Msc-generator

A tool to draw various charts from text description
(version 5.4.0, 19 October 2016)

Zoltan R. Turanyi

This manual is for Msc-generator (version 5.4.0, 19 October 2016), a tool to draw various 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 chart descriptions and produces graphical output in a variety of file formats, or as a Windows OLE object, which can be embedded in documents, such as Word or PowerPoint. It currently supports two kind of charts: Message Sequence Charts (MSCs, this is where the name of the tool comes from) and general graphs.

Message Sequence Charts 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.

The signalling chart part of msc-generator is heavily extended and completely rewritten version of the 0.08 version of Michael C McTernan's mscgen. The original tool was more geared towards describing interprocess communication, this version is more geared towards networking. Msc-generator has a number of enhancements compared to mscgen. The command-line syntax of Msc-generator is compatible to that of mscgen, so any tool integrated with mscgen (such as Doxygen) can also be used with Msc-generator. Since version 4.5 Msc-generator also contains an *mscgen compatibility mode*, which aims to interpret mscgen chart descriptions in a fully backwards compatible manner. See Section 7.15 [Mscgen Backwards Compatibility], page 120.

The graph part of msc-generator uses the graphviz libraries to lay out graphs. It uses the DOT language to describe charts, with a few extensions.

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

1 What's new in Msc-generator 5.4

The improvements added since version 5.3 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.

- signalling,graph: Added support for *procedures*. These are sections of chart text that can be later replayed. Procedures can be defined like

  ```
  defproc <name>($<param_name>[=<default_value], ...) {
      ...actual chart text...
  };
  ```

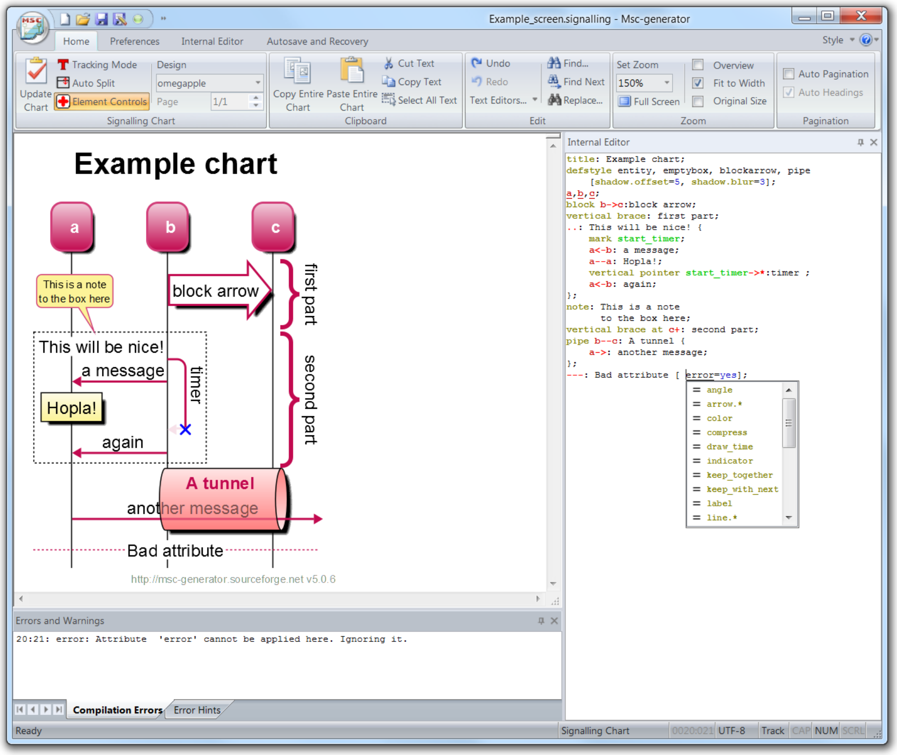
  Parameter names start with a dollar sign and can be used anywhere where a string can be used (attribute values and names, entity and node names, etc.) You can use the `\Q($<param_name>)` text escape to quote the value of a parameter into a label. Procedures can be replayed like `replay <name>(<value>,...);`. You can omit parameters that have default values. (Feature request #5, see Section 6.9 [Procedures], page 67)
- signalling,graph: Support for conditional statements inside procedures, like

  ```
  if $note<>"" then note [label=$note];
  ```
- signalling,graph: Added support for file inclusion using the `include` command, see Section 6.10 [File Inclusion], page 70.
- backend: fixed indentation errors for charts with international characters.
- GUI: Error and warning messages are now colored. F8 and Shift+F8 now jumps between the main messages, skipping additional information (in gray). To make small steps, use Ctrl+F8 and Shift+Ctrl+F8. (You may need to reset keyboard shortcuts in the Customize Quick Access Toolbar button for this to work.)
- GUI: Now requires Win7, but provides compilation progress on the taskbar.
- GUI: Disabled color syntax parsing if internal editor is not visible.
- GUI: Added collapse/expand all buttons for graphs. (Feature request #6)

2 Getting started

Msc-generator is a program that parses textual chart descriptions and produces graphical output in a variety of file formats, or as a Windows OLE object, which can be embedded in documents, such as Word or PowerPoint. It currently supports two kind of charts: Message Sequence Charts (MSCs, this is where the name of the tool comes from) and general graphs.

On Windows Msc-generator is installing as a regular application. You can start it directly, by clicking on a file with .signalling or .graph extension; or by opening an chart embedded in a document (usually by double-clicking it).



```
title: Example chart;
defstyle entity, emptybox, blockarrow, pipe
    [shadow.offset=5, shadow.blur=3];
a,b,c;
block b->c:block arrow;
vertical brace: first part;
..: This will be nice! {
    mark start_timer;
    a<-b: a message;
    a--a: Hopla!;
    vertical pointer start_timer->*:timer ;
    a<-b: again;
};
note: This is a note
      to the box here;
vertical brace at c+: second part;
pipe b--c: A tunnel {
    a->: another message;
};
---: Bad attribute [ error=yes];
```

```
20:21: error: Attribute  'error' cannot be applied here. Ignoring it.
```

The Msc-generator window has the usual elements of a Windows application: menu bar, a ribbon and a status bar. We will briefy discuss these here and give a more detailed description in Chapter 5 [Usage Reference], page 37.

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 edit the chart description there. When you are ready, press the '`Update Chart`' button on the very left of the ribbon (or F2 on the keyboard) and the visual view of the chart will get updated. Any error or warning messages will show up in the error 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.

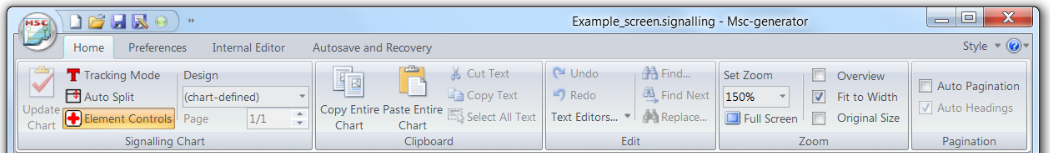
You can also perform undo or redo from the Edit pane of the ribbon or by pressing Ctrl+Z or Ctrl+Y. Similar search and replace operations for the text editor can also be accessed from the Edit pane.

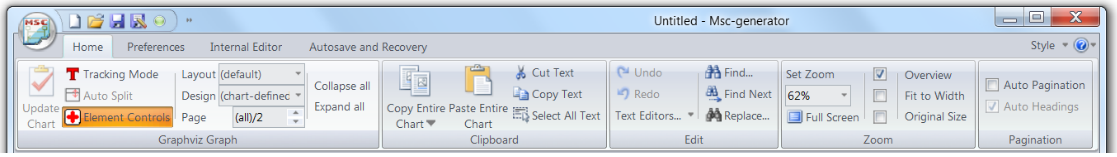
Finally, there is a separate button in the Edit pane to start and stop the internal or an external text editor (see Section 5.3 [External Editor], page 38). The latter is useful in case you prefer to use your own editor.

2.2 The Ribbon

You can find two additional panes on the on the Home category of the ribbon. The Zoom pane enables you to set various zoom options (see Section 2.3 [Zooming], page 5) or to switch to full screen viewing mode.

The first pane always shows the type of chart currently opened (such as Signalling Chart or Graphviz Graph). The buttons in the second column of this 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 6) on or off; or to enable the showing of collapse/expand controls for entity groups and boxes (Section 2.6 [Collapsing and Expanding], page 6)[1].





[1] Some of these buttons may not be available for all chart types. E.g., graphs have no heading and thus does not support AutoSplit.

The third column has additional controls to govern chart appearance. The exact controls depend on the chart type. For graphs, the layout algorithm used can be selected here. For graphs and signalling charts there is a design and page 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 6.7 [Chart Designs], page 65 for more info on chart designs. With the page selectors, you can navigate around in multi-page charts. If '`all`' is selected then pagination is ignored and the whole chart is shown. (See Section 7.12 [Multiple Pages], page 113 for more info on pagination commands for signalling charts. For graphs each graph defined in sequence ends up in a separate page.) Finally, for graphs using the '`Collapse all`' and '`Expand all`' buttons you can collapse or expand all cluster subgraphs with a single click.

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 select an appropriate zoom factor by clicking 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%.[2]

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 this mode, as well. A small toolbar enables you to flip pages, return to the all pages view or to toggle Auto Split (if applicable to the chart, see below). You can exit full screen mode by pressing Escape.

2.4 Tracking Mode

If you click an arrow, entity, graph node 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. The same way, when pressing F4 in the internal editor, if the cursor is inside a chart element, the element is briefly flashed.

If you double-click the chart (try the background) you enter Tracking Mode, where you can select and highlight multiple chart elements at the same time. Visual elements are faintly outlined just by hoovering above them. When actually clicking on them (or pressing F4 when the internal editor cursor is inside them) will make them actually highlighted. Then you can move on to highight more. You can also remove highlighting from an element by clicking it again or by pressing F4 again.

[2] Msc-generator applies a maximum zoom factor for the *Overview* and the *Fit to width* modes, in order to avoid a small chart being enormously magified. This defaults to 150%. On very large screens this may prove to be a tool small factor - thus it is possible to increase this value on the *Preferences* category of the ribbon using the *Max Auto Zoom* edit box.

If you click the chart background or press Escape, all highlighting is removed. Pressing Escape again will make you exit tracking mode. You can enter tracking mode also by the '`Tracking Mode`' button on the Chart pane or by pressing Ctrl+T.

Tracking mode is useful when you are discussing a chart (e.g., in a shared screen session) and you want to talk about various elements.

2.5 Auto Split

When working with a large signalling 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.

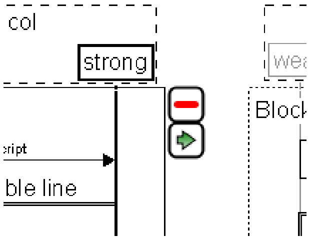
Note that it is possible to define charts where there is no meaningful row of entity headings at the top. In such cases, Msc-generator will get confused and Auto Split is of no use. In case of multi-page charts, Auto Split will show automatic headers only (Instead of '`newpage; heading;`' use rather '`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, but is not available for all charts (e.g., such as graphs).

2.6 Collapsing and Expanding

Msc-generator allows you to collapse boxes, entity groups and cluster subgraphs. This way you can show only a simplified view of the procedure or graph 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 19).

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

For signalling charts 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[3]. Later you can edit the chart by double clicking the chart in the document[4].

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, chart design and layout 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. It can read only signalling charts on Linux, since there it is easy to get native graphviz support.

To use Msc-generator to generate a signalling chart from a text file (containing a singalling chart description in the appropriate language) simply type

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

To use Msc-generator to generate a graph from a text file (containing a graph description in the DOT language) simply type

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

Both of these 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.

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] Make sure you paste the chart using ‘`Paste Special...`’ as an ‘`Msc-generator Chart Object`’.

[4] In place editing is no longer supported from version 3.4.1.

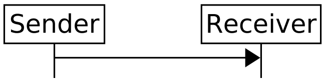
3 Signalling Chart Language Tutorial

In this chapter we give a step-by-step introduction into the language of Msc-generator for signalling charts. At the end you will master most of the language to create charts. Further details (mostly on controlling appearance) are provided in Chapter 7 [Signalling Chart Language Reference], page 71.

3.1 Defining Arrows

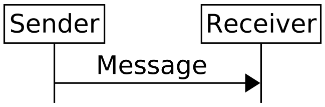
Message sequence charts consits of *entities* and *messages*. The simplest file consists of a single message between two entities: a '`Sender`' and a '`Receiver`'.

```
Sender->Receiver;
```



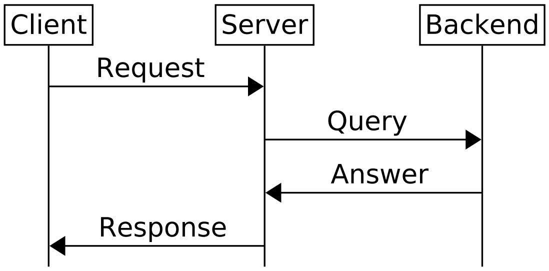
The message may have a label, as well.

```
Sender->Receiver: Message;
```



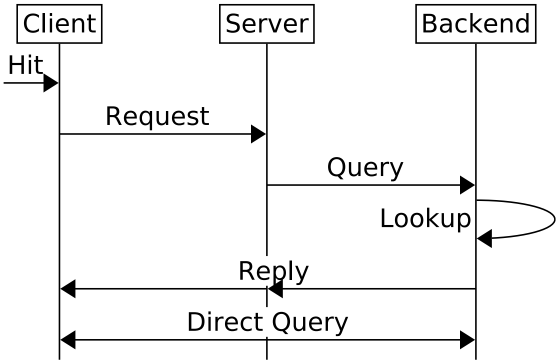
A more complicated procedure would be to request some information from a server, which, in turn, queries a backend. Note that everything in a line after a '`#`' is treated as a comment and is ignored by Msc-generator.

