

# Msc-generator

---

A tool to draw message sequence charts  
(version v4.3.0, 9 September 2014)

Zoltan R. Turanyi

---

This manual is for Msc-generator (version v4.3.0, 9 September 2014), a tool to draw message sequence charts from a textual description.

Please visit <https://sourceforge.net/projects/msc-generator/> to download the latest version.

Msc-generator is a program that parses textual Message Sequence Chart descriptions and produces graphical output in a variety of file formats, or as a Windows OLE embedded object. Message Sequence Charts (MSCs) are a way of representing entities and message interactions between those entities over some time period. MSCs are often used in combination with SDL. MSCs are popular in telecom and data networks and standards to specify how protocols operate. MSCs need not be complicated to create or use. Msc-generator aims to provide a simple text language that is clear to create, edit and understand, and which can be transformed into images. Msc-generator is a potential alternative to mouse-based editing tools, such as Microsoft Visio.

This version of msc-generator is a heavily extended and completely rewritten version of the 0.08 version of Michael C McTernan's mscgen. It has a number of enhancements, but does not support ismaps (clickable URLs embedded into the image). The original tool was more geared towards describing interprocess communication, this version is more geared towards networking.

Msc-generator builds on lex, yacc and cairo. A Linux and Windows port is maintained. The Windows version is written using MFC.

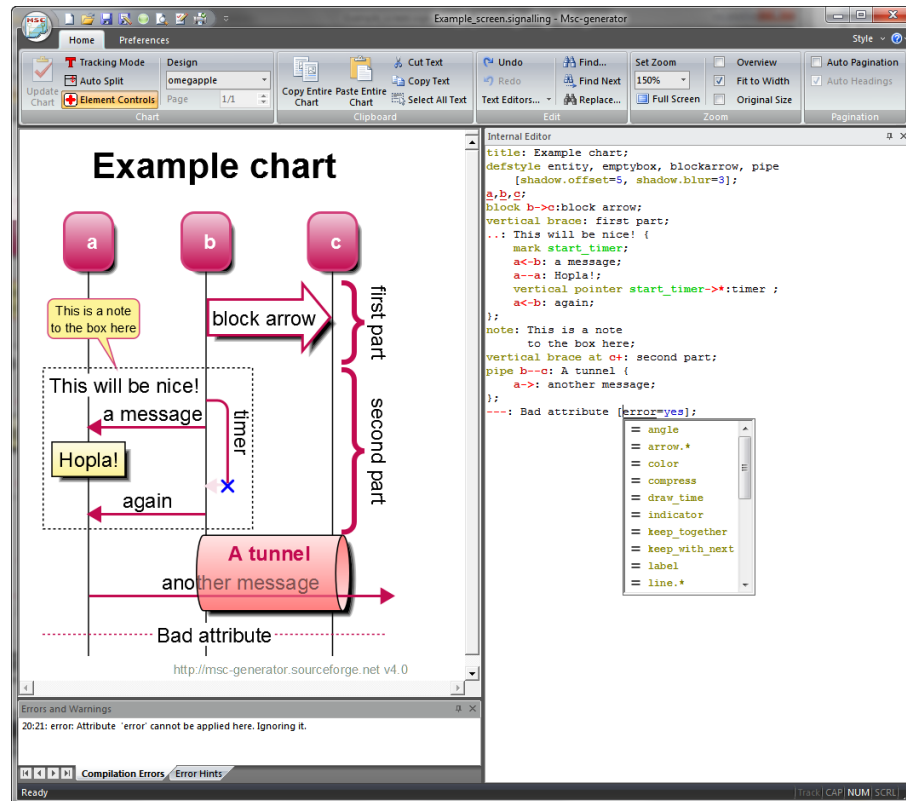
# 1 What's new in Msc-generator 4.4

The improvements added since version 4.2 are listed below. If you are new to Msc-generator, you should probably skip this section and start with [Chapter 2 \[Getting Started\]](#), [page 3](#).

- Added the ability to export label positioning information to the command-line version. If you specify 'lmap' as target (short for label map) you get a text file (with .map extension by default), listing all labels (of arrows, boxes and entities) in the chart with their coordinates (and page number). This is useful if you want to add tooltips or hyperlinks to charts embedded into, e.g., web pages. See [Section 4.10.1 \[Label Maps\]](#), [page 40](#) for more info. (4.3)
- Added the `text at` command and a new symbol `symbol text`. See [Section 5.15.3 \[Inline text\]](#), [page 88](#) for details. (4.4)
- Revamped Windows GUI hinting mechanism. Now it provides hints at more locations. Also added hints for keywords and text formatting escapes. In addition, hints now have tooltip explanations to aid you. (4.3.5)
- Added two built-in markers: `chart.top` and `chart.bottom`
- Some bugfixes submitted by helpful users.

## 2 Getting started

On Windows Msc-generator is installing as a regular application. You can start it directly, by clicking on a file with .signalling extension or by opening an embedded chart.



The Msc-generator window has the usual elements of a Windows application: menu bar, a ribbon and a status bar. We will briefly discuss these here and give a more detailed description in [Chapter 4 \[Usage Reference\]](#), page 32.

You can use the scrollbars to navigate around in the chart. You can also grab the chart by the mouse and drag it (if not all of it fits into the window).

You can also reposition the pane of the internal editor and the error list by clicking on their title bar and dragging them to a new location. On the example above, the internal editor has been moved to the right side from the left (which is the default). You can even create floating windows out of these panes.

If you accidentally close the internal editor, use the 'Text Editors...' button on the ribbon and re-select 'Internal Editor'.

### 2.1 Working with Charts

Msc-generator has a built-in text editor, with color syntax highlighting. You can freely edit the chart description there. When you are ready, press the 'Update Chart' button on the ribbon (or Ctrl+W or F2 on the keyboard) and the visual view of the chart will get updated. Any error or warning messages will show up in a panel at the bottom.

You can use the Main button on the ribbon or the quick access items in the window title to load/save the file. The file format is simply text, the very same that you edit inside Msc-generator's text editor. You can also save the file in various graphics formats using the **Main|Export...** item. Pressing the Main button you also find the usual Print and Print Preview commands.

The Clipboard pane on the ribbon has two set of Copy/Paste operations: one for text in the text editor and a separate set for the entire chart. If you use paste for the entire chart, then its whole content is replaced, whereas if you paste into the editor, the content of the clipboard will be inserted.

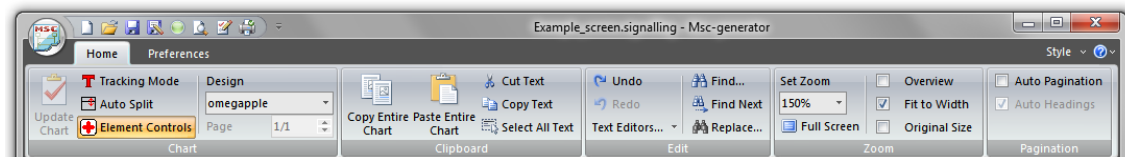
You can also perform undo or redo from the Edit pane of the ribbon or by pressing Ctrl+Z or Ctrl+Y. Similar search and replace operations for the text editor can also be accessed from the Edit pane.

Finally, there is a separate button in the Edit pane to start and stop the internal or an external text editor (see [Section 4.2 \[External Editor\]](#), page 32). The latter is useful in case you prefer to use your own editor.

## 2.2 The Ribbon

You can find two additional panes on the ribbon. The Zoom pane enables you to set various zoom options (see [Section 2.3 \[Zooming\]](#), page 4) or to switch to full screen viewing mode.

The buttons in the second column of the Chart pane enables you to enter tracking mode (see [Section 2.4 \[Tracking Mode\]](#), page 5); to turn automatic splitting ([Section 2.5 \[Auto Split\]](#), page 5) on or off; or to enable the showing of collapse/expand controls for entity groups and boxes ([Section 2.6 \[Collapsing and Expanding\]](#), page 5).



The second column of the Chart pane has two controls. The first one is the design selector. By selecting a chart design here you can override the selection in the source file. This is an easy way of reviewing how your chart would look like in a particular design. See [Section 5.19 \[Chart Designs\]](#), page 93 for more info on chart designs.

The second edit box can be used to select which page of the chart is displayed. If 'all' is selected then pagination is ignored and the whole chart is shown. (See [Section 5.14 \[Multiple Pages\]](#), page 83 for more info on pagination commands.)

## 2.3 Zooming

You can zoom the chart in and out using the commands on the Zoom pane. The zoom drop-down allows setting a specific zoom value. However, the easiest way to zoom is to use the mouse wheel with the Ctrl key pressed.

You can easily set the right zoom factor by selecting certain Zoom pane buttons. **Overview** adjusts zoom to fit the entire chart into the window. This is useful to get an overview

of a chart. *Fit to width* changes the zoom factor to fit the width of the chart to the current window. Finally, *Original Size* sets the magnification back to 100%.

You can also make Msc-generator apply one of the above three zoom adjustments after every update, page change or window re-size by selecting checkboxes besides the above command buttons.

You can also view the chart in full screen mode, by pressing F11. Mouse zooming and panning works in full screen mode. A small toolbar enables you to flip pages, return to the all pages view or to toggle Auto Split (see below). You can exit full screen mode by pressing Escape.

## 2.4 Tracking Mode

If you click an arrow, entity or any other visual element on the chart, it is briefly highlighted and the corresponding text is selected in the editor. This is useful to quickly jump to a certain element in the chart text.

If you double-click the chart (try the background) you enter Tracking Mode, where the above behaviour becomes permanent. Visual elements are selected just by hovering above them. You can enter tracking mode also by the ‘T’ button on the Chart pane or by pressing Ctrl+T. If you move around in the text editor, the visual element corresponding to the text around the current cursor position is highlighted.

You can leave Tracking Mode by pressing Escape.

## 2.5 Auto Split

When working with a large chart, it is sometimes needed to zoom in to an area of it. In case the viewing area is towards the bottom of the chart, it is often difficult to know which entity line belongs to which entity. In such cases turning Auto Split on will result in the splitting of the view into two parts, the upper one showing the entity headings. If zooming is applied Msc-generator always attempts to resize the upper view part to show the entities only.

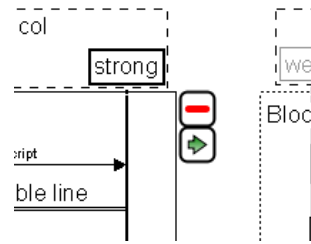
Note that it is possible to define charts where there is no meaningful row of entity headings at the top. In such cases, Msc-generator will get confused and Auto Split is of no use. In case of multi-page charts, Auto Split will show automatic headers only (Instead of ‘`newpage; heading;`’ use rather ‘`newpage [auto_heading=yes];`’ command.)

Note that you can use the slider to change, where the chart is split, both with and without Auto Split. The difference is that when Auto Split is on, the split is reset to headings after a compilation or a page change.

Auto Split also works in Full Screen mode.

## 2.6 Collapsing and Expanding

Msc-generator allows you to collapse boxes and entity groups. This way you can show only a simplified view of the procedure described by the chart text. E.g., instead of many arrows comprising a part of the procedure, a simple box is shown as a summary.



If you move the mouse over a chart element that can be collapsed (or is already collapsed), control icons appear at its top right corner. The control with the minus sign collapses the element, the control with the plus sign expands a collapsed element, while the green arrow collapses the element into a block arrow. The last icon will only appear for boxes, which are not part of a box series (Section 3.4 [Drawing Boxes], page 18).

You can disable the showing of such controls via the red plus button on the Chart pane.

Expanding and collapsing can also be set via the ‘**collapsed**’ attribute and hence is available for the command-line version, as well. It is most useful, however, for interactive work. Any collapse/expand setting via the GUI overrides the one specified by attributes. Such overrides are saved with embedded charts, but naturally not when the chart is saved to disk as text.

If you double click any element that has controls (can be collapsed/expanded) the first control is activated (even if controls are not shown). This essentially toggles collapse/expand status.

## 2.7 Embedding a Chart in a Document

You can take a chart and embed it as a component in a compound document such as a Word, Excel or Powerpoint document. To do this, copy the chart to the clipboard by clicking on the *Copy Entire Chart* button and paste it into the compound document<sup>1</sup>. Later you can edit the chart by double clicking the chart in the document<sup>2</sup>.

Right clicking an embedded chart in a document will bring up a menu of options, where you can select **Edit** or **Open** for editing in a separate window; or **View Full Screen** to view (but not edit) the chart in full screen.

We note that page and chart design settings you select on the ribbon are saved with embedded documents, but not when you save the chart into a file.

## 2.8 Command-line Tool

The command line version of Msc-generator runs on both Linux and Windows. On Windows it is installed to the same directory as the windowed application. That directory is included in the PATH, so you can call it from anywhere.

The command line version of Msc-generator supports PNG, PDF, EPS, SVG file formats, and EMF on Windows. To start it simply type

```
msc-gen -T pdf inputfile.signalling
```

This will give you `inputfile.pdf`. You can change ‘pdf’ to get the other file formats. If you omit the ‘-T’ switch altogether, a PNG will be generated.

<sup>1</sup> Make sure you paste the chart using ‘Paste Special...’ as an ‘Msc-generator Signalling Chart Object’.

<sup>2</sup> In place editing is no longer supported from version 3.4.1.

If Msc-generator has successfully generated an output, it prints `'Success.'`. Instead, or in addition, it may print warnings or errors, when it does not understand something.



## 3 Language Tutorial

In this chapter we give a step-by-step introduction into the language of Msc-generator. At the end you will master most of the language to create charts. Further details (mostly on controlling appearance) are provided in [Chapter 5 \[Language Reference\]](#), page 42.

### 3.1 Defining Arrows

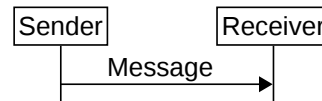
Message sequence charts consists of *entities* and *messages*. The simplest file consists of a single message between two entities: a 'Sender' and a 'Receiver'.

**Sender->Receiver;**



The message may have a label, as well.

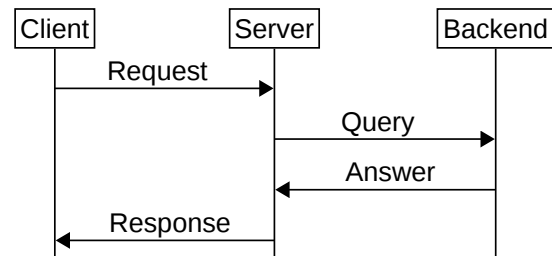
**Sender->Receiver:** Message;



A more complicated procedure would be to request some information from a server, which, in turn, queries a backend. Note that everything in a line after a '#' is treated as a comment and is ignored by Msc-generator.

```

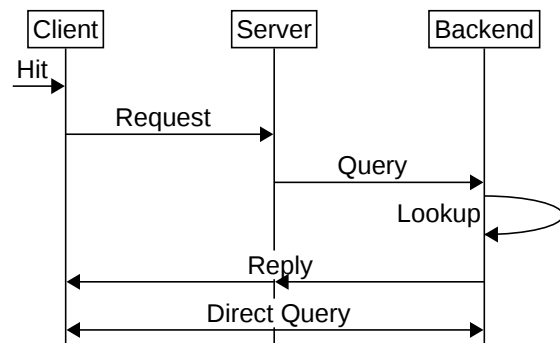
#A more complex procedure
Client->Server: Request;
Server->Backend: Query;
Server<-Backend: Answer;
Client<-Server: Response; #final
  
```



Arrows can take various forms, for example they can be bi-directional or can span multiple entities. They can also start and end at the same entity and can come from or go to "outside"

```

->Client: Hit;
Client->Server: Request;
Server->Backend: Query;
Backend->Backend: Lookup;
Client<-Server<-Backend: Reply;
Client<->Backend: Direct Query;
  
```

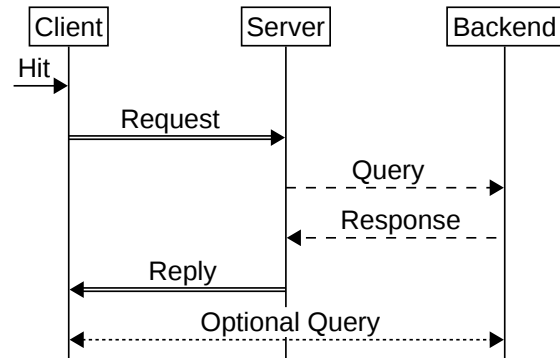


It is also possible to make use of various arrow types, such dotted, dashed and double line. To achieve this the '->' symbol need to be replaced with '>', '>>' and '=>', respectively.

```

->Client: Hit;
Client=>Server: Request;
Server->>Backend: Query;
Server<<Backend: Response;
Client<=Server: Reply;
Client<>Backend: Optional Query;

```

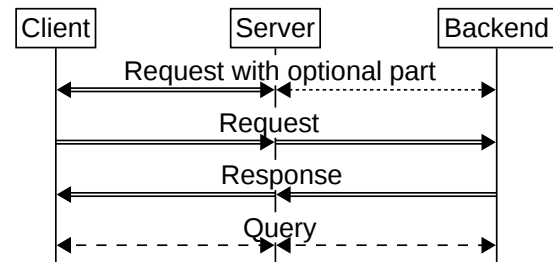


It is also possible to use different line styles for different segments of an arrow - but all must be of the same direction. (That is, it is not possible to write ‘a->b<-c’, for example.) In addition, for multi-segment arrows the dash ‘-’ symbol can be used in the second and following segments, as a shorthand. In this case the added segment will have the same line style as the first one.

```

Client<=>Server<>Backend:
    Request with optional part;
Client=>Server-Backend: Request;
Client<=Server-Backend: Response;
Client<<>>Server-Backend: Query;

```

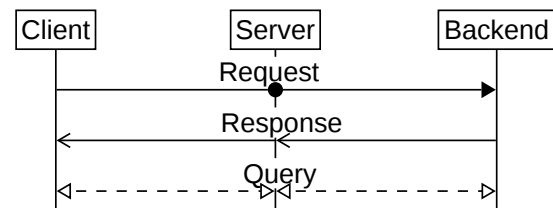


It is possible to change the type of the arrowhead. The arrowhead type is an *attribute* of the arrow. Attributes can be specified between square brackets before or after the label, as shown below. A variety of arrow-head types are available, for a full list of arrow attributes and arrowhead types See [Section 5.3 \[Specifying Arrows\]](#), page 46.

```

Client->>Server-Backend: Request
    [arrow.midtype=dot];
Client<-Server-Backend: Response
    [arrow.type=line];
Client<<>>Server-Backend: Query
    [arrow.type=empty];

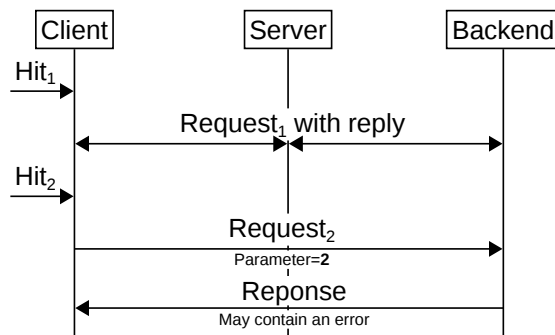
```



Often the message has not only a name, but additional parameters, that need to be displayed. The label of the arrows can be made multi-line and one can apply font sizes and formatting, as well. This is achieved by inserting formatting characters into the label text. Each formatting character begins with a backslash ‘\’. ‘\b’, ‘\i’ and ‘\u’ toggles bold, italics and underline, respectively. ‘\-’ switches to small font, ‘\+’ switches back to normal size, while ‘\^’ and ‘\\_’ switches to superscript and subscript, respectively. ‘\n’ inserts a line break. You can also add a line brake by simply typing the label into multiple lines.

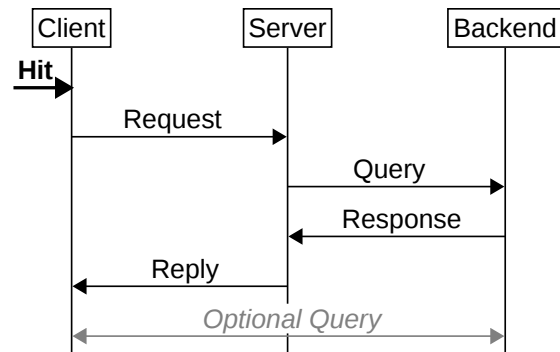
Leading and tailing whitespace will be removed from such lines so you can indent the lines in the source file to look nice.

```
->Client: Hit\1;
Client->Server-Backend: Request\1\+ with reply;
->Client: Hit\2;
Client->Backend: Request\2\-\nParameter=\b2;
Client<-Backend: Reponse
                \-May contain an error;
```



Arrows can further be differentiated by applying *styles* to them. Styles are packages of attributes with a name. They can be specified in square brackets like an attribute that takes no value. Msc-generator has two pre-defined styles ‘**weak**’ and ‘**strong**’, that exists in all chart designs<sup>1</sup>. They will make the arrow look less or more emphasized, respectively. The actual appearance depends on the chart design, in this basic case they represent gray color and thicker lines with bold text, respectively<sup>2</sup>.

```
->Client: Hit [strong];
Client->Server: Request;
Server->Backend: Query;
Server<-Backend: Response;
Client<-Server: Reply;
Client<-Backend: Optional Query
                [weak];
```



Msc-generator places arrows one-by-one below each other. In case of many arrows, this may result in a lot of vertical space wasted. To reduce the size of the resulting diagram, a *chart option* can be specified, which compresses the diagram, where possible. You can read more on chart options, see [Section 5.9.4 \[Compression and Vertical Spacing\]](#), page 73.

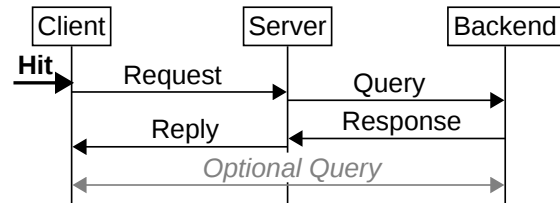
<sup>1</sup> You can define your own styles, as well, see [Section 5.18 \[Defining Styles\]](#), page 91.

<sup>2</sup> For more on chart designs [Section 5.19 \[Chart Designs\]](#), page 93.

```

compress=yes;
->Client: Hit [strong];
Client->Server: Request;
Server->Backend: Query;
Server<-Backend: Response;
Client<-Server: Reply;
Client<->Backend: Optional Query
[weak];

```

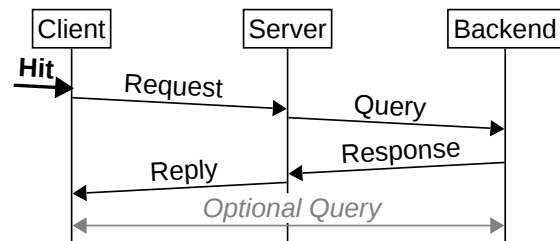


You can use the 'angle' chart option (or attribute) to make the arrows slanted. Simply specify a value in degrees. Note that bi-directional arrows will not be slanted.

```

compress=yes;
angle=3;
->Client: Hit [strong];
Client->Server: Request;
Server->Backend: Query;
Server<-Backend: Response;
Client<-Server: Reply;
Client<->Backend: Optional Query
[weak];

```

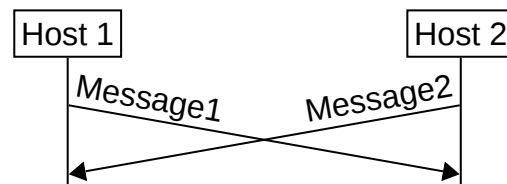


Normally, Msc-generator attempts to avoid overlaps between elements by placing them one below (or sometimes besides) each other. If you want to show messages crossing each other, you need overlapping arrows. For this, you can use the **overlap** keyword. Arrows marked such are allowed to be overlapped by subsequent arrows.

```

hscale=1.5;
angle=10;
H1: Host 1;
H2: Host 2;
overlap H1->H2: \plMessage1;
H2->H1: \prMessage2;

```

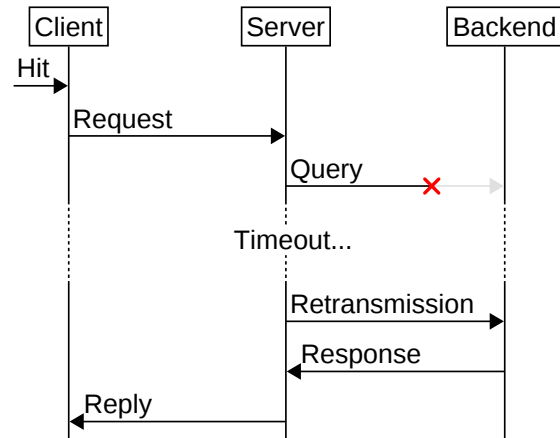


Finally, you can also indicate a lost message by marking the segment of the loss with an asterisk '\*'. For example:

```

defstyle arrow [text.ident = left];
->Client: Hit;
Client->Server: Request;
Server->*Backend: Query;
...: Timeout...;
Server->Backend: Retransmission;
Server<-Backend: Response;
Client<-Server: Reply;

```



## 3.2 Defining Entities

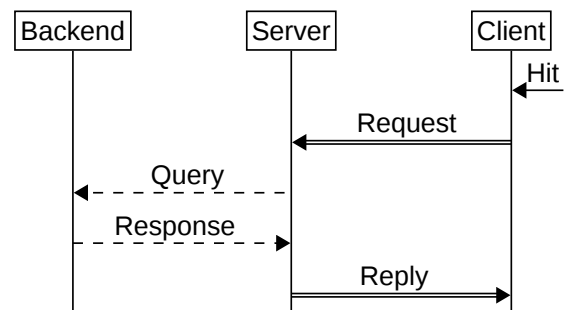
Msc-generator, by default draws the entities from left to right in the order they appear in the chart description. In the examples above, the first entity to appear was always the 'Client', the second 'Server' and the third 'Backend'.

