, valid values for `labelpos` are `t` (on top) and `b` (below). The default value is `t`.

Finally, the `placement` parameter defines the relative distance of the message label to the beginning of the message. Valid values are real numbers in the closed interval $[0, 1]$. The default value is 0.5. While drawing a message, the MSC macro package computes the coordinates of the message label using `placement` and the length and coordinates of the arrow. It then computes a *reference point* for the message label and places it on the coordinates just computed. Figures 2 (page 32) and 3 (page 33) schematically show the reference points for message labels. In the first figure, the labels are located at the default position. In the second figure, all labels are shifted along the arrow by setting `placement` $= 0.9$. The `\lost` and `\found` commands (Section 6.9) and the `\create` (Section 6.13) command use the same method to determine reference points and message label locations. Note that the boxes with the location of the reference points are not generated by the LaTeX code given in Figures 2 and 3; we enriched the LaTeX code with some extra pstricks code (see LaTeX source code of this document).

In addition to label position control, there are three ways to control the shape of messages (see Section 7). These are: `\selfmesswidth` (a parameter to specify the width of the polyline used for drawing self-messages), `\labeldist` (a parameter to specify the distance between the label of a message and the message arrow), and `\messarrowscale{size}` (a command to set the size of the arrow head of a message). `size` should be positive real number.

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.)

The following MSC 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`.




```
\mess{a}{j}{i}[3]
\mess{self}{i}{i}
\nextlevel
\mess*{b}{j}{k}
\mess[b]{c}{k}{envright}
\nextlevel
\mess{d}{k}[.6]{i}
\nextlevel
```

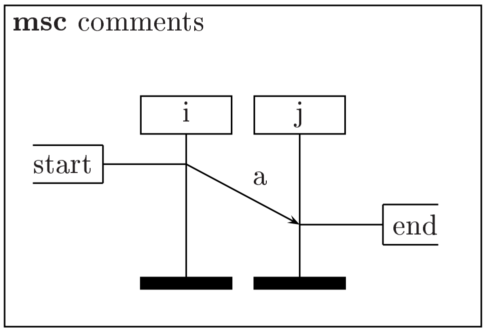
6.5 Comments

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

```
\msccomment[position]{text}{instname}
```

The `instname` parameter defines the instance to which the comment is attached. The text of the comment is specified by the `text` parameter and is processed in LR-mode. The `position` parameter defines the horizontal position of the comment relative to its instance. Valid values of `position` are `l` (left), `r` (right), or any LaTeX length. Its default value is `l`. If the value of `position` is `l` (or `r`), the comment will be placed `\msccommentdist` units to the left (or right) of the `instname` instance. If `position` is a LaTeX length, the comment will be placed `position` units from the `instname` instance. A negative length puts the comment to the left and a positive length puts the comment to the right of the instance.

The following diagram shows how to use comments. In this diagram, the distance between the frame and the instances (`\envinstdist`) is doubled in order to fit the comments inside the frame.




```
\setlength{\envinstdist}{2\envinstdist}
\declinst{i}{}{i}
\declinst{j}{}{j}
\mess{a}{i}{j}[2]
\msccomment{start}{i}
\nextlevel[2]
\msccomment[r]{end}{j}
\nextlevel
```

6.6 Actions

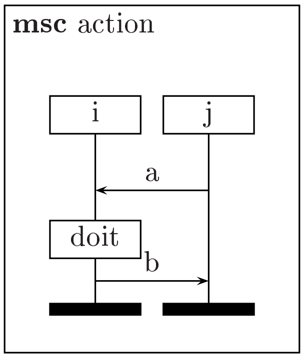
An instance can perform an action, which is denoted by a rectangle.

```
\action{name}{instance}
```

The action is attached at the current level to the `instance`. The `name` is centered inside the action symbol and is processed in LR-mode.

The following parameters determine the detailed drawing of the action symbol (see Section 7): `\actionwidth` (the width of the action symbol), and `\actionheight` (the height of the action symbol).

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




```
\mess{a}{j}{i}
\nextlevel
\action{doit}{i}
\nextlevel[2]
\mess{b}{i}{j}
```

6.7 Timers

There are five commands to draw timer events.

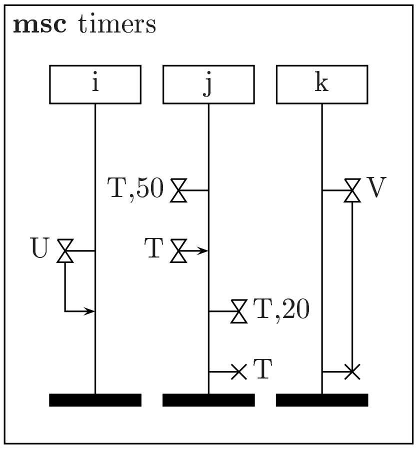
```
\settimer[placement]{name}{instance}
\timeout[placement]{name}{instance}
\stoptimer[placement]{name}{instance}
\settimeout[placement]{name}{instance}[offset]
\setstoptimer[placement]{name}{instance}[offset]
```

Setting of a timer is drawn as a line connecting the `instance` to the *hour glass* symbol. The `name` is put near this symbol. A time-out is represented by an arrow from an *hour glass* symbol to the `instance`. Stopping a timer is drawn as a line connecting the `instance` with the timer stop symbol (denoted by a cross). 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. The default value for `offset` is 2. Likewise, `\setstoptimer` is a combination of the setting of a timer and stopping a timer.

The optional argument `placement` can have values `l` (meaning that the timer is drawn left of the instance axis) and `r` (meaning that the timer is drawn right of the instance axis). By default it is drawn at the left side of the instance.

Several parameters can be used to control the detailed layout of timer symbols (see Section 7): `\timerwidth` (the width of the hour glass and time out symbols), `\selfmesswidth` (the length of the arm between the symbol and the instance axis), and `\labeldist` (the distance between the name and the timersymbol). Furthermore, the size of the arrow head can be controlled with the command `\messarrowscale{size}`.