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



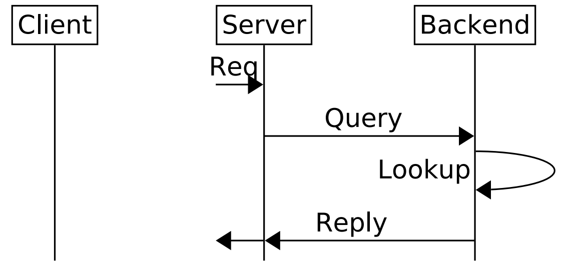
Arrows can take various forms, for example they can be bi-directional or can span multiple entities. They can also start and end at the same entity and can come from or go to "outside"

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



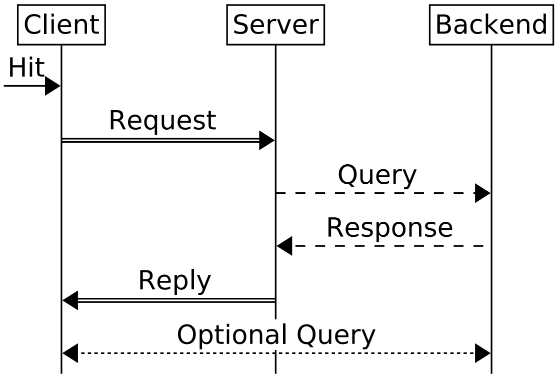
Sometimes one wants to indicate that a message came from an entity not shown, but not at the far left or right. In this case use the pipe symbol '`|`', like `a->|;`.

```
Client;
|->Server: Req;
Server->Backend: Query;
Backend->Backend: Lookup;
|<-Server<-Backend: Reply;
```



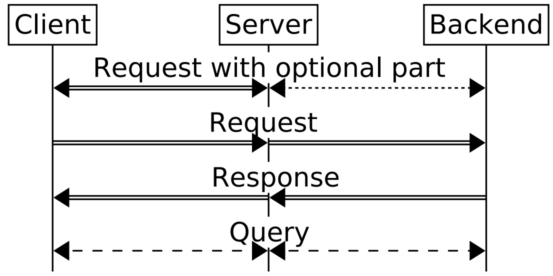
It is also possible to make use of various arrow types, such dotted, dashed and double line. To achieve this the '->' symbol need to be replaced with '>', '>>' and '=>', respectively.

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



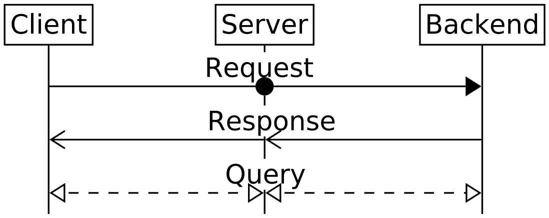
It is also possible to use different line styles for different segments of an arrow - but all must be of the same direction. (That is, it is not possible to write 'a->b<-c', for example.) In addition, for multi-segment arrows the dash '-' symbol can be used in the second and following segments, as a 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;
```



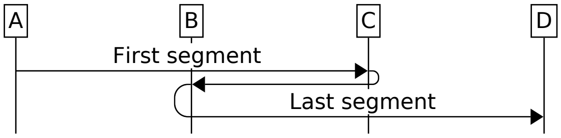
It is possible to change the type of the arrowhead. The arrowhead type is an *attribute* of the arrow. Attributes can be specified between square brackets before or after the label, as shown below. A variety of arrow-head types are available, for a full list of arrow attributes and arrowhead types See Section 7.3 [Specifying Arrows], page 75.

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



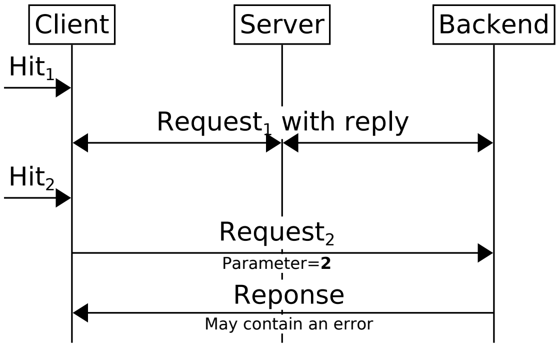
Msc-generator generates an error if the order of entities in a multi-segment arrow does not follow the order of the entities (either left-to-right or back). However, sometimes it is important to show such a message zig-zagging among the entities as one message. This is possible by *joining* arrows. Note that the '`join`' keyword can do more, see Section 7.9.3 [Joining Arrows and Boxes], page 101.

```
A, B, C, D;
A->C: First segment;
join C->B;
join B->D: Last segment;
```



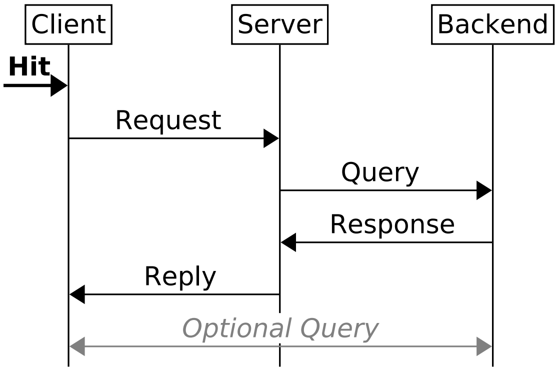
Often the message has not only a name, but additional parameters, that need to be displayed. The label of the arrows can be made multi-line and one can apply font sizes and formatting, as well. This is achieved by inserting formatting characters into the label text. Each formating character begins with 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;
```



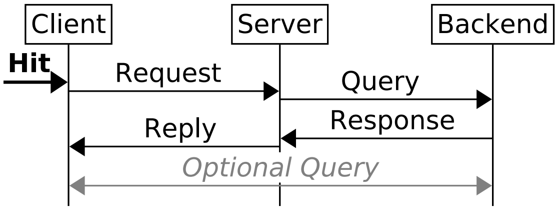
Arrows can further be differentiated by applying *styles* to them. Styles are packages of attributes with a name. They can be specified in square brackets like an attribute that takes no value. Msc-generator has two pre-defined styles '`weak`' and '`strong`', that exits in all chart designs[1]. They 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 thicher lines with bold text, respectively[2].

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



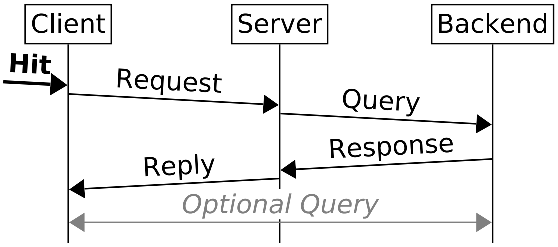
Msc-generator places arrows one-by-one below each other. In case of many arrows, this may result in a lot of vertical space wasted. To reduce the size of the resulting diagram, a *chart option* can be specified, which compresses the diagram, where possible. You can read more on chart options, see Section 7.10.4 [Compression and Vertical Spacing], page 108.

```
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];
```

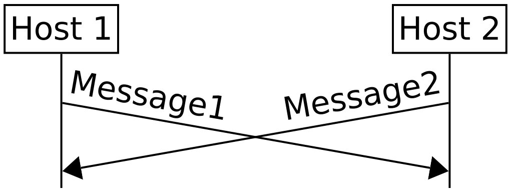


[1] You can define your own styles, as well, see Section 6.6 [Defining Styles], page 64.

[2] For more on chart deisgns Section 6.7 [Chart Designs], page 65.

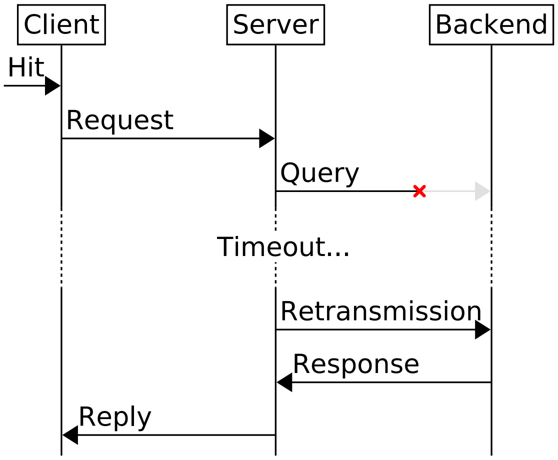
Normally, Msc-generator attempts to avoid overlaps between elements by placing them one below (or sometimes besides) each other. If you want to show messages crossing each other, you need overlapping arrows. For this, you can use the `overlap` keyword. Arrows marked such are allowed to be overlapped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

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

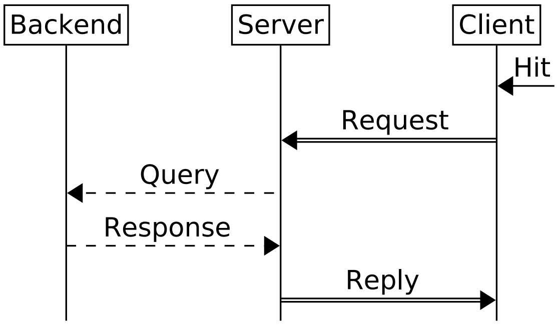


3.2 Defining Entities

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`'.

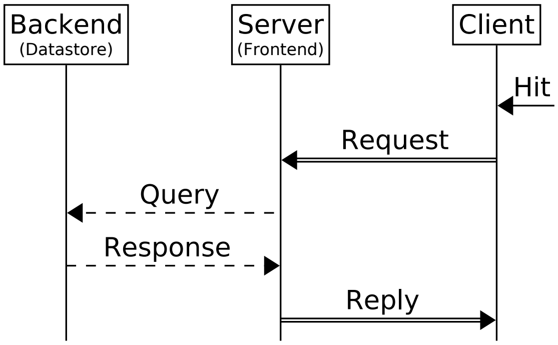
Often one wants to control, in which order entities appear on the chart. This is possible, by listing the entities before actual use. On the example below, the order of the enties are reversed. Note that we have reversed the first arrow to arrive to the '`Client`' from the right.

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



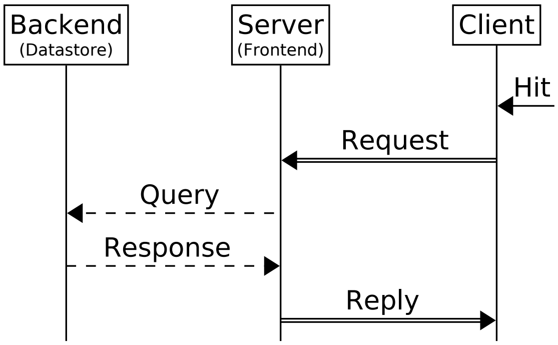
Often the name of the entity need to be multi-line or need to contain formatting characters, or is just too long to type many times. You can overcome this problem by specifying a label for entities. The name of the entity then will be used in the chart description, but on the chart the label of the entity 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\-(Datastore)"],
S [label="Server\n\-(Frontend)"],
C [label="Client"];

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



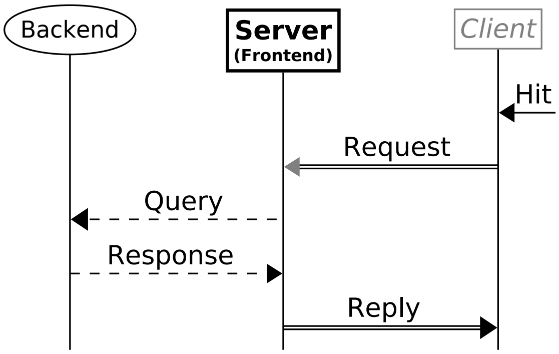
You can also use the colon-notation to specify entity labels, similar to arrows. The above example can thus be written as below. Note that the entity definitions are now terminated by a semicolon – commas would be treated as part of the label.

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

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



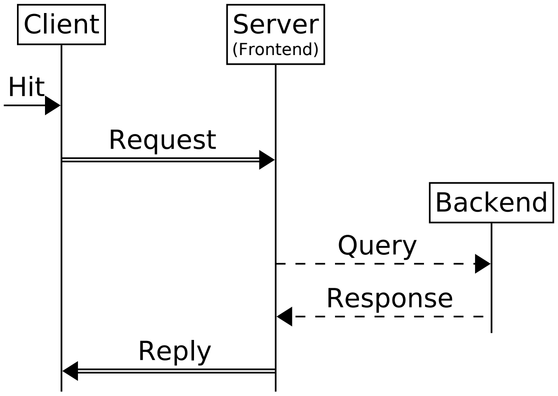
Entities can also be specified as ‘`weak`’ or ‘`strong`’, by applying these styles the same way as for arrows. You can also assign various shapes to the entity headings via the `shape` attribute.