Often one wants to control, in which order entities appear on the chart. This is possible, by listing the entities before actual use. On the example below, the order of the entities are reversed. Note that we have reversed the first arrow to arrive to the 'Client' from the right.

```

Backend, Server, Client;
Client<-: Hit;
Client=>Server: Request;
Server>>Backend: Query;
Server<<Backend: Response;
Client<=Server: Reply;

```



Often the name of the entity need to be multi-line or need to contain formatting characters, or is just too long to type many times. You can overcome this problem by specifying a label for entities. The name of the entity then will be used in the chart description, but on the chart the label of the entity will be displayed. The 'label' is an attribute of the entity and can be specified between square brackets after the entity name, before the comma, as shown below. (You can specify entity attributes only when explicitly defining an entity and not if you just start using them without listing them first.)

```

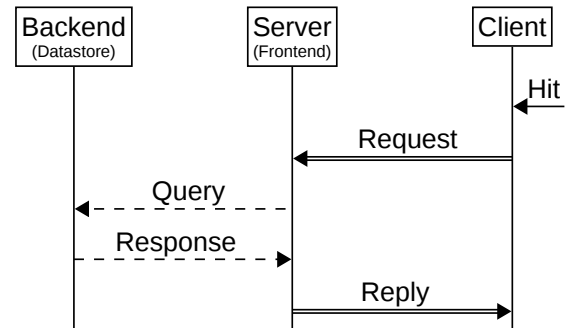
B [label="Backend\n\n- (Datastore)"],
S [label="Server\n\n- (Frontend)"],
C [label="Client"];

```

```

C<-: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C=>S: Reply;

```



You can also use the colon-notation to specify entity labels, similar to arrows. The above example can thus be written as below. Note that the entity definitions are now terminated by a semicolon – commas would be treated as part of the label.

```

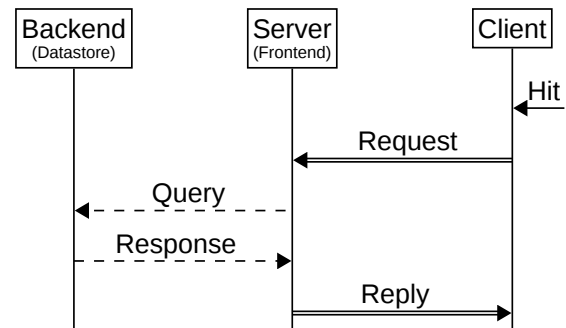
B: Backend\n\n- (Datastore);
S: Server\n\n- (Frontend);
C: Client;

```

```

C<-: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C=>S: Reply;

```



Entities can also be specified as ‘**weak**’ or ‘**strong**’, by applying these styles the same way as for arrows. You can also assign various shapes to the entity headings via the **shape** attribute.

```

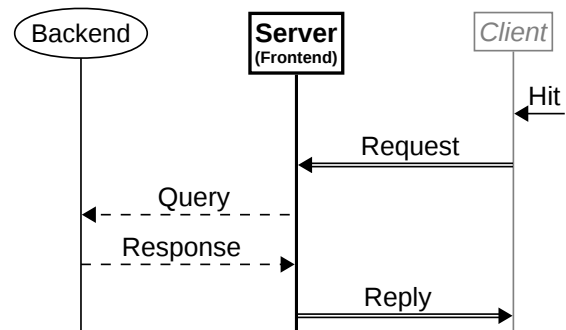
B: Backend [shape=def.oval];
S: Server\n\n- (Frontend) [strong];
C: Client [weak];

```

```

C<-: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C=>S: Reply;

```



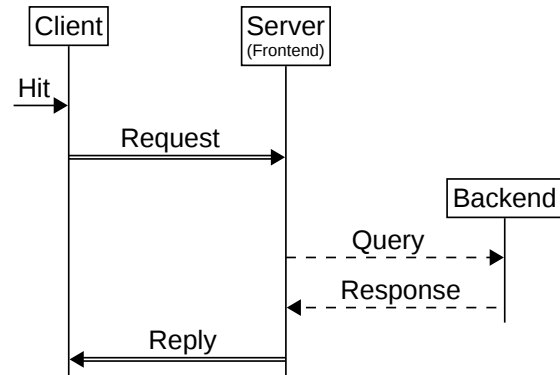
Entities can be turned on and off at certain points in the chart. An entity that is turned off, will not have its vertical line displayed. This is useful if the chart has many entities, but one is involved only in a small part of the process. An entity can be turned off by typing **hide** followed by the name of the entity. You can turn it later back on with the **show** keyword followed by the entities to turn on. When **hide** is used for an entity right

at its definition, it will start hidden and its heading is not drawn at the place of definition. However, when it is later turned on, a heading will be shown.

```

C: Client;
S: Server\n\-(Frontend);
hide B: Backend;
->C: Hit;
C=>S: Request;
show B;
S>>B: Query;
S<<B: Response;
hide B;
C<=S: Reply;

```

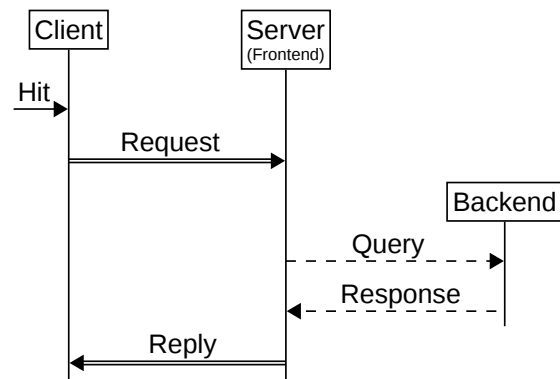


Not showing an entity from the beginning of the chart can also be achieved by simply defining the entity later. Note that this is different from simply starting to use an entity later. When you start using an entity without explicitly defining it first, it will appear at the top of the chart, not only where started using it first. (See earlier examples.)

```

C: Client;
S: Server\n\-(Frontend);
->C: Hit;
C=>S: Request;
B: Backend;
S>>B: Query;
S<<B: Response;
hide B;
C<=S: Reply;

```

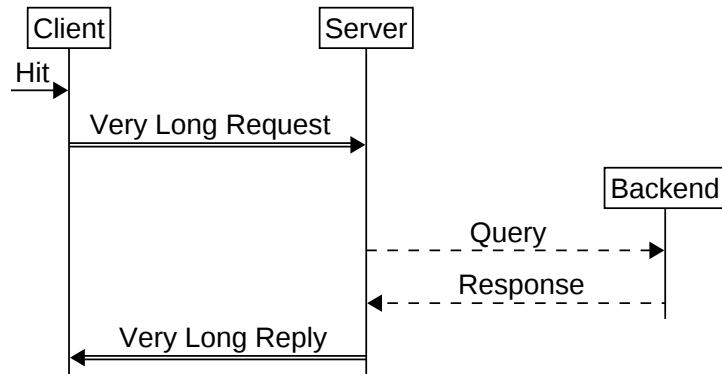


Sometimes the vertical space between entities is just not enough to display a longer label for an arrow. In this case use the 'hscale' chart option to increase the horizontal spacing. It can be set to a numerical value, 1 being the default.

```

hscale=1.3;
C: Client;
S: Server;
->C: Hit;
C=>S: Very Long Request;
B: Backend;
S>>B: Query;
S<<B: Response;
B [show=no];
C<=S: Very Long Reply;

```

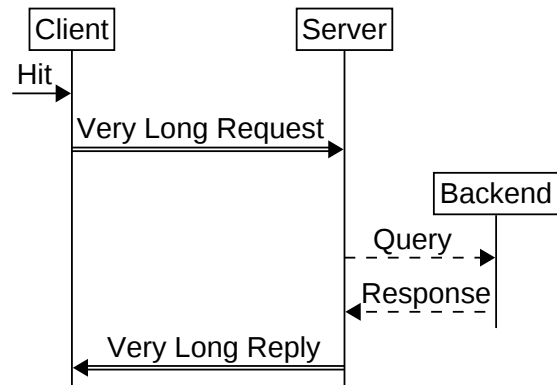


Or you can simply set it to 'auto', which creates variable spacing, just as much as is needed.

```

hscale=auto;
C: Client;
S: Server;
->C: Hit;
C=>S: Very Long Request;
B: Backend;
S>>B: Query;
S<<B: Response;
B [show=no];
C<=S: Very Long Reply;

```

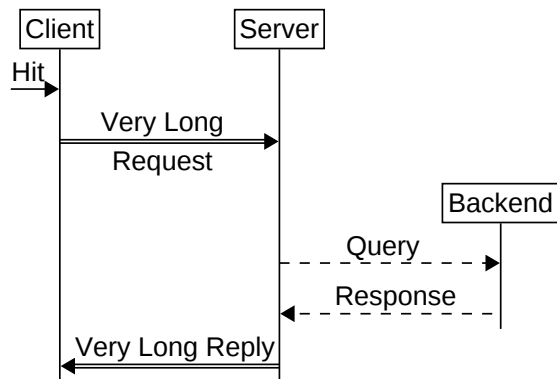


Alternatively, you can instruct Msc-generator to apply word wrapping to the labels of arrows, to fit into the available space, by setting the 'text.wrap' chart option to 'yes'.

```

text.wrap=yes;
C: Client;
S: Server;
->C: Hit;
C=>S: Very Long Request;
B: Backend;
S>>B: Query;
S<<B: Response;
B [show=no];
C<=S: Very Long Reply;

```



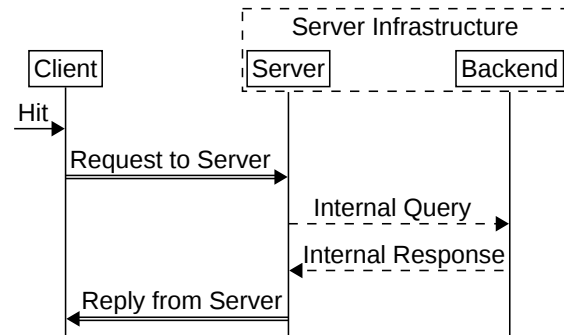
It is possible to define entity groups, to indicate logical relations between various entities. Use curly braces ('{' and '}') after an entity definition (after any potential label and attributes).



```

hscale=auto;
C: Client;
SI: Server Infrastructure {
    S: Server;
    B: Backend;
};
->C: Hit;
C=>S: Request to Server;
S>>B: Internal Query;
S<<B: Internal Response;
C<=S: Reply from Server;

```

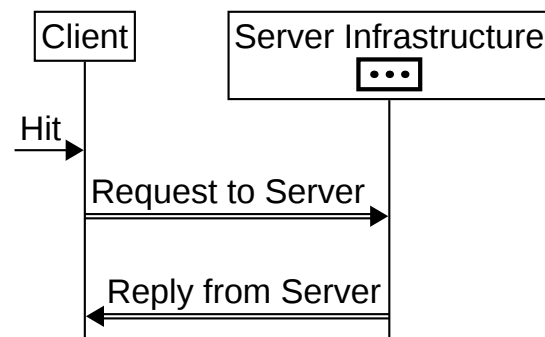


It is also possible to collapse a group entity hiding details of the process. This can be done either via the 'collapsed' attribute or, on Windows, using the GUI. Elements that disappear leave a small indicator (box with 3 dots). The collapsed entity group also includes an indicator to show that further entities are hidden within. (Indicators can be turned off by the 'indicator' chart option.

```

hscale=auto;
C: Client;
SI: Server Infrastructure
    [collapsed=yes] {
        S: Server;
        B: Backend;
    };
->C: Hit;
C=>S: Request to Server;
S>>B: Internal Query;
S<<B: Internal Response;
C<=S: Reply from Server;

```

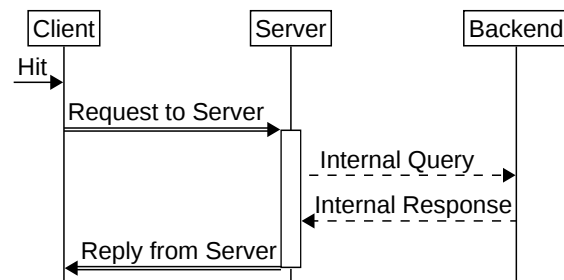


Entities can be *activated*. This results in the entity line becoming a thin rectangle instead. If you do this immediately after an arrow the activation will happen at the tip of the arrow indicating that the cause of the activation is the arrow.

```

hscale=auto;
C: Client;
S: Server;
B: Backend;
->C: Hit;
C=>S: Request to Server;
activate S;
S>>B: Internal Query;
S<<B: Internal Response;
C<=S: Reply from Server;
deactivate S;

```



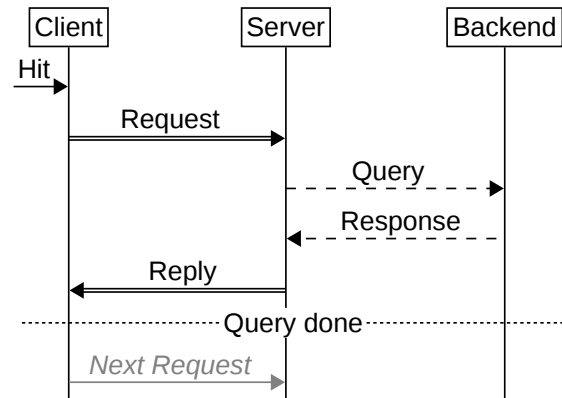
### 3.3 Dividers

In a message sequence chart it is often important to segment the process into multiple logical parts. You can use the ‘---’ element to draw a horizontal line across the chart with some text, e.g., to summarize what have been achieved so far.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C<=S: Reply;
---: Query done;
C->S [weak]: Next Request;

```

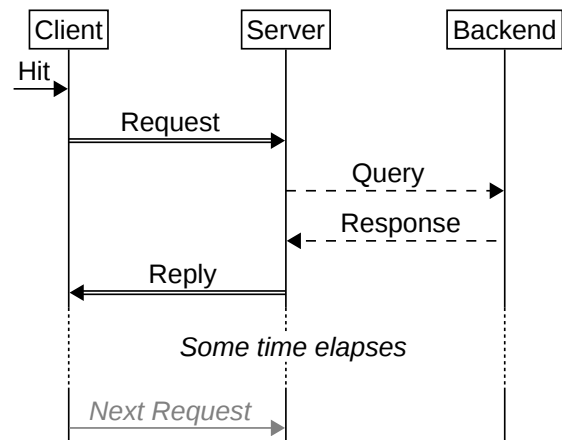


Similar to this, using the ‘...’ element can express the passage of time by making the vertical lines dotted.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C<=S: Reply;
...: \iSome time elapses;
C->S [weak]: Next Request;

```

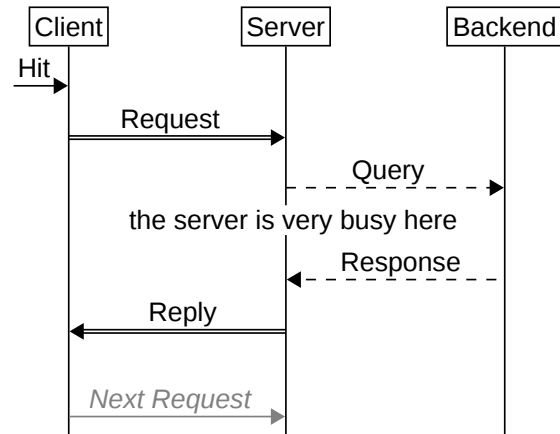


Sometimes one merely wants to add some text to a chart. In that case the empty element can be used either like ‘: text;’.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
C=>S: Request;
S>>B: Query;
: the server is very busy here;
S<<B: Response;
C<=S: Reply;
[];
C->S [weak]: Next Request;

```



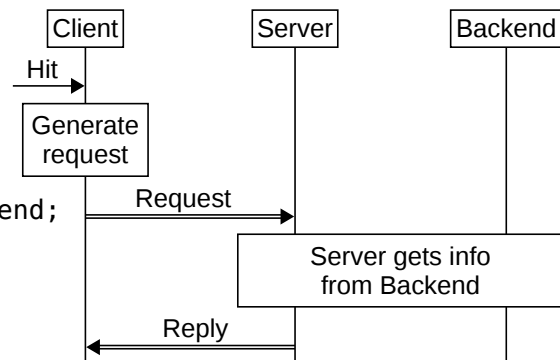
### 3.4 Drawing Boxes

A *box* is a line around one part of the chart. It can be used to add textual comments, group a set of arrows or describe alternative behavior. In their simplest form they only contain text, but they can also encompass arrows. A box spans between two entities, or alternatively around only one.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box C--C: Generate\nrequest;
C=>S: Request;
box S--B: Server gets info\nfrom Backend;
C<=S: Reply;

```

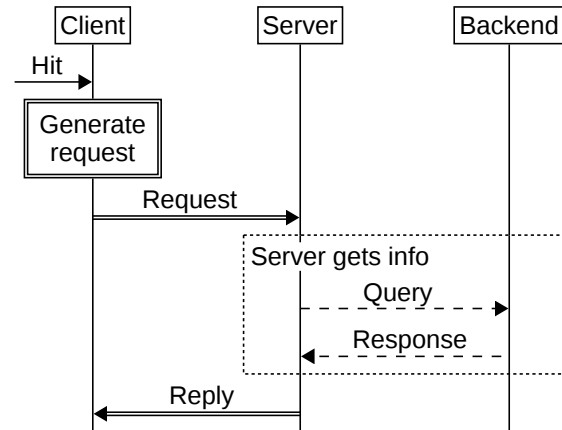


The line around boxes can be dotted, dashed and double line, too, by using ‘. .’, ‘++’ or ‘==’ instead of ‘--’. Boxes can also be used to group a set of arrows. To do this, simply insert the arrow definitions enclosed in curled braces just before the semicolon terminating the definition of the box.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box C==C: Generate\nrequest;
C=>S: Request;
box S..B: Server gets info
{
    S>>B: Query;
    S<<B: Response;
};
C<=S: Reply;

```

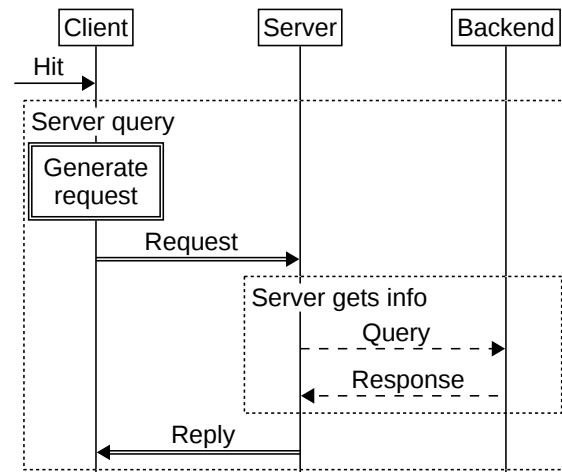


When a box contains arrows, it is not necessary to specify which entities it shall span between, it will be calculated automatically. Also boxes can be nested arbitrarily deep.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box ...: Server query
{
    box C==C: Generate\nrequest;
    C=>S: Request;
    box S..B: Server gets info
    {
        S>>B: Query;
        S<<B: Response;
    };
    C<=S: Reply;
};

```

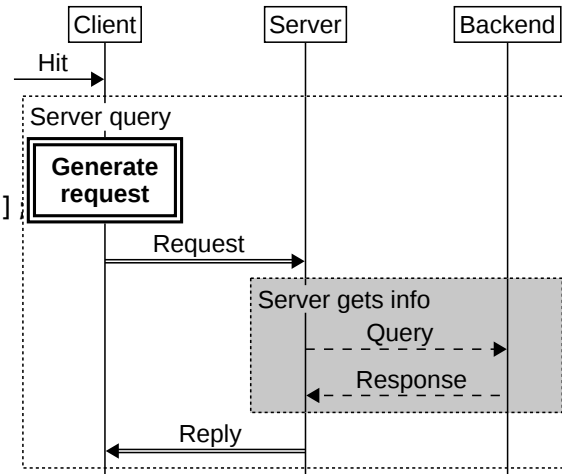


You can shade boxes, by specifying the color attribute. For a full list of box attributes and color definitions, See [Section 5.4 \[Boxes\]](#), page 52, and see [Section 5.10 \[Specifying Colors\]](#), page 74. It is also possible to make a box 'weak' or 'strong'.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box ..: Server query
{
  box C==C: Generate\nrequest [strong]
  C=>S: Request;
  box S..B: Server gets info
    [color=lgray]
  {
    S>>B: Query;
    S<<B: Response;
  };
  C<=S: Reply;
};

```

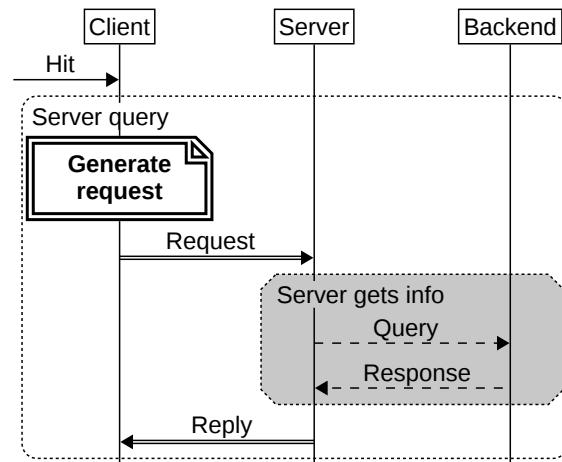


A number of box contours are available via the 'line.corner' attribute.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box ..: Server query
  [line.corner=round]
{
  C==C: Generate\nrequest
    [strong, line.corner=note];
  C=>S: Request;
  box S..B: Server gets info
    [color=lgray,
     line.corner=bevel]
  {
    S>>B: Query;
    S<<B: Response;
  };
  C<=S: Reply;
};

```

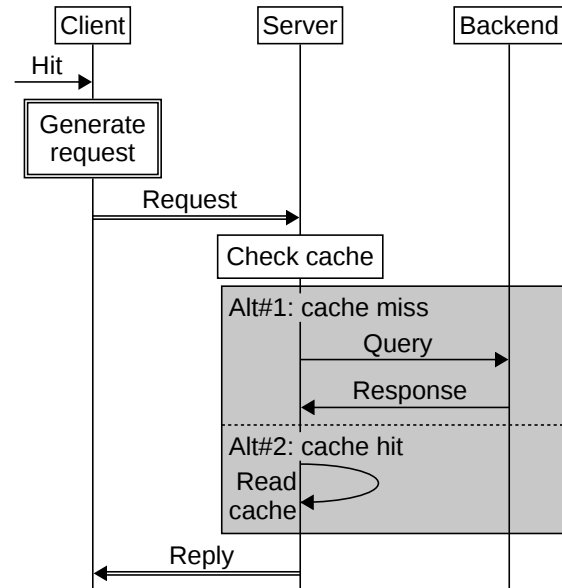


Boxes can express alternatives. To do this, simply concatenate multiple box definition without adding semicolons. These will be drawn with no spaces between. Changing the line style in subsequent boxes impacts the line separating the boxes, otherwise all attributes of the first box are inherited by the subsequent ones.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box C==C: Generate\nrequest;
C=>S: Request;
box S--S: Check cache;
box S--B: Alt\#1: cache miss
    [color=lgray]
{
    S->B: Query;
    S<-B: Response;
}
..: Alt\#2: cache hit
{
    S->S: Read\nocache;
};
C<=S: Reply;

```

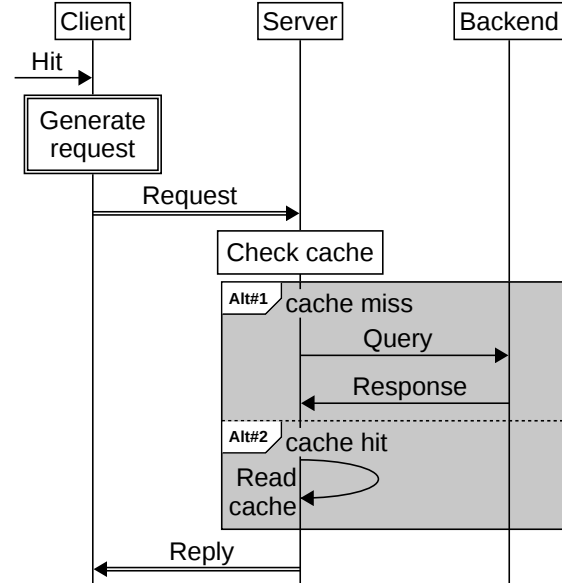


You can use *tags* to label boxes (both standalone or in a series). This can be used to indicate alternatives, loops or optional parts of the sequence.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box C==C: Generate\nrequest;
C=>S: Request;
box S--S: Check cache;
box S--B: cache miss
    [tag="Alt\#1", color=lgray]
{
    S->B: Query;
    S<-B: Response;
}
..: cache hit [tag="Alt\#2"]
{
    S->S: Read\nocache;
};
C<=S: Reply;

```



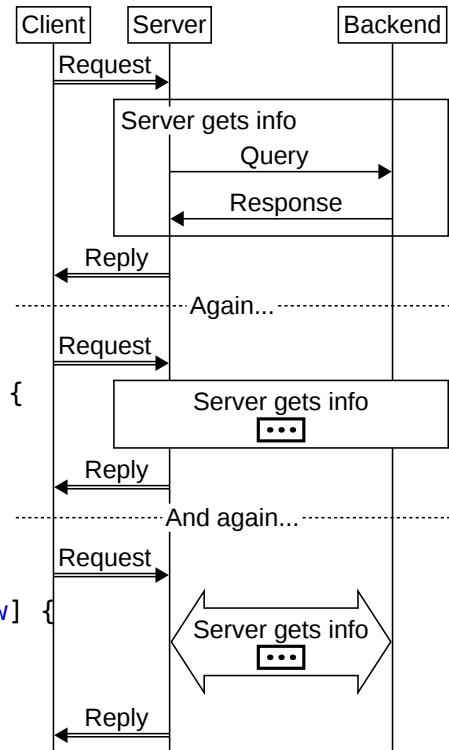
You can observe in the previous example that the '\#' sequence inserts a '#' character into a label. The '\' is needed to differentiate from a comment.

Finally, similar to entity groups, boxes can also be collapsed, if they are not empty. Standalone boxes can be collapsed to an empty box or block arrow by specifying the 'collapsed' attribute (or via the GUI on Windows). This feature is useful to hide or summarize irrelevant parts of the chart and enables quick working with large processes.

```

hscale=auto;
C: Client;
S: Server;
B: Backend;
C=>S: Request;
box S--B: Server gets info {
  S->B: Query;
  S<-B: Response;
};
C<=S: Reply;
---: Again...;
C=>S: Request;
box S--B: Server gets info [collapsed=yes] {
  S->B: Query;
  S<-B: Response;
};
C<=S: Reply;
---: And again...;
C=>S: Request;
box S--B: Server gets info [collapsed=arrow] {
  S->B: Query;
  S<-B: Response;
};
C<=S: Reply;

```



### 3.5 Drawing Things in Parallel

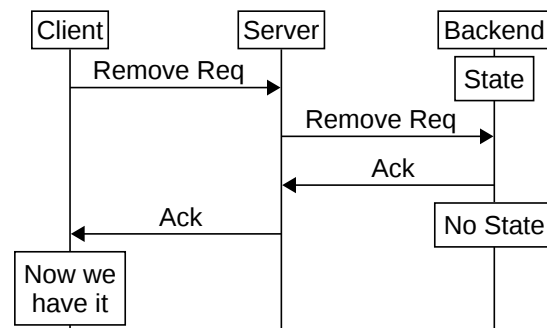
Sometimes it is desired to express that two separate process happen side-by-side. The easiest way to do so is to write ‘parallel’ before any arrow, box or other element. As a result the elements after it will be drawn in parallel with it.

```

C: Client;
S: Server;
B: Backend;

parallel B--B: State;
C->S: Remove Req;
S->B: Remove Req;
S<-B: Ack;
parallel B--B: No State;
C<-S: Ack;
C--C: Now we\nhave it;

```

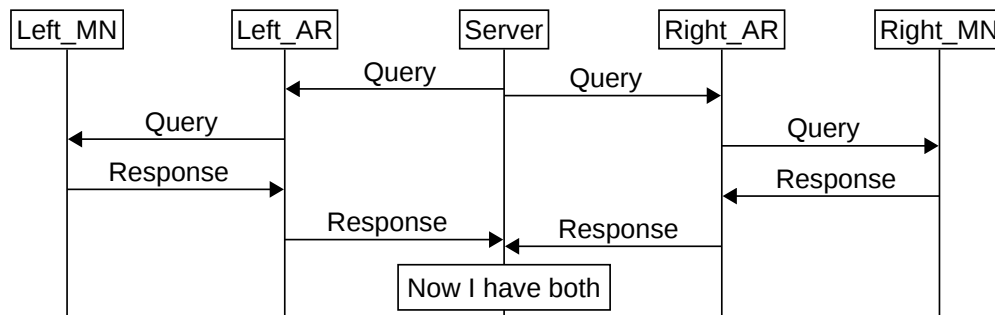


It is also possible to have bigger blocks of action in parallel using *Parallel blocks*. Consider the following example.

```

Left_MN, Left_AR, Server, Right_AR, Right_MN;
{
  Server->Left_AR: Query;
  Left_AR->Left_MN: Query;
  Left_AR<-Left_MN: Response;
  Server<-Left_AR: Response;
} {
  Server->Right_AR: Query;
  Right_AR->Right_MN: Query;
  Right_AR<-Right_MN: Response;
  Server<-Right_AR: Response;
};
box Server--Server: Now I have both;

```



In the above example a central sever is querying two AR entities, which, in turn query MN entities further. The query on both sides happen simultaneously. To display parallel actions side by side, simply enclose the two set of arrows between braces ‘{ }’ and write them one after the other. Use only a single semicolon after the last block. You can have as many flows in parallel as you want. It is possible to place anything in a parallel block, arrows, boxes, or other parallel blocks, as well. You can even define new entities or turn them on or off inside parallel boxes.

The top of each block will be drawn at the same vertical position. The next element below the series of parallel blocks (the "Now I have it" box in our example) will be drawn after the longest of the parallel blocks.

### 3.6 Annotating the Chart

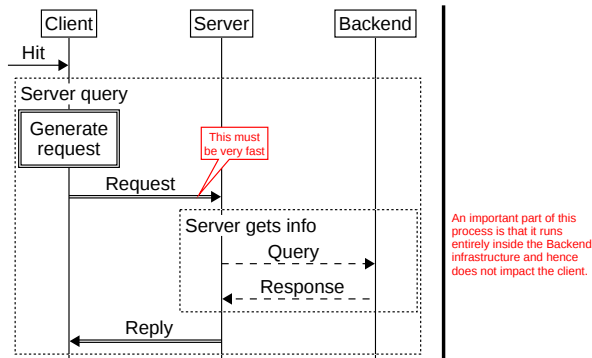
Often it is important to make annotations to the chart detailing what is going on. Msc-generator supports several types of annotations, let’s start with *notes* and *comments*. Both thave a *target element* to which the note or comment is made. Notes appear as small callouts in the chart and should preferably contain short text. Comments, on the other hand appear on the side and allow for more elaborate explanations.



```

defstyle note, comment [text.color=red];
defstyle note [line.color=red];
C: Client;
S: Server;
B: Backend;
->C: Hit;
box ..: Server query
{
    box C==C: Generate\nrequest;
    C=>S: Request;
    note: This must\nbe very fast;
    box S..B: Server gets info
    {
        S>>B: Query;
        S<<B: Response;
    };
    comment:
        An important part of this
        process is that it runs
        entirely inside the Backend
        infrastructure and hence
        does not impact the client.;
    C<=S: Reply;
};

```

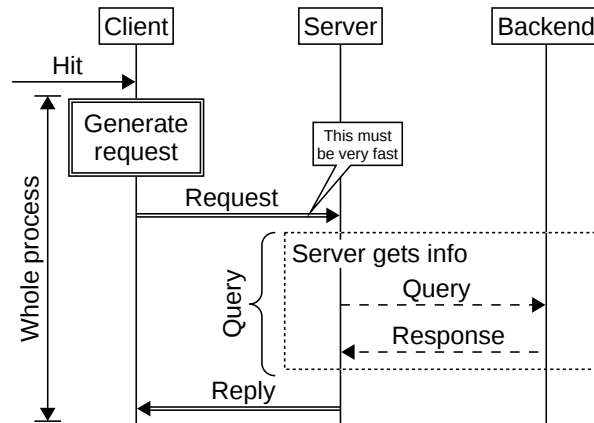


Another way to explain what happens is by annotating a series of events. This can be done by *verticals*, which are elements spanning vertically usually besides an entity line. Here we show two of them, the *range* and *brace* verticals, but there are more.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
mark top;
C==C: Generate\nrequest;
C=>S: Request;
note: This must\nbe very fast;
S..B: Server gets info
{
    S>>B: Query;
    S<<B: Response;
};
vertical brace at S- :Query;
C<=S: Reply;
vertical range top<-> at C-: Whole process;

```



### 3.7 Other Features

There are a few more features that are easy to use and can help in certain situations.

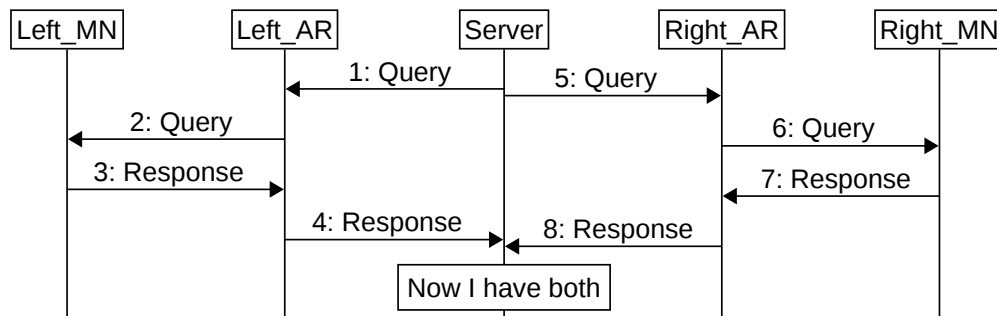
Another useful feature is the numbering of labels. This is useful if you want to insert your chart into some documentation and later refer to individual arrows by number. By specifying the `numbering=yes` chart option all labels will get an auto-incremented number. This includes boxes and dividers, as well. You can individually turn numbering on or off by specifying the `number` attribute. You can set it to `yes` or `no`, or to a specific integer number. In the latter case the arrow will take the specified number and subsequent arrows will be numbered from this value. On the example below, we can observe that in case of

parallel blocks the order of numbering corresponds to the order of the arrows in the source file.