The various timer symbols are shown in the following example.




```
\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}
```

6.8 Time measurements

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

```
\mscmark[placement]{name}{instance}
\measure(*)[placement]{name}{instance1}{instance2}[offset]
\measurestart(*)[placement]{name}{instance}{gate}
\measureend(*)[placement]{name}{instance}{gate}
```

An absolute time stamp is attached to an event on an `instance` with the command `mscmark`. The `name` is the text attached to the mark symbol, which is a dashed polyline. The position of the mark symbol relative to the marked event is determined by the `placement`. (`tl` means top-left, `tr` means top-right, `ml` means middle-left, `mr` means middle-right, `bl` means bottom-left, and `br` means bottom-right).

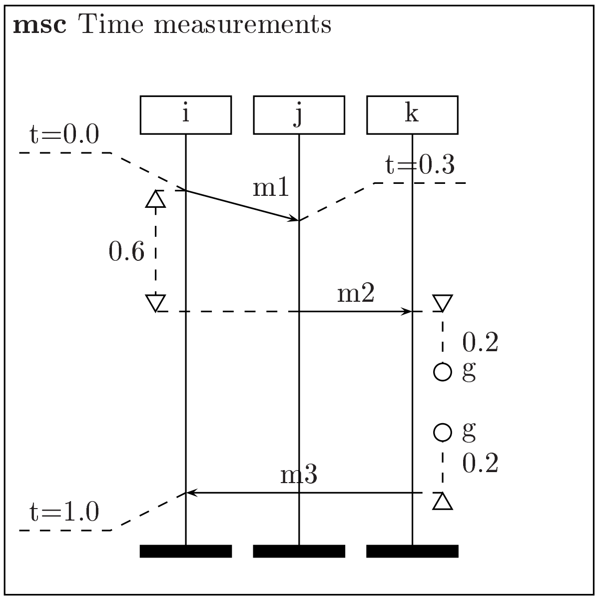
A `\measure` connects two events from `instance1` and `instance2`. The first event is at the current level. The second event is `offset` levels lower than the first event. The `name` is attached to the measure symbol. The measure symbol can be placed at the left or at the right of `instance1`. This is controlled by the optional argument `placement`, which can have values `l` and `r`.

In case the two events are far apart, the measure may be split in two parts. The first part is drawn with the `\measurestart` command and the second part with the `\measureend` command. The points where these two parts should be connected are drawn by a small circle, to which the text `gate` is attached.

There are two equivalent forms of the measurement symbol. The first form, where the arrow heads are at the inside of the measured interval, is the default form. The second form, where the arrow heads are at the outside of the measured interval, is obtained by the commands `\measure*`, `\measurestart*`, and `\measureend*`.

Several parameters can be used to control the detailed layout of the time measurement symbols (see Section 7): `\labeldist` (a parameter to specify the distance between the label of a measurement and the measurement symbol), `\messarrowscale{size}` (a command to set the size of the arrow head), `\selfmesswidth` (specifies the width of the measurement symbols).

The following example illustrates marks and measurements in an MSC. In order to include the marks inside the frame of the diagram, the distance between the frame and the instances (called `\envinstdist`) is increased (before the instances are declared).




```
\setlength{\envinstdist}
          {2\envinstdist}
\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[tr]{t=0.3}{j}
\nextlevel[3]
\mess{m2}{j}{k}
\measurestart*[r]{0.2}{k}{g}
\nextlevel[6]
\mess{m3}{k}{i}
\mscmark[bl]{t=1.0}{i}
\measureend*[r]{0.2}{k}{g}
\nextlevel
```

6.9 Lost and found messages

A lost message is denoted by an arrow starting at an instance and ending at a filled circle. A found message is denoted by an arrow starting at an open circle and ending at an instance.

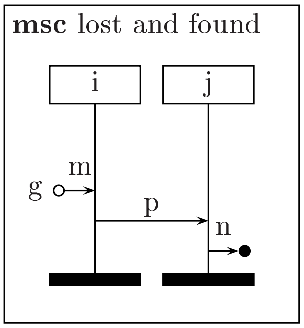
The following commands are used to define lost and found messages.

```
\lost[pos]{name}[labelpos]{gate}{instance}[placement]
\found[pos]{name}[labelpos]{gate}{instance}[placement]
```

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 optional arguments `pos`, `labelpos`, and `placement` have the same function as in the `\mess` command (Section 6.4). That is, `pos` controls the placement of the lost or or found message with respect to the instance axis. Valid values are `l` (left) and `r` (right). The default value is `l`. The optional parameters `labelpos` and `placement` control the placement of `name` with respect to the message arrow. Valid values for `labelpos` are `t` (on top) and `b` (below). The default value is `t`. Valid values for `placement` are real numbers in the closed interval $[0, 1]$ and denote the relative distance of the message label `name` to the beginning of the arrow. The default value for `placement` is 0.5.

Several parameters can be used to control the detailed layout of lost and found messages (see Section 7): `\lostsymbolradius` (the radius of the circle), `\selfmesswidth` (the length of the arrow), and `\labeldist` (the distance between the name and the arrow). Furthermore, the size of the arrow head can be controlled with the command `\messarrowscale{size}`.

The following example shows a found and a lost message.




```
\found{m}{g}{i}
\nextlevel
\mess{p}{i}{j}
\nextlevel
\lost[r]{n}{}{j}
```

6.10 Conditions

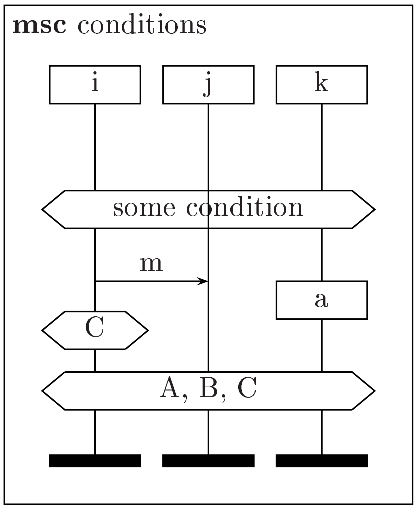
A condition is denoted by a hexagon. It is used to express that the system has entered a certain state. A condition relates to a number of instances. All conditions which take part in the condition are covered by the condition symbol. The other instances are drawn through the condition symbol. The following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

```
\condition{text}{instancelist}
```

The `text` is placed in the center of the condition. The `instancelist` 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 LaTeX. The order in which the instances are listed is immaterial.

There are two parameters which control the shape of the condition symbol (see Section 7): `\conditionheight` (the height of the condition symbol), and `\conditionoverlap` (the width of the part of the condition symbol which extends over the rightmost/leftmost contained instance axis).