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

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



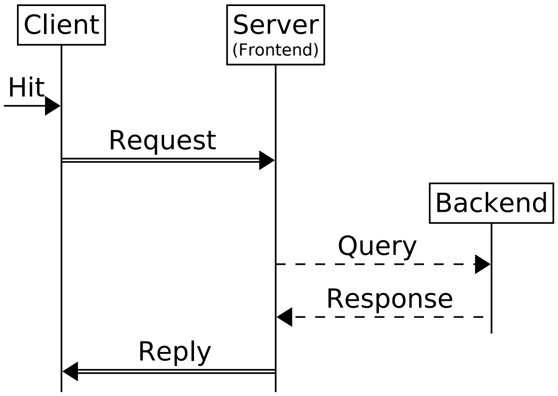
Entities can be turned on and off at certain points in the chart. An entity that is turned off, will not have its vertical line displayed. This is useful if the chart has many entities, but one is involved only in a small part of the process. An entity can be turned off by typing `hide` followed by the 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;
```



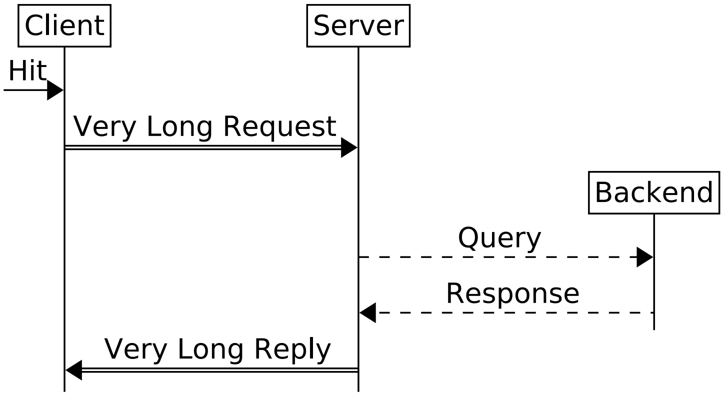
Not showing an entity from the beginning of the chart can also be achieved by simply defining the entity later. Note that this is different from simply starting to use an entity later. When you start using an entity without explicitly defining it first, it will appear at the top of the chart, not only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



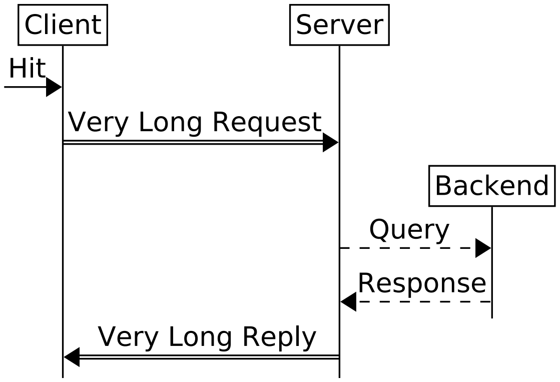
Sometimes the vertical space between entities is just not enough to display a longer label for an arrow. In this case use the ‘`hscale`’ chart option to increase the horizontal spacing. It can be set to a numerical value, 1 being the default.

```
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;
```



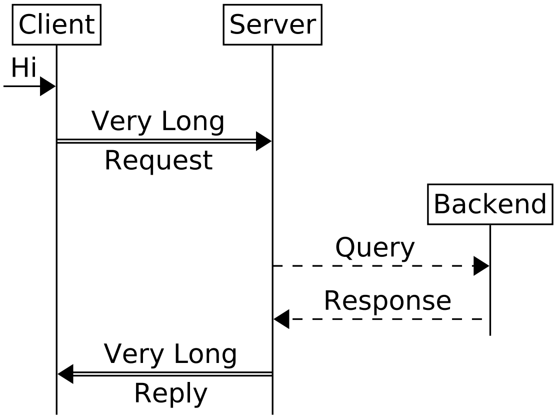
Or you can simply set it to ‘`auto`’, which creates variable spacing, just as much as is needed.

```
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;
```



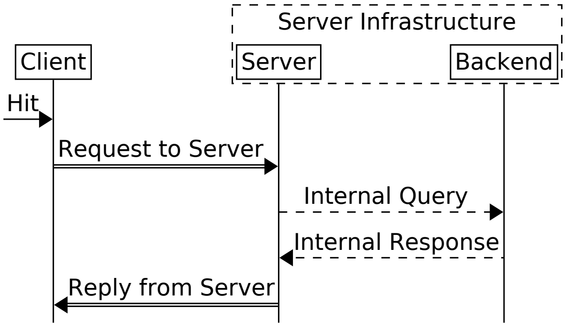
Alternatively, you can instruct Msc-generator to apply word wrapping to the labels of arrows, to fit into the available space, by setting the ‘`text.wrap`’ chart option to ‘`yes`’.

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



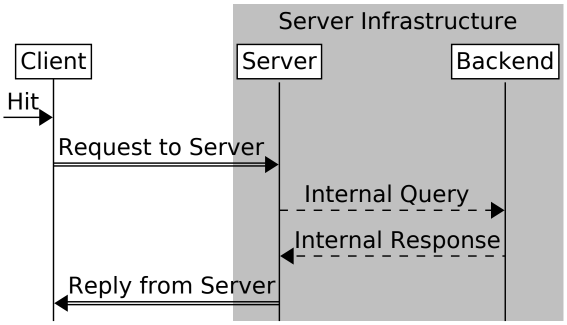
It is possible to define entity groups, to indicate logical relations between various entities. Use curly braces (‘`{`’ and ‘`}`’) after an entity definition (after any potential label and attributes).

```
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;
```



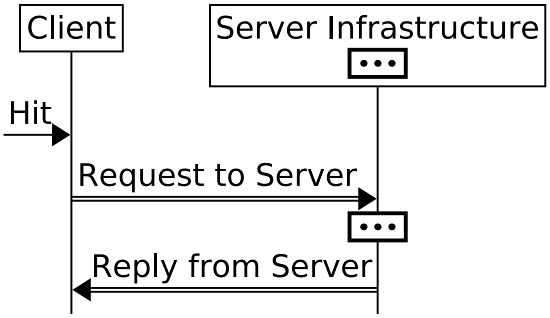
Instead of a group heading, you can also shade the background behind entities in an entity group.

```
hscale=auto;
C: Client;
SI: Server Infrastructure
  [large=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;
```



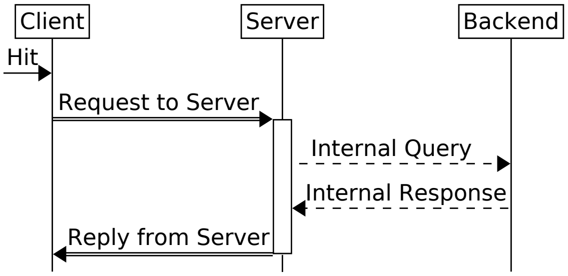
It is also possible to collapse a group entity hiding details of the process. This can be done either via the '`collapsed`' attribute or, on Windows, using the GUI. Elements that disappear leave a small indicator (box with 3 dots). The collapsed entity group also includes an indicator to show that further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

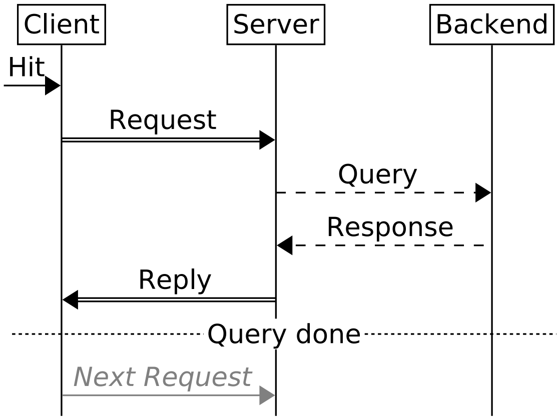
```
hscale=auto;
C: Client;
S: Server;
B: Backend;
->C: Hit;
C=>S: Request to Server;
activate S;
S>>B: Internal Query;
S<<B: Internal Response;
C<=S: Reply from Server;
deactivate S;
```



3.3 Dividers

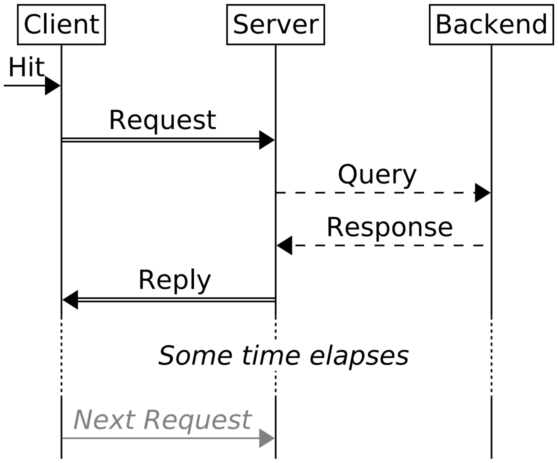
In a message sequence chart it is often important to segment the process into multiple logical parts. You can use the ‘`---`’ element to draw a horizontal line acorss the chart with some text, e.g., to summarize what have been achieved so far.

```
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;
```



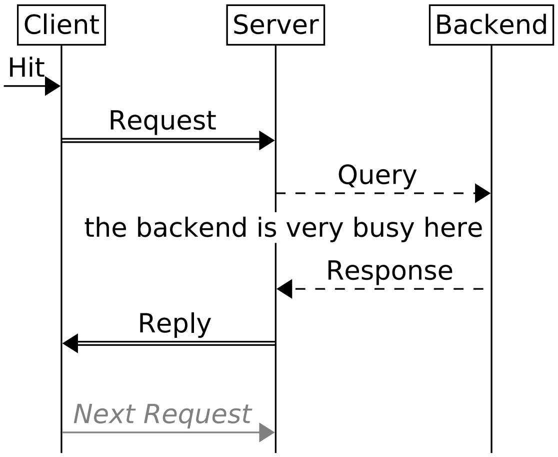
Similar to this, using the ‘`...`’ element can express the passage of time by making the vertical lines dotted.

```
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;
```



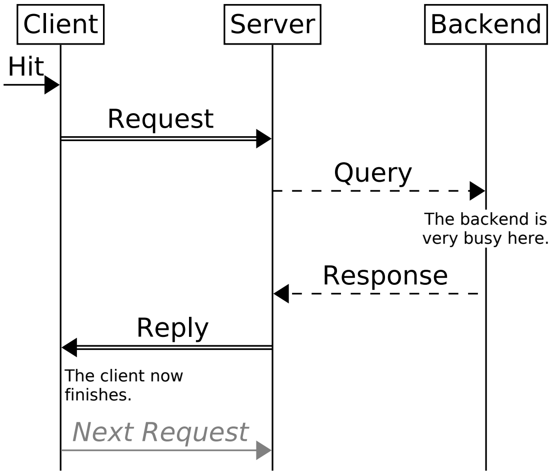
Sometimes one merely wants to add some text to a chart. In that case the empty element can be used either like ‘`: text;`’. The symbol ‘`|||`’ simply inserts an empty row.

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



The above construct always places the text in the middle of the chart. Smaller chart explanations are better done via the `text at` command.

```
C: Client;
S: Server;
B: Backend;
->C: Hit;
C=>S: Request;
S>>B: Query;
text at B: The backend is
           very busy here.;
S<<B: Response;
C<=S: Reply;
text at C+: The client now
            finishes.;
C->S [weak]: Next Request;
```

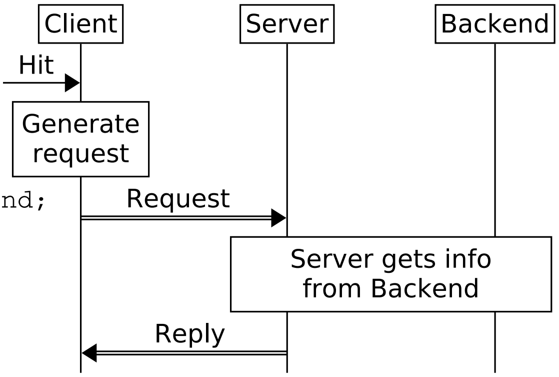


More options to comment and annotate your chart can be found in Section 3.6 [Annotating the Chart], page 25.

3.4 Drawing Boxes

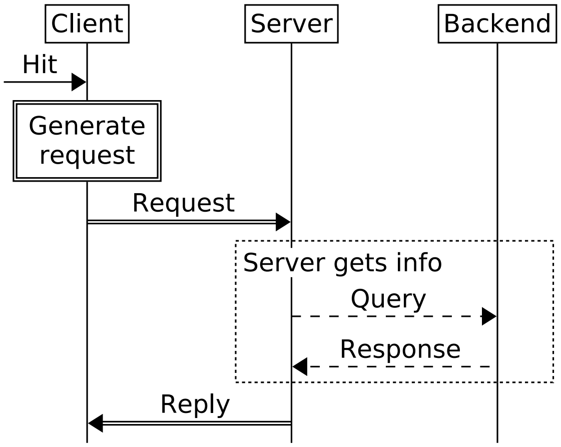
A *box* is a line around one part of the chart. It can be used to add textual comments, group a set of arrows or describe alternative behavior. In their simplest form they only contain text, but they can also encompass arrows. A box spans between two entities, or alternatively around only one.

```
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;
```



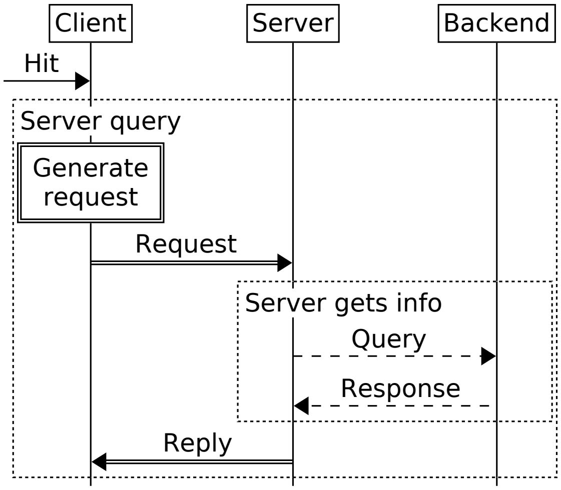
The line around boxes can be dotted, dashed and double line, too, by using '..', '++' or '==' instead of '--'. Boxes can also be used to group a set of arrows. To do this, simply insert the arrow definitions enclosed in curled braces just before the semicolon terminating the definition of the box.