```

numbering=yes;
Left_MN, Left_AR, Server, Right_AR, Right_MN;
{
  Server->Left_AR: Query;
  Left_AR->Left_MN: Query;
  Left_AR<-Left_MN: Response;
  Server<-Left_AR: Response;
} {
  Server->Right_AR: Query;
  Right_AR->Right_MN: Query;
  Right_AR<-Right_MN: Response;
  Server<-Right_AR: Response;
};
Server--Server: Now I have both [number=no];

```



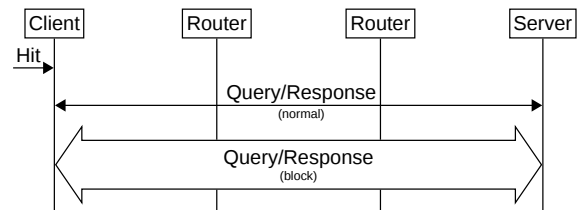
Sometimes a block of actions would be best summarized by a block arrow. This can be achieved by typing 'block' in front of any arrow declaration.

```

C: Client;
R1: Router;
R2: Router;
S: Server;

->C: Hit;
C->S: Query/Response\n\-(normal);
block C->S: Query/Response\n\-(block);

```



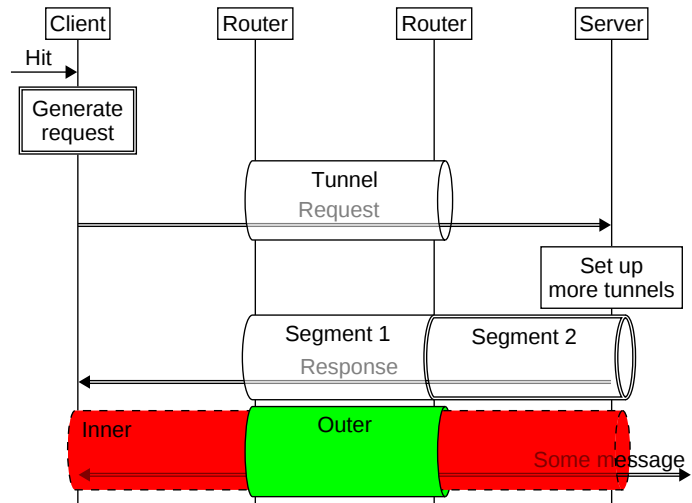
Similar, many cases you want to express a tunnel between two entities and messages travelling through it. To achieve this, just type 'pipe' in front of any box definition. You can define a series of connected or disconnected pipe segment each with its own visual style or even encapsulate pipes. More on this in [Section 5.5 \[Pipes\]](#), page 55.

```

C: Client;
R1: Router;
R2: Router;
S: Server;

->C: Hit;
C==C: Generate\nrequest;
pipe R1--R2: Tunnel {
    C=>S: Request;
};
S--S: Set up\nmore tunnels;
pipe R1--R2: Segment 1 []
    R2==S: Segment 2
{
    C<=S:Response;
};
pipe R1--R2: Outer
    [solid=255, color=green] {
        pipe C++S: \plInner
            [color=red] {
                C<=>: \prSome message;
            };
    };
};

```



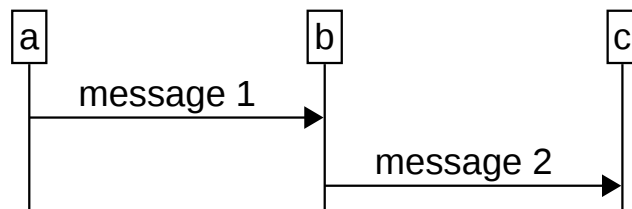
Adding a title to the chart is easy. Just type `title:` followed by the title text.

```

title: This is the title;
a,b,c;
a->b: message 1;
b->c: message 2;

```

## This is the title



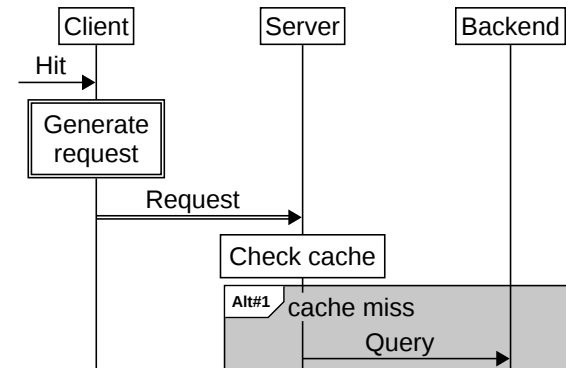
Another handy feature is multi-page support. This is useful when describing a single procedure in a document in multiple chunks. By inserting the `newpage;` command, the rest of the chart will be drawn to a separate file. You can specify as many pages, as you want. In order to display the entity headings again at the top of the new page, add the `auto_heading=yes` attribute. Breaking a page is possible even in the middle of a box, see the following example.

```

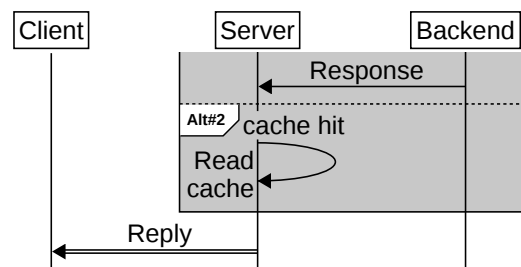
C: Client;
S: Server;
B: Backend;
->C: Hit;
box C==C: Generate\nrequest;
C=>S: Request;
box S--S: Check cache;
box S--B: cache miss
    [tag="Alt\#1", color=lgray]
{
    S->B: Query;
#break here
newpage [auto_heading=yes];
S<-B: Response;
}
...: cache hit [tag="Alt\#2"]
{
    S->S: Read\ncache;
};
C<=S: Reply;

```

Chunk one:



Chunk two:

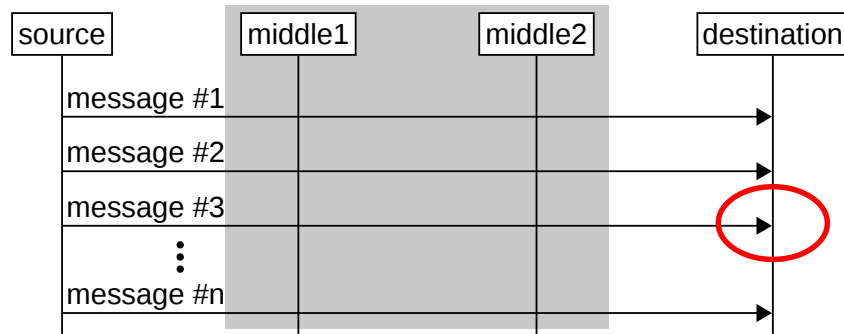


From version 3.3 you can draw arbitrary circles and rectangles onto the chart. They syntax is quite rich to allow free placement. You can even specify to draw below the entity lines or over other drawn elements. More detailed description can be found in [Section 5.15 \[Free Drawing\]](#), page 84, but here are a few examples.

```

mark top;
source ,middle1, middle2, destination;
vspace 10;
source->destination: \plmessage \#1;
source->destination: \plmessage \#2;
mark a_top;
source->destination: \plmessage \#3;
mark a_bottom [offset=10];
symbol ... center at source-middle1;
source->destination: \plmessage \#n;
mark bottom;
symbol rectangle top-bottom left at middle1 -40 right at middle2 +40
    [fill.color=lgray, line.type=none,
    draw_time=before_entity_lines];
symbol arc a_top-a_bottom center at destination
    [xsize=60, line.color=red, line.width=3,
    fill.color=none];

```



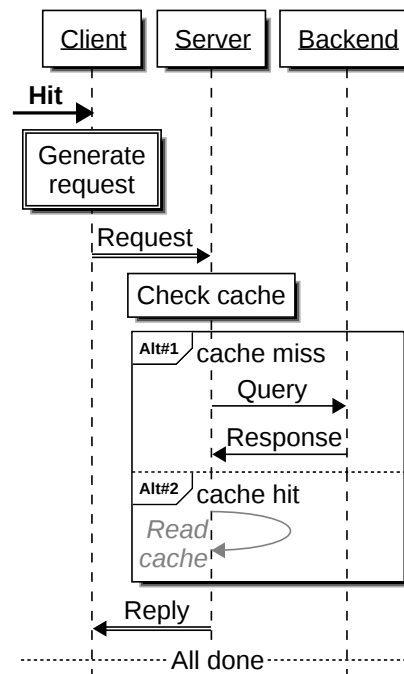
Finally, an easy way to make charts visually more appealing is through the use of *Chart Designs*. A chart design is a collection of colors and visual style for arrows, boxes, entities and dividers. The design can be specified either on the command line after double dashes, or at the beginning of the chart by the `msc=<design>` line.

Currently several designs are supported. ‘plain’ was used as demonstration so far. Below we give an example of the others.

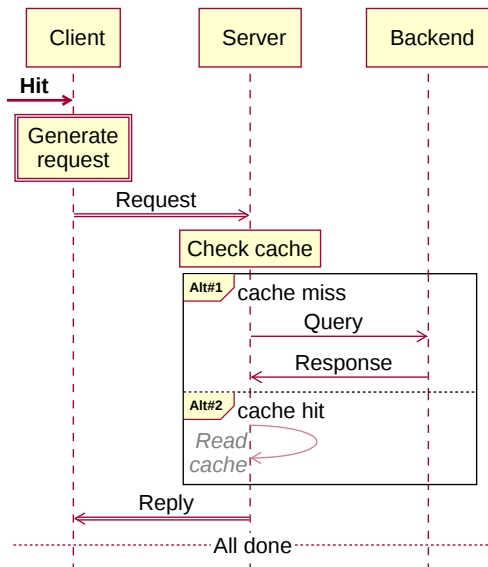
```

msc=qs;
C: Client;
S: Server;
B: Backend;
->C: Hit [strong];
box C==C: Generate\nrequest;
C=>S: Request;
box S--S: Check cache;
box S--B: cache miss [tag="Alt\#1"]
{
    S->B: Query;
    S<-B: Response;
}
..: cache hit [tag="Alt\#2"]
{
    S->S: Read\ncache [weak];
};
C<=S: Reply;
---: All done;
  
```

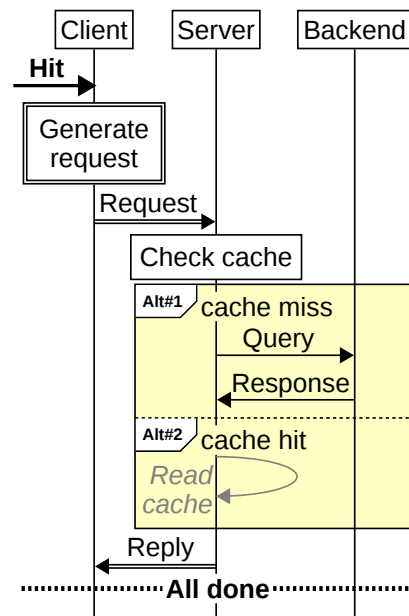
The ‘qs’ design:



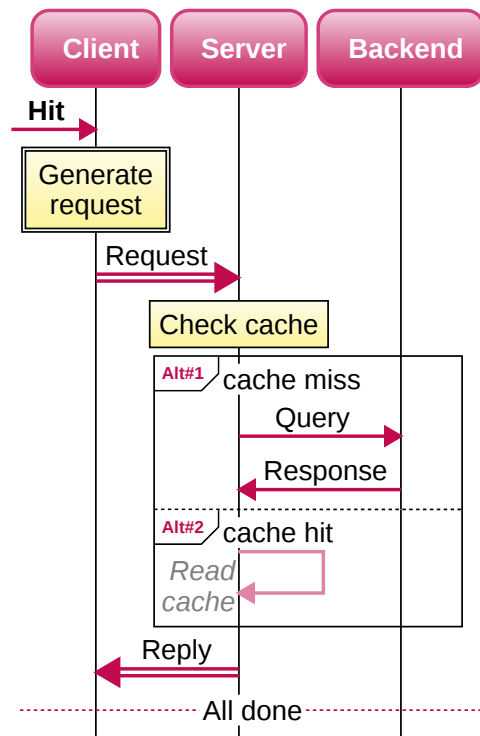
The 'rose' design:



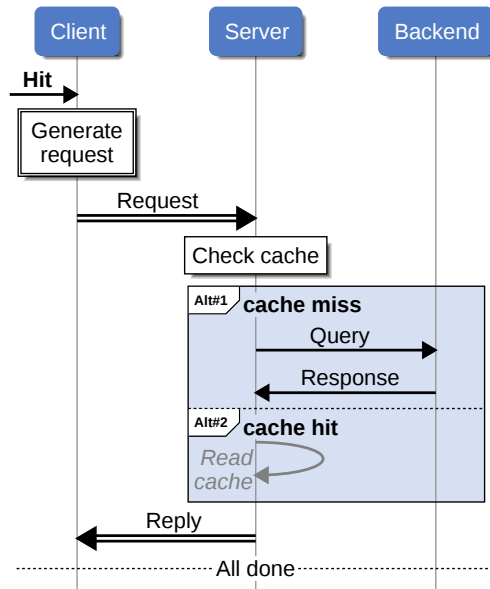
The 'mild\_yellow' design:



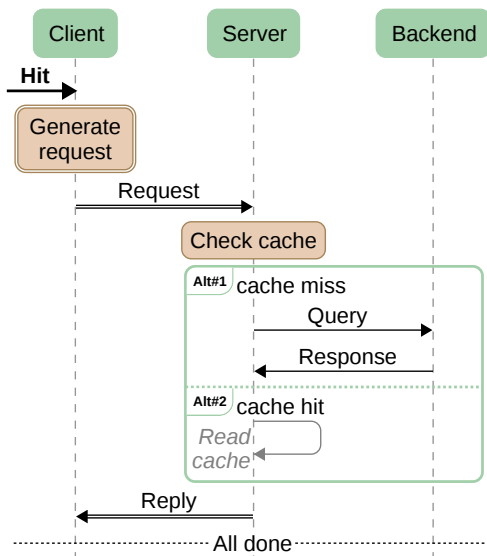
The 'omegapple' design:



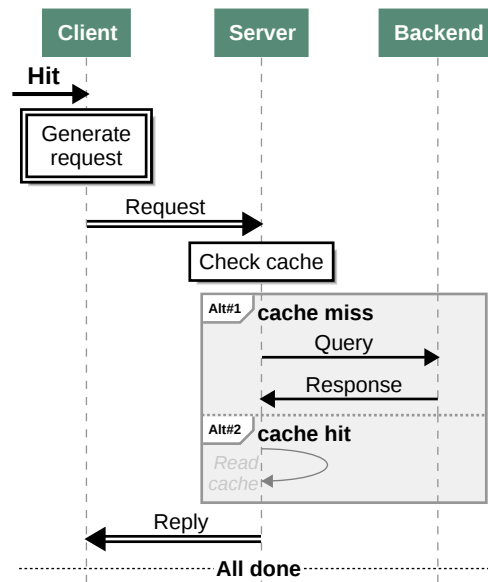
The 'modern\_blue' design:



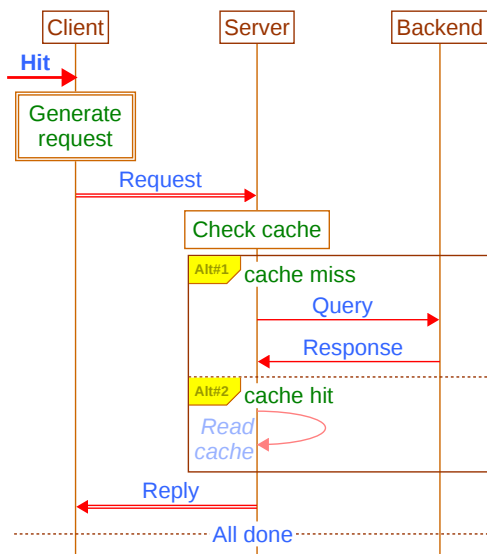
The 'round\_green' design:



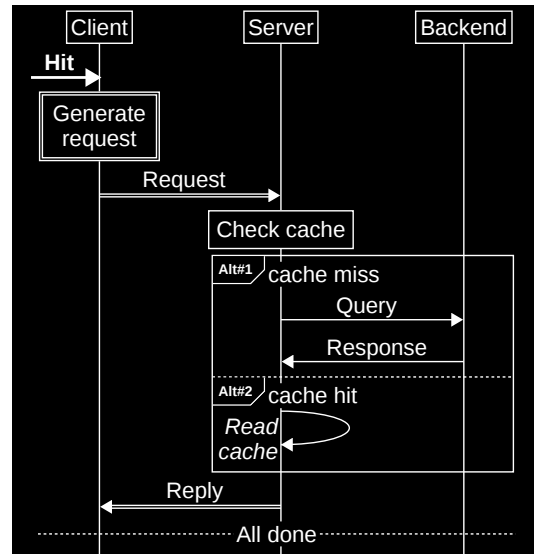
The 'green\_earth' design:



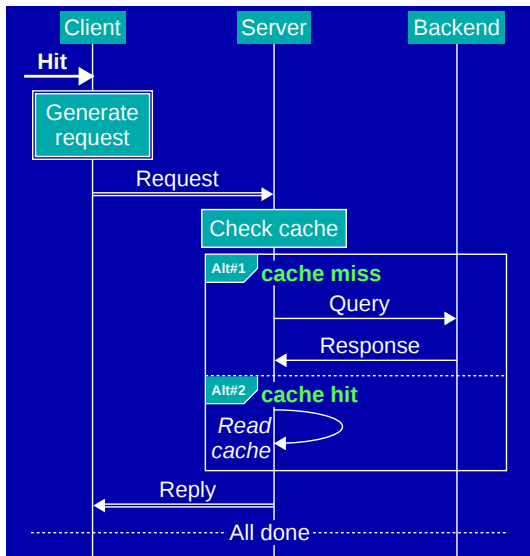
The 'colores' design:



The 'black\_on\_white' design:



And the the ‘norton\_commander’ tribute design.





## 4 Usage Reference

### 4.1 Design Library

Msc-generator comes with a bunch of Chart Designs and Entity Shapes (see [Section 5.19 \[Chart Designs\]](#), page 93 and [Section 5.2.6 \[Entity Shapes\]](#), page 46). The designs available on a Windows machine can be viewed in the design selector combo box on the ribbon.

On Windows the file describing the default designs and shapes are installed besides the executable. On Linux these files are incorporated into the executable so that msc-gen can be a standalone file.

Specifically on Windows at startup Msc-generator looks for a file called `designlib.signalling` in the directory where the executable is located. If not found, the file `original_designlib.signalling` is searched (of which a default one is placed there by the installer)<sup>1</sup>. If any found, the content is parsed as regular chart text before any chart.

You are free to create and modify your own design files<sup>2</sup>. On Linux place these files to the `~/.msc-genrc/` folder or to your folder of preference specified by the `MSC_GEN_RC` environment variable. On Windows place these to the roaming appdata folder, which is under `Users\<user name>\AppData\Roaming\Msc-generator` on Windows 7.

Design and Defining Shapes shall have the `.signalling` extension. Any such file in the above directories will be read by Msc-generator in no specified order.

### 4.2 External Editor

Although there is a built-in editor in Msc-generator, you can also use an external text editor of your choice. When you press `Ctrl+E` or click on `'Text Editors...|External Editor...'` button on the ribbon, an external text editor is started, where you can edit the chart description. If you perform save in the text editor, the chart drawing is updated, so you can follow your changes. Also, if there were errors or warnings, they are displayed in a the usual manner. If you select an error, Msc-generator will instruct the external editor to jump to the location of the error (if the external editor supports this functionality.)

During the time you are working with an external editor, the built-in text editor becomes read-only. You can exit the external editor any time to return to the built-in one. By pressing `Ctrl+E` or clicking on the `'Text Editors...|External Editor...'` ribbon button again, Msc-generator attempts to close the external editor (which will probably prompt you to save outstanding changes).

You can select the text editor to start in `Preferences|External Editor`. You can select between the Windows Notepad, Notepad++ or any editor of your preference. The author finds Notepad++ a very good editor, so I included specific support<sup>3</sup>.

---

<sup>1</sup> This mechanism was provided to enable the user to (re)define chart designs and is retained for backwards compatibility only. (The idea was that `original_designlib.signalling` is overwritten, when a newer version of Msc-generator is installed, whereas `designlib.signalling` is not. The current recommended practice is to add your own designs to new files in the roaming AppData folder, see below.

<sup>2</sup> However, please avoid any construct in design, which result in visual elements. Also try not to create files that result in warnings, errors.

<sup>3</sup> You can download Notepad++ from <http://notepad-plus.sourceforge.net/>

Note that Msc-generator does not support unicode or wide character systems for charts. Write your labels in ASCII only. There are no guarantees for non-ASCII characters to display correctly or at all.

### 4.3 Smart Indent

The internal editor supports automatic indentation for TAB, RETURN and BACKSPACE keys, (if the **Smart Indent** option is turned on). TAB and Shift+TAB works also with selections as in most programming editors.

In addition Msc-generator detects the beginning of multi-line labels and aligns all subsequent lines of the label to that. This also works when you select a block of text and press TAB or Ctrl+TAB. In the below example, Smart Indent would make the second lines of the labels to start exactly aligned with the first character of the first line above.

If the **TAB key indents** option is selected, pressing the TAB key anywhere in a line will indent the current line to its right position as opposed to inserting a tabulator. When multiple lines are selected all of them get automatically indented.

```
a->b: Label
    in two lines;
aaa->bbb: Another label
    in two lines;
aaaaaa->bbbbbb: A third label.
                Two lines, too.;
```

### 4.4 Color Syntax Highlighting

The internal editor also supports Color Syntax Highlighting. On the preferences pane you can select one of four color schemes. In most schemes entities that are used the first time are underlined; this helps to detect mistyped entity names. The examples in this document were colored using the ‘**Standard**’ color scheme.

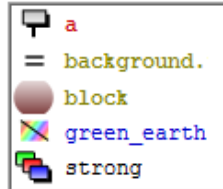
In the preferences it is also possible to select to underline parse error locations. In this case you get instant feedback on syntax problems. Finally, it is also possible to request error messages for any error that has been underlined in the internal editor. These explanatory messages appear in the same window as compilation errors, but they are prefixed with ‘**Hint**’. If the error they refer to is corrected, they disappear.

Note that during text edit Msc-generator does not perform a full parsing of the text to enhance performance. For example, correctness of attribute names and values is not verified, merely syntax.

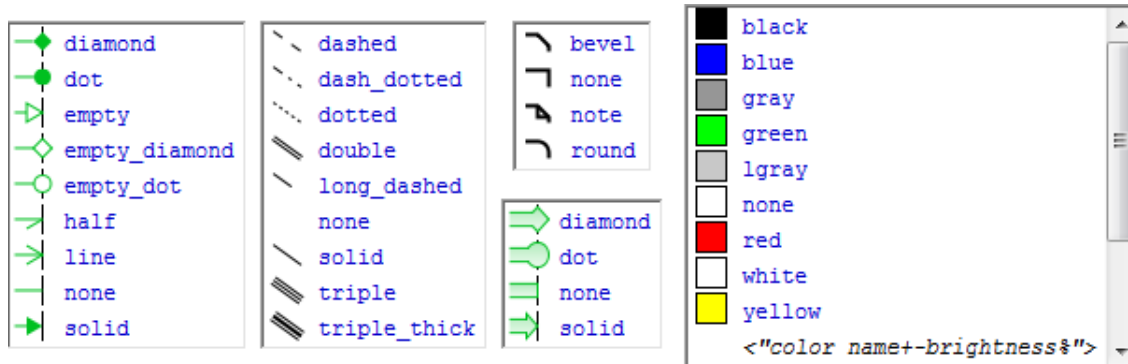
### 4.5 Typing Hints and Autocompletion

When turned on, the internal editor can also provide suggestions on how to complete the phrase you started typing. You can use the up/down arrow keys to select between the offered alternatives and press enter or TAB to select it. Alternatively, you can continue typing the keyword or hit any non-alphanumeric character, which will automatically select the highlighted hint and continue after.

The hints provided are associated with a small icon showing the type of the symbol. On the example below, an entity name ('a'), an option name, a keyword, a design name and a style name is shown.



Various attribute values offer a graphic representation to ease selection.<sup>4</sup> The items in italics do not represent actual text to be inserted into the chart, so you cannot select them. They are more like descriptions of what you can write there.

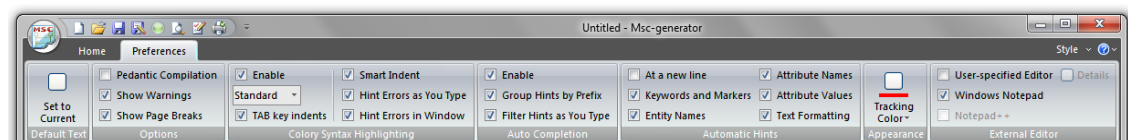


In the preferences you can control how much suggestions you will get and how they are displayed. You can turn hints entirely off.

If you press Ctrl+Space and there are meaningful suggestions, the hint box pops up even if automatic hints are turned off. If there is only one possible way to finish what you have started typing, that ending is automatically inserted (word auto-completion).

## 4.6 Options

Selecting the **Preferences** category on the ribbon allows you to set a few options of Msc-generator.



In the first category you can specify what is the chart that pops up when a new chart is started. Just press the button and the current text will become the default. You can place your frequently used constructs here to be readily available when you start a new chart; or just delete everything here to start real empty.

<sup>4</sup> Shapes defined in the current file do not appear with a thumbnail representation. Only shapes defined in design libraries do.

Under ‘Options’ you can set a few compilation options. When pedantic is set Msc-generator generates a warning if an entity is not declared explicitly before use. Turning the second option on will suppress the generation of warning messages altogether (including the ones generated due to the pedantic option). ‘Show Page breaks’ governs if a dashed line is drawn to show where page breaks are when watching all of the pages. See [Section 5.14 \[Multiple Pages\]](#), page 83 for more information.

In the ‘Color Syntax Highlighting’ panel you can select if you want to use color syntax highlighting in the built-in editor and if yes, which color scheme. There are four pre-defined schemes: Minimal, Standard, Colorful and Error oriented. The first three applies increasing amount of color, while the last is a minimalist scheme but with potential errors heavily highlighted<sup>5</sup>. At the moment you can not customize individual colors in the schemes.

If the ‘TAB key indents’ check box is checked pressing the TAB key will automatically indent the current line instead of inserting a TAB character.

In the right column of checkboxes you can control smart label indenting ([Section 4.3 \[Smart Indent\]](#), page 33) and whether you want to see errors as you type underlined and/or in the error window, [Section 4.4 \[Color Syntax Highlighting\]](#), page 33.

In the ‘Auto Completion’ panel you can govern, if the system provides language auto-completion (using Ctrl+Space) and Hints. Using the to checkbox you can turn Hints and auto-completion completely off (not even Ctrl+Space works). The two bottom checkboxes govern if the list of hints is grouped along dots (to reduce the length of the list) and if hints that are not matching what you have typed so far shall be removed or not. If grouping is on, attributes starting with the same text, such as `line.color` and `line.width` appear as a combined entry as `line.*`. Pressing the dot ‘.’ key will automatically auto-complete the common part. If filtering is turned on, only those hints are displayed which begin the same as the word under the cursor. If you continue typing, the list is narrowed by every character. If filtering is off all values valid at the location of the cursor are shown. Msc-generator can provide you hints even without pressing the Ctrl-Space. In the subsequent panel you can govern in what language contexts do you want to receive such automatic hints. In general it is best to experiment with these settings and see what you like.

On the last panel you can specify which external text editor to use. You can select any editor using the first option. In this case you have to give a command-line to start the editor and one to invoke to jump to a certain line by pressing the button to the right. The latter can be omitted if the editor does not provide a command line option to jump to a certain location in an existing editor window. Use ‘%n’ for the filename and ‘%l’ for the line number; these will be replaced to the actual filename and linenummer at invocation.

## 4.7 Working with Multi-page Charts

Msc-generator supports multi-page charts. These may be useful when you want to print a long chart. You can manually start a new page by typing the `newpage;` command or let Msc-generator automatically paginate for you. See more in [Section 5.14 \[Multiple Pages\]](#), page 83.

You can select on the ribbon which page to view. This setting is also saved with embedded charts, and of course only the selected page is shown in the container document. You

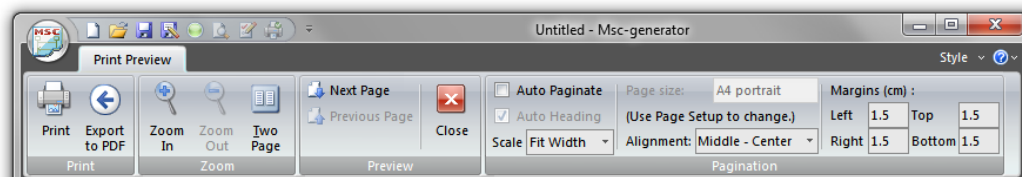
---

<sup>5</sup> We note here that all four schemes underline entities at their first use. This is to help you avoid a mis-typed entity name.

can also select to view all pages. When viewing all pages, Msc-generator marks page breaks with a dashed or dotted line for manual and automatic page breaks, respectively, and also prints page numbers to the left. This behaviour can be turned off in the preferences. (See [Section 4.6 \[Options\]](#), [page 34](#).)

The last pane on the Home category of the ribbon governs automatic pagination. The first checkbox turns it on. The paper size can be selected in ‘Print|Print Setup...’, whereas margins, page alignment and scaling can be selected in Print Preview. Ticking the the second checkbox will result in a heading to be displayed for the active entities at the top of every page.

You can preview a multi-page chart before printing using the Print Preview option from the main menu. It behaves similar to Print Preview in other programs.



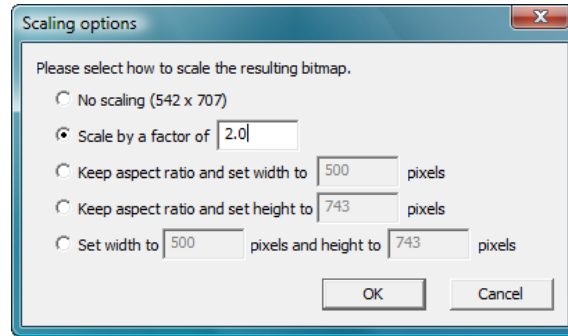
Here you can print or export to PDF<sup>6</sup>. You can also set Automatic Pagination and Auto Headings from here, but you can also set, how the chart pages are sized; how they are aligned within the physical pages and also what margins to apply. The former option can take *Fit width* and *Fit page*<sup>7</sup>. In the former case the scaling factor is selected to fit the width of the chart to the page, this is perhaps the most useful selection to print long charts combined with automatic pagination. In the latter case, the scaling is set uniformly for all pages to fit the longest one. With automatic pagination this is in effect equivalent to *Fit width*.

## 4.8 Scaling Options

If the chart is exported to a bitmap image (PNG or BMP), after selecting the filename an additional dialog box appears where you can set scaling options. In all but the last option the original aspect ratio of the chart is kept. After the ‘No scaling’ option the native size of the chart is shown.

<sup>6</sup> When you export a multi-page chart from Print Preview, Msc-generator automatically exports to a multi-page PDF file. When you use the Export option from the main menu, Msc-generator asks what you want.

<sup>7</sup> They correspond to the `-s=width` and `-s=auto` command-line options, respectively.



## 4.9 Advanced OLE Considerations

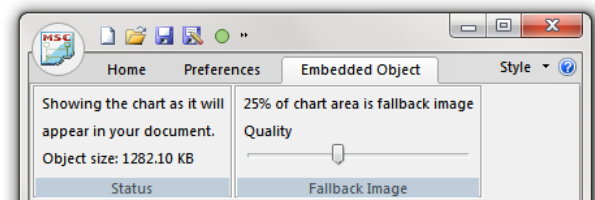
### 4.9.1 Graphics of Embedded Charts

The technology used to embed charts into other document, called OLE, has certain limitations on graphics<sup>8</sup>. To work around these, Msc-generator employs a few simplifications.

- Due to clipping limitations, certain arrowheads, like 'line' draw differently.
- Due to font limitations, the label of slanted arrows are drawn with limited resolution and looks somewhat different than non-slanted text.
- Due to missing gradient fill support, gradient fills and shadows are approximated. At large magnification this becomes visible.
- Due to the limited size of the coordinate space, placement of elements in very large charts appear imprecise.
- Due to lack of transparency support, translucent areas (such as pipes) are drawn on a bitmap, a *fallback image* and then inserted, see below.

If a chart contains a lot of fallback images, the size of the embedded object can become large, several megabytes for a chart. To control the size of the embedded chart and eventually that of your container document, for embedded charts a third category ("Embedded Object") appears on the ribbon, allowing you to adjust the quality of the fallback images.

Note that this issue is fixed in newer versions of Microsoft Office, which are able to compress the images in embedded objects.



When this category is selected, Msc-generator shows the chart as it will appear in the container document. Fallback image locations are briefly highlighted when switching to this category. The ribbon category shows how large the embedded object will be (if not compressed by the container applications) and what percentage of the chart is drawn on

<sup>8</sup> Only drawing operations permitted in old-style Windows Metafiles (developed in 16-bit Windows times) are permitted by design.

fallback images (if any). There is a slider allowing you to set the resolution of the fallback images. You can observe the resulting image size and visual quality immediately.

### 4.9.2 Linking

You can also choose to insert a Link to a copied chart instead of embedding it into the document. In this case updating the source chart will get reflected in the document, as well. You can also insert a link to only a page of a chart, by copying that page to the clipboard via the drop-down menu of the *Copy Entire Chart* button.

Note, however, that you cannot insert a link to a chart that is not saved on disk, but is yet ‘Untitled’. In addition, not all container applications implement the full range of linking features.

- LibreOffice and OpenOffice do not allow links to be inserted into documents. You can only embed charts in their documents.
- Microsoft PowerPoint allows links to be inserted into a slide, but does not allow other programs to link to a chart embedded in a slidepack. This includes the case when you want to insert a link into a slidepack that points to a chart embedded in the very same slidepack.
- Microsoft Excel implements full linking features, that is it allows you to insert links into worksheets, but also allows you to insert a link pointing to a chart embedded in a worksheet into other documents (or the same worksheet). You can even insert links that point to a single page of a chart embedded in a worksheet. (You can do this by opening the embedded object in Msc-generator and select ‘Copy Page #1’, and use Paste Special to insert a link.)
- Microsoft Word allows you to insert link to charts that are saved in files or are embedded in some other container (such as an Excel worksheet). It also allows others (including Word itself) to link to a full chart embedded in a Word document, but does not allow linking to a page of a chart embedded in a word document. If you invoke ‘Copy Entire Chart’ or ‘Copy Page #x’ from within Msc-generator for a chart embedded in a Word document, the link will not work. However, if you copy the chart to the clipboard from Word (and then you can copy all of it) then if you insert a link via Paste Special, you will get a valid link.

There is a suspected bug in Word 2003 that fails linking to a single page of a chart embedded in a Word document.<sup>910</sup>

## 4.10 Command-Line Referece

The syntax of the command-line version is the same on Linux and Windows<sup>11</sup>.

Note that the command line syntax below is a superset of the command-line options of the mscgen tool. This means that by renaming `msc-gen` to `mscgen` you can use Msc-

<sup>9</sup> Interestingly you can link to charts embedded in Excel documents. As for PowerPoint, I think it does not support linking to any object in a presentation.

<sup>10</sup> To link to a full chart embedded in Word, make sure you place chart to the clipboard using Word and not using Msc-generator.

<sup>11</sup> The only two exceptions are in how pathnames are written on the two systems and the fact that the Windows version will look for a designlib.signalling file for design definitions, while the Linux version will not.

generator's extra features and rich language in every tool that is integrated with mscgen. These tools include Doxyge, Sphinx and Msctexen.