The following example contains some conditions.




```
\condition{some condition}{i,k}
\nextlevel[3]
\mess{m}{i}{j}
\action{a}{k}
\nextlevel
\condition{C}{i}
\nextlevel[2]
\condition{A, B, C}{i,j,k}
\nextlevel[2]
```

6.11 Generalized ordering

A generalized order is treated much like a regular message (see Section 6.4). 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[pos]{sender}{receiver}[leveloffset]
```

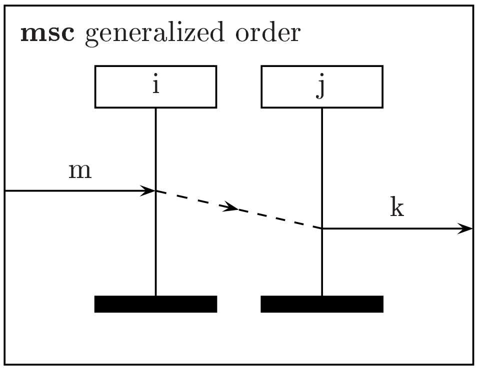
The `sender` and `receiver` are the nicknames of the instances which are connected by the generalized ordering symbol. At the `receiver` instance, the generalized ordering symbol ends at the current level plus the `leveloffset`. The `leveloffset` is an optional integer value, with default 0.

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 optional argument `pos`. It can have values `l` (meaning that the ordering symbol is drawn left of the instance axis) and `r` (meaning that the ordering symbol is drawn right of the instance axis). By default it is drawn at the left side of the instance. For non-self orders, the `pos` argument is ignored.

Additionally, there are two ways to control the shape of the generalized ordering symbol (see Section 7). These are: `\selfmesswidth` (a parameter to specify the width of the polyline used for drawing orderings on a single instance axis). and `\messarrowscale{size}` (a command to set the size of the arrow head in the ordering symbol).

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.




```
\mess{m}{envleft}{i}
\order{i}{j}[1]
\nextlevel
\mess{k}{j}{envright}
```

6.12 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).

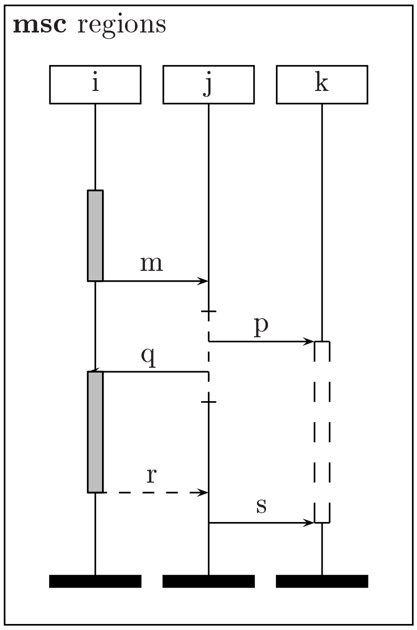
The following commands are used to draw an instance region.

```
\regionstart{type}{instname}
\regionend{instname}
```

If the `\regionstart` command is used the instance axis of `instname` is drawn in the shape determined by `type`, starting from the current level. The type can have values `coregion`, `suspension`, and `activation`.

The shape of an instance region can be controlled with the following parameters (see Section 7): `\regionbarwidth` (the width of the coregion start and end symbol), `\instwidth` (the width of the fat instance rectangle), `\regionwidth` (the width of the activation and suspension rectangle).

In the following example, several instance regions are demonstrated. Notice the relative order of `\mess`, `\regionstart`, and `\regionend` commands. Interchanging lines `\regionstart{activation}{i}` and `\mess{q}{j}{i}` makes the arrow head of the 'q' message partly invisible. Also note that the second activation region on i ends with a method reply (which is produced by the command `\mess*`).




```
\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
```

In some situations, the gray-colored activation regions can hide message-labels. *Level back-up* can help in these situations, see the *Tricks* section in the reference manual [2].

6.13 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{createdinst}
\create{name}
       [labelpos]
       {creator}
       [placement]
       {createdinst}
       {instancenameabove}
       {instancenamewithin}
\stop{instance}
```

In order to reserve space for an instance which will be created dynamically, the command `dummyinst` must be used. This command is mixed with the declarations of normal instances (see the `declinst` command, Section 6.3). The argument `createdinst` is the nickname of the instance that will be created later.

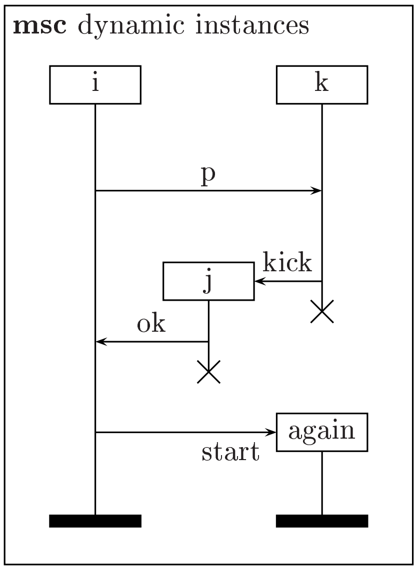
An instance can be created with the `\create` command. This command results in a dashed horizontal message arrow labeled with `name`. The arrow starts at the current level at the instance with nickname `creator` and ends at the current level at the instance head of the instance with nickname `createdinst`. This instance must have been declared first with the `dummyinst` command. The name of the created instance consists of two parts. The part called `instancenameabove` is placed above the instance head and the `instancenamewithin` is centered within the instance head.

As with normal messages, placement of the message label is controlled by the optional parameters `labelpos` and `placement`. That is, `labelpos` denotes the relative position of the message label with respect to the message arrow. Valid values are `t` (on top) and `b` (below). The default value is `t`. The optional parameter `placement` denotes the relative distance of the message label with respect to the beginning of the arrow. Valid values are real numbers in the closed interval $[0, 1]$. The default value is 0.5. See Figure 2 (page 32) and its description in Section 6.4 for more information on the placement of message labels.

An instance is stopped with the `\stop` command. The `instance` 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.