```
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;
```



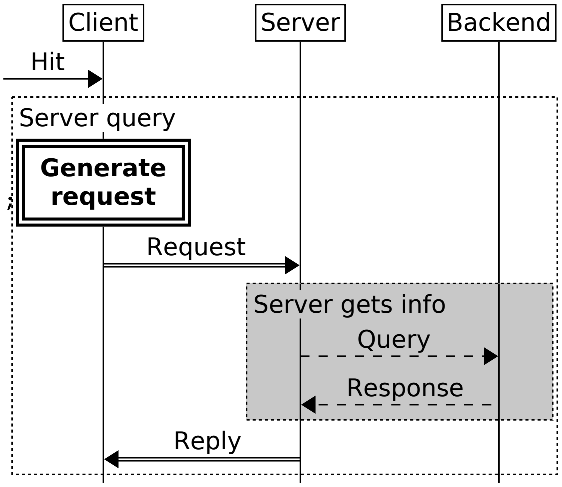
When a box contains arrows, it is not necessary to specify which entities it shall span between, it will be calculated automatically. Also boxes can be nested arbitrarily deep.

```
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;
};
```



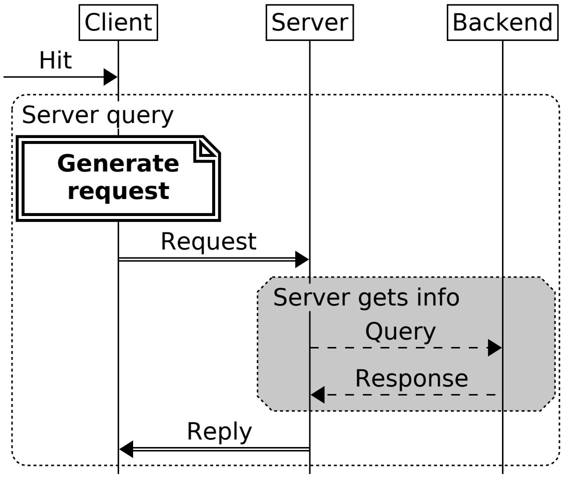
You can shade boxes, by specifying the color attribute. For a full list of box attributes and color definitions, See Section 7.4 [Boxes], page 85, and see Section 6.2 [Specifying Colors], page 58. It is also possible to make a box ‘weak’ or ‘strong’.

```
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;
};
```



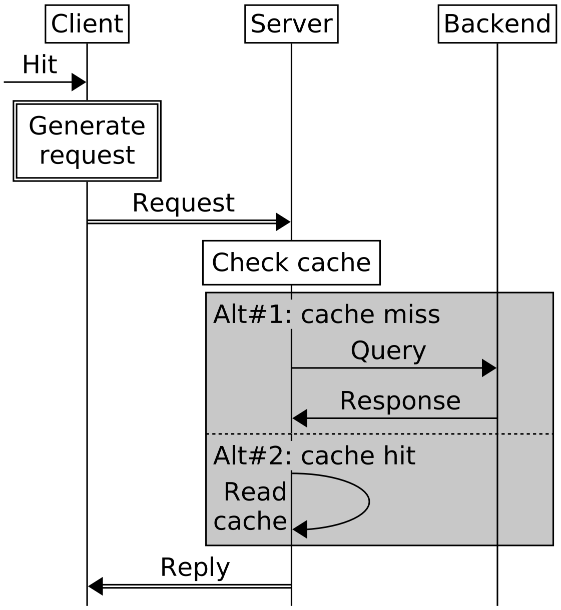
A number of box contours are available via the ‘line.corner’ attribute.

```
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;
};
```



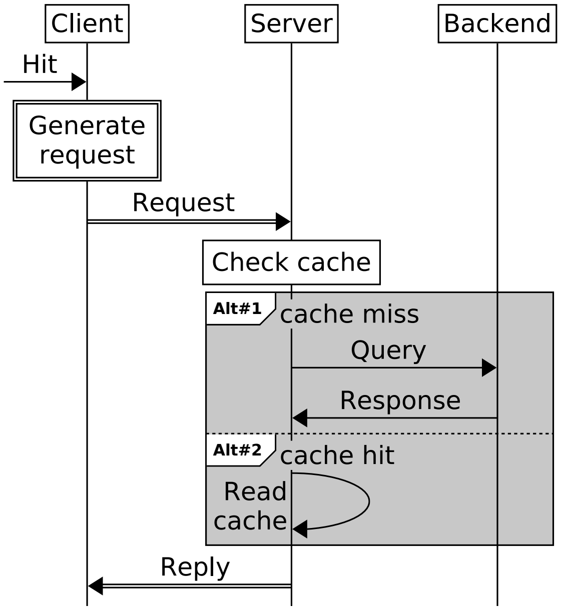
Boxes can express alternatives. To do this, simply concatenate multiple box definition without adding semicolons. These will be drawn with no spaces between. Changing the line style in subsequent boxes impacts the line separating the boxes, otherwise all attributes of the first box are inherited by the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

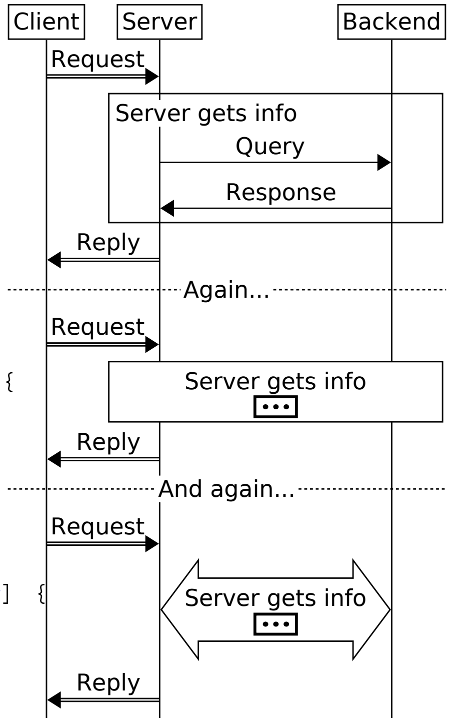
```
C: Client;
S: Server;
B: Backend;
->C: Hit;
box C==C: Generate\nrequest;
C=>S: Request;
box S--S: Check cache;
box S--B: cache miss
     [tag="Alt\#1", color=lgray]
{
     S->B: Query;
     S<-B: Response;
}
..: cache hit [tag="Alt\#2"]
{
     S->S: Read\ncache;
};
C<=S: Reply;
```



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.

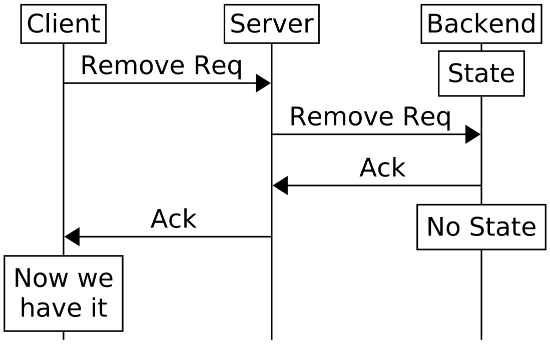
```
hscale=auto;
C: Client;
S: Server;
B: Backend;
C=>S: Request;
box S--B: Server gets info {
     S->B: Query;
     S<-B: Response;
};
C<=S: Reply;
---: Again...;
C=>S: Request;
box S--B: Server gets info [collapsed=yes] {
     S->B: Query;
     S<-B: Response;
};
C<=S: Reply;
---: And again...;
C=>S: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



3.5 Drawing Things in Parallel

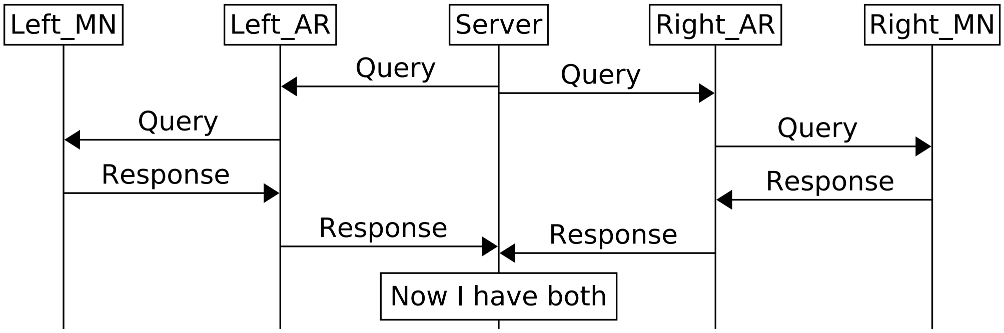
Sometimes it is desired to express that two separate process happen side-by-side. The easiest way to do so is to write ‘`parallel`’ before any arrow, box or other element. As a result the elements after it will be drawn in parallel with it.

```
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;
```



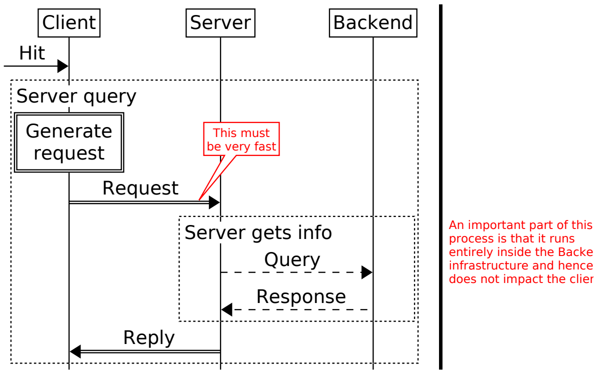
In the above example a central sever is querying two `AR` entities, which, in turn query `MN` entities further. The query on both sides happen simultaneously. To display parallel actions side by side, simply enclose the two set of arrows between braces '`{}`' and write them one after the other. Use only 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

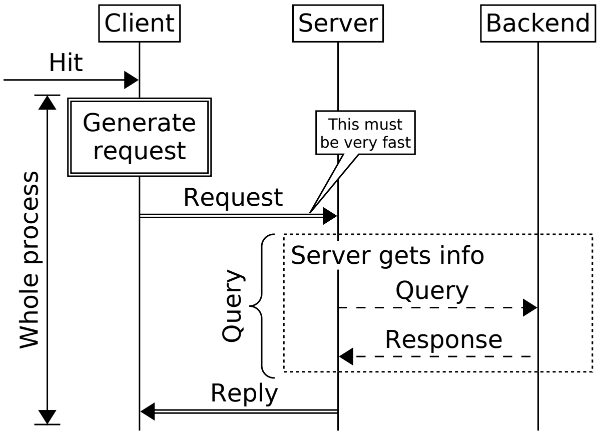
Often it is important to make annotations to the chart detailing what is going on. Msc-generator supports several types of annotations, let's start with *notes* and *comments*. Both thave a *target element* to which the note or comment is made. Notes appear as small callouts in the chart and should preferably 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;
```

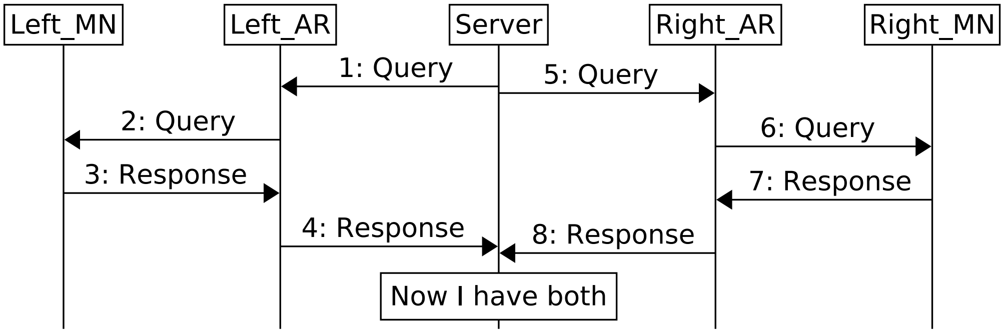


3.7 Other Features

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

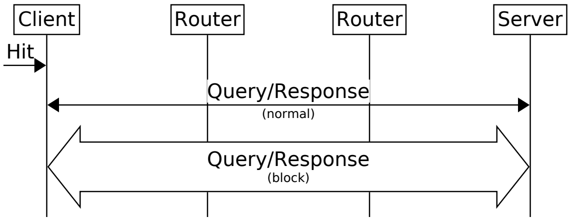
One useful feature is the numbering of labels. This is useful if you want to insert your chart into some documentation and later refer to individual arrows by number. By specifying the `numbering=yes` chart option all labels will get an auto-incremented number. This includes boxes and dividers, as well. 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];
```



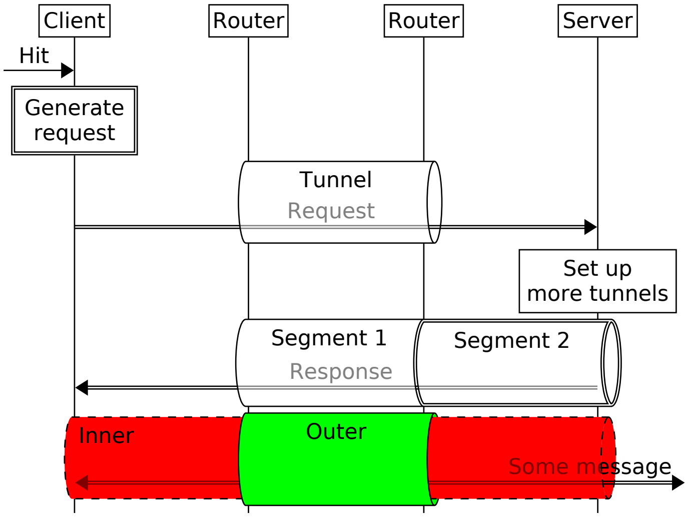
Sometimes a block of actions would be best summarized by a block arrow. This can be achieved by typing '`block`' in front of any arrow declaration.