```
Usage: msc-gen [-T type] [-o file] [infile] [-Wno] [-Pno] [--pedantic]
              [-p[=page size] [-m{lrud}=margin]] [-a[h]]
              [[-x=width] [-y=height] | [-s=scale]]
              [-F font] [-D design_file] [--nodesigns]
              [--chart_option=value] [--chart_design]

msc-gen -l
```

**-T type** Specifies the output file type, which maybe one of `png`, `eps`, `pdf`, `svg` or `emf` (on Windows only). Default is `png`. The token `ismap` is also accepted, but produces an empty map file. You can also specify `lmap`, which will not generate any graphics, but a text file listing each label in the chart with their coordinates. This is useful if you want to assign tooltips of clickable regions. See [Section 4.10.1 \[Label Maps\]](#), page 40 below for details.

**-o file** Write output to the named file. If omitted the input filename will be appended by the appropriate extension and used as output. If neither input nor output file is given, `mscgen_out.{png,eps,pdf,svg,emf}` will be used.

**infile** The file from which to read input. If omitted or specified as `-`, input will be read from the standard input.

**-p=[page size]**

Full-page output. (PDF only now.) In this case the chart is drawn on fixed-size pages (following pagination) with one pixel equalling to 1/72 inches. If a chart page is larger than a physical page it is simply cropped with a warning. Setting the scale with the `-s` option enables zooming. Page size can be set to ISO sizes from A0 to A6, and to US sizes, such as letter, legal, ledger and tabloid. Append a `'p'` or an `'l'` for portrait and landscape, respectively (except for `'tabloid'` and `'ledger'`, which are by definition portrait and landscape, resp.). E.g., use `'A4p'`, `'A2l'` or `'letter_l'`. Default is `'A4p'`.

**-m{lrud}='margin'**

Useful only for full-page output, specifies the margin. A separate option is needed to specify the left, right, upwards and downwards margins, denoted by the second letter of the option. Margins are to be specified in inches (number only) or in centimeters, if appended with `'cm'` (no spaces). The default margin is half inches everywhere.

**-va=<center|up|down>**

**-ha=<center|left|right>**

Set the vertical and horizontal alignment within a page for full-page output.

**-a[h]** Automatic pagination. Used only with full-page output. If specified, scale cannot be `'auto'`. Specifying `-ah` will insert a heading after automatically inserted page breaks.

**--pedantic**

When used all entities are expected to be declared before being used. Arrows with entities not declared before will trigger an error. (But the entity will be implicitly declared and the arrow included.)



- x=width** Specifies chart width (in pixels). Meant to be used for bitmaps (PNG and BMP), but works for all graphics output.
- y=height** Specifies chart height (in pixels). If only one of **-x** or **-y** is specified, the aspect ratio is kept. Meant to be used for bitmaps (PNG and BMP), but works for all graphics output.
- s=scale** Can be used to scale chart size up or down. Default is 1.0. Cannot be used together with any of **-x** or **-y**. Meant to be used for bitmaps (PNG and BMP) or full-page output (**-p**), but works for all graphics output. For full-page output, you can set *scale* to **'width'** which results in the chart width being set to the page width, or **'auto'**, which scales such that all pages fits. For full-page output, you can specify multiple **-s** options, which makes msc-gen to try them in the order specified until one is found for which no pages need to be cropped. If none is such, the last one will be used and a warning will be given.
- F font** Use specified font. This must be a font name available in the local system, and overrides the MSCGEN\_FONT environment variable if that is also set.
- D design\_file** Load file containing additional chart design definitions. You can have multiple of this option to load several design files, after the default ones. See [Section 5.19 \[Chart Designs\], page 93](#) for more info.
- nodesigns** If you specify this no design files will be loaded (no even the ones you specify with **-D**. This is useful to increase performance when you do not use them anyway.
- chart\_option=value** Any chart option (see [Section 5.13 \[Chart Options\], page 80](#)) can be specified on the command line. These are overridden by options in the file. Do not use any space before or after the equal sign.
- chart\_design** The design pattern of the chart can be specified on the command line (see [Section 5.19 \[Chart Designs\], page 93](#)). This will override any design specified in the file.
- Wno** No warnings displayed.
- Pno** No progress indicator displayed.
- l** Display program licence and exit.
- h** Display program help and exit.

### 4.10.1 Label Maps

When you specify **lmap** as output file format, Msc-generator creates a text file with one line for each text label in the chart (and no graphics output). The default extension will be **.map**. The lines in the output file contains the following information separated by space.

`<type> <page> <x1> <y1> <x2> <y2> <first line>`

The **type** character tells, what chart element contained this label. The following characters are possible

- A            Arrow, including block arrows (including boxes collapsed to arrows)
- E            Entity heading. Each appearance of the entities will result in one line.
- B            Box that has content (unless collapsed)
- b            Box that contains just a label (or collapsed)
- P            Pipes
- V            Verticals (all forms, including boxes, block arrows, ranges, braces and brackets)
- D            Divides, titles, subtitles, discontinuity lines and plain text (like `[label="aaa"] ;`).
- N            Floating notes
- C            Comments (on the side or at the end)

The second item **page** gives which page the label is on. One label is mentioned only once even if it spans multiple pages.

The following four numbers give the upper left and lower right corner of the bounding box of the label (and not the corresponding element). It is given in pixels for bitmap output and in logical coordinates matching the logical size of the output image for vector graphics output. The coordinates are relative to the top left corner of the page origin and are rounded to integers for ease of use. So if you run Msc-generator twice, once with a graphics output format and once with label map output (leaving all scaling and other switches the same), the coordinates of the label map shall match the graphics output perfectly.

Finally the line ends with the first line of the label (which may contain spaces), potentially with the number prepended (if any) in the number format used in the chart (e.g., roman numbers). Note that the coordinates specify the bounding box of the entire label, not just the first line given here.

## 5 Language Reference

### 5.1 Titles

The `title` and `subtitle` commands can be used to specify titles for the chart. You must supply a label, perhaps using the colon syntax.

```
title: This is a title;
subtitle: This is a subtitle;
```

The title and subtitle include text and a box around the text - the latter being omitted in the ‘plain’ design. You can turn it on by setting the ‘`line.*`’ and ‘`fill.*`’ attributes. The default attributes are taken from the default styles `title` and `subtitle`, changing these will affect all titles in the chart. Entity lines are not drawn behind the titles by default, this can be changed by setting the `vline.*` attribute.

### 5.2 Specifying Entities

Entities can be defined at any place in the chart, not only at the beginning.

Entity names can contain upper or lowercase characters, numbers, dots and underscores. They are case sensitive and must start with a letter or underscore and cannot end in a dot. If you want other characters, you have to put the entity name between quotation marks every time it is mentioned. This, however, makes little sense: you can set the label of the entity to influence how the entity is called on the drawn chart.

It is also possible to define entities without attributes (having all attributes set to default) by typing

```
entityname, ...;
```

It is also possible to change some of the attributes later in the chart, well after the definition of the entity. The syntax is the same as for definition — obviously the name identifies an already defined entity.

Note that typing several entity definition commands one after the other is the same as if all entity definitions were given on a single line. Thus

```
a;
b;
c;
```

is equivalent to

```
a, b, c;
```

Also, `heading` commands are combined with the definitions into a single visual line of entity headings.

#### 5.2.1 Entity Positioning

Entities are placed on the chart from left to right in the order of definition. This can be influenced by the `pos` and `relative` attributes.

Specifying `pos` will place the entity left or right from its default location. E.g., specifying `pos=-0.25` for entity B makes B to be 25% closer to its left neighbour. Thus `pos` shall be

specified in terms of the unit distance between entities. (Which is 130 points - a historic value kept for backwards compatibility.)

The next entity **C**, however, will always be from a unit distance from the entity defined just before it, so in order to specify a 25% larger space, on the right side of entity **B**, one needs to specify `pos=0.25` for **C**.

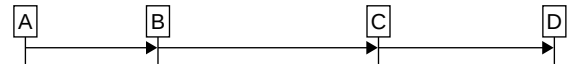
```
A, B, C, D;  
A->B-C-D;
```



```
A, B [pos=-0.25], C, D;  
A->B-C-D;
```

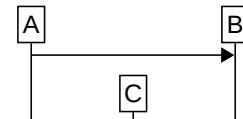


```
A, B [pos=-0.25], C [pos=+0.25], D;  
A->B-C-D;
```



The attribute `relative` can be used to specify the base of the `pos` attribute. Take the following input, for example. In this case **C** will be placed halfway between **A** and **B**.

```
A, B;  
A->B;  
C [pos=0.5, relative=A];
```



Note that specifying the `hscale=auto` chart option makes entity positioning automatic. This setting overrides `pos` values with the exception that it maintains the order of the entities that can be influenced by setting their `pos` attribute. See [Section 5.13 \[Chart Options\]](#), page 80. In most cases it is simpler to use `hscale=auto`, you need `pos` only to fine-tune a chart, if automatic layout is not doing a good job.

### 5.2.2 Group Entities

A group entity can contain other entities. Groups can be nested arbitrary deep. To specify a group entity, use curly braces after an entity definition (but before the colon or comma). Between the braces you can list entity definitions, style/color definitions or chart options<sup>1</sup>. The curly braces open a new scope, so any style or color definition or chart option takes its effect only within the group of entities between the curly braces. See [Section 5.17 \[Scoping\]](#), page 90 for more information.

Any entity you specify in the group must be a newly defined entity. It is not possible to place already defined entities into a group. Similar, an already defined entity cannot be

<sup>1</sup> Only some of the chart options can be used, the ones that merely change the context and do not draw. E.g. the `'background'` options cannot be used. Practically only the `'indicator'` chart option makes any sense.

made a group entity later by adding entities to it. Nor can a group be later extended with additional entities.

The position of a group entity is derived from its members so the ‘**pos**’ and ‘**relative**’ attributes cannot be used.

Group entities can be *collapsed*, by setting the ‘**collapsed**’ attribute to yes (or via the GUI on Windows). A collapsed group entity does not show its member entities, but is displayed as a non-grouped entity. Arrows and boxes in the chart are modified (or even removed) to reflect the collapse. If the ‘**indicator**’ attribute of the entity is set to yes, a small indicator is shown both inside the collapsed entity and for each arrow or box removed.

### 5.2.3 Entity Attributes

The following entity attributes can only be set at the definition of the entity.

<b>label</b>	This specifies the text to be displayed for the entity. It can contain multiple lines or any text formatting character. See <a href="#">Section 5.11 [Text Formatting]</a> , <a href="#">page 75</a> . If the label contains non alphanumeric characters, it must be quoted between double quotation marks. The default is the name of the entity.
<b>pos</b>	This attribute takes a floating point number as value and defaults to zero. It specifies the relative horizontal offset from the entity specified by the <b>relative</b> attribute or by the default position of the entity. The value of 1 corresponds to the default distance between entities. See a previous section for an example. Grouped entities cannot have this attribute.
<b>relative</b>	This attribute takes the name of another entity and specifies the horizontal position used as a base for the <b>pos</b> attribute. Grouped entities cannot have this attribute.
<b>shape</b>	This attribute takes the name of the shape you want for the entity headings. See <a href="#">Section 5.2.6 [Entity Shapes]</a> , <a href="#">page 46</a> .
<b>shape.size</b>	This attribute specifies the size of the shape to use for the entity headings. Only has effect if a valid shape is specified via the <b>shape</b> attribute. It takes one of <b>tiny</b> , <b>small</b> , <b>normal</b> , <b>big</b> or <b>huge</b> with <b>small</b> as default.
<b>collapsed</b>	This attribute can be used to collapse a group entity. Only group entities can have this attribute.
<b>indicator</b>	If set to yes (default) a small indicator will be displayed in a collapsed entity and also for any arcs that disappeared because of the collapse of this entity. On non-collapsed group entities it has no effect. Only grouped entities can have this attribute.

The following attributes can be changed at any location and have their effect downwards from that location.

<b>show</b>	This is a binary attribute, defaulting to yes. If set to no, the entity is not shown at all, including its vertical line. This is useful to omit certain entities from parts
-------------	--

of the chart where their vertical line would just crowd the image visually. See more on entity headings in [Section 5.2.5 \[Entity Headings\]](#), page 45.

**active** This is a binary attribute, defaulting to no. If set the entity line becomes a thin long rectangle indicating that the entity is active. You can set the fill of the rectangle via the `vfill.*` attributes. The commands `activate` and `deactivate` are shorthand for setting or clearing this attribute. Using the keywords is equivalent to setting the attributes, except that when the keywords are used just after an arrow, the activation/deactivation will take place immediately at the tip of the arrow, and not after.

**color** This sets the color of the entity text, the box around the text and the vertical line to the same color. It is a shorthand to specify `text.color`, `line.color` and `vline.color` to the same value.

`line.*`

`vline.*`

`fill.*`

`vfill.*`

`text.*`

**shadow.\*** See [Section 5.9 \[Common Attributes\]](#), page 63 for the description of these attributes.

### 5.2.4 Implicit Entity Definition

It is not required to explicitly define an entity before it is used. Just typing the arrow definition `a->b;` will automatically define entities ‘a’ and ‘b’ if not yet defined. This behaviour can be disabled by specifying the `--pedantic` command-line option or specifying `pedantic=yes` chart option. See [Section 5.13 \[Chart Options\]](#), page 80. Disabling implicit definition is useful to generate warnings for mis-typed entity names<sup>2</sup>.

Implicitly defined entities always appear at the very top of the chart. If you want an entity to appear only later, define it explicitly.

### 5.2.5 Entity Headings

By default, when an entity is defined, its heading is drawn at that location. If the entity name is preceded by the `hide` keyword or the `show=no` attribute is specified at the entity definition then the entity heading is not drawn at the location of the definition. It is drawn later, if/when the entity is turned on by using `show` followed by the entity name or by setting `show=yes`. Note that multiple entities can be listed after both `show` and `hide`. It is also possible to specify other attributes for entities after these keywords.

Mentioning an entity after its definition either preceded by `show` or with `show=yes` will cause an entity heading to be drawn into the chart even if the entity is already shown. This can be useful for long charts, see [Section 3.2 \[Defining Entities\]](#), page 12 for examples.

You can display all of the entity headings using the `heading;` command, as well. This command displays an entity heading for all (currently showing) entities. This may be useful after a `newpage;` command, see [Section 5.16 \[Commands\]](#), page 89. However, the

<sup>2</sup> To this end, color syntax highlighting underlines an entity name appearing the first time. This allows quickly realizing if the name of an entity is misspelled.

best practice is to use `'newpage [auto_heading=yes];'` instead, since it only shows the heading when the chart is viewed per-page (which is the same for page breaks inserted by automatic pagination).

### 5.2.6 Entity Shapes

The shape of an entity heading can be altered from the default box-like appearance to something custom using the `shape` attribute. Its value is a string, the name of the shape. The actual appearance of shapes is defined in separate files. Msc-generator comes with a few default shapes (their name all start with `def.`), but you can define your own shapes or add third-party Defining Shapes. See [Section 5.20 \[Defining Shapes\]](#), page 94 for more.

For some shapes, the label of the entity is written inside the shape, for some it is written below. This is decided by the author of the shape. If the label is written inside, it is scaled to fit. You can influence the size of the shape via the `shape.size` attribute, which takes the values `tiny`, `small`, `normal`, `big` or `huge` with `small` as default.

Note that the above two attributes can be set in the `entity` and the `entitygroup_collapsed` style, which will influence all entities in a chart at once.

## 5.3 Specifying Arrows

Arrows are probably the most important elements in a message sequence chart. They represent the actual messages. Arrows can be specified using the following syntax.

```
entityname arrowsymbol entityname [attr = value | style, ...];
```

*arrowsymbol* can be any of `'->'`, `'<-'` or `'<->'`, the latter for bidirectional arrows. `a->b` is equivalent to `b<-a`. This produces an arrow between the two entities specified using a solid line. Using `'>'/<>'`, `'>>'/<<>>'` or `'=>'/<=>'`, will result in dotted, dashed or double line arrows, respectively. These settings can be redefined using styles, see [Section 5.18 \[Defining Styles\]](#), page 91.

It is possible to omit one of the entity names, e.g., `a->;`. In this case the arrow will expand to/from the chart edge, as if going to/coming from an external entity.

It is possible to specify multi-segment arrows, such as `a->b->c` in which case the the arrow will expand from `'a'` to `'c'`, but an arrow head will be drawn at `'b'`, as well. This is used to indicate that `'b'` also processes the message indicated by the arrow. The arrow may contain any number of segments, and may also start and end without an entity, e.g., `->a->b->c->d->;`. As a syntax relaxation, additional line segments can be abbreviated with a dash (`'-'`), such as `a<=>b-c-d;`. Subsequent segments inherit the line type and direction of the first one. This enables quick changes to these attributes with minimal typing, as only the first arrow symbol needs to be changed. As a further possibility, different arrow symbols can also be used for different segments, such as `a->b=>c>>d-e;`, but all the arrow symbols must be of the same direction. It is therefore not possible to mix arrows of different directions, such as `a->b<-c;` or `a->b<->c;`. Note that specifying different arrow symbols affect only the line attributes of the segments, not the arrowhead, text or other attributes.

If the entities in a multi-segment arrow are not listed in the same (or exact reverse) order as in the chart, Msc-generator gives an error and ignores the arrow. This is to protect against unwanted output after rearranging entity order.

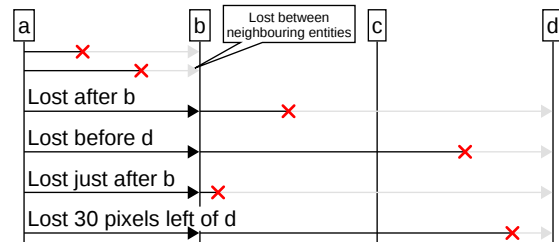
Arrows can also be defined starting and ending at the same entity, e.g., `a->a;`. In this case the arrow will start at the vertical line of the entity and curve back to the very same line. Such arrows cannot be multi-segmented.

Only non-grouped entities can be used in an arrow definition. If an entity used to define an arrow is not shown due to the collapse of its group entity, Msc-generator will automatically use the collapsed group entity when drawing the arrow, instead. If the arrow becomes degenerate (spanning between only a single collapsed group entity) or disappears entirely, an indicator will be shown instead, if the ‘`indicator`’ attribute of the collapsed group entity was set to yes (default).

### 5.3.1 Lost Messages

Sometimes one want to express that a message is lost. You can do this in Msc-generator in two ways. Either, you can add an asterisk between the two entities where the message is lost; or you can add a ‘lost at’ clause after the arrow specification before the label or attributes. This causes a small x to be drawn at the place specified and the dimming of the remainder of the arrow.

```
a, b, c, d;
a*->b;
a->*b;
note: Lost between
      neighbouring entities;
a->b*->d: \plLost after b;
a->b->*d: \plLost before d;
a->b->d lost at b++:
    \plLost just after b;
a->b->d lost at d -30:
    \plLost 30 pixels left of d;
```



The first three ones are the quick one. The message lost will be indicated around the entity after or before the asterisk. Specifically, it will be between this and its neighbouring visible entity. If that visible entity is also part of the arrow, the loss will be at one third the distance between them, else it will be halfway. Using the second form, you can specify exactly where the loss happened. It can be placed onto an entity, left or right from it, or between two entities. These are specified as ‘lost at <entity>’, ‘lost at <entity>-’, ‘lost at <entity>+’ or ‘lost at <entity1>-<entity2>’, respectively. You can add two plus or minus symbols to increase distance. You can also specify any offset in addition by adding a number after, such as in ‘lost at <entity> <number>’. The number will be interpreted in pixels and shifts the vertical left or right depending on its sign.

The appearance of the loss symbol (the x) can be influenced using the `x.line.width`, `x.line.color` and the `x.size` attributes. The latter takes the same values as arrowhead sizes: `tiny`, `small`, `normal`, `big` or `huge`, with `normal` as default.

The appearance of the lost portion of the message can also be influenced via the `lost.text.*`, `lost.line.*` and the `lost.arrow.*` attributes. Anything specified here will be added to the text, line and arrowhead format of the arrow. Currently only the color of



the line and the arrowhead is overlaid with `++white,128` (for plain designs), making them weaker, but you can change to dash lines, specify a narrower line or empty arrowheads.

### 5.3.2 Arrow Attributes

Arrows can have the following attributes.

<code>label</code>	This is the text associated with the arrow. See <a href="#">Section 5.9.2 [Labels]</a> , page 69 for more information on how to specify labels. In Msc-generator the first line of the label is written above the arrow, while subsequent lines are written under it. Future versions may make this behaviour more flexible.
<code>text.*</code>	All text formatting attributes described in <a href="#">Section 5.9 [Common Attributes]</a> , page 63 can be used to manipulate the appearance of the label <sup>3</sup> .
<code>number</code>	Can be set to <code>yes</code> , <code>no</code> or to a number, to turn numbering on or off, or to specify a number, respectively. See <a href="#">Section 5.9.3 [Numbering]</a> , page 70.
<code>refname</code>	Can be set to any string and is used to give a name to the arrow, which can be used to reference this arrow. Use the <code>\r(name)</code> escape in labels to insert the number of the referenced arrow. See <a href="#">Section 5.9.3 [Numbering]</a> , page 70.
<code>compress</code>	Can be set to <code>yes</code> or <code>no</code> to turn compressing of this arrow on or off. See <a href="#">Section 5.9.4 [Compression and Vertical Spacing]</a> , page 73.
<code>vspacing</code>	Can be set to a number interpreted in pixels or to the string <code>compress</code> . Governs how much vertical space is added before the arrow (can be negative). This attribute is another form (superset) of the <code>compress</code> attribute; <code>compress=yes</code> is equivalent to <code>vspacing=compress</code> , whereas <code>compress=no</code> is equivalent to <code>vspacing=0</code> .
<code>angle</code>	This takes a number in degrees and makes the arrow slanted. Arrows pointing to the same entity cannot have such an attribute. This attribute takes its default value from the <code>angle</code> chart option (or is zero in the absence of such an option, which corresponds to horizontal arrows).
<code>slant_depth</code>	This is similar in effect to the <code>angle</code> attribute, but instead of specifying the angle of the slant in degrees, you can use this attribute to specify how many pixels shall the end of the arrow be below the start of it. If you specify both the <code>angle</code> attribute and <code>slant_depth</code> the latter takes effect.
<code>color</code>	This specifies the color of the text, arrow and arrowheads. It is a shorthand to setting <code>text.color</code> , <code>line.color</code> and <code>arrow.color</code> to the same value.
<code>line.color</code> , <code>line.width</code>	Set the color and the width of the line, see <a href="#">Section 5.9 [Common Attributes]</a> , page 63.

---

<sup>3</sup> A special note on left and right text margins (to be specified via `\ml()` and `\mr()` escapes). Msc-generator always adds enough text margins to prevent the label to overlap with the arrowhead. Thus, if you specify less margin, it will have no effect.

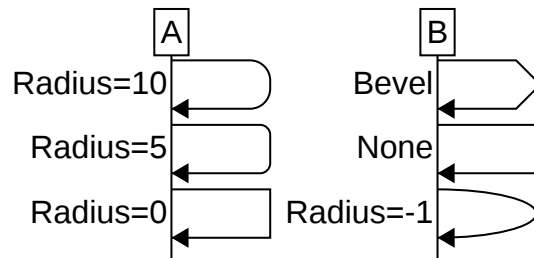
**line.corner**

This attribute specifies how the line shall be drawn at corners. It impacts boxes and entities drawn with this line, for arrows it is effective for arrows that start and end at the same entity. Its value can be `none`, `round`, `bevel` or `note`. See the example below. Setting `line.corner` without `line.radius` will result in the default radius of 10.

**line.radius**

For arrows starting and ending at the same entity, this specifies the roundness of the arrow corners. 0 is fully sharp (equivalent to `line.corner=none`, positive values are meant in pixels, a negative value will result in a single arc (for any corner setting). If only `line.radius` is set and not `line.corner` the result will be a round corner.