The size of the stop symbol is determined by the parameter `\stopwidth` (see Section 7). The following parameters apply to the instance head symbol: `\instheadheight` (the height of the instance head symbol), `\instwidth` (the width of the instance head symbol), and `\labeldist` (the distance between the instance head symbol and the part of the instance name drawn above the head symbol).

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




```
\declinst{i}{}{i}
\dummyinst{j}
\declinst{k}{}{k}

\mess{p}{i}{k}
\nextlevel[3]
\create{kick}{k}{j}{}{j}
\nextlevel
\stop{k}
\nextlevel
\mess{ok}{j}{i}
\nextlevel
\stop{j}
\nextlevel[2]
\create{start}[b]{i}[.75]{k}{}{again}
\nextlevel[2]
```

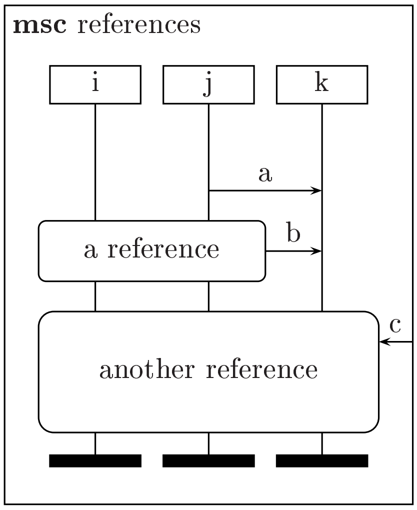
6.14 MSC references

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[lo][ro]{nickname}{text}{leftinstance}{rightinstance}
\referenceend{nickname}
```

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 `nickname`. The `text` is placed in the center of the reference symbol. The reference covers all instances from `leftinstance` to `rightinstance`. The distance between the left (right) edge of the reference symbol and the leftmost (rightmost) covered instance axis is defined by the optional overlap value `lo`. The default value for `lo` in large/normal/small MSCs is 1.5/1/0.75 (see Section 7 for selecting large, normal or small drawing mode). If the second optional value `ro` is present too, the optional value `lo` defines the left overlap and the optional value `ro` defines the right overlap.

The left and right edge of the reference symbol are `nicknameleft` and `nicknameright`, where `nickname` 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. This is shown in the following example.




```
\mess{a}{j}{k}
\nextlevel
\referencestart{r}{a reference}{i}{j}
\nextlevel
\mess{b}{rright}{k}
\nextlevel
\referenceend{r}
\nextlevel
\referencestart{p}{another reference}{i}{k}
\nextlevel
\mess{c}{envright}{pright}
\nextlevel[3]
\referenceend{p}
```

6.15 Inline expressions

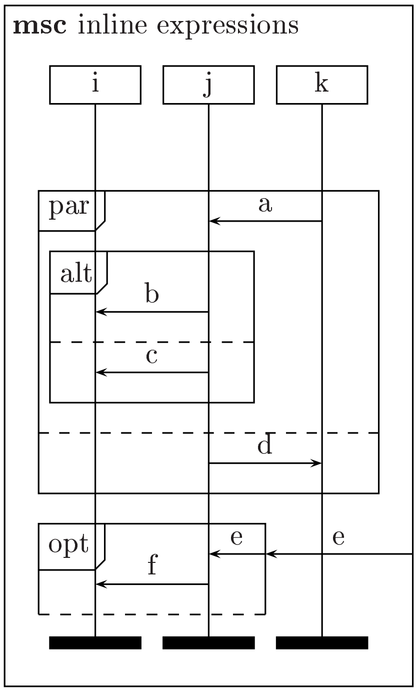
An inline expression is a part of an MSC on which an operation is defined. A rectangle surrounds the part of the MSC containing the operands. The operands are separated by horizontal dashed lines. The operator is placed in the upper left corner of the inline expression symbol. The following commands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

```
\inlinestart[lo][ro]{nickname}{operator}{leftinstance}{rightinstance}
\inlineseparator{nickname}
\inlineend(*){nickname}
```

The inline expression starts at the level where the `\inlinestart` command occurs. The inline expression spans over the instances from `leftinstance` to `rightinstance`. The `operand` is placed in the upper left corner of the rectangle. At every level where a corresponding `\inlineseparator` command occurs, a dashed line is drawn. The inline expression ends at the level where the `\inlineend` command occurs. The `nickname` is used to match corresponding commands. 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.

The left and right edge of the inline expression symbol are named `nicknameleft` and `nicknameright`, 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.

The distance between the left (right) edge of the inline expression symbol and the leftmost (rightmost) included instance axis is defined by the optional overlap value `lo`. The default value for large/normal/small MSCs is 1.5/1/0.75 (see Section 7 for selecting large, normal or small drawing mode). If the second optional value `ro` is present too, the optional value `lo` defines the left overlap and the optional value `ro` defines the right overlap. These options make it possible to easily draw nested inline expressions as in the following example.




```
\inlinestart{exp1}{par}{i}{k}
\nextlevel
\mess{a}{k}{j}
\nextlevel
\inlinestart[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}
\mess{e}{exp3right}{j}
\nextlevel
\mess{f}{j}{i}
\nextlevel
\inlineend*{exp3}
```

6.16 Gates

A gate determines a connection point for messages.

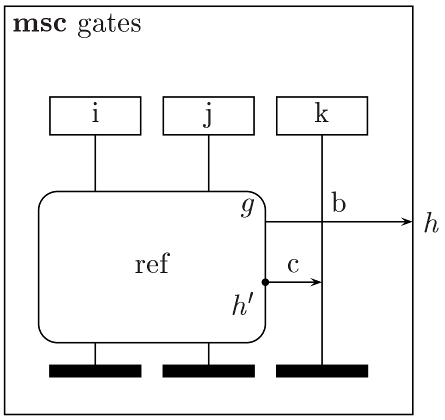
The following command can be used to draw gates.