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

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



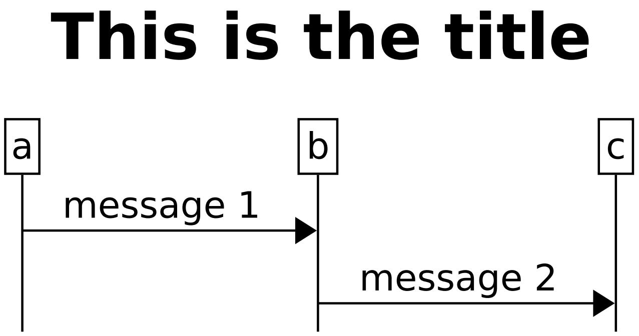
Similar, many cases you want to express a tunnel between two entities and messages travelling through it. To achieve this, just type '`pipe`' in front of any box definition. You can define a series of connected or disconencted pipe segment each with its own visual style or even encapsulate pipes. More on this in Section 7.5 [Pipes], page 88.

```
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.

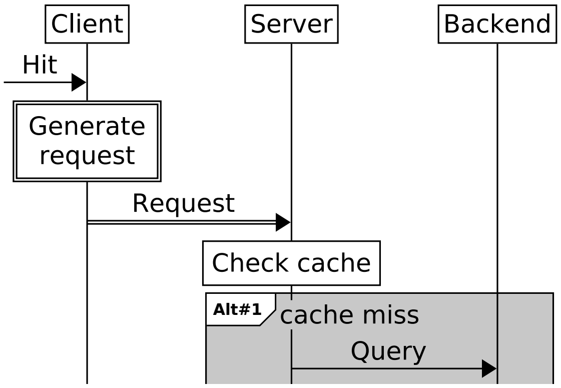
```
title: This is the title;
a,b,c;
a->b: message 1;
b->c: message 2;
```



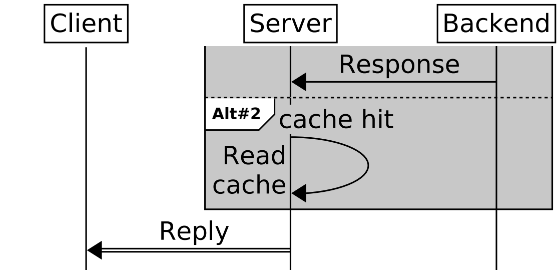
Another handy feature is multi-page support. This is useful when describing a single procedure in a document in multiple chunks. By inserting the `newpage;` command, the rest of the chart will be drawn to a separate file. You can specify as many pages, as you want. In order to display the entity headings 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:

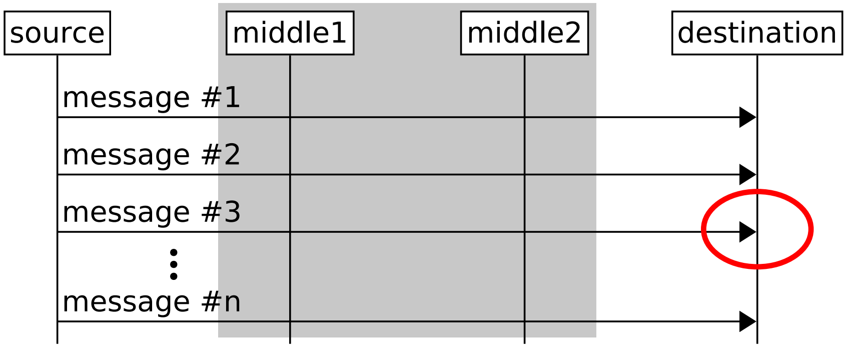


Chunk two:



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 7.13 [Free Drawing], page 114, 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];
```

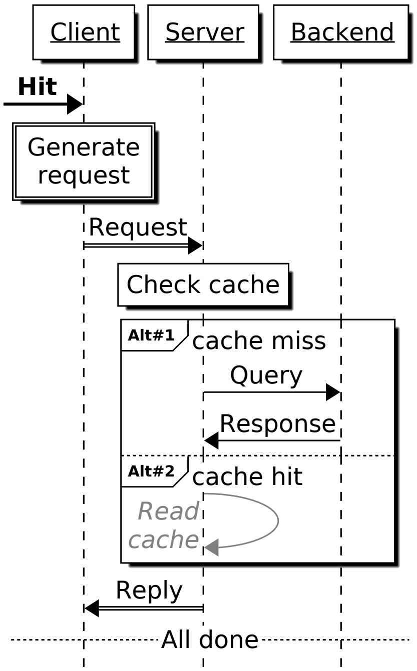


Finally, an easy way to make charts visually more appealing is through the use of *Chart Designs*. A chart design is a collection of colors and visual style for arrows, boxes, entities and dividers. The design can be specified either on the command line after double dashes, or at the beginning of the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

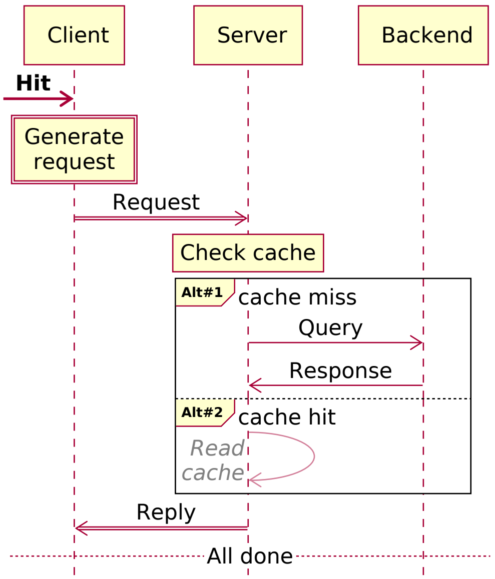
Currently several designs are supported. ‘`plain`’ was used as demonstration so far. Below we give an example of the others.

```
msc=qsd;
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;
```

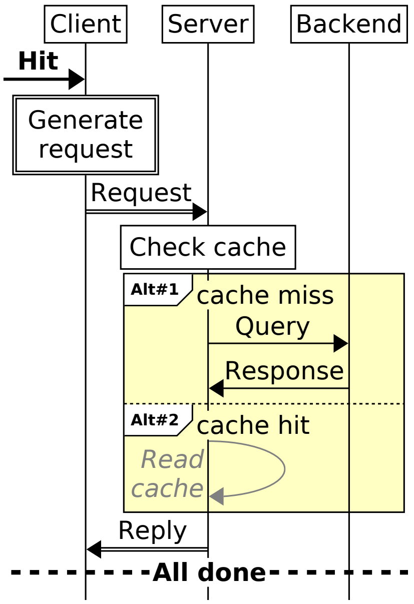
The ‘`qsd`’ design:



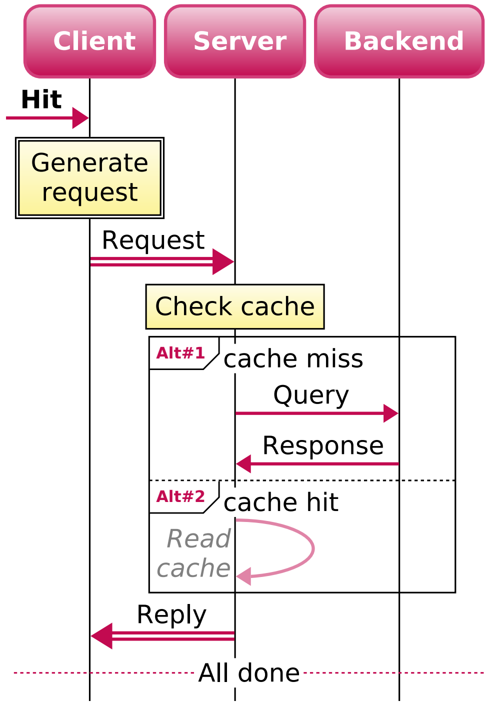
The ‘rose’ design:



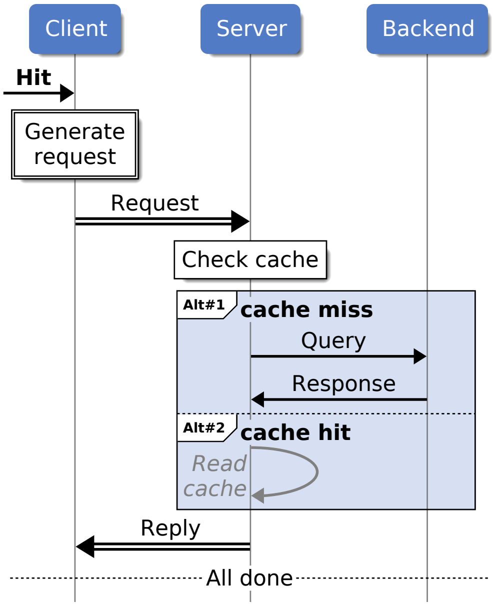
The ‘mild_yellow’ design:



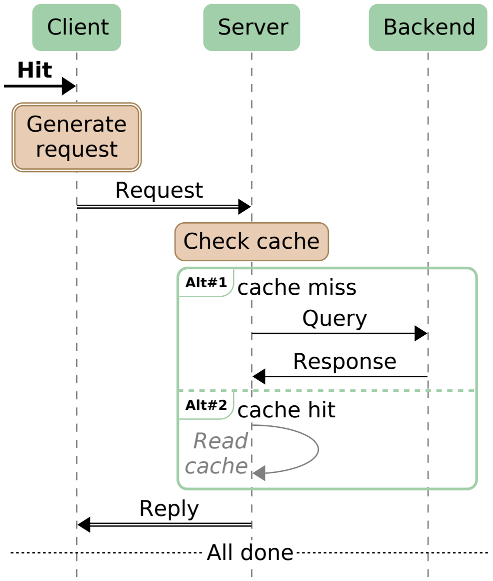
The ‘omegapple’ design:



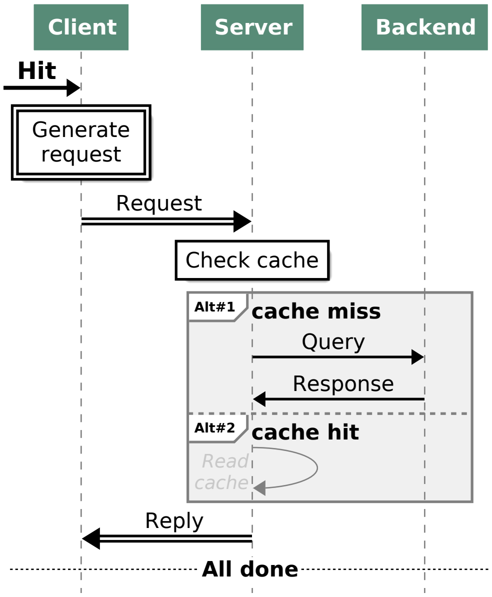
The ‘modern_blue’ design:



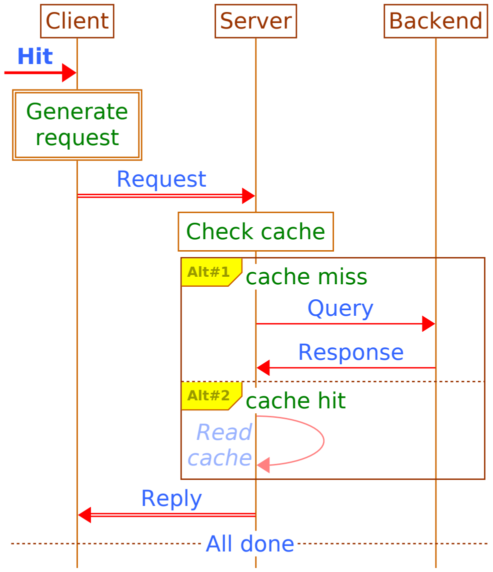
The ‘round_green’ design:



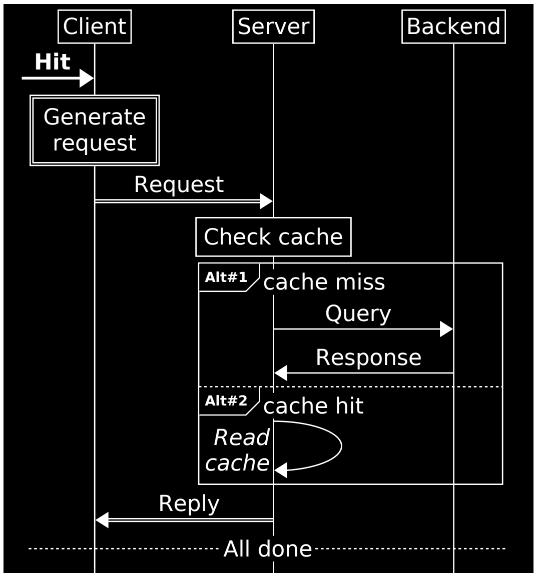
The ‘green_earth’ design:



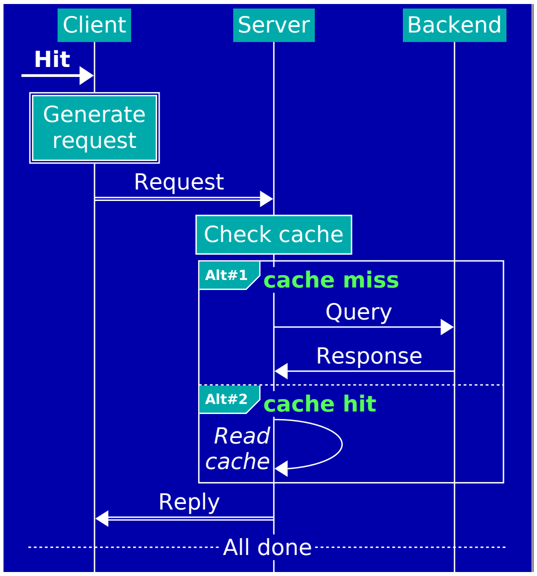
The ‘colores’ design:



The ‘black_on_white’ design:



And the the 'norton_commander' tribute design.