```
hscale=auto;
{
  A->A: Radius=10 [line.radius=10];
  A->A: Radius=5 [line.radius=5];
  A->A: Radius=0 [line.radius=0];
} {
  B->B: Bevel [line.corner=bevel];
  B->B: None [line.corner=none];
  B->B: Radius=-1 [line.radius=-1];
};
```

**arrow.size**

The size of the arrowheads. It can be `tiny`, `small`, `normal`, `big` or `huge`, with `small` as default.

**arrow.color**

The color of the arrowheads.

**arrow.type**

Specify the arrowhead type. The values can be `half`, `line`, `empty`, `solid`, which draw a single line, a two-line arrow, an empty triangle and a filled triangle, respectively. The above 4 types also exist in `double` and `triple` variants, which draw two or three of them. `sharp` and `empty_sharp` draws a bit more pointier arrowhead, filled or empty, respectively. `diamond` and `empty_diamond` draws a filled or empty diamond, while `dot` and `empty_dot` draws a filled or empty circle. Specifying `none` will result in no arrowhead at all. This attribute sets both the `endtype` and `midtype`, see below.

**arrow.endtype**

Sets the arrow type for arrow endings only. This refers to the end of the arrow, where it points to. In case of bidirectional arrows, both ends are drawn with this type. It defaults to a filled triangle.

**arrow.midtype**

This attribute sets the arrowhead type used for intermediate entities of a multi-segment arrow. It defaults to a filled triangle.

**arrow.starttype**

This attribute sets the arrowhead type used at the starting point of an arrow. It defaults to no arrowhead.

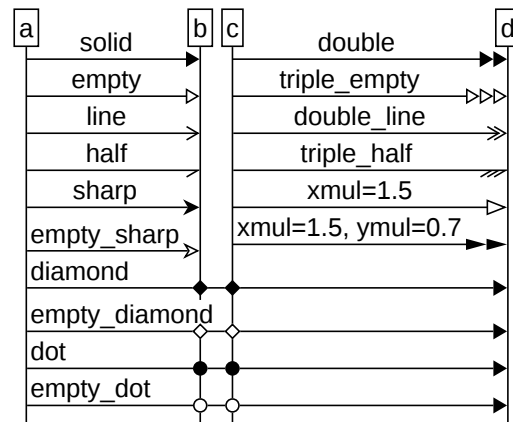
**arrow.xmul****arrow.ymul**

These attributes change the width or the height of the arrowhead. The default value is ‘1’. They are multipliers, thus the value of ‘1.1’ results in a 10% increase, for example.

```

hscale=auto, compress=yes;
{
  a->b: solid [arrow.type=solid];
  a->b: empty[arrow.type=empty];
  a->b: line[arrow.type=line];
  a->b: half[arrow.type=half];
  a->b: sharp[arrow.type=sharp];
  a->b: empty_sharp[arrow.type=empty_sharp];
} {
  c->d: double[arrow.type=double];
  c->d: triple_empty[arrow.type=triple_empty];
  c->d: double_line[arrow.type=double_line];
  c->d: triple_half[arrow.type=triple_half];
  c->d: xmul=1.5 [arrow.type=empty, arrow.xmul=1.5];
  c->d: xmul=1.5, ymul=0.7 [arrow.type=double,
    arrow.xmul=1.5, arrow.ymul=0.7];
};
a->b-c-d: \pldiamond [arrow.midtype=diamond];
a->b-c-d: \plempty_diamond [arrow.midtype=empty_diamond];
a->b-c-d: \pldot [arrow.midtype=dot];
a->b-c-d: \plempty_dot [arrow.midtype=empty_dot];

```

**lost.text.\***

The values specified here will be added to the values of **text.\*** when drawing the label of the lost part of the message.

**lost.line.\***

The values specified here will be added to the values of **line.\*** when drawing the line of the lost part of the message.

**lost.arrow.\***

The values specified here will be added to the values of **arrow.\*** when drawing the arrowheads in the lost part of the message.

**x.size** The size of the loss symbol for lost messages. It can be **tiny**, **small**, **normal**, **big** or **huge**, with **normal** as default.

**x.line.width****x.line.color**

The linewidth and color of the loss symbol for lost messages.

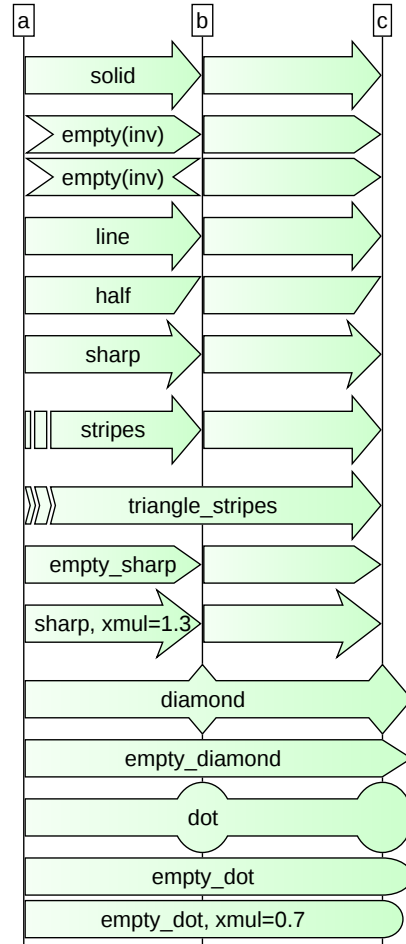
Note that default values can be changed using styles, see [Section 5.18 \[Defining Styles\]](#), page 91.

### 5.3.3 Block Arrows

When typing `block` in front of any arrow definition, it will become a *block arrow*. The label of a block arrow is displayed inside it. In addition to the attributes above, block arrows also have fill and shadow attributes, similar to entities.

All arrowheads explained above for regular arrows are supported, except the `double` and `triple` ones. In general, types with `empty` in them, draws a variant of the arrowhead which is not taller than the body of the block arrow. The ones with `line` draw the same as the ones without. Three additional types `empty_inv`, `'stripes'` and `'triangle_stripes'` types are supported, as well. See the example below for a detailed list of all types supported for block arrows.

```
defstyle blockarrow [fill.color="green+80",
                    fill.gradient=right];
block a->b-c: solid [arrow.type=solid];
block a->b-c: empty(inv) [arrow.type=empty,
                        arrow.starttype=empty_inv];
block a->b-c: empty(inv) [arrow.endtype=empty,
                        arrow.starttype=empty_inv,
                        arrow.midtype=empty_inv];
block a->b-c: line [arrow.type=line];
block a->b-c: half [arrow.type=half];
block a->b-c: sharp [arrow.type=sharp];
block a->b-c: stripes
    [arrow.starttype=stripes];
block a->c: triangle_stripes
    [arrow.starttype=triangle_stripes];
block a->b-c: empty_sharp
    [arrow.type=empty_sharp];
block a->b-c: sharp, xmul=1.3
    [arrow.type=sharp,
     arrow.xmul=1.3];
block a->b-c: diamond [arrow.type=diamond];
block a->b-c: empty_diamond
    [arrow.type=empty_diamond];
block a->b-c: dot [arrow.type=dot];
block a->b-c: empty_dot
    [arrow.type=empty_dot];
block a->b-c: empty_dot, xmul=0.7
    [arrow.type=empty_dot,
     arrow.xmul=0.7];
```



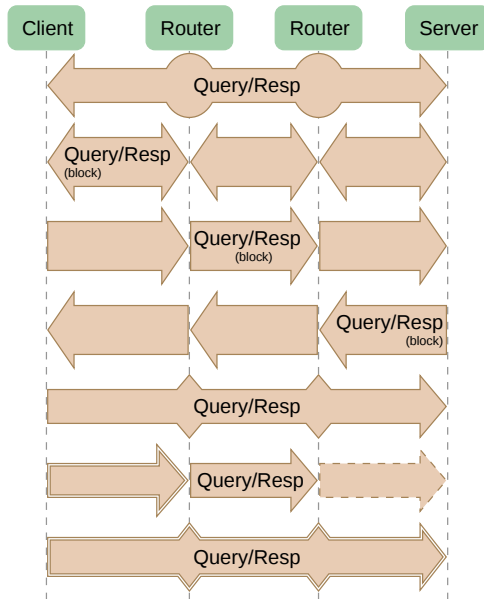
If the arrow has multiple segments and the type of the inner arrowheads is either of `half`, `line`, `empty`, `solid` or `sharp` the block arrow is split into multiple smaller arrows. In this case the arrow label is placed into the leftmost, rightmost or middle one of the smaller arrows, depending on the value of the `text.ident` attribute.

It is also possible to use different arrow symbols leading to different line types, but only if the middle arrow type is such that the arrow is split into multiple contours. If not, the whole arrow is drawn with the line type of the first segment.

```

msc = round_green;
hscale=auto;
C [label="Client"], R1 [label="Router"],
R2 [label="Router"], S [label="Server"];
block C<->R1-R2-S: Query/Resp [arrow.midtype=dot];
block C<->R1-R2-S: Query/Resp\n\-(block) [text.ident=left];
block C ->R1-R2-S: Query/Resp\n\-(block) [text.ident=center];
block C<- R1-R2-S: Query/Resp\n\-(block) [text.ident=right];
block C ->R1-R2-S: Query/Resp [arrow.midtype=diamond];
block C=>R1->R2>>S: Query/Resp;
block C=>R1->R2>>S: Query/Resp [arrow.midtype=diamond];

```



Block arrows can also be slanted using the `angle` attribute.

## 5.4 Boxes

Boxes enable 1) to group a set of arrows by drawing a rectangle around them; 2) to express alternatives to the flow of the process; and 3) to add comments to the flow of the process. The first two use is by adding a set of arrows to the box, while in the third case no such arrows are added, making the box *empty*.

The syntax definition for boxes is as follows.

```

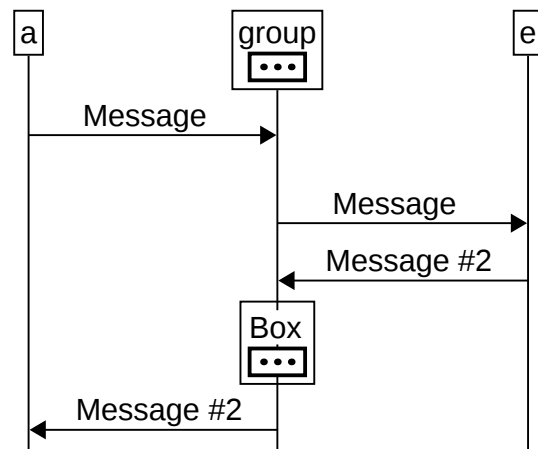
[box] entityname boxsymbol entityname [attr = value | style, ...]
{ element; ... };

```

The box symbol is optional at the beginning of the line. The *boxsymbol* can be ‘..’, ‘++’, ‘--’ or ‘==’ for dotted, dashed, solid and double line boxes, respectively.

As with arrows the two entity names specify the horizontal span. These can be omitted (even both of them), making the box auto-adjusting to cover all the elements within. If there are no elements within and you omit one or both entities the default is to span to the edge of the chart. Specifying the entity names therefore, is useful if you want a deliberately larger or smaller box, or if you specify an *empty* box. Contrary to arrows, you can use group entities when specifying a box. The box will then cover all member entities in that group. Specifying the leftmost or rightmost member entity instead of the group entity makes a difference only if the group entity is collapsed. In the former case the box may disappear, in the latter case it will not. See the example below.

```
a, group [collapsed=yes] {
  b, c, d;
}, e;
a->b: Message;
box b--d: Box {
  b->c->d: Message;
};
d->e: Message;
d<-e: Message \#2;
box group--group: Box {
  b<-c-d: Message \#2;
};
a<-b: Message \#2;
```



Boxes take attributes, controlling colors, numbering, text indentation quite similar to arrows. Specifically boxes also have a `label` attribute that can also be shorthand, as for arrows. For example: `... Auto-adjusting empty box;` is a valid definition. The valid box attributes are `label`, `number`, `refname`, `compress`, `vspacing`, `color`, `text.*`, `line.*`, `shadow.*` and `fill.*`. The latter specifies the background color of the box, while `line.*` specifies the attributes of the line around. Note that `color` for boxes is equivalent to `fill.color`. `text.ident` defaults to centering for empty boxes and to left indentation for ones having content.

After the (optional) attributes list, the content of the box can be specified between braces ‘{’ and ‘}’. Anything can be placed into an box, including arrows, dividers, other boxes or commands. If you omit the braces and specify no content, then you get an empty box, which is useful to make notes, comments or summarize larger processes into one visual element by omitting the details.

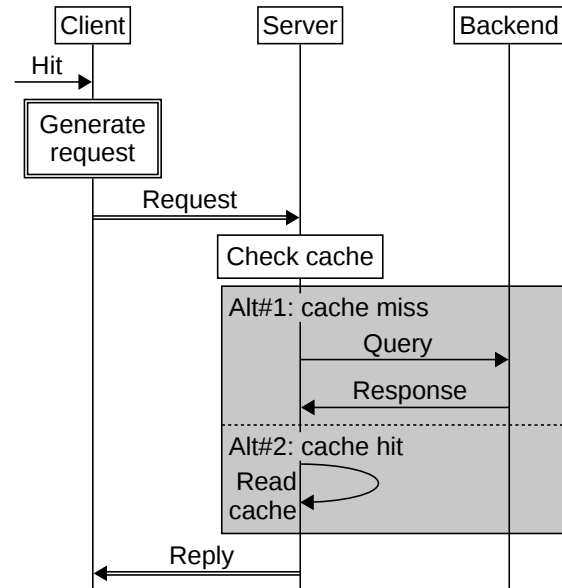
### 5.4.1 Box Series

If a box definition is not followed by a semicolon, but another box definition, then the second box will be drawn directly below the first one. This is useful to express alternatives, see the below example.

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box C==C: Generate\nrequest;
C=>S: Request;
box S--S: Check cache;
box S--B: Alt\#1: cache miss
    [color=lgray]
{
    S->B: Query;
    S<-B: Response;
}
..: Alt\#2: cache hit
{
    S->S: Read\nocache;
};
C<=S: Reply;

```



The subsequent boxes will inherit the fill, line and text attributes of the first one, but you can override them. The line type of subsequent boxes ('--' in the example) will determine the style separating the boxes — the border will be as specified in the first one. The horizontal size of the combined box is determined by the first definition, entity names in subsequent boxes are ignored.

Boxes can be collapsed, similar to group entities. The 'indicator' attribute governs if collapsed boxes show a small indicator to indicate that there is hidden content inside.

### 5.4.2 Box Tags

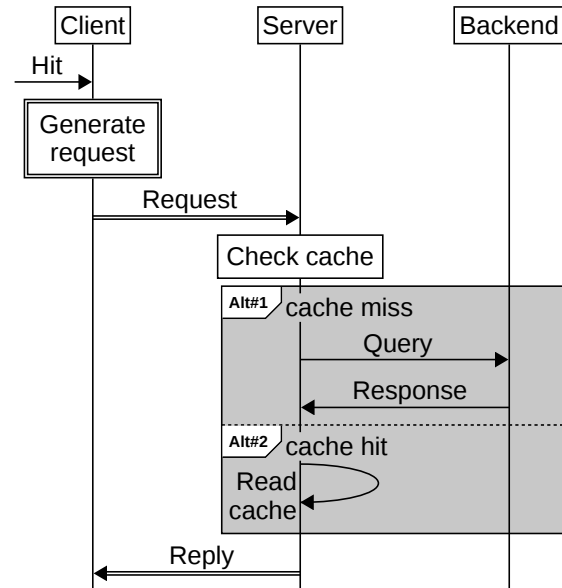
Boxes can also have *tags*, which are small enclosed labels at the top-left corner of the box. Tags are useful to label the content of the box, such as alternatives, loops or optional sections, while keeping the ability to add a regular box label, as well.

To specify a tag, use the **tag** attribute, and specify the label of the tag. If the label contains non-alphanumeric characters (or spaces), you need to put them in between quotation marks. You can specify the appearance of the tag via the **tag.line.\***, **tag.fill.\*** and the **tag.text.\*** attributes. Especially the **tag.line.corner** attribute can be used to influence the bottom right corner of the tag (but not the other corners).

```

C: Client;
S: Server;
B: Backend;
->C: Hit;
box C==C: Generate\nrequest;
C=>S: Request;
box S--S: Check cache;
box S--B: cache miss
  [tag="Alt\#1", color=lgray]
{
  S->B: Query;
  S<-B: Response;
}
..cache hit [tag="Alt\#2"]
{
  S->S: Read\nocache;
};
C<=S: Reply;

```



## 5.5 Pipes

By typing `pipe` in front of a box definition, it is turned into a pipe. Pipes can represent tunnels, encapsulation or other associations (e.g., encryption) in networking technologies. Using them one can visually express as messages travel within the tunnels or along other associations.

Pipes take all the attributes of boxes, plus two extra ones, called `solid` and `side`. `solid` controls the transparency of the pipe. It can be set between 0 and 1 (or alternatively 0 and 255, similar to color RGB values). The value of 0 results in a totally transparent pipe: all its contents is drawn in front of it. The value of 1 results in a totally opaque pipe, all its content is "inside" the pipe, not visible. Values in between result in a semi-transparent pipe. `side` can be set to `left` or `right` and governs which side the pipe can be looked into from<sup>4</sup>.

For pipes the `line.radius` attribute governs, how wide the oval is at the two ends of the pipe. The default value is 5. Note that `line.corner` has no effect for pipes. Both `line.radius` and `side` can only be set on the first of the pipe segments, see below.

<sup>4</sup> Beware that if you embed the chart in a Windows document, then using a lot of transparency can increase the size of the embedded object excessively.



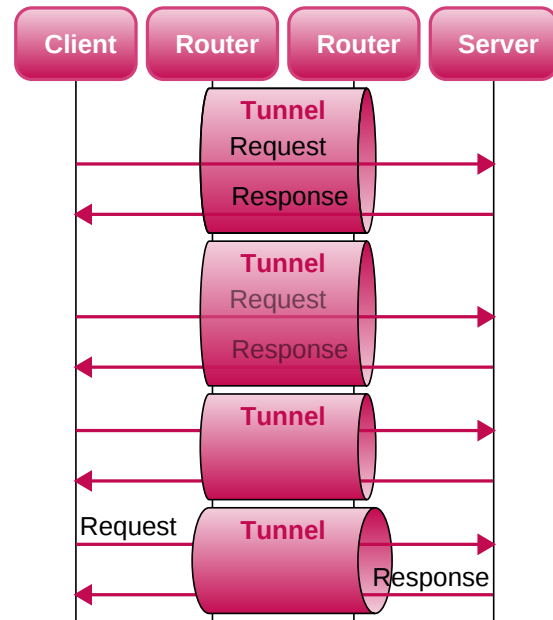
```

msc=omegapple;
C: Client;
R1: Router;
R2: Router;
S: Server;

defstyle pipe [fill.color=rose];
defstyle pipe [fill.gradient=down];

pipe R1--R2: Tunnel [solid=0] {
  C->S: Request;
  C<-S: Response;
};
pipe R1--R2: Tunnel [solid=0.5] {
  C->S: Request;
  C<-S: Response;
};
pipe R1--R2: Tunnel [solid=1] {
  C->S: Request;
  C<-S: Response;
};
pipe R1--R2: Tunnel
  [solid=1, line.radius=10] {
  C->S: \plRequest;
  C<-S: \prResponse;
};

```



On the example above one can observe, that the last two pipes are smaller than the first two, even though they have exactly the same two arrows within. This is because in case of the first two arrows the label of the pipe itself is visible at together with the two arrows within. In contrast, the last two pipes are fully opaque so the pipe label can be drawn over its content.

Note the two `defstyle` commands before the pipes, as well. They are re-defining the default fill for pipes. You can read more about this in [Section 5.18 \[Defining Styles\]](#), page 91.

Similar to boxes multiple subsequent pipe definitions can be placed after each other without a semicolon. In case of boxes this results in a series of vertical connected boxes. In

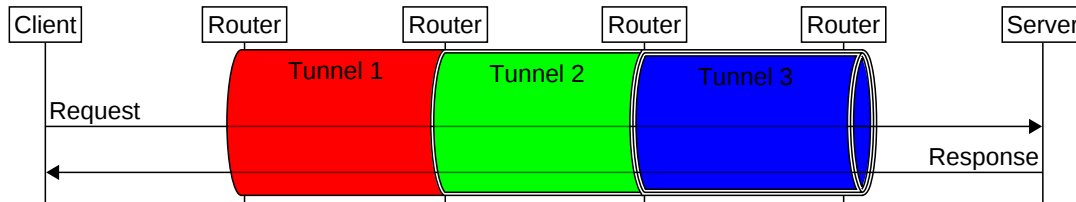
case of pipes this results in a series of horizontal pipe segments besides each other. However, contrary to boxes only one set of content can be specified.

```

C: Client; R1: Router;
R2: Router; R3: Router;
R4: Router; S: Server;

pipe R1--R2: Tunnel 1 [color=red]
     R2==R3: Tunnel 2 [color=green]
     R3==R4: Tunnel 3 [color=blue, line.type=triple]
{
  C->S: \plRequest;
  C<-S: \prResponse;
};

```



## 5.6 Verticals

A *vertical* is one of the following.

- a box or an arrow with a general direction of up and down as opposed to regular arrows or boxes, which go from left to right;
- a *brace* or *bracket*, to show grouping of things;
- a *range* to mark a time period and to comment on it; or
- a *pointer* which points from one point at an entity's timeline to another representing some cause/effect or a timer. This also has a version, where the pointer "lost" (e.g., to indicate a timer expiring).

```
vertical [shape | [from] symbol [to]] [at position] [attributes...];
```

The *shape* keyword can be one of **box**, **brace**, **bracket**, **range** or **pointer** or can be omitted. These variations result in different shapes, see the examples further down.

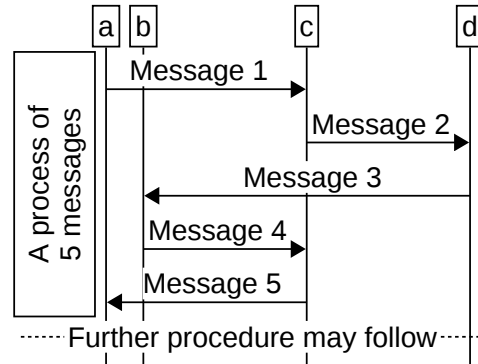
The *from* and *to* represent *markers* and specify the vertical position of the vertical. Markers can be placed with the **mark** command to mark a vertical position. The third line of the example below places a marker named **top** just below the entity headings. Then this marker is referenced by the vertical as the upper edge of it. The other marker is omitted in the example, it is then assumed to be the current vertical position. The **mark** command can have an **offset** attribute, which takes a number and shifts the position down by that many pixels (up for negative numbers).

There are two built-in markers, that are available without the **mark** command: **chart.top** and **chart.bottom** referring to the top and bottom of the entire chart, respectively.

```

hscale=auto;
a, b, c, d;
mark top;
a->c: Message 1;
c->d: Message 2;
d->b: Message 3;
b->c: Message 4;
c->a: Message 5;
vertical top-- at a-:
    A process of\n5 messages;
---: Further procedure may follow;

```



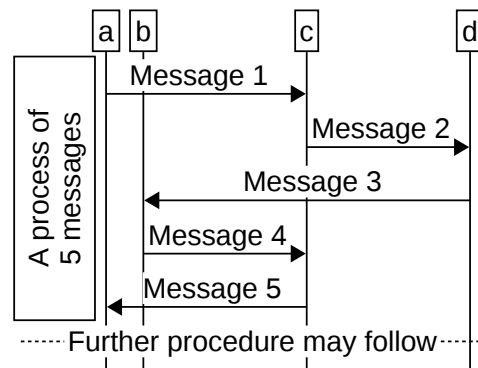
Between the two positions, one of the box or arrow symbols can be used: ‘--’, ‘..’, ‘++’, ‘==’, ‘->’, ‘=>’, ‘>’ or ‘>>’. If we omitted the *shape* specifier these symbols result in a box or a block arrow. The arrow symbols can be used in bidirectional or reverse variants, as well. For ranges and pointers the box symbols result in no arrowheads, for braces and brackets there is no difference between the box and arrow symbols, they only control the line style.

You can omit both markers. In this case the vertical spans besides the chart element before it. You can group a set of chart elements with curly braces and specify a vertical immediately after to make it span along the entire group. (This is a simpler way than to use the `mark` command.) You can even omit the *symbol* making it default to `->`<sup>5</sup>. The above chart can also be written as below.

```

hscale=auto;
a, b, c, d;
{
    a->c: Message 1;
    c->d: Message 2;
    d->b: Message 3;
    b->c: Message 4;
    c->a: Message 5;
};
vertical box at a-:
    A process of\n5 messages;
---: Further procedure may follow;

```

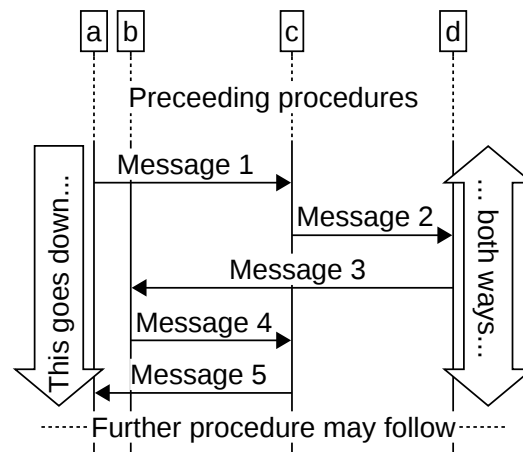


Verticals can contain a label, which can be rotated 90 degrees compared to other elements. This can be set via the `side` attribute, which specifies from which direction the text is readable from (`left`, `right` or `end`, which means the regular horizontal typeset). If you sent `side=end` to typeset the label horizontally, you can use word wrapping by setting `text.wrap=yes`. If you do so (even if via the default `text` attribute), you must specify a text width for each such vertical to do the wrapping in. Use the `text.width` attribute for this purpose. Verticals with vertically typeset text ignore the `text.wrap` attribute and do no label word wrapping.

<sup>5</sup> But you cannot omit both the *shape* and *symbol*.

The text after the ‘at’ keyword determines the horizontal location of the vertical. The horizontal position is defined in relation to entity positions. It can be placed onto an entity, left or right from it, or between two entities. These are specified as ‘<entity>’, ‘<entity>--’, ‘<entity>+’ or ‘<entity1>--<entity2>’, respectively. You can also specify any distance from an entity by adding a number after the first form, such as in ‘at <entity> <number>’. The number will be interpreted in pixels and shifts the vertical left or right depending on its sign. Use a space before the number.

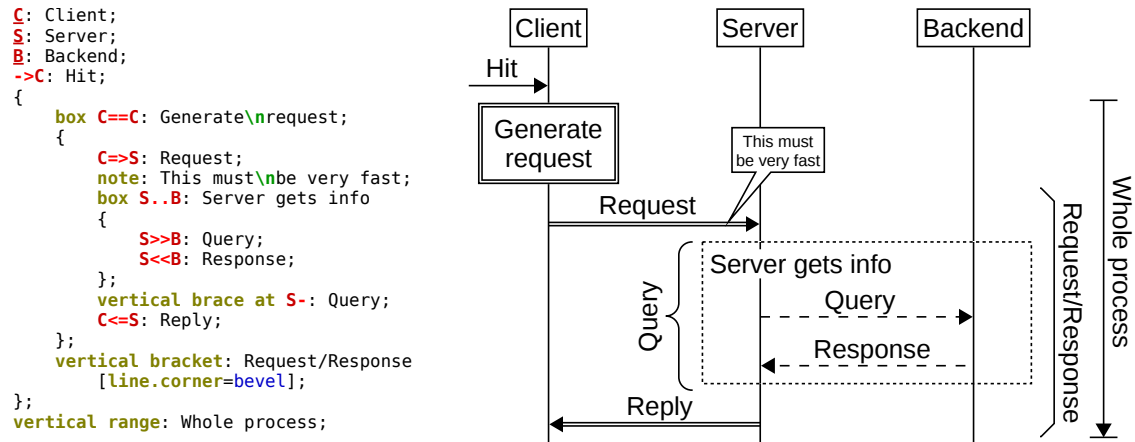
```
hscale=auto;
a, b, c, d;
...: Preceding procedures;
{
  a->c: Message 1;
  c->d: Message 2;
  d->b: Message 3;
  b->c: Message 4;
  c->a: Message 5;
};
vertical -> at a-:
    This goes down...;
vertical <->:
    ... both ways...;
---: Further procedure may follow;
```



You can also omit the **at** clause, which results in the vertical being placed besides the entities it spans (by default on the right side of them using ‘<entity>+’). If, however, the **side** attribute is set to **right** (to mean that the text is readable from the right direction), the vertical is placed left of the entities it spans besides.