```
\gate(*)[hpos][vpos]{gatename}{instname}
```

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 `instname` (which can also be the left and right edge of e.g. an MSC reference). The `gatename` is attached to the gate. The positioning of the `gatename` relative to the gate is determined by the values of `hpos` (horizontal position) and `vpos` (vertical position). Possible values for `hpos` are `l` (left, default) and `right` (right). Possible values for `vpos` are `t` (top, default), `c` (center), and `b` (bottom).

There are several parameters to control the size and shape of the gate symbol (see Section 7). These are `\gatesymbolradius` (sets the radius of the gate symbol), `\labeldist` (the vertical distance between the gate name and the gate).

The next example shows a number of 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
\nextlevel
\mess{c}{rright}{k}
\gate*[l][b]{$h'$}{rright}
\nextlevel[2]
\referenceend{r}
```

6.17 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}[headerpos]{hmscname}(llx,lly)(urx,ury)
\end{hmsc}
\hmscstartsymbol{nickname}(x,y)
\hmscendsymbol{nickname}(x,y)
```

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

In order to draw HMSCs, a new environment is defined, which is called `hmsc`. The command to begin this environment has several arguments. The argument `headerpos` is optional. It controls positioning of the header of the HMSC. 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 HMSC is formed from the keyword `msc`, followed by the `hmscname`. The size of the HMSC frame is determined by the coordinates of the lower-left corner, `(llx,lly)`, and the coordinates of the upper-right corner, `(urx,ury)`.

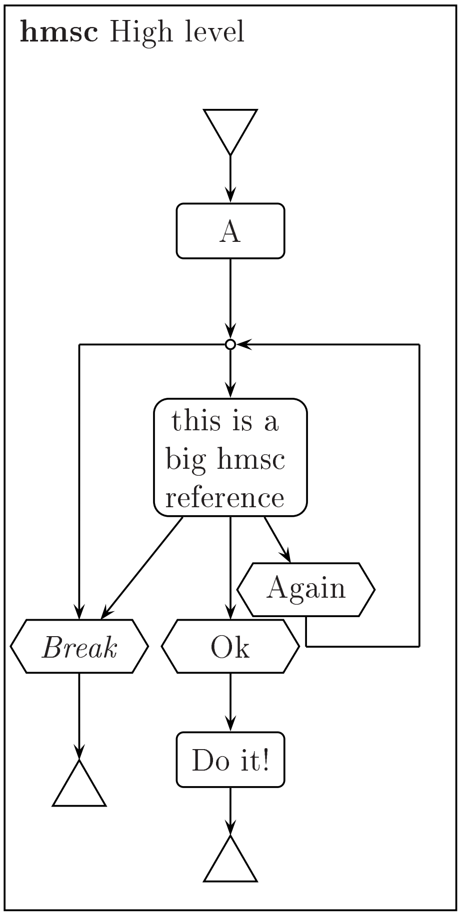
The HMSC grid is not drawn, but used to control the positioning of the HMSC symbols (`startsymbol`, `endsymbol`, `reference`, `condition`, and `connection`). The center of each symbol is drawn on the grid point with coordinates `{x,y}`. Each symbol also has a `nickname` for later reference.

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: `(x1,y1)(x2,y2)...(xk,yk)`. This means that the connecting line goes through the points with coordinates (x1,y1), (x2,y2), ..., (xk,yk).

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 `\hmscconditionheight` (the height of the condition symbol), `\hmscconditionwidth` (the width of the condition symbol, excluding the left and right angular parts), `\hmscreferenceheight` (the height of the reference symbol), `\hmscreferencewidth` (the width of the reference symbol), `\messarrowscale{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}
```

6.18 MSC documents

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

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

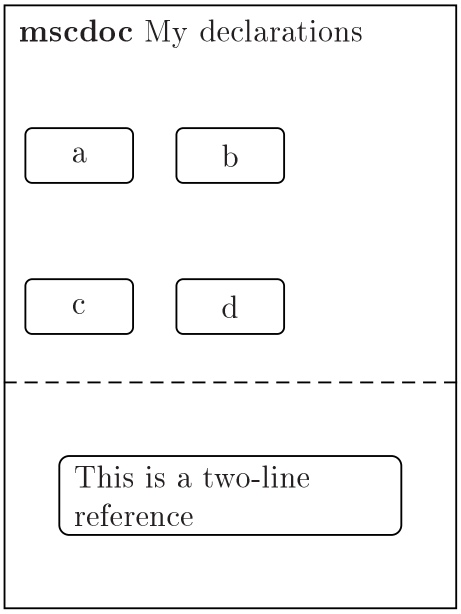
As for MSC and HMSC, a new environment is defined, which is named `mscdoc`. The command to begin an MSCdoc has several arguments. The argument `headerpos` is optional. It controls positioning of the header of the MSCdoc. This argument can have values `l` (for a left aligned header), `c` (for a centered 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.




```
\begin{mscdoc}{My declarations}%
              (0,0)(6,8)
\reference{a}(1,6)
\reference{b}(3,6)
\reference{c}(1,4)
\reference{d}(3,4)
\separator{3}
\setlength{\mscdocreferencewidth}
          {4.5\mscunit}
\setlength{\mscdocreferenceheight}
          {2\baselineskip}
\reference{%
 \parbox{4\mscunit}%
 {\raggedright This is a
  two-line reference}}
 (3,1.5)
\end{mscdoc}
```

7 Style parameters

By means of a collection of parameters, the graphical appearance of an MSC can be fine tuned to the user's taste. The general parameters are displayed in Figure 4 on page 34.

The value of a parameter can be changed using standard LaTeX commands, e.g.

```
\setlength{\levelheight}{1cm}
```

The following list describes all parameters. The default values for drawing MSCs at large, normal and small size are included. See the command `\setmscvalues{size}` below for restoring the parameters to their original values.

`\actionheight` Height of action symbols.
(large/normal/small value 0.75/0.6/0.5 cm.)

`\actionwidth` Width of action symbol.
(large/normal/small value 1.25/1.25/1.2 cm.)

`\bottomfootdist` Distance between bottom of foot symbol and frame.
(large/normal/small value 1.0/0.7/0.5 cm.)

`\commentdist` Distance between a comment and its instance.
(large/normal/small value 0.5/0.5/0.5 cm.)

`\conditionheight` Height of condition symbols.
(large/normal/small value 0.75/0.6/0.5 cm.)

`\conditionoverlap` Overlap of condition symbol.
(large/normal/small value 0.6/0.5/0.4 cm.)

`\envinstdist` Distance between environments and nearest instance line.
(large/normal/small value 2.5/2.0/1.2 cm.)

`\firstlevelheight` Height of level just below head symbols. Should not be changed inside the MSC environment.
(large/normal/small value 0.75/0.6/0.4 cm.)

`\hmscconditionheight` Height of HMSC condition symbol.
(large/normal/small value 0.375/0.3/0.25 cm.)