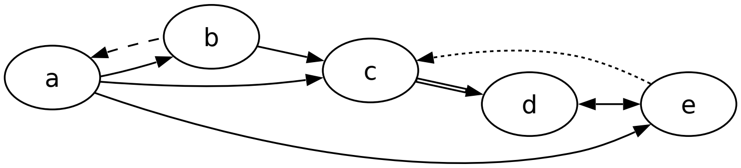
4 Graph Language Tutorial

The language of the graphs (the DOT language) is documented well on the graphviz pages at http://www.graphviz.org/. Here we give a brief introduction (using Msc-generator additions).

You need to start with either the `graph` or `strict graph` to specify a graph, the second version does not allow multiple edges between the same two nodes.

After this, you can list nodes and edges enclosed between curly braces. Although there is no need to terminate the lines with a semicolon, we suggest to do so for better readability. Node names are underlined the first time they are used (as with signalling chart entities).

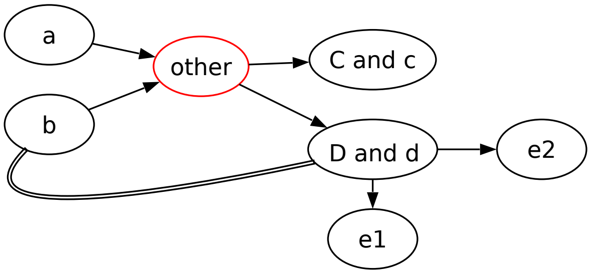
```
graph {
    rankdir = LR;
    a b;
    a->c;
    a->e;
    a->b->c=>d<->e
    a<<b;
    c<e;
};
```



You can use various arrow symbols (`->` or `<=`, for example) to generate edges with different style. The `rankdir=LR` line instructs the DOT layout algorithm to lay out from left to right as opposed to from top to bottom.

You can group nodes using curly braces. This is useful to add edges to all of them in one go or to instruct the dot algorithm to lay them out on the same level. The latter can be observed with node '`e1`' being on the same rank (left-right position) as '`D and d`'. This is because both of them are mentioned inside a set of curly braces with `rank=same`. Whithout this, '`e1`' would have been laid out right of '`D and d`' as it happens with '`e2`'.

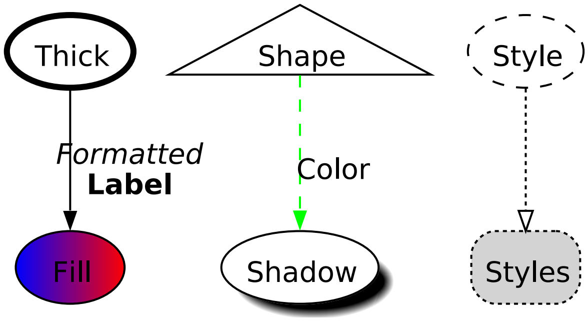
```
graph {
    rankdir = LR;
    a; b;
    c [label="C and c"];
    d::D and d;
    other [color=red];
    {a, b} -> other -> {c, d};
    b:sw == d;
    {rank=same; d->e1;}
    d->e2;
};
```



On the example above you can also see how to assign attributes. The `label` attribute can be substituted with the colon-syntax, but note that for graphs, you need to use two colons to indicate the label (as opposed to signalling charts), since single colons are used to indicate ports and compass direction. The latter is useful to impact at which angle an edge arrives/leaves the node. In the example above `sw` represents southwest, and the `==` arrow symbol results in an edge with no arrows, but double-lined.

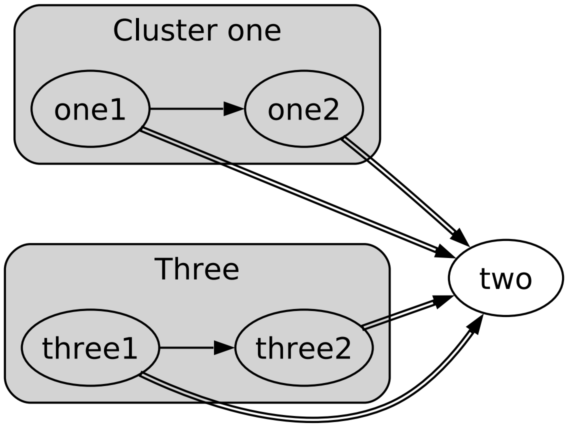
Graphviz supports many attributes for nodes, edges and layout algorithms, try experimenting. You can also use the text formatting escapes for labels used by the signalling chart syntax, but only if you specify your label with the double colon syntax. Note that due to the peculiar design of graphviz the `style` attribute can have values that affect the line, fill or the shape of nodes. Multiple such values can be specified separated using the attribute multiple times.

```
graph {
    a::Thick [penwidth=3];
    b::Fill [fillcolor="red:blue"
             style=filled];
    c::Shape [shape=triangle];
    d::Shadow [shadow_offset=7,
               shadow_blur=5];
    e::Style [style=dashed];
    f::Styles [style=dotted,
               style=filled,
               shape=box,
               style=rounded];
    a->b::\iFormatted\i\n\bLabel;
    c>>d::Color [color=green];
    e>f [arrowhead=onormal];
};
```



You can use the `subgraph` keyword to make a part of the graph separate. By graphviz convention, if you select a name that begings with '`cluster_`' the subgraph will be visibly shown. This convention is only honoured by some of the layout algorithms. As a syntactic sugar, Msc-generator defines the `cluster` keyword, that prepends '`cluster_`' to the name you supply and automatically sets the label[1]

```
graph {
    rankdir = LR;
    subgraph cluster_one {
        label="Cluster one";
        style=rounded;
        style=filled;
        one1->one2;
    } => two;
    cluster Three {
     style="rounded,filled";
     three1->three2;
    } => two;
};
```



[1] If you do not specify any name, a numeric one is generated, but is not used as label.

More details on the syntax of graphviz and the Msc-generator extensions can be found in Chapter 8 [Graph Language Reference], page 123.

5 Usage Reference

5.1 Multiple Chart Types

From version 5.0 Msc-generator supports multiple chart types. Each chart type has its own textual language, which are all some extent similar (use attributes in square brackets, options and curly braces to structure the chart).

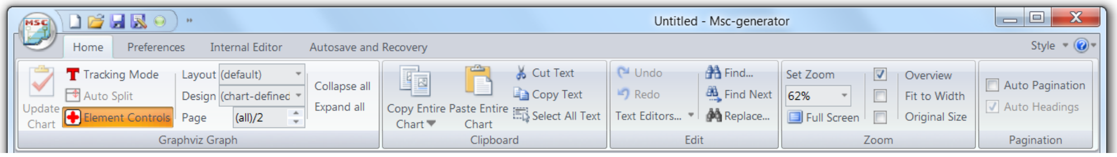
The Windows GUI supports color syntax highlighting, hinting, auto-complete, small indent, element controls and tracking for all (most) languages. The Windows GUI asks what type of chart do you want to create every time it is started or when a new chart is created.

Each language has a set of assiociated file extensions. By default, the file extension is used to determine the type of chart when a file is opened in the GUI or processed on the command line. This can be overridden on the command line. There is always a *primary* extension for each language, this is used to name the language (on the command line) and also for chart designs, see below.

The currently supported languages and their extensions is listed below (the first extension is the primary one).

Chart type	Extensions	Comment
Signalling Chart	.signalling, .msc	This chart type is the original chart type of Msc-generator.
Graphviz Graph	.graph, .dot	This language is the superset of the DOT language.

The first panel on the Home tab displays the current file type and the controls relevant for that given chart type. For example, on the picture below, the controls for graphs are shown. Compared to signalling charts there are two differences. First, you can also select the graphviz layout algorithm to apply. This is equivalent to using `layout=<layout>;` inside a graph. Second, you can collapse and expand all cluster subgraphs with one click. (Applicable only when the '`dot`' layout algorithm is used.)



5.2 Design Library

For each of its languages Msc-generator comes with a bunch of Chart Designs (and Entity Shapes for signalling charts, see Section 6.7 [Chart Designs], page 65 and Section 7.2.6 [Entity Shapes], page 75). The designs available on a Windows machine can be viewed in the design selector combo box on the ribbon.

The files describing the chart designs (and shapes) are of the same language that the language of the chart they provide designs for. These files shall have the primary extension of the chart type.

Note that you should define only designs and shapes in design libraries (for languages that support them). Colors, styles and procedures shall be defined as part of a design. If you want to define one color, style or procedure to be available without applying a design, add them to the `plain` design. Any styles, colors or procedures defined in a design library outside a design will probably be unavailable later - however, the behaviour is undefined and Msc-generator may generate errors for this in the future. Similar, avoid commands in the design library that actually generate chart elements.

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 for each supported chart type Msc-generator looks for a file called `designlib.xxx` in the directory where the executable is located (xxx is the primary extension of the chart type). If not found, the file `original_designlib.xxx` is searched (of which a default one is placed there by the installer)[1]. If any found, the content is parsed as regular chart text before any user chart text.

You are free to create and modify your own design files[2]. Designs (and shapes) defined in these will be added to the list of available designs in the GUI and can be referred to from files processed via the command line. On Linux place your design 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 and 8. Any file in the above directories with the primary extension of a supported chart type will be read by Msc-generator in no specified order before processing your chart. All other files are silently ignored.

5.3 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 visually. Also, if there were errors or warnings, they are displayed in 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).

[1] 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.xxx` is overwitten, when a newer version of Msc-generator is installed, whereas `designlib.xxx` is not. The current recommended practice is to add your own designs to new files in the roaming AppData folder, see below.

[2] However, please avoid any construct in design, which result in visual elements. Also try not to create files that result in warnings, errors.

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[3].

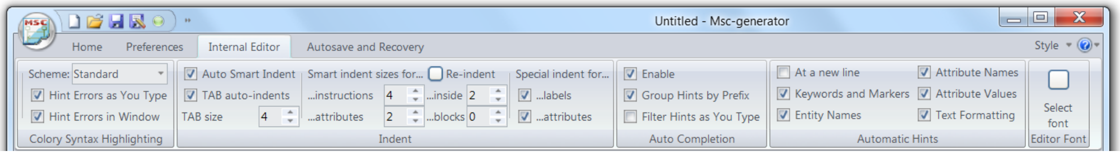
Since version 4.6 Msc-generator supports unicde characters via both UTF-8 and UTF-16 input, thus any of these can be used by the external editor to save the file. See Section 5.12 [International Text Support], page 46 for more.

5.4 Smart Indent

The internal editor supports automatic indentation for TAB, RETURN and BACKSPACE keys. 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 attribute lists and can align all subsequent lines of the label or attribute list to the first.

The parameters of automatic indentation can be set on the `Indent` panel of the `Internal Editor` ribbon category. Checking the `Auto Smart Indent` check box makes the tool automatically indent after presing RETURN, BACKSPACE, opening or closing braces and square brackets. This way indentation is automatic as you type.



Scheme: Standard
☑ Hint Errors as You Type
☑ Hint Errors in Window

☑ Auto Smart Indent
☑ TAB auto-indents
TAB size 4
Smart indent sizes for...
...instructions 4
...attributes 2
☐ Re-indent
...inside 2
...blocks 0
Special indent for...
☑ ...labels
☑ ...attributes

☑ Enable
☑ Group Hints by Prefix
☐ Filter Hints as You Type

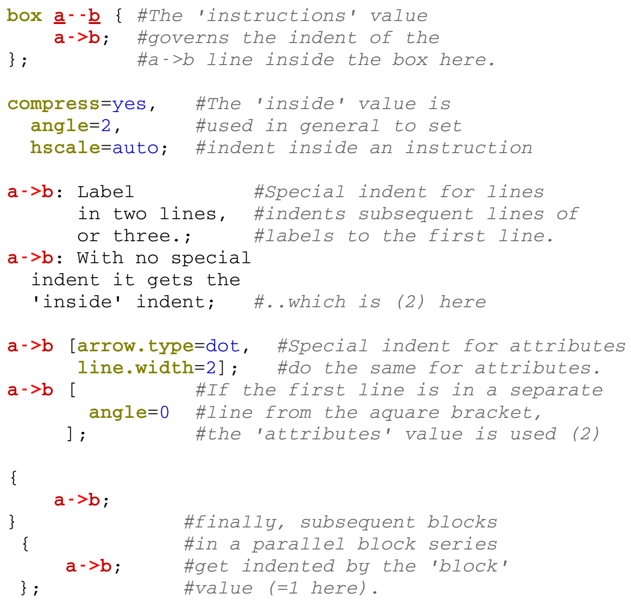
☐ At a new line
☑ Keywords and Markers
☑ Entity Names
☑ Attribute Names
☑ Attribute Values
☑ Text Formatting

The `TAB auto-indents` checkbox governs what happens when you press the TAB key. If it is on, the whole line of the cursor (or the whole selection) is auto-indented. If it is off, simply a TAB character is inserted or the whole selection indednted uniformly by the TAB size, set below.

The amount of space added by smart indent can be set in the next four edits. `instructions` govern how much instructions inside braces are indented; `attributes` are used to indent attribute lists; `blocks` govern, how a subsequent block in a parallel block, box or pipe series is indented, while `inside` governs, how anything else inside an instruction is indented, such as options in an option list, entities in an entity list, etc.