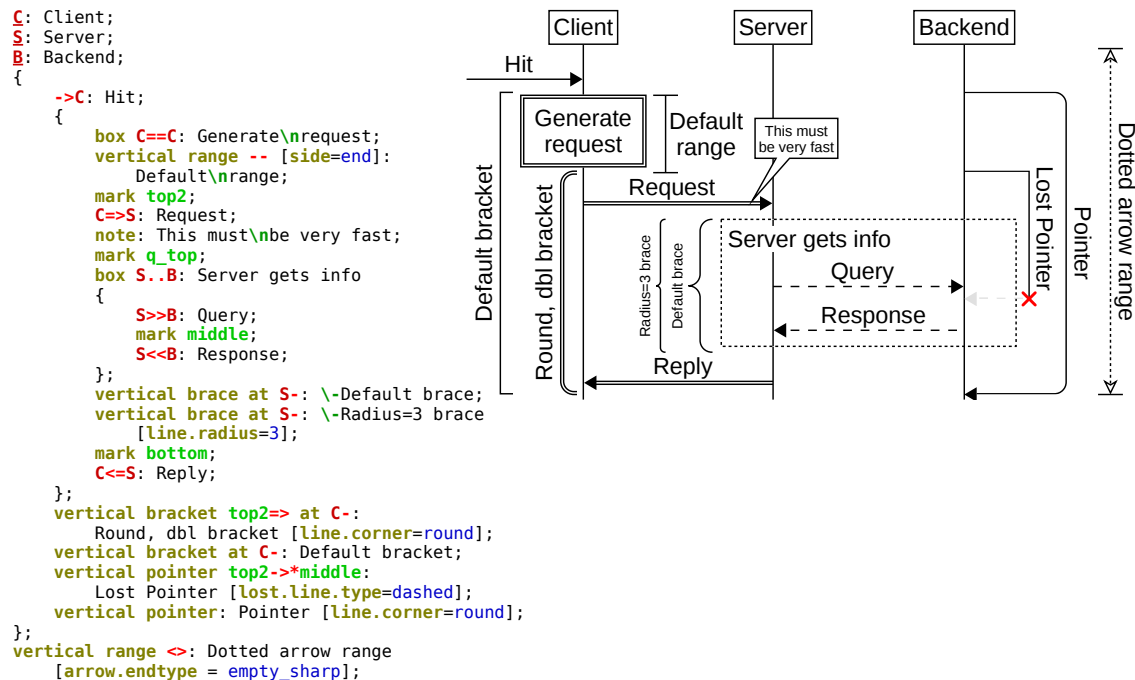
Verticals specified to be besides an entity (with ‘<entity>--’, ‘<entity>--’, ‘<entity>+’ or ‘<entity>++’ horizontal locations) are placed further from the entity line if there are boxes or elements in the way. Only those elements are considered, which are specified in the input file before vertical. If the vertical references markers below it, it may overlap with later elements, thus it is a good idea only to mark the top of the vertical and specify the vertical itself at the bottom location (as in all the examples)<sup>6</sup>. The **makeroom** attribute is a boolean value defaulting to yes. When it is turned off verticals are not considered when entity distances are calculated with **hscale=auto**. When **makeroom** is on, Msc-generator attempts to take the vertical into account when laying out entities. In a well-designed case you can even nest verticals, as a vertical specified earlier will be considered by subsequent verticals (but only if its **makeroom** attribute is set to yes).

<sup>6</sup> Note that Msc-generator does not lay out verticals entirely correctly in relation to parallel blocks.



e Below is a picture demonstrating all shapes of verticals. Here are a few tips on them.

The radius of the curves of the brace vertical can be adjusted with the `line.radius` attribute and defaults to 8. The width of the bracket vertical can also be influenced with the same attribute. In addition you can set the `line.corner` attribute to `round` or `bevel` to influence the corners of the bracket. The range vertical can display either an arrow or just a simple line depending on whether you use the arrow or box symbols. In case you specify an arrow, you can adjust the arrowhead via the `arrow.*` attributes.



The pointer vertical can be marked with an asterisk as being lost. If so it displays the same loss symbol that is used at lost messages (see [Section 5.3.1 \[Lost Messages\]](#), page 47) and you can use the `x.size` and `x.line.*` attributes to control its appearance. Note that

you cannot control the exact location of the loss symbol (via the `lost at` construct), it is always at the bottom of the pointer.

## 5.7 Dividers

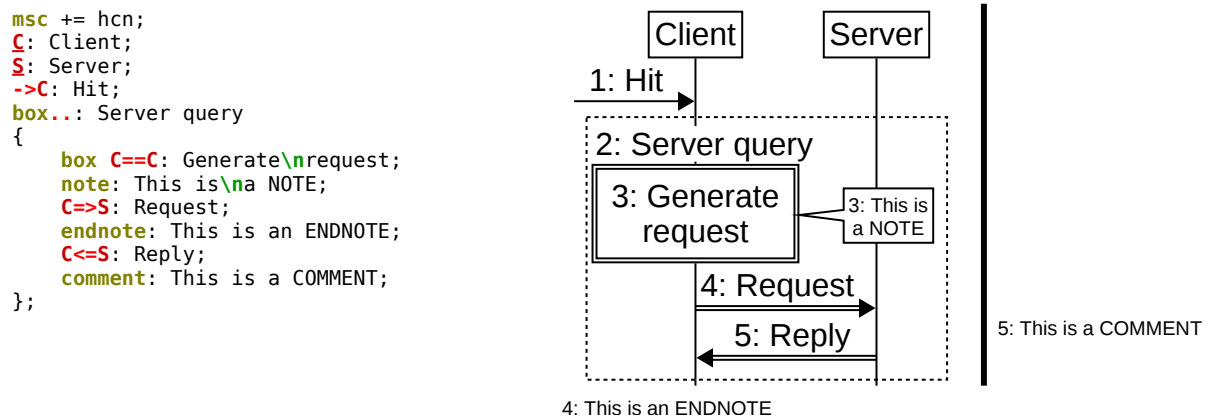
Dividers are called like this as they divide the chart to parts. Three types of dividers are defined. ‘---’ draws a horizontal line across the entire chart with potentially some text across it. ‘...’ draws no horizontal line, but makes all vertical entity lines dotted, thereby indicating the elapse of time.

The third type of divider is a simple vertical space. This can be specified by entering just attributes in square brackets. The extreme ‘[]’ simply inserts a lines worth of vertical space. You can add text, too by specifying a label. See [Section 3.3 \[Dividers\]](#), page 17 for examples.

Dividers take the `label`, `color`, `text.*`, `line.*`, `compress`, `vspacing`, `number` and `refname` attributes with the same meaning as for arrows. In addition, the type of the vertical line can be specified with `vline.*`, with `vline.type` defaulting to `dotted` for ‘...’ dividers and to `solid` for ‘---’ dividers. Other values are `dashed`, `none` and `double`. Again, note that the default values can be changed by using styles, see [Section 5.18 \[Defining Styles\]](#), page 91.

## 5.8 Notes and Comments

The ‘`note`’, ‘`comment`’ and ‘`endnote`’ commands enable you to make annotations to the chart that are visible to the reader. Notes are placed onto the chart drawing area in a callout; comments are placed onto a column left or right from the chart; whereas endnotes are placed at the bottom of the chart. Notes are suitable for shorter comments, whereas the latter two fit longer explanations better.



Each note, comment and endnote has a *target element*. The target element is the element preceeding the ‘`note`’, ‘`comment`’ or ‘`endnote`’ command<sup>7</sup>. In case of notes the tip of the callout will point to the target element, whereas side notes will be typeset beside

<sup>7</sup> Note that some elements cannot be targets, such as chart options. In this case the preceeding element becomes the target.

their target. You can issue multiple notes, comments and/or endnotes to the same target. If numbering is enabled for a note, comment or endnote, it inherits the numbering of its target (if any).

The syntax is simple, issue one of the three commands with attributes. You must specify a label, but similar to arrows or entities, the colon syntax can be used.

```
note: This is a note [attributes];
note at <tip>: Note pointing to <tip> [attribute];
comment: Comment text [attributes];
endnote: Endnote text [attribuest];
```

Note and comment text is typeset in a smaller font by default. You can change both of the above by changing the ‘note’, ‘comment’ or ‘comment’ styles.

### 5.8.1 Notes

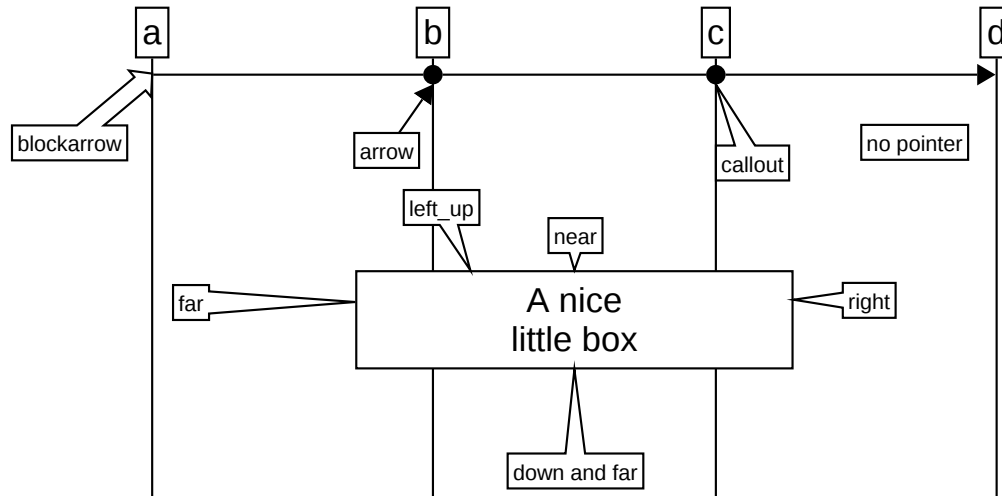
For notes the tip of the callout can be guided using the `at` keyword. After it you can specify either an entity or a marker. This is useful if you want to make a note to a specific part of an arrow.

You can use the `note.pointer` attribute to define, what the tip looks like. It can take four values: `none`, `callout`, `arrow` or `blockarrow`.

The position of the note is selected automatically by Msc-generator, but you can influence the choice via the `note.pos` attribute. It can take one of the following values: `near`, `far`, `left`, `right`, `up`, `down`, `left_up`, `left_down`, `right_up` or `right_down`. The first two can be used to specify the distance from the element, whereas the rest dictate which direction the note shall be. You can set this attribute twice if needed, once for distance and a second time for direction.

The ‘note’ style contains text, fill and line attributes and also ‘note.layout’ and ‘note.pos’ to define default note layout.

```
a,b,c,d;
a->b-c-d [arrow.midtype=dot];
note at a: blockarrow [note.pointer=blockarrow];
note at b: arrow [note.pointer=arrow];
note at c: callout;
note at d: no pointer [note.pointer=none];
vspace 80;
box b--c: A nice\nlittle box;
note: right [note.pos = right];
note: left_up [note.pos = left_up];
note: far [note.pos = far];
note: near [note.pos = near];
note: down and far [note.pos=down, note.pos=far];
vspace 40;
```



### 5.8.2 Comments and Endnotes

Comments can be set either to the left or the right side of the chart as dictated by the `side` attribute. This attribute can also take the value `end`, which will turn the comment to an endnote. In fact endnotes are comments with their `side` attribute set to `end`. So you can convert all your comments to endnotes by redefining the `comment` style, as below. For ease of use the `comment.text` and the `comment.side` chart options can also be used to set comment properties<sup>8</sup>.

```
defstyle note [text.size.normal=16, text.size.small=10];
defstyle comment [side=end];
comment.side=right;
comment.text.italics=yes;
```

When the chart contains comments on the side a line is drawn separating the comments from the chart text. You can change the properties of this line via the `'comment.line.*'` chart options. Only the width, color and type of the line can be changed (not its radius or corner). You can turn this line off by selecting the `'none'` line type. Similar, the background of the comments can be set via the `'comment.fill.*'` chart options. These options can also be made part of designs. Finally, the space available on the side for comments can be adjusted with the `hspace left|right comment` command, see [Section 5.15.1 \[Spacing\]](#), [page 84](#).

## 5.9 Common Attributes

As discussed earlier, attributes can influence how chart elements look like and how they are placed. There is a set of attributes that apply to several types of elements, so we describe them collectively here.

Attribute names are case-insensitive. Attributes can take string, number or boolean values. String values shall be quoted in double quotes (`"`) if they contain non-literal characters or spaces<sup>9</sup>. Quoted strings themselves can contain quotation marks by preceeding them with a backslash `"`. Numeric values can, in general be floating point numbers (no

<sup>8</sup> These are equivalent to changing the `'comment'` style. There is no such shortcut for endnotes, yet.

<sup>9</sup> Specifically strings that contain characters other than letters, numbers, underscores or dots, must be quoted. If the string starts with a number or a dot or it ends with a dot, it must also be quoted. The only exception to this are built-in style names, see [Section 5.18 \[Defining Styles\]](#), [page 91](#).



exponents, though), but for some attributes these are rounded to integers. Boolean values can be specified via **yes** or **no**. The syntax of color attributes is explained in [Section 5.10 \[Specifying Colors\]](#), page 74.

The attributes below can be part of a *style*, see [Section 5.9.1 \[Styles\]](#), page 68.

**line.color**

Specifies the color of the line for the element. For arrows and dividers this is the horizontal line. For block arrows, boxes, pipes and entities this is the line around the element. Unless you use a single color name you must quote the color specification, see [Section 5.10 \[Specifying Colors\]](#), page 74 for the syntax of colors.

**line.width**

Specifies the width of the line.

**line.type**

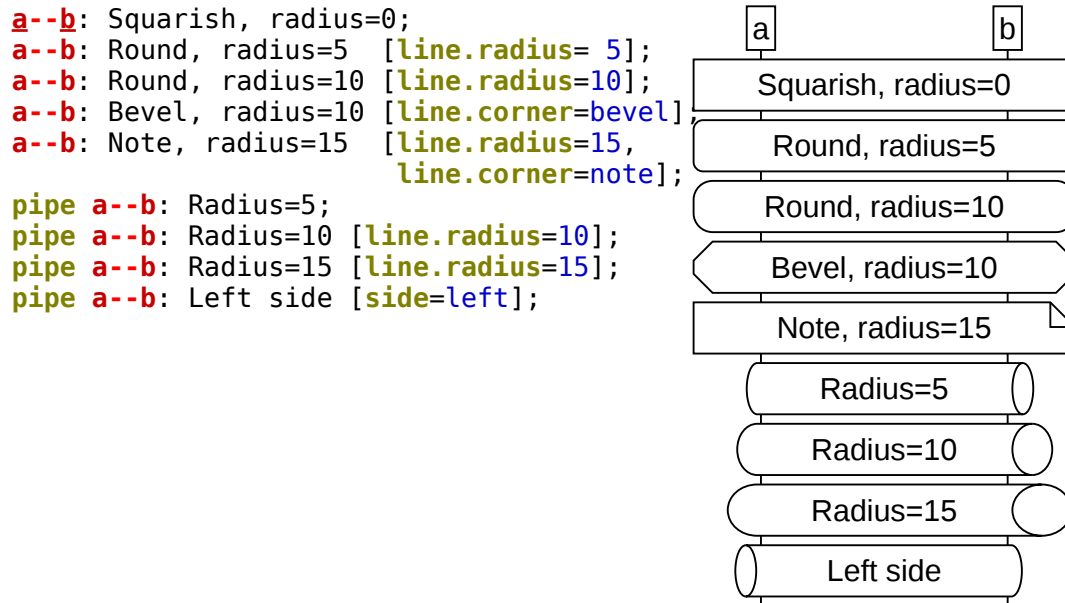
Specifies the type of the line. Its value can be **solid**, **dashed**, **dotted**, **double** or **none**.

**line.radius**

For arrows it has effects only on arrows starting and ending in the same entity (see [Section 5.3.2 \[Arrow Attributes\]](#), page 48). For entities and boxes, this specifies the size of the corners. 0 is fully sharp, values are meant in pixels. If no **line.corner** is specified setting radius to a positive value will result in round corners. For pipes, it specifies the width of the oval, in other words from how left we look at the pipe.

**line.corner**

For boxes and entities this attribute specifies how the corners of the box are drawn. Its value can be **none**, **round**, **bevel**, **note**. It has no effect on other elements.



**vline.\*** Specifies the color, width or type of the vertical line stemming from entities. This is useful to indicate some change of state for the entity. **vline.radius** and **vline.corner** has no effect. These attributes can be used for entities and dividers.

#### **fill.color**

Defines the background color of the box, entity, block arrow or pipe. Specifying **none** results in no fill at all. Unless you use a single color name you must quote the color specification, see [Section 5.10 \[Specifying Colors\]](#), page 74 for the syntax of colors.

#### **fill.color2**

If this attribute is specified then the fill gradient will not be between **fill.color** and a lighter variant, but between **fill.color** and the value specified here. If no gradient specified or **button** is used, this attribute has no effect.

#### **fill.gradient**

Defines the gradient of the fill. It can take five values **up**, **down**, **in**, **out** and **button**. The first two results in linear gradients getting darker in the direction indicated. The second two results in circular gradients with darker shades towards the center or edge of the entity box, respectively. The last one mimics light on a button.

```

hscale = auto;
defstyle entity
  [fill.color="yellow-25",
   text.format= "\mu(10)\md(10)\ml(10)\mr(10)"];
Up      [fill.gradient=up],
Down    [fill.gradient=down],
In       [fill.gradient=in],
Out      [fill.gradient=out],
Button  [fill.gradient=button];

```

**shadow.offset**

If not set to zero, then the entity or box will have a shadow (default is 0). The value of this attribute then determines, how much the shadow is offset (in pixels), in other words how "deep" the shadow is below the entity or box.

**shadow.color**

The color of the shadow. This attribute is ignored if shadow.offset is 0. Unless you use a single color name you must quote the color specification, see [Section 5.10 \[Specifying Colors\]](#), page 74 for the syntax of colors.

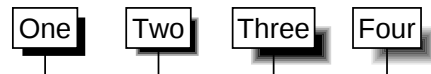
**shadow.blur**

Specifies how much the shadow edge is blurred (in pixels). E.g., if shadow.offset is 10 and shadow.blur is 5, then half of the visible shadow will be blurred. Blurring is implemented by gradually changing the shadow color's transparency towards fully transparent. This attribute is ignored if shadow.offset is 0.

```

hscale = 0.5;
One     [shadow.offset= 5],
Two     [shadow.offset= 5, shadow.blur= 2],
Three   [shadow.offset=10, shadow.blur= 5],
Four    [shadow.offset=10, shadow.blur=10];

```

**text.ident**

This can be `left`, `center` or `right` and specifies the line alignment of the label. The default is centering, except for non-empty boxes, where the default is left. It can be abbreviated as simply `ident`.

**text.color**

Sets the color of the label. Unless you use a single color name you must quote the color specification, see [Section 5.10 \[Specifying Colors\]](#), page 74 for the syntax of colors.

**text.format**

Takes a (quoted) string as its value. Here you can specify any of the text formatting escapes that will govern the style of the label, see [Section 5.11 \[Text Formatting\]](#), page 75. Specifying them here or directly at the beginning of the label has the same effect, so having this attribute is more useful for styles.

**text.wrap**

Can be set to `yes` or `no`. If disabled (default), the label will follow the line breaks inserted by the user. If enabled, these line breaks are ignored and the line is typeset to fill available space, see [Section 5.11 \[Text Formatting\]](#), page 75.

**arrow.\*** Styles can also contain arrow formatting attributes. These are described in [Section 5.3.2 \[Arrow Attributes\]](#), page 48.

`lost.text.*`

`lost.line.*`

`lost.arrow.*`

The values specified here will be added to the values of `text.*` `line.*` or `arrow.*` when drawing the text, line or arrowheads of the lost part of the message, see [Section 5.3.1 \[Lost Messages\]](#), page 47. Only applicable for arrows.

`x.size`

`x.line.*` The controls the appearance of the loss symbol for lost messages, see [Section 5.3.1 \[Lost Messages\]](#), page 47.

`tag.line.*`

`tag.fill.*`

`tag.text.*`

These attributes apply only to boxes (applicable to the `box`, `emptybox` and `box_collapsed` style) and govern the style of tags, if the `tag` attribute of the box is set. (The `tag` attribute is not part of the style, you must set it individually on each box you want to have a tag.)

`shape` This attribute takes the name of the shape you want for the entity headings. See [Section 5.2.6 \[Entity Shapes\]](#), page 46. They can be made part of style but have effect only on entities.

`shape.size`

This attribute specifies the size of the shape to use for the entity headings. Only has effect if a valid shape is specified via the `shape` attribute. It takes one of `tiny`, `small`, `normal`, `big` or `huge` with `small` as default. They can be made part of style but have effect only on entities.

`note.layout`

`note.pos` These govern how notes are laid out. See [Section 5.8 \[Notes and Comments\]](#), page 61 on how to use them. They can be made part of style but have effect only on notes.

`side` This attribute can take either `left` or `right`. For pipes it specifies which side the pipe can be looked from into. For verticals it tells which side the text can be read from. For comments it specifies which side of the chart the comment is placed on. It has no effect on any other elements.

`solid` This attribute can be used to set the transparency of a pipe. See [Section 5.5 \[Pipes\]](#), page 55 for more information.

`number` This attribute governs if the arrow, box, etc. is numbered or not. See [Section 5.9.3 \[Numbering\]](#), page 70 for details.

`compress` If this attribute is set to `yes`, the element is drawn as close to the ones above it as possible without touching those. It is useful to save space, see [Section 5.9.4 \[Compression and Vertical Spacing\]](#), page 73 for a detailed description.

`vspacing` Can be set to a number interpreted in pixels or to the string `compress`. Governs how much vertical space is added before the element (can be negative). This attribute is another form (superset) of the `compress` attribute; `compress=yes`

is equivalent to `vspacing=compress`, whereas `compress=no` is equivalent to `vspacing=0`.

#### `collapsed`

This attribute can be used for group entities and boxes to collapse them.

#### `indicator`

If this is set to `yes` on a collapsed group entity or box, indicators will show hidden entities and other chart elements.

The attributes below can be specified for most elements, but cannot be made part of a style

**label** This gives the label of the element (for elements having one). It can be abbreviated with the colon notation, see [Section 5.9.2 \[Labels\]](#), page 69.

**refname** Use this attribute to name the element for later reference. Used primarily to refer to elements via their numbers using the ‘`\r(name)`’ escape in labels.

#### `draw_time`

Use this attribute to draw elements earlier or later and thereby control how they overlap. See more in [Section 5.15.2 \[Symbols\]](#), page 85.

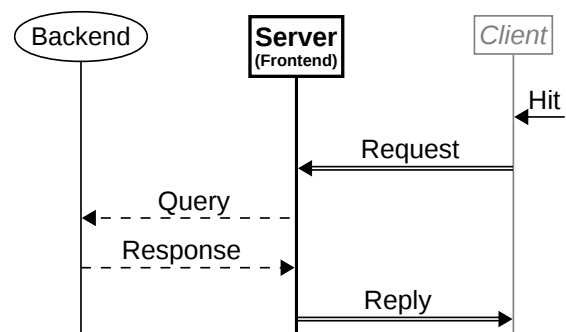
**parallel** This can take a `yes` or a `no` and is equivalent to prepending the element with the `parallel` keyword, see [Section 5.12 \[Parallel Blocks\]](#), page 79.

### 5.9.1 Styles

Styles are packages of attribute definitions with a name. Applying a style to any element can be easily done by simply stating the name of the style wherever an attribute is allowed, see the example below.

```
B: Backend [shape=def.oval];
S: Server\n\-(Frontend) [strong];
C: Client [weak];
```

```
C<-: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C<=S: Reply;
```



Styles can contain any of the attributes listed in the above section. If a style contains an attribute not applicable for the element that you apply the style to, that attribute is simply ignored. For example, applying a style with `fill.color=red` attribute setting to an arrow, will ignore this attribute since arrows take no fill attributes.

You can define your own styles or redefine existing ones. See [Section 5.18 \[Defining Styles\]](#), page 91 for more on this.

### 5.9.2 Labels

Entities, arrows, boxes, pipes and dividers have a `label` attribute, which specifies the text to be displayed for the element. Each element displays it at a different place, but the syntax to describe a label is the same for all. For entities the label defaults to the name of the entity, while for the rest it defaults to the empty string. Labels have to be quoted if they contain any character other than letters, numbers, underscores and the dot, or if they start with a dot or number or end with a dot. You can use all character formatting features in labels, see [Section 5.11 \[Text Formatting\]](#), page 75.

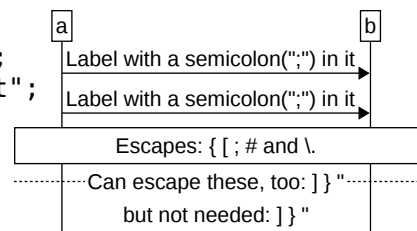
To avoid typing `[label="..."]` many times it is possible to specify the label attribute in a simpler way. After the definition of the element, just type a colon, the text of the label unquoted and terminate with a semicolon (or opening brace ‘{’ or bracket ‘]’). You can write attributes before or after the label. Thus all lines below result in the same text.

```
a->b [label="This is the label", line.width=2];
a->b: This is the label [line.width=2];
a->b [line.width=2]: This is the label;
```

If the label needs to contain a opening bracket (‘[’), opening brace (‘{’), hashmark (‘#’) or a semicolon (‘;’) use quotations or precede these characters by a backslash ‘\’<sup>10</sup>. This is needed since these characters would otherwise signal the end of the label (or the beginning of a comment) If you want a real backspace, just type ‘\\’.

When using the colon notation, heading and trailing spaces are removed from the label. If these are needed, place the entire label between two quotation mark ‘”’<sup>11</sup>.

```
hscale=auto;
a->b:    Label with a semicolon(";") in it;
a->b: "   Label with a semicolon("\;") in it";
box a--b: Escapes: \{ \[ \; \# and \.;
---: Can escape these, too: \] \} \";
: but not needed: ] } ";
```



Labels can span multiple lines. You can insert a line break by adding the ‘\n’ escape sequence. Alternatively you can simply break a label and continue in the next line. In this case leading and trailing whitespace is removed from each line.

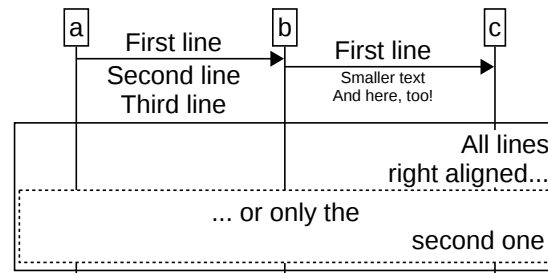
<sup>10</sup> This character is often called the *escape character* making an *escape sequence* together with the character it follows.

<sup>11</sup> In this case there is no need to escape the opening bracket or brace, the hashmark or the semicolon, since the end of the label is clearly indicated by the terminating quotation mark. If, on the other hand you need quotation marks in the label use ‘\’’. Also, you cannot break the text in multiple lines in the input file, you have to use the ‘\n’ escape to insert line breaks. This mode is provided only for backwards compatibility.

```

compress=yes;
a->b: First line
      Second line #comment
      Third line;
b->c: First line
      \-Smaller text
      And here, too!;
box a--c: \prAll lines
           right aligned... {
           box a..c: ... or only \prthe
                     second one;
};

```



### 5.9.3 Numbering

Arrows, boxes and dividers (any element with a label, except entities) can be auto-numbered. It is a useful feature that allows easier reference to certain steps in a procedure from explanatory text. To assign a number to an element, simply set its **number** attribute to **yes**. You can also assign a specific number, in that case the element will get that number and subsequent elements will be numbered (if they have **number** set to **yes**) from that number upwards.

Notes and comments will not increase numbering, instead they carry the number of the element they are referring to. If the target element had no number comments will have none, even if numbering is turned on for them.

Styles can also control numbering. If a style has its **number** attribute set to **yes** or **no**, any element that you assign the style to will have its attribute set likewise. See [Section 5.9.1 \[Styles\]](#), page 68 for more.

In order to minimize typing, the **numbering** chart option can be used. It can be set to **yes** or **no** and serves as the default for freshly defined elements. You can set the value of **numbering** at any time and impact elements defined thereafter. You can use scoping to enable or disable numbering for only blocks of the chart, see [Section 5.17 \[Scoping\]](#), page 90.

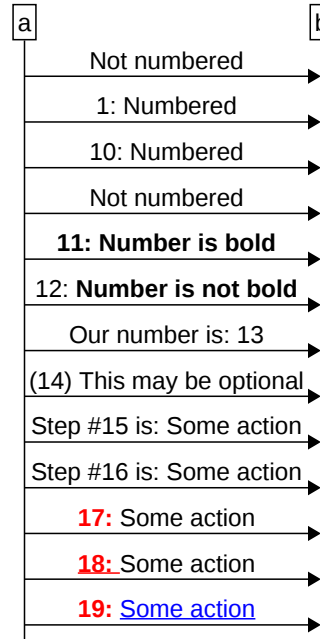
Most of the time you just declare **numbering=yes** at the beginning of the chart and are done with it. However, if you want to control that only some parts of the elements (e.g., only concrete messages and not boxes, for example) got a number, you may need the other alternatives.

```

hscale=auto;

a->b: Not numbered;
a->b: Numbered [number=yes];
a->b: Numbered [number=10];
a->b: Not numbered;
numbering=yes;
a->b: \bNumber is bold;
a->b: \\bNumber is not bold;
a->b: Our number is: \N;
a->b: (\N) This may be optional;
numbering.pre="Step #";
numbering.post=" is: ";
a->b: Some action;
a->b: Some action;
numbering.pre="\c(red)\b";
numbering.post=": \s()";
a->b: Some action;
a->b: \c(blue)\uSome action;
a->b: \\c(blue)\uSome action;

```



If numbering is turned on for a label, the number is inserted at the beginning of the label and is followed by a semicolon and a space by default. More precisely, the number is inserted after any initial text formatting sequences, so that it has the same formatting as the label itself (see [Section 5.11 \[Text Formatting\], page 75](#))<sup>12</sup>. The above default can be changed by inserting the `\N` escape sequence into a label. This causes the number appear where the `\N` is inserted, as opposed to the beginning of the label. In this case, the colon and the space is omitted, only the number itself is inserted.