`\hmscconditionwidth` Width of HMSC condition symbol.
(large/normal/small value 1.0/0.8/0.7 cm.)

`\hmscconnectionradius` Radius of HMSC connection symbol.
(large/normal/small value 0.06/0.05/0.04 cm.)

`\hmscreferenceheight` Height of HMSC reference symbol.
(large/normal/small value 0.375/0.3/0.25 cm.)

`\hmscreferencewidth` Width of HMSC reference symbol.
(large/normal/small value 1.0/0.8/0.7 cm.)

`\hmscstartsymbolwidth` Width of HMSC start symbol.
(large/normal/small value 0.75/0.6/0.3 cm.)

`\inlineoverlap` Overlap of inline symbol.
(large/normal/small value 1.5/1.0/0.75 cm.)

`\instbarwidth` Default width of vertical instance bars (applies to fat instances only).
(large/normal/small value 0.0/0.0/0.0 cm.)

`\instdist` Distance between instance axes.
(large/normal/small value 3.0/2.2/1.5 cm.)

`\instfootheight` Height of foot symbols. Should not be changed inside the MSC environment.
(large/normal/small value 0.25/0.2/0.15 cm.)

`\instheadheight` Height of head symbols. Should not be changed inside the MSC environment.
(large/normal/small value 0.6/0.55/0.5 cm.)

`\instwidth` Width of header and foot symbols.
(large/normal/small value 1.75/1.6/1.2 cm.)

`\labeldist` Distance between labels and the symbols to which they belong (for instance, message labels and arrows).
(large/normal/small value 1.0/1.0/1.0 ex.)

`\lastlevelheight` Height of level just above foot symbols. Should not be changed inside the MSC environment.
(large/normal/small value 0.5/0.4/0.3 cm.)

`\leftnamedist` Distance between left of the frame and (left of) MSC, HMSC, or MSCdoc title.
(large/normal/small value 0.3/0.2/0.1 cm.)

`\levelheight` Height of a level.
(large/normal/small value 0.75/0.5/0.4 cm.)

`\lostsymbolradius` Radius of the lost and found symbols.
(large/normal/small value 0.15/0.12/0.08 cm.)

`\markdist` Horizontal distance from a mark to its instance.
(large/normal/small value 1.0/1.0/1.0 cm.)

`\measuredist` Horizontal distance from a measure to its (closest) instance.
(large/normal/small value 1.0/1.0/1.0 cm.)

`\measuresymbolwidth` Width of a measure symbol.
(large/normal/small value 0.75/0.6/0.4 cm.)

`\mscdocreferenceheight` Height of reference symbol in an MSCdoc.
(large/normal/small value 0.375/0.3/0.25 cm.)

`\mscdocreferencewidth` Width of reference symbol in an MSCdoc.
(large/normal/small value 1.0/0.8/0.7 cm.)

`\referenceoverlap` Overlap of reference symbol.
(large/normal/small value 1.5/1.0/0.75 cm.)

`\regionbarwidth` Width of region bars.
(large/normal/small value 0.5/0.4/0.2 cm.)

`\selfmesswidth` Length of horizontal arms of self messages.
(large/normal/small value 0.75/0.6/0.4 cm.)

`\stopwidth` Width of the stop symbol.
(large/normal/small value 0.6/0.5/0.3 cm.)

`\timerwidth` Width of the *timer* symbols.
(large/normal/small value 0.4/0.3/0.2 cm.)

`\topheaddist` Distance between top of head symbols and frame.
(large/normal/small value 1.5/1.3/1.2 cm.)

`\topnamedist` Distance between top of the frame and (top of) MSC, HMSC, or MSCdoc title.
(large/normal/small value 0.3/0.2/0.2 cm.)

In addition there are several commands which allow the user to adjust the MSC drawing to his own taste.

`\messarrowscale`{*scalefactor*} Sets the scale factor (a positive real number) of message arrow heads. (large/normal/small value 2/1.5/1.2)

`\setmscscale`{*scalefactor*} Sets the scale factor of the MSC environment to *scalefactor*. the scale factor is supposed to be a real number. Scaling is done when the MSC environment ends (`\end{msc}`). The default of *scalefactor* is 1. A more consistent way for varying the size of the MSC can be obtained by using the `\setmesvalues` command as described below. (default value 1.)

`\psset{linewidth=D}` This command sets the width of all lines in MSCs, HMSCs, and MSCdocs to length `D`. If this command is issued outside the msc environment, then the value is set for the complete document. If it is used directly after the start of the msc environment it only holds for this MSC.
(large/normal/small value 0.8/0.7/0.6 pt.)

`\setfootcolor`{*color*} Sets the color of the foot symbols of MSC instances. Possible values are *black*, *white*, *gray*, or *lightgray*. For more color values, see the documentation of the LaTeX 2_ε color package.

The following command can be used to set the above mentioned style parameters to suitable values.

`\setmscvalues`{*size*} Sets all parameters of the MSC macro package to predefined values. Valid values for *size* are: `small`, `normal`, and `large`. (The default value of size is normal. This can be used for drawings with at maximum six instances on a sheet of A4 paper. For sizes large and small, a maximum of four and nine instances respectively fit on a sheet of A4 paper.)

Caution has to be taken when changing the value of a parameter within the MSC definition. The following parameters can be changed within an MSC definition without unexpected side effects:

\actionheight, \actionwidth, \bottomfootdist, \conditionheight, \conditionoverlap, \inlineoverlap, \instfootheight, \instwidth (however, this may cause different sizes of corresponding instance header and footer symbols), \labeldist, \lastlevelheight, \levelheight, \lostsymbolradius, \referenceoverlap, \regionbarwidth, \selfmesswidth, \stopwidth, \timerwidth, and \topnamedist.

In addition to the parameters specific to the MSC macro package, standard LaTeX commands can be used to change the type style and other details. For example, if the command \sffamily is included directly after the start of the msc environment, the text in the MSC is drawn using a sans serif font. Likewise, the text size can be changed by inserting, e.g., the command \small. The \raisebox and \parbox commands can also be used to position and format names.

8 Example

Figure 5 on page 35 shows the MSC defined in the following LaTeX 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}{envright}
\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: Peter Peters, Michel Reniers.