[3] You can download Notepad++ from `http://notepad-plus.sourceforge.net/`

Finally the last two check-boxes govern, if special indentation for labels and attribute list are applied or not. See the below figure for explanation.



```
box a--b { #The 'instructions' value
    a->b;  #governs the indent of the
};         #a->b line inside the box here.

compress=yes,   #The 'inside' value is
  angle=2,      #used in general to set
  hscale=auto;  #indent inside an instruction

a->b: Label          #Special indent for lines
      in two lines,  #indents subsequent lines of
      or three.;     #labels to the first line.
a->b: With no special
  indent it gets the
  'inside' indent;   #..which is (2) here

a->b [arrow.type=dot,  #Special indent for attributes
      line.width=2];   #do the same for attributes.
a->b [          #If the first line is in a separate
       angle=0  #line from the aquare bracket,
     ];         #the 'attributes' value is used (2)

{
    a->b;
}              #finally, subsequent blocks
 {             #in a parallel block series
     a->b;     #get indented by the 'block'
 };            #value (=1 here).
```

5.5 Color Syntax Highlighting

The internal editor also supports Color Syntax Highlighting. On the `Internal Editor` pane you can select one of four color schemes. 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[4]. At the moment you can not customize individual colors in the schemes. 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 (see `Hint Errors as You Type`). 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 (see `Hint Errors in Window`). 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.

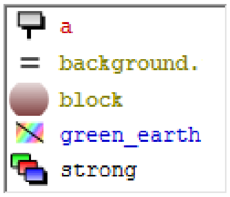
5.6 Typing Hints and Autocompletion

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

[4] We note here that all four schemes underline entities at their first use. This is to help you avoid a mis-typed entity name.

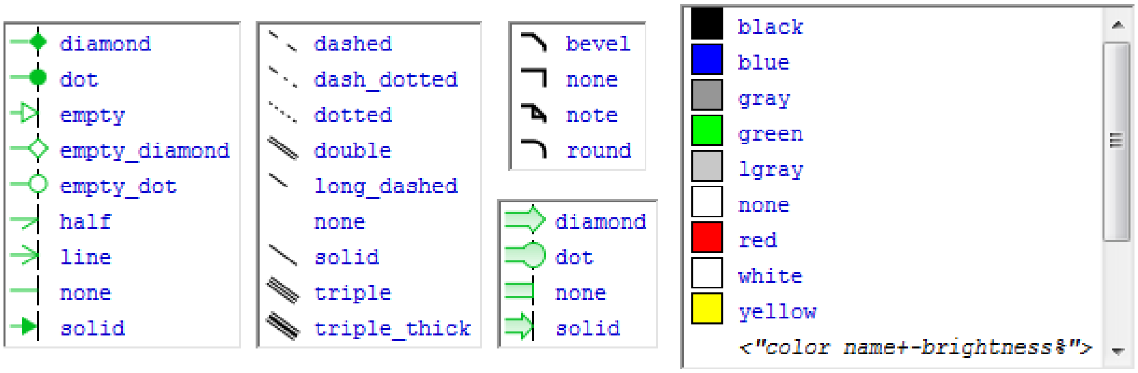
typing the keyword or hit any non-alphanumeric character, which will automatically select the highlighted hint and continue after.

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.



a
background.
block
green_earth
strong

Various attribute values offer a graphic representation to ease selection.[5] 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.



diamond
dot
empty
empty_diamond
empty_dot
half
line
none
solid

dashed
dash_dotted
dotted
double
long_dashed
none
solid
triple
triple_thick

bevel
none
note
round

diamond
dot
none
solid

black
blue
gray
green
lgray
none
red
white
yellow
<"color name+-brightness%">

On the Internal Editor category of the ribbon you can control how much suggestions you will get and how they are displayed. You can turn hints entirely off. The two bottom checkboxes in the Auto Completion panel 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.

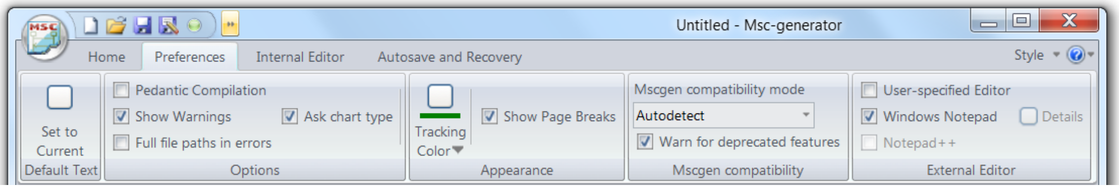
Msc-generator can provide you hints even without pressing the Ctrl-Space. In the subsequent panel you can govern in what language contexts do you want to receive such automatic hints. In general it is best to experiment with these settings and see what you like.

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).

[5] Shapes defined in the current file do not appear with a thumbnail representation. Only shapes defined in design libraries do.

5.7 Options

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



☐ Pedantic Compilation
☑ Show Warnings ☑ Ask chart type
☐ Full file paths in errors
☑ Show Page Breaks
Autodetect
☑ Warn for deprecated features
☐ User-specified Editor
☑ Windows Notepad ☐ Details
☐ Notepad++

In the first category you can specify what is the text of the chart that pops up when a new chart is started. Just press the button and the current text will become the default for the current chart type. 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.

Under '`Options`' you can set a few compilation options. When pedantic is set Msc-generator enforces stricter language interpretation. For signalling chart, it means generating a warning if an entity is not declared explicitly before use. For graphs, it means generating a warning for each use of directed edges in undirected graphs and vice versa (graphviz does not allow such mixing), but when turned off, mixing directed and undirected edges in a graph becomes possible. Turning the second option on will supress the generation of warning messages altogether (including the ones generated due to the pedantic option). Checking the third option will show the full path of the filename in error messages. This is useful when using `include` and suspecting the precise identity of the included file with an error. When '`Ask char type`' is set, Msc-generator displays the list of available chart types at startup and when creating an empty document and you can select one. If unchecked, the last file type is used always.

On the '`Appearance`' panel you can first select the color of the tracking overlay (what flashes when you click on a chart element). '`Show Page breaks`' governs if a dashed line is drawn to show where page breaks are when watching all of the pages. See Section 7.12 [Multiple Pages], page 113 for more information. Lastly you can also set the maximum zoom factor selected by the *Overview* and *Fit to Width* automatic zoom modes. You may want to increase this for very large screens. See Section 2.3 [Zooming], page 5.

The '`Mscgen compatibility`' panel is applicable only to signalling charts and governs how Msc-generator switches to backwards compatibility mode with mscgen. See Section 7.15 [Mscgen Backwards Compatibility], page 120 for more.

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 linenumber at invocation.

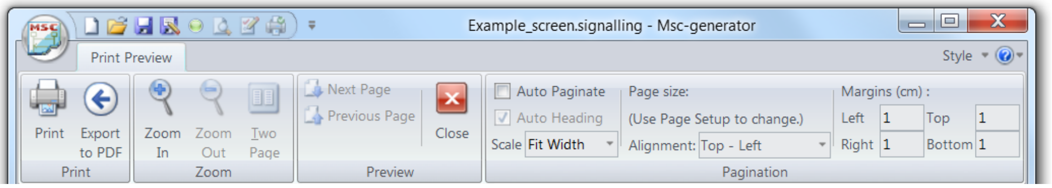
5.8 Working with Multi-page Charts

Msc-generator supports multi-page charts. These may be useful when you want e.g., to print a long signalling chart. For signalling charts you can manually start a new page by typing the `newpage;` command or let Msc-generator automatically paginate for you. See more in Section 7.12 [Multiple Pages], page 113. In case of graphs, each graph is laid out to a separate page.

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 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 5.7 [Options], page 42.)

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.



☐ Auto Paginate
☑ Auto Heading
Scale Fit Width
Page size:
(Use Page Setup to change.)
Alignment: Top - Left
Margins (cm) :
Left 1 Top 1
Right 1 Bottom 1

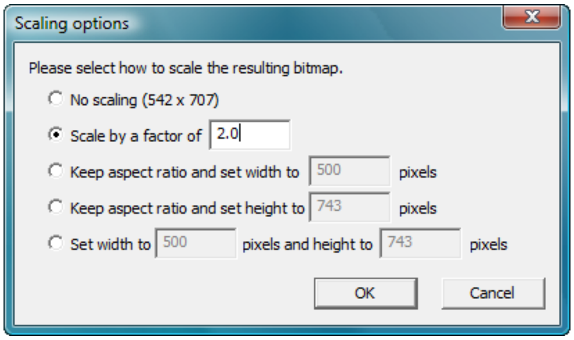
Here you can print or export to PDF[6]. You can also set Automatic Pagination and Auto Headings from here, but you can also set, how the chart pages are sized; how they are aligned within the physical pages and also what margins to apply. The former option can take *Fit width* and *Fit page*[7]. In the 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*.

5.9 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.

[6] 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.

[7] They correspond to the `-s=width` and `-s=auto` command-line options, respectively.



Please select how to scale the resulting bitmap.

○ No scaling (542 x 707)

● Scale by a factor of 2.0

○ Keep aspect ratio and set width to 500 pixels

○ Keep aspect ratio and set height to 743 pixels

○ Set width to 500 pixels and height to 743 pixels

5.10 Advanced OLE Considerations

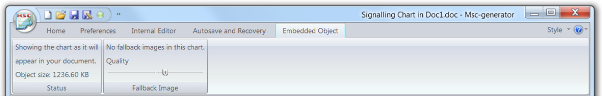
5.10.1 Graphics of Embedded Charts

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

- Due to clipping limitations, certain arrowheads, like '`line`' draw differently on signalling charts.
- Due to font limitations, the label of slanted arrows on signalling charts 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, transculent 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 new 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.



Showing the chart as it will appear in your document.
Object size: 1236.60 KB

No fallback images in this chart.
Quality

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 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.

[8] Only drawing operations permitted in old-style Windows Metafiles (developed in 16-bit Windows times) are permitted by design.

5.10.2 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 geta 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.[9]

5.11 Autosave and Recovery

Since version 5.0 The Msc-generator GUI supports the Restart Manager introduced to Windows in Vista. That is, if Msc-generator crashes (or needs to be stopped due to an installation), its status is saved and is automatically restarted.

The documents you are working on are autosaved.[10] Clicking on the '`Autosave and Recovery`' category on the ribbon opens a list of autosaved files on the side.[11] Double-clicking a file will recover it either by overwriting the current chart text or by inserting it at the cursor (you can select after double-clicking). You are also offered the option to

[9] To link to a full chart embedded in Word, make sure you place chart to the clipboard using Word and not using Msc-generator.

[10] The autosave location is `C:\Users\<username>\AppData\Local`, you can manually recover autosaved files from there.

[11] Any autosaved version of the currently open document is omitted from the list.

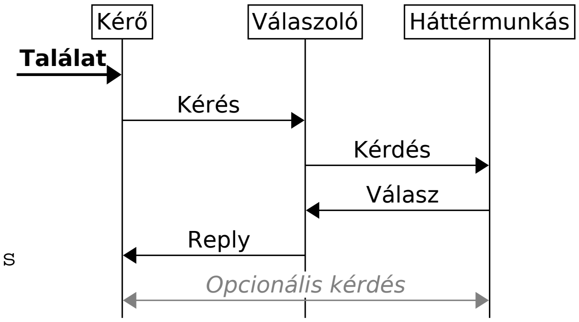
delete the autosaved file after recovrey. On the ribbon you can filter the list of autosaved file to the current chart type and make Msc-generator to open the list of autosaved files at startup. You can also delete all autosaved files at once.[12]

Note that in the rare event of an Msc-generator crash[13] the restart function kicks in only if the application was open for more than 60 seconds (to prevent restart loops). Your documents are autosaved even before this time, nevertheless.

5.12 International Text Support

Msc-generator supports international text via both UTF-8 and UTF-16[14]. The type of encoding is determined automatically when the file is loaded. If it is ASCII or valid UTF-8, it is treated as so. Else it is treated as UTF-16. The Windows GUI displays the detected encoding on the status bar. You can change it by double clicking the indicator on the status bar - next time you save the file, it will be saved using the encoding displayed there. Below is an example of a simple chart shown in the language tutorial in my native language, Hungarian.

```
->Kérő: Találat [strong];
Kérő->Válaszoló: Kérés;
Válaszoló->Háttérmunkás: Kérdés;
Válaszoló<-Háttérmunkás: Válasz;
Kérő<-Válaszoló: Reply;
Kérő<->Háttérmunkás: Opcionális kérdés
        [weak];
```



Note that not all font faces contain characters for all possible unicode codepoints. This has to be considered when selecting a font[15]. If you mix characters of different languages, you need to ensure that they are displayed via a font that has coverage for all those characters.

In labels, you can use the `\$` escape followed by exactly four hexadecimal digits, to insert any unicode character to a label, e.g., `\$00a9` inserts the copyright symbol.

Note that if you plan to use non-ASCII characters in the chart text you need to select a font that can display them not only in the chart, but also for the internal editor. See Section 5.14 [Fonts], page 50.

[12] If you filter the list of autosaved files to the current chart type then clicking the '`Delete autosaved files currently showing`' button will not delete the autosaved files of other chart types.

[13] The graphviz library crashes on some chart input. This is something I cannot easily fix. Hence the addition of the autosave and recovery functions.

[14] UTF-8 is a superset of ASCII, where international characters are encoded in multiple bytes all larger than 127 (which is the largest ASCII character). In contrast, in UTF-16 all characters are encoded on two bytes, ASCII and international characters alike. The UTF-16 file usually start by the characters 0xff and 0xfe and their order determines in which order the two bytes of a character comes. UTF-8 is widely used in Linux and other Unix systems, whereas Windows has adopted UTF-16 as the system encoding. Msc-generator supports both file formats and both byte orders of UTF-16.

[15] In signalling charts you can select a font via the `text.font.face` chart option or attribute, whereas in graphs use the `fontname` attribute

5.13 Command-Line Referece

The syntax of the command-line version is the same on Linux and Windows[16].