The colon and space can be changed to some other value by setting the `numbering.post` chart option to the string you want to append to the number. Similar, any string the `numbering.pre` option is set to will be prepended to the number (empty by default). Both options are ignored when using the `\N` escape sequence to set the label position.

Note that for the last two arrows formatting escapes were added to the `numbering.pre` option. These are reversed by the `\s()` escape in the `numbering.post` option. See [Section 5.11 \[Text Formatting\], page 75](#) for more details.

The format of the number can be set with the `numbering.format` chart option. You can specify any of `'123'`, `'iii'`, `'III'`, `'abc'`, or `'ABC'` for arabic, lowercase and uppercase roman numbers or lowercase and uppercase letters, respectively<sup>13</sup>. You can also prepend or append any text before or after the above strings, those will be prepended or appended to the number (and will be included also when the number is inserted via the `\N` escape).

Note that the value of the `numbering` options is subject to scoping, that is any change lasts only up to the next closing brace.

<sup>12</sup> You can use the `\|` formatting escape to insert a non-visible break into a stream of formatting escapes. The number will be inserted there.

<sup>13</sup> Using `'arabic'`, `'letters'` or `'roman'` is also valid (both uppercase or lowercase).



Note also, that when using roman numbers or letters, you can use such numbers as the value of the `number` attribute, as shown below for `'7c'`.

It is also possible to have multi-level numbering (such as 1.1). To achieve this, use the `numbering.append` chart option and specify the format of the second level including any separator. Use the same format as for `numbering.format` above.

It is possible to change the format of a multi-level label via the `numbering.format` option. Simply use multiple of the number format strings (such as `'123'` or `'roman'`) as in the `'Exotic format'` line of the example above. If you use less number format strings than the current number of levels (as in the `'Only the last number'` line of the example), Msc-generator displays only the end of the number, omitting levels from the top. Those levels, however, are still maintained, just are not displayed.

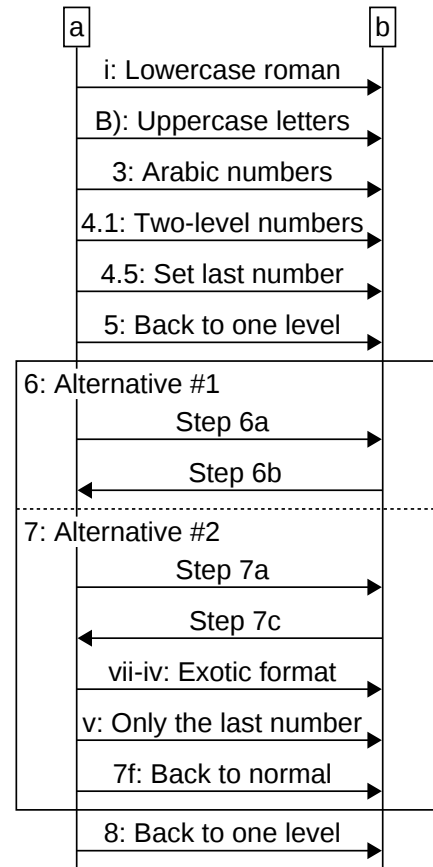
The `numbering.append` option can only be used to add levels. There is no explicit way to decrease the number of levels, you have to use scoping to achieve that. On the example above, the second level appended in the scope of `'Alternative #1'` is cancelled at the end of the scope, so we need to append a second level also in `'Alternative #2'`, which then restarts from `'a'`.

```

hscale=auto, numbering=yes;

numbering.format = "roman";
a->b: Lowercase roman;
numbering.format = "ABC";
a->b: Uppercase letters;
numbering.format = "123";
a->b: Arabic numbers;
{
  numbering.append = ".123";
  a->b: Two-level numbers;
  a->b: Set last number [number=5];
};
a->b: Back to one level;
box a--b: Alternative \#1 {
  numbering.append = "abc";
  a->b: Step \N;
  b->a: Step \N;
}
a..b: Alternative \#2 {
  numbering.append = "abc";
  a->b: Step \N;
  b->a: Step \N [number=c];
  numbering.format = "roman-roman";
  a->b: Exotic format;
  numbering.format = "roman";
  a->b: Only the last number;
  numbering.format = "123abc";
  a->b: Back to normal;
};
a->b: Back to one level;

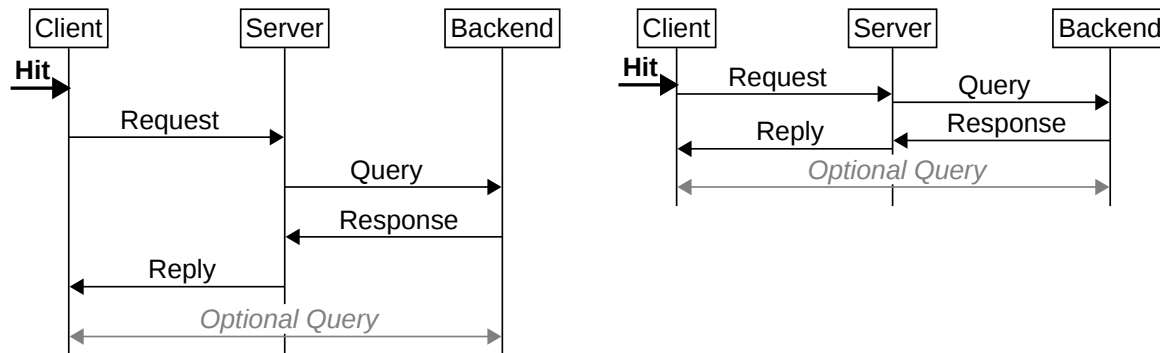
```



Finally, if an element is named using the `refname` attribute, you can reference the number of that element in another label using the `\r(name)` escape sequence. Note that the value of the `numbering.pre` and `numbering.post` options are ignored when inserting the number of a referenced element, similar to how the `\N` escape inserts numbers. Specifying an empty `\r()` escape inserts the number of the current element and is thus equivalent to `\N`.

### 5.9.4 Compression and Vertical Spacing

In this section we explain how Msc-generator sets the vertical space between elements. It can apply a set amount of vertical space or can use the *compression* mechanism. The latter aims to reduce the height of chart graphics by vertically pushing chart elements closer to each other. See the two examples below copied from the end of [Section 3.1 \[Defining Arrows\]](#), [page 8](#). They differ only in that the second begins with `compress=yes`.



Each element (except entities) has a `compress` and a `vspacing` attribute. When the former is set to `yes`, the element is first placed fully under the element before it, then it is shifted upwards until it bumps into some already drawn element. The same effect can be achieved by using `vspacing=compress`. If compression is not used the element is placed below the previous element by the value of the `vspacing` attribute (understood in pixels). E.g., using `vspacing=10` adds 10 pixels between the element and the element before it. The `vspacing` attribute is the superset of the `compress` attribute, setting `compress` to `yes` or to `no` is equivalent to `vspacing=compress` and `vspacing=0`, respectively.

Compression and vertical spacing can be set individually for each element, but to save typing by setting the `compress` or `vspacing` chart option, you can effectively set the `compress` or `vspacing` attribute of all elements after. This is similar, how the `numbering` chart option effects the `number` attribute. If you then want to exempt specific elements from compression or add more space individually (so that they are somewhat further from the element above), just specify the `compress` or `vspacing` attribute for the element in question.

Styles can also influence compression and vertical spacing the same way as numbering, that is you can set the `compress` or `vspacing` options for a style, which will effect compression and vertical spacing of elements you assign the style to.

Note that to insert extra vertical spacing you can also use the `vspace` command, see [Section 5.15.1 \[Spacing\]](#), page 84.

## 5.10 Specifying Colors

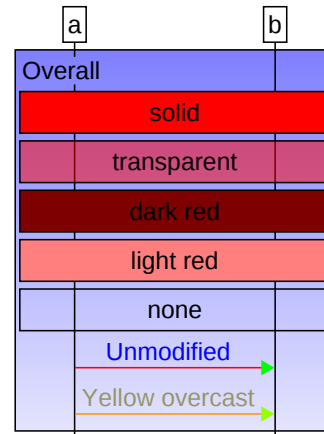
Msc-generator has the following color names defined initially: `none`, `white`, `black`, `red`, `green`, `blue`, `gray` and `lgray`, the first for completely transparent color, and the last for light gray. When you specify a color by name, no quotation marks are needed.

Color names can be appended with a '+' or '-' sign and a number between [0..100] to make a color lighter or darker, respectively, by the percentage indicated. Any color +100 equals white and any color-100 equals black. Aliases can be further appended with a comma and a value between [0..255] (or [0..1.0] similar to RGB values). This specify color opaqueness: 0 means fully transparent and 255 means fully opaque.

```

a, b;
--: Overall [fill.color = blue+50,
           fill.gradient=up] {
  a--b [fill.color=red]:    solid;
  a--b [fill.color=red,128]: transparent;
  a--b [fill.color=red-50]: dark red;
  a--b [fill.color=red+50]: light red;
  a--b [fill.color=none]:   none;
  defstyle arrow [line.color=red,
                  arrow.color=green,
                  text.color=blue];
  a->b: Unmodified;
  a->b: Yellow overcast [color=++yellow,150];
};

```



You can specify colors giving the red, green and blue components separated by commas. An optional fourth value can be added for the alpha channel to control transparency. Values can be either between zero and 1.0 or between 0 and 255. If all values are less than or equal to 1, the former range is assumed<sup>14</sup>. If any value is negative or above 255 the definition is invalid. Note that you must not enter spaces between the color name, its lighter/darker or transparency modifier or between the RGB values.

You can mark a color a *overlay* color by prepending the ‘++’ symbol. An overlay color applied to an attribute is not set as the color value. Instead, it is overlaid over and thus is combined with the existing color. (See the arrows in the example above.) You can also assign an overlay color to a style or a color name (see below). (This is how their **weak** style is implemented.)

It is possible to define your own color names using the **defcolor** command as below.

```
defcolor alias=color definition, ... ;
```

Color names are case-sensitive and can only contain letters, numbers, underscores and dots, but can not start with a number or a dot and can not end with a dot. Aliases can also be later re-defined using the **defcolor** command, by simply using an existing alias with a different color definition.

Msc-generator honors scoping. Color definitions (or re-definitions) are valid only until the next closing brace ‘}’. This makes it possible to override a color only for parts of the chart, returning to the default later. Note that you can start a new scope any time by placing an opening brace. See [Section 5.17 \[Scoping\]](#), page 90 for more on scopes.

## 5.11 Text Formatting

Entity, divider, arrow, pipe and box labels –any text displayed in the chart– can contain *formatting escapes*. Each formatting escape begins with the backslash ‘\’ character. You can also use the backslash to place special characters into the label. Below is the list of escape sequences available.

<sup>14</sup> This mechanism allows both people thinking in range [0..1] and in [0..255] to conveniently specify values. (Internally values are stored on 8 bits.)

<code>\n</code>	Inserts a line break.
<code>\-</code>	Switches to small font.
<code>\+</code>	Switches to normal (large) font.
<code>\^</code>	Switches to superscript.
<code>\_</code>	Switches to subscript.
<code>\b</code>	Toggles bold font.
<code>\B</code>	Sets font to bold.
<code>\i</code>	Toggles italics font.
<code>\I</code>	Sets font to italics.
<code>\u</code>	Toggles font underline.
<code>\U</code>	Sets font to underlined.
<code>\f(font face name)</code>	Changes the font face. Available font face names depend on the operating system you use. On Windows, you can use all the fonts available, but only OpenType and TrueType fonts provide correct alignment. On Linux you can use whatever font backend your cairo library was compiled for. This typically includes FreeType. If you specify no font, just <code>f()</code> , the font used at the beginning of the label is restored.
<code>\0..\9</code>	Inserts the specified number of pixels as line spacing below the current line.
<code>\c(color definition)</code>	Changes the color of the text. Color names or direct rgb definitions can both be used, as described in <a href="#">Section 5.10 [Specifying Colors]</a> , <a href="#">page 74</a> . No quotation is needed. You can also omit the color and just use <code>\c()</code> , which resets the color back to the one at the beginning of the label.
<code>\s(style name)</code>	Applies the specified style to the text <sup>15</sup> . Naturally only the <code>text.*</code> attributes of the style are applied. You can omit the style name and specify only <code>\s()</code> , which resets the entire text format to the one at the beginning of the label <sup>16</sup> . See <a href="#">Section 5.9.1 [Styles]</a> , <a href="#">page 68</a> for more information on styles.
<code>\mu(num)</code>	
<code>\md(num)</code>	
<code>\ml(num)</code>	
<code>\mr(num)</code>	
<code>\mi(num)</code>	Change the margin of the text or the inter-line spacing. The second character stands for up, down, left, right and internal, respectively. ‘num’ can be any

<sup>15</sup> Note that the `\s` formatting escape was used to switch to small font in 1.x versions of Msc-generator (since 2.0 `\-` is used for that). In order to work with old format charts, if the style name is not recognized, Msc-generator will give a warning but fall back to using small font.

<sup>16</sup> Any formatting escapes strictly at the beginning of a label (up to the first non-formatting escape or literal character) are included in the text format, so if you start a label with ‘`\b`’ then ‘`\s()`’ will restore a bold font. To prevent this use the ‘`\|`’ escape to create an invisible non-formatting character.

nonnegative integer and is interpreted in pixels. Intra-line spacing comes in addition to the line-specific spacing inserted by `\0..\9`. Defaults are zero. You can also omit the number, which restores that particular value to the one in effect at the beginning of the label. Note that Msc-generator always adds enough left and right margins to arrow labels to avoid overlapping the label with the arrowhead. Thus if you specify less margin, it may not show as you expect.

 $\backslash mn(num)$ 

**\ms(num)** Changes the size of the normal or small font. This applies only to the label, where used, not globally for the entire chart. Defaults are `\mn(16)\ms(10)`. You can also omit the number, which restores that particular value to the one at the beginning of the label.

\p1 \pc \pr

Changes the indentation to left, centered or right. Applying at the beginning of a line (t.i., before any literal character) will apply new indentation to that line and all following lines within the label. Applying after the beginning of a line will only impact subsequent lines.

\{ \[ \" \; \# \} \]

These produce a literal ‘{’, ‘[’, ‘”’, ‘;’, ‘#’, ‘}’ or ‘]’, respectively, since these are characters with special meaning and would, otherwise signal the end of a label. The last two can actually be used without the backslash, but result in a warning.

VI

This escape is a non-formatting escape that generates no output. It can be used at the beginning of a label to delimit those formatting escapes that are included in the default formatting restored by the ‘\s()’ escape and used to format the label number, from those which are just to be applied at the beginning of the label.

 $\backslash N$ 

This escape marks the position of the label number within the label. If omitted the number is prepended to the beginning of the label (after the initial formatting escapes). If no number is specified for the label, this escape has no effect. You can specify ‘\N’ multiple times, with each occurrence being replaced by the number. Note that if you omit ‘\N’, the number inserted at the beginning of the label is augmented by the value of the `numbering.pre` and `numbering.post` options, whereas with the ‘\N’ option, those are not used.

 $\backslash r(\textit{refname})$ 

This escape inserts the number of the referenced element. Use the `refname` attribute to name elements. Similar to the ‘`\N`’ escape, the value of the `numbering.pre` and `numbering.post` options are ignored. When no name is given (that is ‘`\r()`’) the escape is equivalent to ‘`\N`’.

Font size commands (including superscript or subscript) last until the next font size formatting command. For example in order to specify a subscript index, use `label="A\_i\+ value"`.

Any unrecognized escape characters in a label are removed with a warning. Unrecognized escapes and plain text in `text.format` attributes is ignored with a warning.

Note that the `text.*` chart options can be used to set the default text formatting.

### 5.11.1 Word Wrapping

Before Msc-generator 3.6 the user was required to manually specify line breaks in labels. Using the `text.wrap` attribute you can instruct Msc-generator to break lines automatically depending on how much horizontal space is available. For labels with this attribute set the line breaks of the source file inside the label are ignored. However, the line breaks inserted into the label via the `\n` escape sequence are still honoured. You can set the `text.wrap` attribute of labels globally via the `text.wrap` chart option, but you can also override this setting individually for each label.

You cannot set this attribute for entities. Their label is always typeset with `text.wrap=no` exactly as you specify in the source file.

Note that this feature is most useful if you do not use automatic horizontal scaling `hscale=auto`, since in that case the distance between entities is determined from the size of the labels - and with `text.wrap=yes` there is no inherent size for most labels. For notes, which float and whose width is not determined by the spacing of entities, a new `width` attribute is inserted, which can specify the width of the note making word wrapping meaningful.

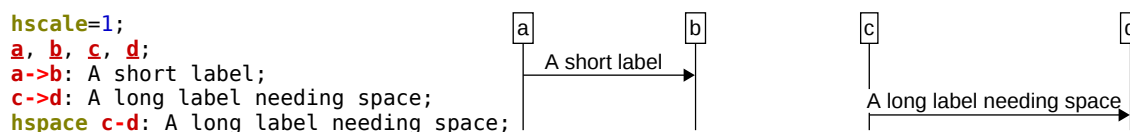
### 5.11.2 Long Labels

You have effective 3 easy way to typeset long labels.

- Word wrapping: Use `text.wrap=yes` (and perhaps a fixed `hscale`), in this case the long labels wrap into multiple lines.
- Automatic scaling: Use `hscale=auto` and no word wrapping, in this case entities are spaced apart, so that there is enough space for all labels.
- None: No word wrapping or automatic scaling (default): long labels expand beyond their available space, which may be sometimes ugly.

You can some combinations, as well.

- Even with `hscale=auto` you can make some long labels word wrap by applying `text.wrap=yes` only to the specific arrow, box, divider or comment. Specifying a long label with word wrapping will not cause entities to be spaced apart to make room for it, but instead the label is typeset into the space available (determined by other labels). Adding horizontal spacing with the `hspace` command can be applied to manually push entities somewhat apart (but perhaps not to the full length of the long label, which will be wrapped into the space available).
- Even with a fixed `hscale` You can push entities further by using the `hspace` command and thereby make enough room for a long label. You can create exactly as much as needed by using the label text as the argument for `hspace`, see below



## 5.12 Parallel Blocks

Sometimes it is desired to express that two separate process happen side-by-side. *Parallel blocks* allow this. Simply place the the parallel blocks between ‘{ }’ marks and write them one after the other, as in [Section 3.5 \[Drawing Things in Parallel\]](#), [page 22](#). You can specify as many parallel blocks as you want. The last parallel block shall be terminated with a semicolon. The order of the blocks is irrelevant, with the exception of numbering, which goes in the order the blocks are specified in the source file. It is possible to place anything in a parallel block, arrows, boxes, or other parallel blocks, as well.

By default, elements in parallel blocks are laid out so that they do no overlap. (At the cost of potentially showing elements sequentially even if they were intended to be shown in parallel.) This behaviour is new in version 3.6, so a chart option ‘`classic_parallel_layout`’ is provided, which returns to the old behaviour. In that mode, no attempt is made to avoid overlaps. The top of each block is drawn at the same vertical position. If you start with two arrows, they may be aligned and appear as a single arrow. To avoid this use the `nudge` command in one of the blocks which inserts a small vertical space top mis-align accidentally aligned arrows. This is not needed in case of the new algorithm. Also if elements spanning the entire width of the chart are specified in a parallel block (such as the `heading` command and dividers) they trigger a warning.

The next element below the series of parallel blocks will be drawn after the longest of the parallel blocks.

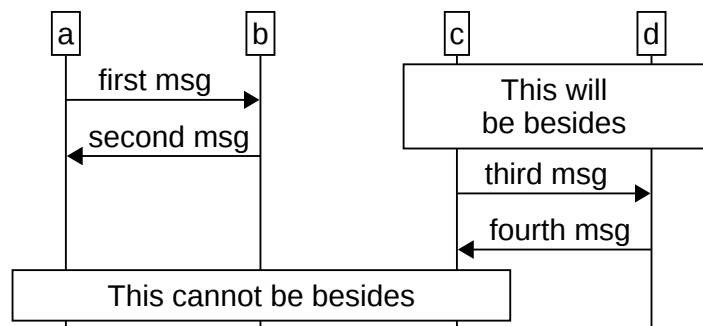
### 5.12.1 Parallel Keyword

Specifying the keyword `parallel` in front of an element will make the rest of the chart be drawn in parallel with it. To be more precise the effect only lasts till the end of the scope, so elements after the next closing brace will be drawn sequentially under<sup>17</sup>.

You can place `parallel` in front of really any element, including entity definitions or even parallel blocks. You can even combine several elements using braces.

```
hscale = 0.8;
parallel {
    a->b: first msg;
    a<-b: second msg;
};
box c--d: This will
        be besides;

parallel {
    c->d: third msg;
    c<-d: fourth msg;
};
box a--c: This cannot be besides;
```



<sup>17</sup> This is how this works exactly: first, the element marked with `parallel` is placed. Then the rest of the elements in the scope are placed below it and are moved as one block up at most to the top of the element marked with `parallel`. The move stops if any element in the block being moved bumps into an already placed element, thus overlaps are avoided.



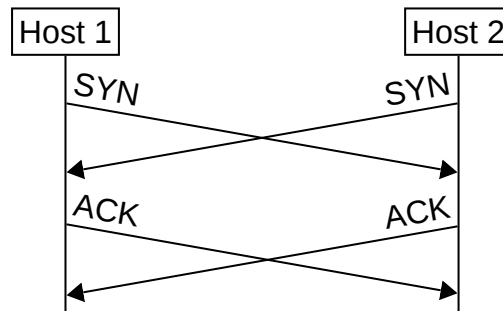
### 5.12.2 Overlap Keyword

Sometime one explicitly wants two elements to overlap. One prime case is to show slanted messages to cross each other. You could do it via parallel blocks allowing overlap by setting the ‘`classic_parallel_layout`’ chart option to yes, but that is ugly. A better way is offered by the `overlap` keyword.

Specifying this keyword in front of any element will result in ignoring this element at the layout of subsequent elements. The next element will be laid out exactly at the same vertical position as the element marked with `overlap` drawing on top of it. Subsequent are then laid out below and can still overlap the element marked with `overlap`. This effect is in place till the next closing brace (or the end of the file).

Thus this keyword is similar to the `parallel` keyword, but it allows direct overlap, not just side-by-side layout.

```
hscale=1.5;
angle=10;
H1: Host 1;
H2: Host 2;
overlap {
    H1->H2: \plSYN;
    H2->H1: \prACK;
};
H2->H1: \prSYN;
H1->H2: \plACK;
```



### 5.13 Chart Options

Chart options are global settings that impact overall chart appearance or set defaults for chart elements. Chart options can be specified at any place in the input file, but typically they are specified before anything else. The syntax is as below.

```
option = value, ... ;
```

The following chart options are defined.

- |               |  |
|---------------|--|
| <b>msc</b>    | This option takes a chart design name as parameter and sets, how the chart will be drawn. It is usually specified as the first thing in the file before any other chart option. However, it can be specified multiple times, in which case its effect takes place downward from the chart option. If not specified then the ‘ <code>plain</code> ’ design is used. Note that this option can be overridden from the command line and also from the Windows GUI. Also note that only full designs can be applied with the ‘ <code>=</code> ’ symbol, partial designs shall use ‘ <code>+=</code> ’. See <a href="#">Section 5.19 [Chart Designs]</a> , page 93 for more on chart designs. |
| <b>hscale</b> | This option takes a number or <code>auto</code> , and specifies the default horizontal distance between entities. The default is 1, so to space entities wider apart, use a larger value. When specifying <code>auto</code> entity positions will be automatically set according to the spacing needs of elements. In this case the <code>pos</code> attribute of entities will be ignored except when influencing the order of the entities. See the end of   |

Section 3.2 [Defining Entities], page 12 for examples. Similar to `msc`, if you specify this attribute multiple times, the last one takes precedence.

#### `numbering`

This option takes `yes` or `no` value, the default is `no`. Any element you define will take the default value of its `number` attribute from this option. See more on numbering in Section 5.9.3 [Numbering], page 70.

#### `compress`

This option takes a boolean value, and defaults to `off`. Any element you define will take the default value of its `compress` attribute from this option. See more on numbering in Section 5.9.4 [Compression and Vertical Spacing], page 73.

#### `vspacing`

Can be set to a number interpreted in pixels or to the string `compress`. Governs how much vertical space is added before each element (can be negative). This option is another form (superset) of the `compress` option; `compress=yes` is equivalent to `vspacing=compress`, whereas `compress=no` is equivalent to `vspacing=0`.

#### `angle`

Specifies the default value for arrow slanting. Its value is measured in degrees, can take values from 0 to 45 degrees and its default value is zero.

#### `indicator`

Similar to the `compress` option above this chart option can be used to influence the default value of the `indicator` attribute for grouped entities and boxes. The simplest way to turn all indicators on or off is to specify this chart option at the beginning of the file.

#### `auto_heading`

Sets the default value for the ‘`auto_heading`’ attribute of ‘`newpage`’ commands. Setting to `yes` will cause all ‘`newpage`;’ commands to create an entity heading on the subsequent page making additional ‘`heading`;’ commands unnecessary. The default is `no`.

#### `classic_parallel_layout`

If set to `yes`, parallel blocks are laid out with an old algorithm, which allows and ignores overlaps between the elements in the different parallel blocks. Defaults to `no`, and is kept only for backwards compatibility.

#### `pedantic`

This option takes a boolean value. It defaults to `no`, but can also be set by the command line or using `Edit|Preferences...` on Windows. When turned on, then all entities must be defined before being used. If an entity name is not recognized in an arrow or box definition an error is generated. However, the implicit definition is accepted. Setting `pedantic` affects only the definitions after it and you can set it multiple times on and off. However it makes little sense.

#### `text.ident`

#### `text.format`

#### `text.color`

#### `text.wrap`

This chart option can be used to set the default text format. It will be the default for all labels. Any styles or attributes specified will overwrite the formatting specified here. Its syntax is the same as that of the `text.*` attributes.

**numbering.pre****numbering.post**

These options specify what shall be prepended and appended to label numbers. Their default value is the empty string and a semicolon followed by a space, respectively. The value of these options are ignored when a label number is inserted due to the ‘\N’ escape sequence. See [Section 5.9.3 \[Numbering\]](#), page 70 for more.

**numbering.format**

Specifies the format of automatic numbering for labels. Can be an arbitrary string (usually quoted) and may also contain formatting escapes. Any occurrence of ‘123’, ‘arabic’, ‘iii’, ‘roman’, ‘abc’, ‘letters’ (or uppercase versions) will be replaced to the actual number in the specified format. The string can contain multiple of the strings above, that will be interpreted as a multi-level numbering format. It is an error to describe more levels than the chart has at the location of the option. In this case an error is printed and the option is not changed. Describing fewer levels will result in Msc-generator omitting the top level numbers from labels. For example, if the numbering is at 2.4.1 and one specifies ‘123.123’ for number format, Msc-generator will display only 4.1. Such truncation, however, will not change the number of levels, merely how the number is displayed.

**numbering.append**

This option can be used to append a new level to numbering. Its syntax is the same as for **numbering.format**. E.g., opening a second level of arabic numbers separated by a colon from the first level can be done by specifying ‘.123’ (use quotation marks). It is possible to add more than levels at once. All added levels start from the value of 1 (or ‘i’ or ‘a’, for roman numbers or letters, respectively).

**background.color****background.gradient**

These are similar to **fill.\*** attributes and specify the background color of the chart. By default the background is transparent. The only exception The only exceptions are PNG images, which cannot have transparency, so the default background color is white. You can change the background color multiple times, each change taking effect at the place where you issue the background chart option. This is useful to split your chart to multiple sections visually. By setting **background.color=none** you can restore transparent background for the rest of the chart. Note that most image formats cannot handle partially semi-transparent backgrounds. For such targets either set the background to a solid color or leave it fully transparent.

**file.info**

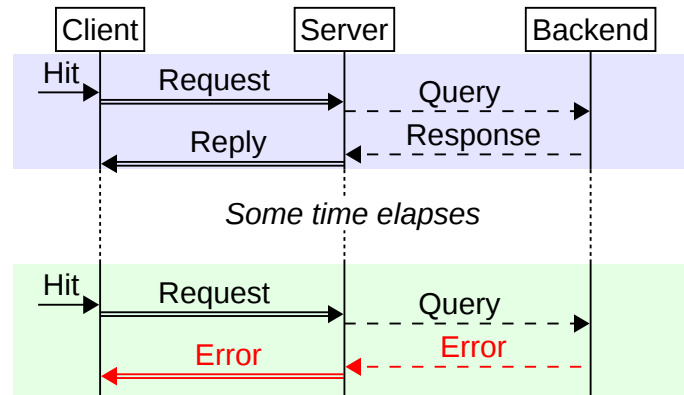
It takes a (quoted) string of human-readable text as value. It is useful to describe what is this file and what it contains. It is used so far only to annotate design libraries, so that if you open an OLE object with a shape not present in your system you can get some info on what file it is from. You can specify this option multiple times their values get concatenated.

`file.url` It takes a quoted URL as value providing a potential place to download this file from.

```

compress=yes;
C: Client;
S: Server;
B: Backend;
background.color="blue+90";
->C: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C<=S: Reply;
background.color=none;
...:\iSome time elapses;
background.color="green+90";
->C: Hit [compress=no];
C=>S: Request;
S>>B: Query;
S<<B: Error [color=red];
C<=S: Error [color=red];

```



```

comment.line.*
comment.fill.*

```

If you have comments on the chart these govern the background of the comments and the attributes of the line separating the comments from the chart. As with background changing them applies downwards from the point of the chart option. See [Section 5.8 \[Notes and Comments\]](#), page 61 for more information on comments.