References

[1] V. Bos and S. Mauw. *A LaTeX macro package for Message Sequence Charts—Maintenance document—Describing MSC macro package version 1.13*, June 2002. Included in MSC macro package distribution.

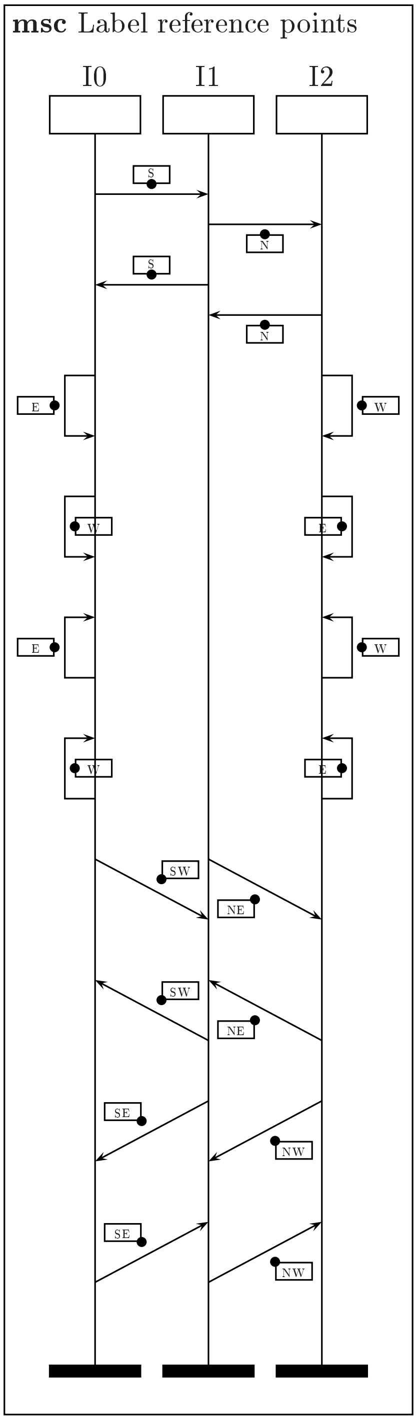
[2] V. Bos and S. Mauw. *A LaTeX macro package for Message Sequence Charts—Reference Manual—Describing MSC macro package version 1.13*, June 2002. Included in MSC macro package distribution.

[3] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LaTeX Graphics Companion*. Addison-Wesley, 1997.

[4] ITU-TS. ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). Geneva, 1997.

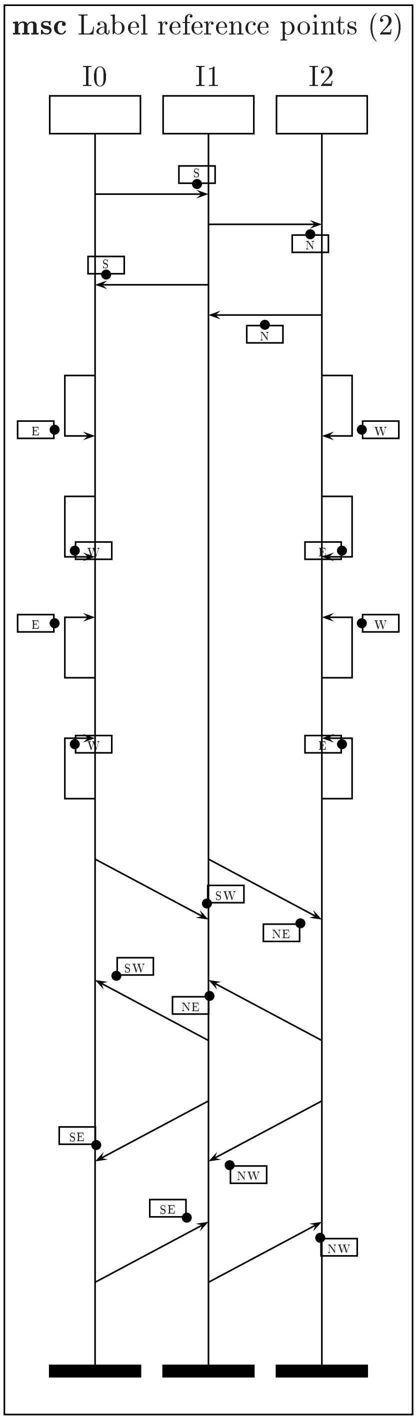
[5] L. Lamport. *LaTeX—A Document Preparation System—User's Guide and Reference Manual*. Adsison-Wesley, 2nd edition, 1994. Updated for LaTeX 2_ε.

[6] E. Rudolph, P.Graubmann, and J. Grabowski. Tutorial on message sequence charts (MSC'96). In *FORTE*, 1996.




```
\begin{msc}{Label reference points}
\declinst{m0}{I0}{}
\declinst{m1}{I1}{}
\declinst{m2}{I2}{}
\nextlevel

\mess{S}{m0}{m1}
\nextlevel
\mess{N}[b]{m1}{m2}
\nextlevel[2]

\mess{S}{m1}{m0}
\nextlevel
\mess{N}[b]{m2}{m1}
\nextlevel[2]

\mess{E}{m0}{m0}[2]
\mess[r]{W}{m2}{m2}[2]
\nextlevel[4]
\mess{W}[r]{m0}{m0}[2]
\mess[r]{E}[l]{m2}{m2}[2]
\nextlevel[6]

\mess{E}{m0}{m0}[-2]
\mess[r]{W}{m2}{m2}[-2]
\nextlevel[4]
\mess{W}[r]{m0}{m0}[-2]
\mess[r]{E}[l]{m2}{m2}[-2]
\nextlevel[2]

\mess{SW}{m0}{m1}[2]
\mess{NE}[b]{m1}{m2}[2]
\nextlevel[6]

\mess{SW}{m1}{m0}[-2]
\mess{NE}[b]{m2}{m1}[-2]
\nextlevel[2]

\mess{SE}{m1}{m0}[2]
\mess{NW}[b]{m2}{m1}[2]
\nextlevel[6]

\mess{SE}{m0}{m1}[-2]
\mess{NW}[b]{m1}{m2}[-2]
\nextlevel[2]
\end{msc}
```

Figure 2: Reference points of message labels




```
\begin{msc}{Label reference points (2)}
\declinst{m0}{I0}{}
\declinst{m1}{I1}{}
\declinst{m2}{I2}{}
\nextlevel

\mess{S}{m0}[.9]{m1}
\nextlevel
\mess{N}[b]{m1}[.9]{m2}
\nextlevel[2]

\mess{S}{m1}[.9]{m0}
\nextlevel
\mess{N}[b]{m2}{m1}
\nextlevel[2]

\mess{E}{m0}[.9]{m0}[2]
\mess[r]{W}{m2}[.9]{m2}[2]
\nextlevel[4]
\mess{W}[r]{m0}[.9]{m0}[2]
\mess[r]{E}[l]{m2}[.9]{m2}[2]
\nextlevel[6]

\mess{E}{m0}[.9]{m0}[-2]
\mess[r]{W}{m2}[.9]{m2}[-2]
\nextlevel[4]
\mess{W}[r]{m0}[.9]{m0}[-2]
\mess[r]{E}[l]{m2}[.9]{m2}[-2]
\nextlevel[2]

\mess{SW}{m0}[.9]{m1}[2]
\mess{NE}[b]{m1}[.9]{m2}[2]
\nextlevel[6]

\mess{SW}{m1}[.9]{m0}[-2]
\mess{NE}[b]{m2}[.9]{m1}[-2]
\nextlevel[2]

\mess{SE}{m1}[.9]{m0}[2]
\mess{NW}[b]{m2}[.9]{m1}[2]
\nextlevel[6]

\mess{SE}{m0}[.9]{m1}[-2]
\mess{NW}[b]{m1}[.9]{m2}[-2]
\nextlevel[2]
\end{msc}
```

Figure 3: Reference points of shifted message labels

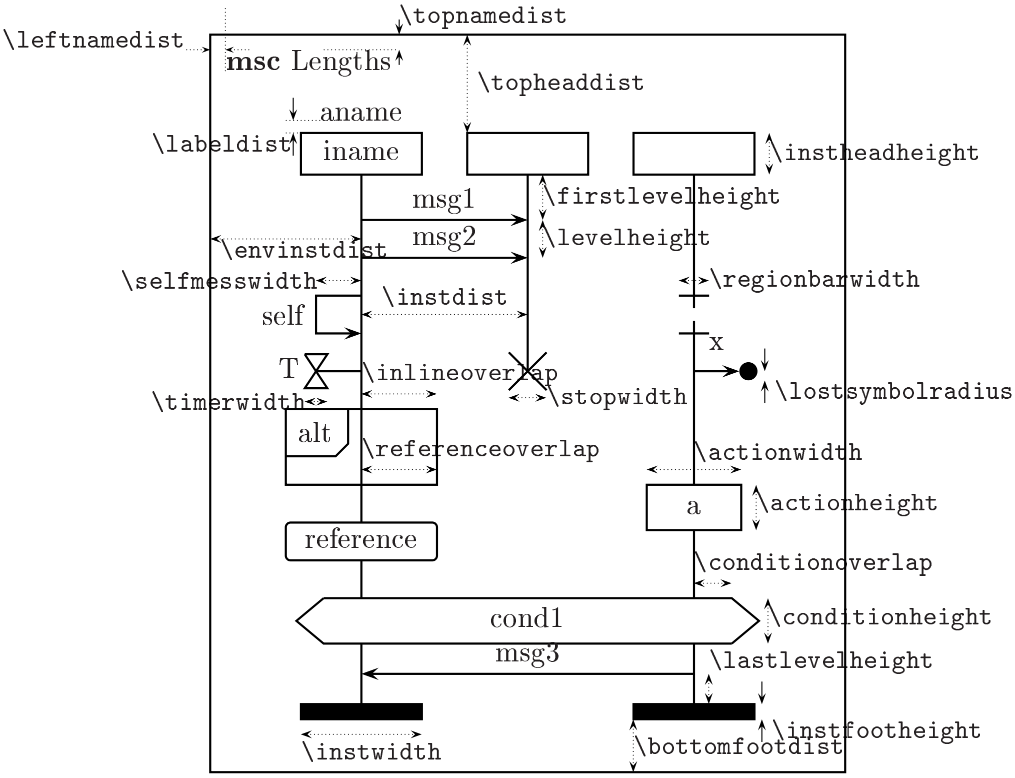


Figure 4: User controllable parameters

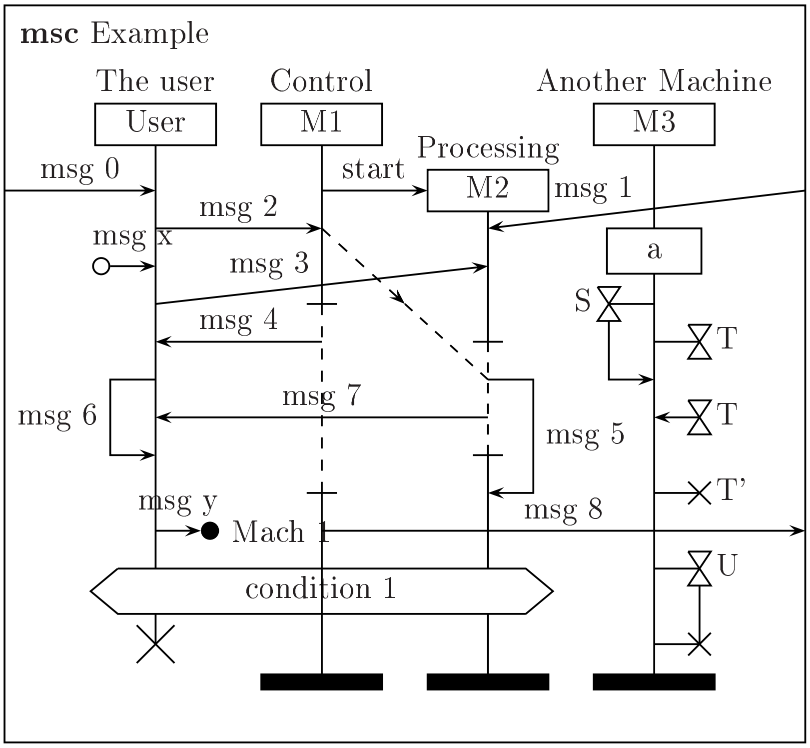


Figure 5: A menagerie of MSC symbols