## 5.14 Multiple Pages

Msc-generator supports multi-page charts. These may be useful when you want to print a long chart. Also, when you only want to show some parts of a chart in a compound document, but want to keep the rest of the text, too. In the latter case just put the parts to show on a different page and show only that page in the compound document.

By default the whole chart is a single page. The chart can be manually broken into multiple pages by inserting `'newpage;'` commands. The chart then can be viewed either as a whole or page by page. You can have as many pages in a document as you want. Adding the `'[auto_heading=yes]'` option to the command will result in displaying an automatic entity heading at the top the page after the page break - but only when the chart is viewed page-by-page. If you want this for all such manually inserted, simply set the `'auto_heading'` chart option to yes.

You can also make Msc-generator to paginate the chart for a given page size. On the command line this is available via the `'-p -a'` options, on Windows, there is a checkbox on the ribbon. You can ask Msc-generator to insert headings to the top of the new pages by specifying `'-ah'` or ticking the `'Auto Headings'` checkbox.

The command-line version of Msc-generator creates as many output files as many pages there are. If there is more than one page, it appends the page number to the filename you specify. Specifying the ‘-p’ option for PDF output allows you to have a single, multi-page output file. In the Windows GUI if you export from Print Preview to PDF, a single multi-page file is created using the page size, orientation, margins and alignment selected in Print Preview.

## 5.15 Free Drawing

Sometimes one wants to add simple drawing elements to a chart, such as circle an arrowhead or comment, dots or other shapes. Msc-generator supports naturally only limited drawing capabilities, but here they are.

### 5.15.1 Spacing

Arbitrary vertical space can be added using the `vspace` command.

```
vspace number [attributes];
vspace: label [attributes];
```

In the first form the vertical space is specified as a number in points. In the second form, the height of the given label will be used. This command also has a specific attribute, called `compressable`, which specifies if the space should be ignored if `compress` is on. It defaults to `no`.

Horizontal spacing between the entities can be controlled either via the `pos` and `relative` entity attributes or can be made fully automatic by specifying `hscale=auto`;; see [Section 5.2.1 \[Entity Positioning\]](#), page 42 and [Section 5.13 \[Chart Options\]](#), page 80.

The `hspace` command is useful in the latter case to force a certain horizontal distance between two (not necessarily neighbouring) entity. The space can be larger than the one specified with `hspace` if the layout requires so, but never smaller.

```
hspace entity-entity number [attributes];
hspace entity-entity: label [attributes];
hspace left comment number [attributes];
hspace right comment number [attributes];
```

The syntax is similar to that of the `vspace` command, both a number or a label can be used to specify the horizontal distance. Before the distance, the two entities need to be specified. Any one can be omitted, in this case the distance is proscribed between the edge of the chart and the entity<sup>18</sup>. Two special versions of the `hspace` command exist to specify the spacing for the comments on the right and left sides.

The `hspace` command can be specified anywhere in the file with the same effect.

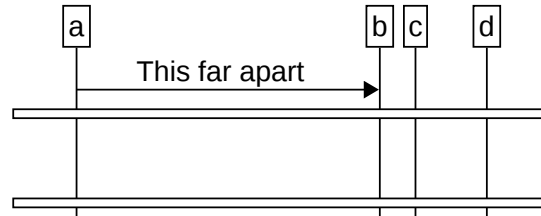
---

<sup>18</sup> Note that the edge will not be the physical edge, merely the invisible line from which arrows connect to when only one entity is specified, such as `a->`; or `->a`;

```

hscale = auto;
a, b, c, d;
a->b:
    This far apart;
hspace a-b:
    This faaaaaaaaar apart;
a--d;
vspace 40;
a--d;
hspace c-d 40;

```



### 5.15.2 Symbols

Currently Msc-generator can draw circles (ellipses), ellipses (three dots) and rectangles (optionally with text) or just plain text. We call these *symbols*.

```
symbol arc|rectangle|...|text marker-marker hpos1 hpos2 [attributes];
```

By specifying either `arc`, `rectangle`, `...` or `text` after the `symbol` keyword one instructs Msc-generator to draw one circle/ellipsis, rectangle, ellipses or just text<sup>19</sup>, respectively.

The vertical position of the symbols can be specified two ways. Either they are *in-line*, which means they occupy space and the layout engine takes them into account when laying out entities above below. In this case symbols will be drawn at the vertical position where they are specified in the file, just like any other element (except verticals). To achieve in-line placement, just omit the markers (and the dash in-between) from the above syntax.

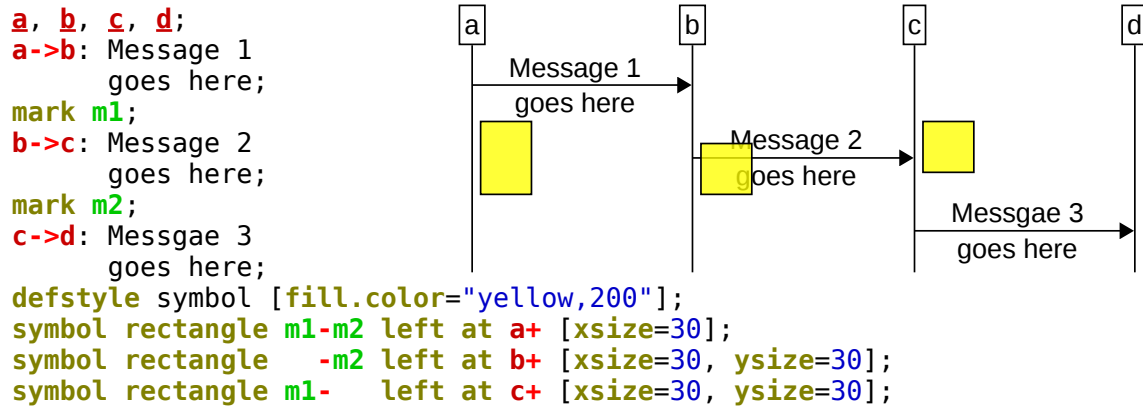
Otherwise it is possible to specify the vertical position where the symbol should appear. This can be done via markers, similar as for verticals, see [Section 5.6 \[Verticals\], page 57](#). In this case however, the layout engine will ignore the symbol and it will be drawn either behind or in front of other elements.

The vertical size of the object can be specified two ways. Either you specify two markers (as above), in which case the symbol will vertically span from one to the other; or you omit one of the markers, in which case the `ysize` attribute specifies the height (in points)<sup>20</sup>. If the dash is in front of the marker, the bottom of the symbol will be aligned with the marker. If the dash is after the marker, then the marker designates the top of the symbol.

In the example below we see three rectangles. One stretches between two markers, the second is bottom aligned, while the third is top aligned.

<sup>19</sup> We have to note that `text` is just syntactic sugar for a `rectangle` with no line or fill. Rectangles can also contain text.

<sup>20</sup> In case of rectangles and text, you can use the natural size of the label you specify as the height of the symbol by omitting the `ysize` attribute.



The horizontal position of the symbol is specified via one or two *horizontal position specifiers*. They specify the horizontal position of either the left or right edge of the symbol or of its center. This is governed by the first keyword

```

left|center|right at entity-entity [number]
left|center|right at entity--
left|center|right at entity-
left|center|right at entity [number]
left|center|right at entity+
left|center|right at entity++

```

Then, after the `at` keyword one specifies either one entity with additional modifiers or two entities. In the former case the horizontal position will be at the middle of the entity's line or somewhat left or right of it depending on the modifiers. In the latter the horizontal position will be between the two entities. Two of the forms can also take a number, which is interpreted as pixels and will shift the position to the right for positive values and to the left for negative values.

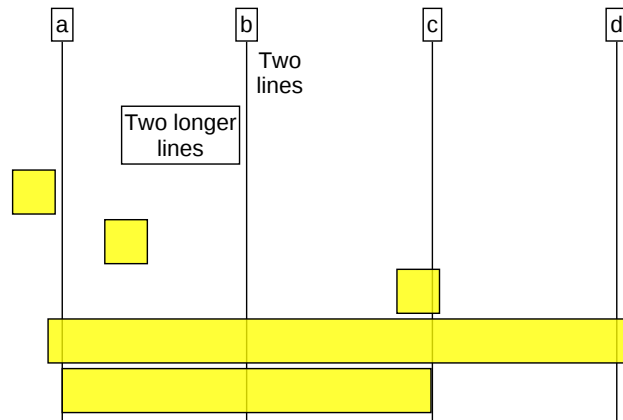
If you specify two such horizontal position specifiers one after the other, they describe both the placement of the symbol and its width. If you specify one, the width of the symbol can be specified using the `xsize` attribute<sup>21</sup>. This may sound a bit complicated, so here is an example with 5 in-line symbols.

```

a, b, c, d;
hspace -a 100; #make room on left side
symbol text left at b+: Two\nlines;
symbol rectangle right at b-: Two longer\nlines;
defstyle symbol [fill.color="yellow,200"];
symbol rectangle right at a- [xsize=30, ysize=30];
symbol rectangle left at a +30 [xsize=30, ysize=30];
symbol rectangle center at c-- [xsize=30, ysize=30];
symbol rectangle left at a-- right at d++ [ysize=30];
symbol rectangle center at b left at a [ysize=30];

```

<sup>21</sup> Similar to height, in case of rectangles and text, you can use the natural size of the label you specify as the width of the symbol by omitting the `xsize` attribute.



Whether the symbol is drawn behind or in front of other elements can be controlled by the `draw_time` attribute. It can take the following values.

#### `before_entity_lines`

Elements with this property will be drawn before the entity lines are laid out in the order as they are specified in the chart description.

#### `after_entity_lines`

Elements with this property will be drawn just after the entity lines are laid out, but before regular elements are drawn.

**default** This is the default, elements with no `draw_time` will be drawn this time in the order as specified in the chart description.

#### `after_default`

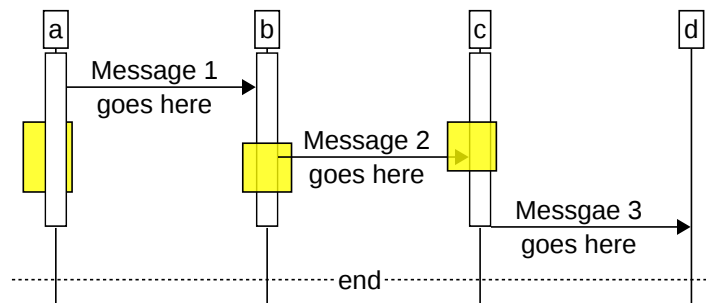
Elements with this property will be drawn last, after all the above elements in the order as they are specified in the chart description.

Note that from v3.3.4 any element can specify the `draw_time` attribute. It will not impact that layout only the drawing order (what is called the *z-order*).

```

a, b, c, d;
vspace 0;
activate a, b, c;
a->b: Message 1
    goes here;
mark m1;
b->c: Message 2
    goes here;
mark m2;
c->d: Messgae 3
    goes here;
deactivate a, b, c;
---: end;
defstyle symbol [fill.color="yellow,200"];
symbol rectangle m1-m2 center at a- [xsize=30, draw_time=before_entity_lines];
symbol rectangle -m2 center at b [xsize=30, ysize=30, draw_time=after_entity_lines];
symbol rectangle m1- center at c- [xsize=30, ysize=30, draw_time=after_default];

```



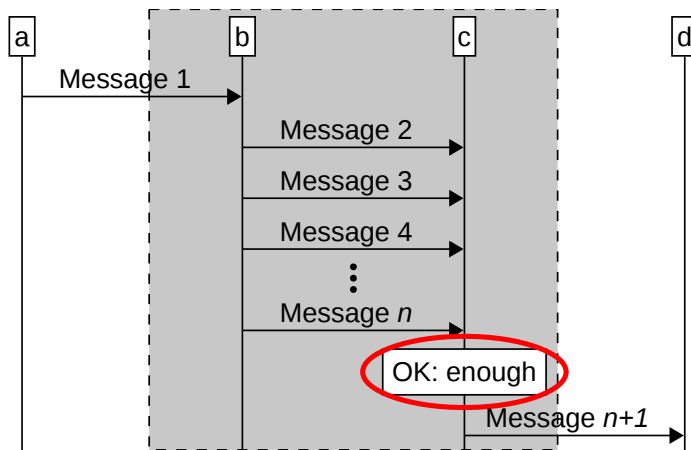
As you can see the first (leftmost) rectangle was drawn below the entity lines, the second (middle) one between the entity lines and the arrows, while the last (rightmost) one was drawn on top of the arrows.



Finally we show a few examples of how symbols may be used.

```
mark top;
a, b, c, d;
symbol rectangle top-bottom left at a-b +10 right at c-d -10
[fill.color=lgray, line.type=dashed, draw_time=before_entity_lines];

a->b: Message 1;
b->c: Message 2;
b->c: Message 3;
b->c: Message 4;
symbol ... center at b-c;
b->c: Message \in;
mark circletop [offset=-5];
box c--c: OK: enough;
mark circlebottom [offset=+5];
symbol arc circletop-circlebottom center at c
[fill.color=none, line.width=3, line.color=red, xsize=120];
c->d: Message \in+1;
mark bottom;
```



### 5.15.3 Inline text

Sometimes one just wants to add some text to the diagram and in this case the `symbol text` syntax may be a bit heavy and difficult to do. As an easier way to do that Msc-generator offers the `text at` command.

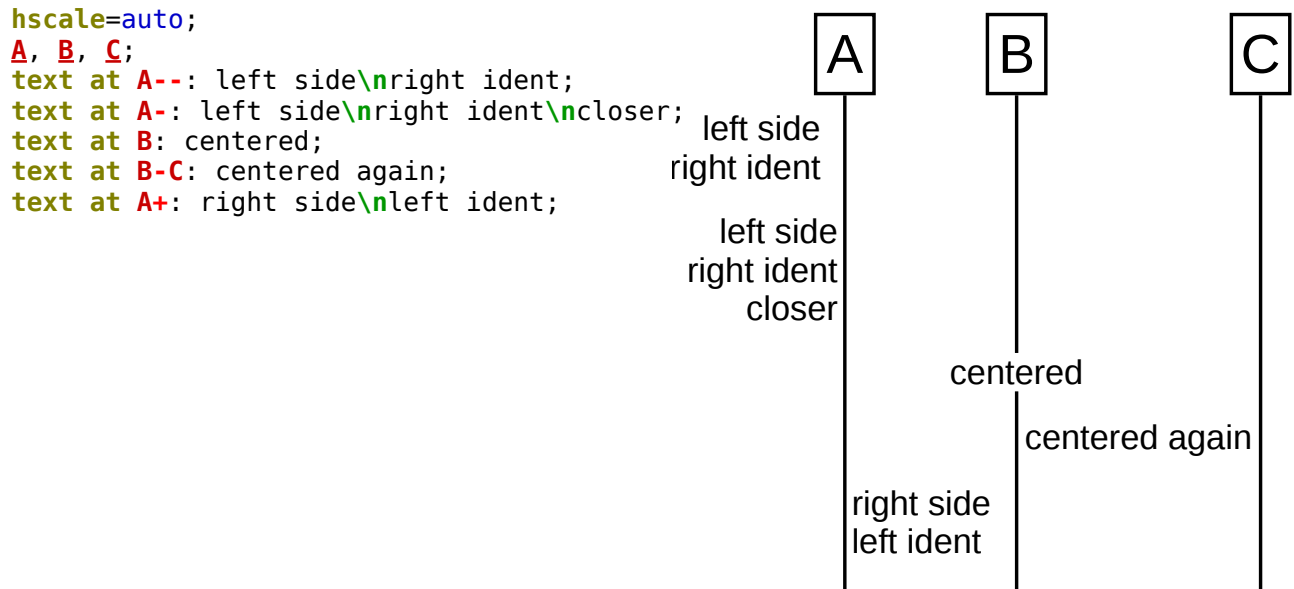
```
text at pos [attributes]: label;
```

This draws just text (you must specify a label) at the vertical position the command is written. You can use simple horizontal position specifiers, like below to place the text centered in-between two entities; left of an entity, centered around an entity or right of an entity. You can optionally specify a number, which will be interpreted as a pixel offset to the right (negative value to the left.)

```
entity-entity [number]
entity-- [number]
```

```
entity- [number]
entity [number]
entity+ [number]
entity++ [number]
```

You can influence the default appearance via the `text` built-in style.



## 5.16 Commands

Besides entity definitions, arrows, dividers, boxes, parallel block definitions and options, msc-generator also has a few commands.

- |                |   |
|----------------|---|
| <b>nudge</b>   | This command inserts a small vertical space useful to misaligning two arrows in parallel blocks, see <a href="#">Section 5.12 [Parallel Blocks]</a> , page 79.  |
| <b>hspace</b>  | This command forces horizontal distance between two (not necessarily neighbouring) entity. See <a href="#">Section 5.15.1 [Spacing]</a> , page 84.  |
| <b>vspace</b>  | This command inserts an arbitrary size vertical space, see <a href="#">Section 5.15.1 [Spacing]</a> , page 84.  |
| <b>newpage</b> | This command starts a new page, see <a href="#">Section 5.14 [Multiple Pages]</a> , page 83.  |
| <b>heading</b> | This command displays all entity headings that are currently turned on. It is useful especially after a newpage command. Note that if there are any immediately preceding or following entity definition commands before or after <b>heading</b> , only one copy of the entity headings is drawn. |
| <b>show</b>    |   |
| <b>hide</b>    | Prepending these in front of an entity definition (or later mention) will set the 'show' attribute of those entities (there can be a comma separated list) to yes or no, respectively.  |

**activate****deactivate**

Prepending these in front of an entity definition (or later mention) will set the ‘active’ attribute of those entities (there can be a comma separated list) to yes or no, respectively. In addition, when these commands are used to activate or deactivate certain entities immediately after an arrow, the activation or deactivation will take place at the tip of the arrow and not after it. This is to indicate that the activation or deactivation happened as a result of the arrow. This effect is not applied if an entity is activated or deactivated by setting its **active** attribute.

**mark**

This command creates a *marker* by storing the vertical position of this command. Symbols, verticals and notes can then refer to this location. See [Section 5.6 \[Verticals\]](#), [page 57](#) for more information.

**note****comment****endnote**

These comments are useful to annotate the chart, see [Section 5.8 \[Notes and Comments\]](#), [page 61](#).

**symbol****text**

These commands can be used to draw arbitrary graphics to the chart, see [Section 5.15 \[Free Drawing\]](#), [page 84](#).

**defcolor**

This command is used to define or re-define color names, see [Section 5.10 \[Specifying Colors\]](#), [page 74](#).

**defstyle**

This command is used to define or re-define styles, see [Section 5.18 \[Defining Styles\]](#), [page 91](#).

**defdesign**

This command is used to define new designs, see [Section 5.19 \[Chart Designs\]](#), [page 93](#).

**defshape**

This command is used to define new shapes, see [Section 5.20 \[Defining Shapes\]](#), [page 94](#).

## 5.17 Scoping

Each time an opening brace is put into the file, a new *scope* begins. Scopes behave similar as in programming languages, meaning that any color name or style definitions take their effect only within the scope, up to the closing brace. Thus if you redefine a style just after an opening brace, the style returns to its original definition after the closing brace. (See [Section 5.18 \[Defining Styles\]](#), [page 91](#).)

Scoping also applies to the **numbering** (including **pre**, **post**, **format** and **append**), **compress**, **vspacing**, **indicator**, **angle** and **text.\*** chart options. Any changes to these take effect only until the next closing brace. Scoping explicitly does not apply to **background.\*** and **comment.\*** options. Those take effect until the next such option or all the way to the bottom of the chart.

You can nest scopes arbitrarily deep and can also use the parallel block syntax with a single block to manually open a new scope, such as below.

```

...numbering is off here...
{
    #number only in this scope
    numbering=yes;
    ...various elements with numbers...
};
...other elements with no numbers...

```

## 5.18 Defining Styles

It is possible to define a group of attributes as a style and later apply them collectively. Styles are useful if you have e.g., two types of signals on a diagrams and want to visually distinguish between them. Then, instead of re-typing all the required attributes for each arrow, simply define two styles for them. Also, if you later want to change the appearance of these arrows, you just need to change the style and not every arrow individually.

Styles can be defined using the `defstyle` command, as below.

```
defstyle stylename, ... [ attribute=value | style, ... ], ... ;
```

First you list the name of the style(s) to define then the attributes and their intended values. Similar to color names, style names are case-sensitive and can only contain letters, numbers, underscores and dots, but can not start with a number or a dot and can not end with a dot. You do not have to specify all possible attributes, just those you want to modify with the style. The rest of the attributes will remain unspecified. When you apply the style to an element, attributes of the element that are unspecified in the style are left unchanged.

Any of the attributes listed in [Section 5.9 \[Common Attributes\]](#), [page 63](#) can be added to a style. You can also enlist styles among the attributes. In this case the newly defined style inherits all the attributes specified in that style. If you apply a style to an element, those attributes of the style, which not applicable to that particular element type are simply ignored. For example, applying a style including `fill.color` to an arrow will silently ignore the value of the `fill.color` attribute.

The same syntax above can be used to extend and modify styles. You can add new attributes to an existing style or modify existing attributes. This is when listing multiple styles comes in handy. You can set attributes to the same value in multiple styles in a single command.

It is also possible to unset an attribute by specifying the attribute name, followed by the equal sign, but no value.

### 5.18.1 Default Styles

There are a number of default, built-in styles that govern the default appearance of elements. By modifying these you can impact, e.g., all the arrows in a chart. This is how chart designs operate: by modifying the built-in styles.

First there is a built-in style for each element: `arrow`, `box`, `emptybox`, `divider`, `blockarrow`, `pipe entity`, `entitygroup`, `symbol`, `indicator`<sup>22</sup>, `title`, `subtitle`, `note`,

<sup>22</sup> The style `indicator` determines the appearance of the small symbols that indicate elements hidden due to a collapsed box or entity group.

`comment`, `endnote`, `vertical`<sup>23</sup>, `vertical_brace`, `vertical_bracket`, `vertical_range`, `vertical_pointer`, `symbol`<sup>24</sup> and `text`<sup>25</sup>.

There are also predefined styles for grouped entities and boxes for when they are collapsed: `entitygroup_collapsed`, `box_collapsed` and `box_collapsed_arrow`, the latter is used when a box is collapsed to a bidirectional arrow.

If you want to change a set of attributes for multiple elements (such as both for arrows and dividers) simply list these separated by commas before the attributes.

```
defstyle arrow, divider [line.width=2];
```

It will apply to both.

Then there are further styles defined for each arrow, box and divider element, called *refinement styles*. These are partial (by default specify only line type) and will be applied to the element after the main style for the element (listed above).

- for arrows: ‘->’, ‘=>’, ‘>’ and ‘>>’<sup>26</sup>.
- for block arrows: ‘block->’, ‘block=>’, ‘block>’ and ‘block>>’.
- for boxes: ‘--’, ‘==’, ‘++’ and ‘..’
- for pipes: ‘pipe--’, ‘pipe==’, ‘pipe++’ and ‘pipe..’
- for dividers: ‘---’ and ‘...’
- for verticals: `vertical->`, `vertical>`, `vertical>>`, `vertical=`, `vertical--`, `vertical++`, `vertical..` and `vertical==`.

Redefining enables you to quickly define, e.g., various arrow styles and use the various symbols as shorthand for these. Usually style names containing non-letter characters have to be quoted, but for the above styles the parser is expected to recognize them without quotation. So both below are valid.

```
defstyle "->" [arrow.size=tiny];
defstyle -> [arrow.size=tiny];
```

Note that re-defining an existing style do not erase the attributes previously set in the style. Only the new attribute definition is added - changing the value of the attribute if already set in the style. This the example above keeps the `line.type=solid` setting in ‘->’ style.

Finally there are two more pre-defined styles: **strong** and **weak**. By adding these to any element you will get a more and less emphasized look, respectively. The benefit of these compared to making elements stronger or weaker by yourself is that they are defined in all chart designs in a visually appropriate manner. Thus you do not need to change anything when changing chart design just keep using them unaltered.

<sup>23</sup> referring to vertical boxes

<sup>24</sup> referring to all symbols

<sup>25</sup> referring to `text at` commands

<sup>26</sup> These are also applied to bi-directional arrows and arrows pointing from an entity back to itself. Thus there is no separate ‘<->’ style, for example.

As a related comment we note that chart designs modify all the above styles and the default value for the `hscale`, `compress`, `vspacing`, `numbering`, `indicator`, `angle` and `text` chart options, too.

Thus, in summary the actual attributes of an element are set using the following logic.

1. If you specify an attribute directly at the element (perhaps via applying a style), the specified value is used<sup>27</sup>.
2. Otherwise, if the attribute is set in the refinement style (at the point and in the scope of where the element is defined), the value there is used. (Usually only line styles are set, so this is why you can use e.g., `=>` to make an arrow double-lined.)
3. Otherwise, if the attribute is set in the default style of the element, the value there is used.
4. Otherwise, the value of the applicable chart option is used, such as `text.*`, `compress`, `vspace`, `indicator`, `numbering`, `auto_heading` and `angle`. In order for these chart options to be effective default styles usually have no value specified for these attributes. You can set these attributes in styles, e.g., to set font type for empty boxes, which will take precedence over chart options.

## 5.19 Chart Designs

A chart design is a collection of color and style definitions, and the value of the `hscale`, `numbering`, `compress`, `vspacing`, `text`, `background` and `comment` attributes. For numbering you can turn it on or off and specify the format of the top level number - but you cannot specify multiple levels.

There are *full designs* and *partial designs*. A full design contains a value for all the chart options, default colors and styles. A partial design contains values only for some of these. E.g., the `thick_lines` design is a partial one - it merely makes all lines of width 2 in all the default styles, but leaves color, line type, fill or any other attribute or chart option unchanged.

To apply a full style, use the `msc = <style_name>` chart option. To apply a partial style use the `msc += <style_name>` chart option.

Currently the following partial designs ship with Msc-generator: `hcn`, `thick_lines`, `all_blue`, `feng_shui_notes`. The first one simply sets `hscale` to auto and turns on compression and numbering. The second one makes lines of all default styles of width 2. The third makes the color of lines in all default styles blue. The last makes notes rounded and red on yellow background. Try them.

You can define or re-define chart designs by using the syntax below.

```
defdesign designname {
    [ msc=parent design ]
    [ msc+=partial design ]
    options, ...
    color definitions, ...
    style definitions, ...
}
```

---

<sup>27</sup> If you specify the attribute several times, the last one is used.

First you can name an existing full design to inherit from using the ‘`msc=`’ option. If specified the design will become a full design, too. Thus in each such design definition the styles mentioned in [Section 5.18.1 \[Default Styles\]](#), [page 91](#) are always present and fully specified. If omitted, the style will become a partial style. Then you can specify optional multiple ‘`msc+=`’ options to bring in partial designs. Finally, you can define colors, styles in any order and/or set one or more of the attributes mentioned above.

It is possible to add your design definitions to a file having the `.signalling` extension and making them available in all charts for use. See [Section 4.1 \[Design Library\]](#), [page 32](#).

## 5.20 Defining Shapes

This section describes how you can define new shapes that can be used in entity headings. To define a new shape use the `defshape` command.

```
defshape shape_name {
    defining lines;
    ...
};
```

The *shape\_name* is the name of the shape to define. You can use letters, numbers, underscores and dots; and you must start with a letter. Each *defining line* shall start with a capital letter, one of the list below. After this letter comes a space delimited list of arguments, terminated by a semicolon. Comments can start with a hashmark ‘`#`’ as usual and empty lines are also permitted.

- S** Starts a new *section* in the new file. Each shape may contain three optional sections. You shall include the number of the section after the **S** separated by a space. Sections contain paths, which describe the shape. Anything that comes after this line will belong to this section all the way until the next **S** or the end of the shape definition.
- 0 Section 0 specifies the background of the shape. This shall be a (set of) closed path(s), which will be filled by Msc-generator using the value of `fill.color` attribute.
  - 1 Section 1 specifies the foreground of the shape. This shall be a (set of) closed path(s), which will be filled by Msc-generator using the value of the `line.color` attribute.
  - 2 Section 2 specifies an alternative, additional way to specify the foreground. It shall be a set of potentially open paths, which will be drawn (stroked) using the value of the `line.color` attribute. Contrary to section 1, other line attributes will also be applied, such as `line.width` and `line.type`. You can specify both Section 1 and 2, in this case both will be drawn.
- M** This command is used within the specification of a path, and moves the (imaginary) cursor to a given point. Two space-separated numbers after the **M** specifies the x and y coordinates. Note that you can use floating-point numbers and any scaling or position - Msc-generator will normalize the size of each shape based on their height.

- L** This command is used within the specification of a path, and draws a straight line from the current position of the (imaginary) cursor to a given point. Two space-separated numbers after the **L** specifies the x and y coordinates.
- C** This command is used within the specification of a path, and draws a curved line from the current position of the (imaginary) cursor to a given point. Six space-separated numbers after the **C** specifies the x and y coordinates of the target point and two control points, respectively. The curved line is computed as a cubic bezier curve.
- E** This command is used within the specification of a path, and closes a path. It takes no arguments.
- T** This optional item specifies where the entity label shall be drawn. It takes two points (four numbers) as arguments specifying the opposing corners of a rectangle. If this line is omitted, the label will be displayed below the shape. This line can be specified before, inside, between or after sections.
- H** This optional item specifies which portion of the shape shall be shown in the hints popup box on Windows in the internal editor (see [Section 4.5 \[Typing Hints and Autocompletion\]](#), page 33). This also takes 4 numbers specifying a rectangle as above. If omitted the whole shape is shown (on a miniature scale). This line can be specified before, inside, between or after sections.

See [Section 4.1 \[Design Library\]](#), page 32 to see where to find and put your own files defining Shapes.