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-generator's extra features and rich language in every tool that is integrated with mscgen. These tools include Doxygen, Sphinx and Msctexen[17]. For more on the compatibility of the languages see Section 7.15 [Mscgen Backwards Compatibility], page 120.

```
Usage: msc-gen [OPTIONS] [infile]
       msc-gen -l
       msc-gen --help
       msc-gen --version
```

Generic options:

`-T` *type*
Specifies the output file type, which maybe one of `png`, `eps`, `pdf`, `svg`, `ismap`, `lmap` or `emf` (on Windows only). Default is `png`. Output type `ismap` generates an NCSA format ismap file contain link information, see Section 6.4 [Links], page 62. You can also specify `lmap`, which will not generate any graphics either, 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 5.13.1 [Label Maps], page 49 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. You can optionall specify the `-i` switch before the input file name (for mscgen compatibility).

-S <lang>
Forces Msc-generator to interpret the input file as a specific type of chart. This overrides the guess from the filename extension. Use the primary extension of the chart type here. Current chart types are 'signalling' for Signalling Charts and 'graph' for Graphviz graphs.

`--utf16`
Forces the input file to be interpreted as UTF-16, even if it looks like UTF-8 or ASCII.

`--utf8`
Forces the input file to be interpreted as UTF-8, even if it does not look like it.

`-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 physcal 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

[16] The only two exceptions are in how pathnames are written on the two systems and where the design libraries (e.g., `designlib.signalling`) will be searched.

[17] This is thanks to the original mscgen author Michael McTernan and many others

'`ledger`', which are by definition portrait and landscape, resp.). E.g., use '`A4p`', '`A2l`' or '`letter_l`'. Deafult 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.

`-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 beused 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. See Section 5.14 [Fonts], page 50. Does not apply to graph charts.

`-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 6.7 [Chart Designs], page 65 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 7.11 [Chart Options], page 110) 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 6.7 [Chart Designs], page 65). This will overridde any design specified in the file.

`-Wno` No warnings displayed.

`-Pno` No progress indicator displayed.

`-l` Display program licence and exit.

`-h`
`--help` Display program help and exit.

`--version`

Display version information and exit.

Options specific to Signalling Charts:

`--force-mscgen`

Forces the chart to be interpreted in mscgen mode. Note that many Msc-generator attributes, commands and keywords are still recognized. This setting makes conflicting syntax be intrepreted as mscgen would do. Without this switch Msc-generator uses the mscgen mode only if the chart starts with the text `msc {`.

`--prevent-mscgen`

Prevents the chart to be interpreted in mscgen mode. Note that some mscgen attributes and symbols are still recognized. This setting makes conflicting syntax be intrepreted as Msc-generator would do. Without this switch Msc-generator uses the mscgen mode if the chart starts with the text `msc {`.

`-Wno-mscgen`

Disables warnings for deprecated constructs kept only for backwards compatibility with mscgen. Has no effect with –force-mscgen, in that case no such warnings are emitted.

`--pedantic`

When used, Msc-generator enforces stricter language interpretation. For signalling charts, it will generate a warning if an entity is not declared explicitly before its use. For graphs, Msc-generator will generate a warning for each use of directed edges in undirected graphs and vice versa (graphviz does not allow such mixing). When not specified, it is OK to mix directed and undirected edges in any kind of graph.

5.13.1 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[18] (and no graphics output). The default extension will be `.map`. The lines in the output file contains the followin information separated by space.

[18] Note that box tags are not included in the label map.

```
<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 appeareance 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 swicthes 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.

In this release of Msc-generator, label maps are not emitted for graphs.

5.14 Fonts

For signalling charts the fonts used for labels, comments, etc. can be selected in 5 ways (listed in decreasing order of preference).

1. Using the \f(*font name*) text formatting escape sequence. This can be applied even in the middle of a label.
2. Using the `text.font.face` attribute, in which case it applies to the whole label. The value of this attribute can also be set by styles.
3. Using the `text.font.face` chart option. It affects all subsequent labels (until the next closing brace, if any).
4. Using the `-F` command-line option.
5. By setting the `MSCGEN_FONT` environment variable.

The fonts available are system dependent. On Windows, you can use all the Windows 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 have fontconfig installed, use the `fc-list` command to list available fonts. Pick the family name in the list for use in Msc-generator.

For graphs, the font is selected exclusively via the `fontname` attribute.

On the Windows GUI you can use the last panel of the '`Internal Editor`' ribbon category to select the font of the internal text editor. The two checkboxes can be used to filter the list of font families offered by the lisbox below them. Note that if you plan to use non-ASCII characters in the chart text you need to select a font that can display them not only in the chart, but also for the internal editor.

6 Reference for Common Language Elements

Msc-generator languages all share a common underlying logic and design.

- They all define elements and relations between elements, such as entities and arrows for signalling charts and nodes and edges for graphs.
- These elements and relations usually can have a text label, the format of which (such as bold or italics) can be influenced via *text formatting escapes*.
- These elements and the relations all have *attributes* to govern appearance, which can be specified in square brackets.
- Attributes can be collected in *styles*, which can be applied to elemets and relations in a simple way.
- Each element and relation type has a *default style*, which can be changed to change the appearance for all such elements or relations.
- Curly braces can be used to group elements and to isolate a *scope* - any changes to styles or colors are limited to the current scope.
- You can define *designs*, which contain styles, colors and can be applied to the whole chart to change its appearance in one go. These are collected into *design libraries*, which are read at program startup and are available by default.
- You can also include other files in the middle of the any document.
- You can define and later replay *proecedures* to avoid frequent copy/paste and reuse frequent constructs instead.

This chapter contains all the language elements that are common to all or most chart types supported by Msc-generator. Variations specific to each language are indivially described.

6.1 Common Attributes

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[1]. Quoted strings themselves can contain quotation marks by preceeding them with a backslash ‘`"`’. Numeric values can, in general be floating point numbers (no 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 6.2 [Specifying Colors], page 58.

The attributes below can be part of a *style*, see Section 6.1.4 [Styles], page 56.

[1] Specifically for signalling charts: 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 it ends with a dot, it must also be quoted. The only exception to this are built-in style names, see Section 6.6 [Defining Styles], page 64. For graphviz graphs strings can not start with number and can only contain alphanumeric characters - else they have to be quoted.

6.1.1 Line and Fill Attributes

Currently line attributes apply only in signalling charts. Graphviz has a totally different way of specifying line appearance via the `style=` attribute. Further languages planned to be added will use the below attributes.

`line.color`
: Specifies the color of the lines. Unless you use a single color name you must quote the color specification, see Section 6.2 [Specifying Colors], page 58 for the syntax of colors.[2]

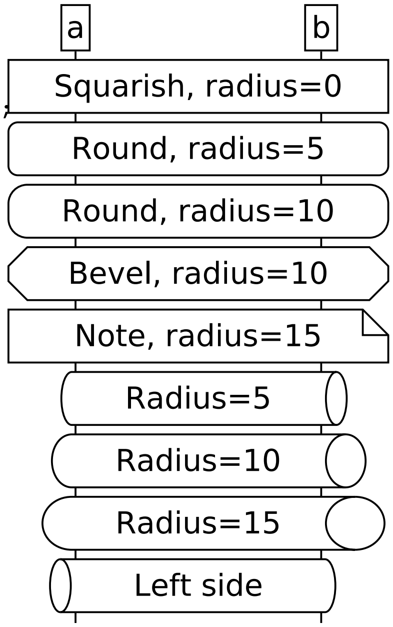
`line.width`
: Specifies the width of the line.

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

`line.corner`
: For boxes-like elements 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.

`line.radius`
: Specifies the size of the corner above. In some cases it is also used to specify the turning radius of turning lines (such as arrows turning back, etc.).

```
a--b: Squarish, radius=0;
a--b: Round, radius=5  [line.radius= 5];
a--b: Round, radius=10 [line.radius=10];
a--b: Bevel, radius=10 [line.corner=bevel];
a--b: Note, radius=15  [line.radius=15,
                        line.corner=note];
pipe a--b: Radius=5;
pipe a--b: Radius=10 [line.radius=10];
pipe a--b: Radius=15 [line.radius=15];
pipe a--b: Left side [side=left];
```



`fill.color`
: Defines the background of an element covering an aread. Specifying `none` results in no fill at all. Unless you use a single color name you must quote the

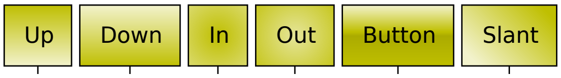
[2] For signalling charts this is not necessary.

color specification, see Section 6.2 [Specifying Colors], page 58 for the syntax of colors.[3]

`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 textual 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. In addition, it can also be set to a numerical value, which result in a linear gradient and specifies the angle of it.

```
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];
Slant   [fill.gradient=45];
```



6.1.2 Shadow Attributes

`shadow.offset`
: If not set to zero, then the element 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. In graphviz, this attribute is called `shadow_offset`.

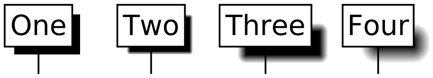
`shadow.color`
: The color of the shadow. This attribute is ignored if shadow.offset is 0. Unless you use a single color name you must quote the color specification, see Section 6.2 [Specifying Colors], page 58 for the syntax of colors.[4] In graphviz, this attribute is called `shadow_color`.

`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. In graphviz, this attribute is called `shadow_blur`.

[3] For signalling charts this is not necessary.

[4] For signalling charts this is not necessary.

```
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];
```



6.1.3 Text Formatting Attributes

Currently line attributes apply only in signalling charts. Graphviz labels can only be styled using text formatting escapes in a colon-label. Further languages planned to be added will use the below attributes.

`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 6.2 [Specifying Colors], page 58 for the syntax of colors.

`text.bgcolor`
: Sets the background color of the label, none by default.

`text.font.face`
: Specify the font face family using this attribute, such as `Arial` or `Helvetica`. The fonts available depend on the platform. See Section 5.14 [Fonts], page 50.

`text.font.type`
: Select between normal or small font, superscript or subscript using the values `normal`, `small`, `superscript` and `subscript`, respectively.

`text.bold`
`text.italic`
`text.underline`
: You can set them to `yes` or `no`.

`text.gap.up`
`text.gap.down`
`text.gap.left`
`text.gap.righ`
: These four attribues can be used to set the margins around the label in pixels.

`text.gap.spacing`
: This sets the line spacing in pixels.

`text.size.normal`
`text.size.small`
: These sets the height of the normal font type (see `text.font.type` above) or the height of small, superscript and subscript, respectivel.

`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 6.3 [Text

Formatting], page 59. 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.link_format`

Similar to the above, you can specify the formatting applied to links. See Section 6.4 [Links], page 62.

`text.wrap`

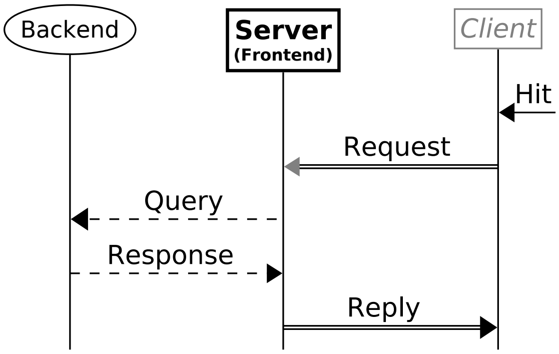
Can be set to `yes` or `no`. If disabled (default), the label will follow the line breaks inserted by the user. If enabled, these line breaks are ignored and the line is typeset to fill available space, see Section 6.3 [Text Formatting], page 59.

`text.`

6.1.4 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 whereever an attribute is allowed. See in the example below, how the `strong` and `weak` styles are applied to entities.

```
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;
```



For signalling charts styles can contain any of the attributes listed in Section 7.10 [Signalling Chart Attributes and Styles], page 102. For graphs any graphviz attribute can be part of a style. 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 6.6 [Defining Styles], page 64 for more on this in signalling charts.

6.1.5 Labels

Entities, arrows, boxes, pipes, dividers (signalling charts), nodes, edges and clusters (graphs) all 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 (signalling charts), nodes and clusters (graphs) the label defaults to the name of the elements, 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 6.3 [Text Formatting], page 59.

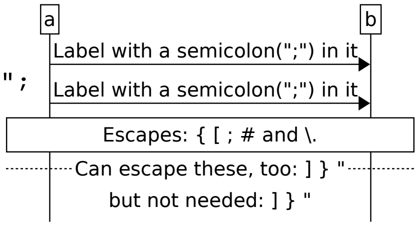
To avoid typing `[label="..."]` many times it is possible to specify the label attribute in a simpler way. After the definition of the element, just type a colon (two colons for graphs), the text of the label unquoted and terminate with a semicolon (or opening brace '`{`' or bracket '`[`'). You can write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

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

If the label needs to contain a opening bracket ('`[`'), opening brace ('`{`'), hashmark ('`#`') or a semicolon ('`;`') use quotations or preceed these characters by a backslash '\'[5]. 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 '`"`'[6].

```
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: ] } ";
```

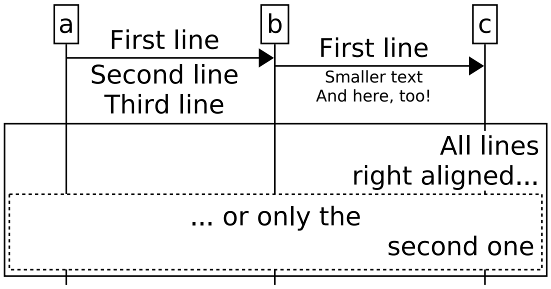

a
b
Label with a semicolon(";") in it
Label with a semicolon(";") in it
Escapes: { [; # and \.
Can escape these, too:] } "
but not needed:] } "

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.

[5] This character is often called the *escape character* making an *escape sequence* together with the character it follows.

[6] 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 with mscgen.

```
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;
};
```

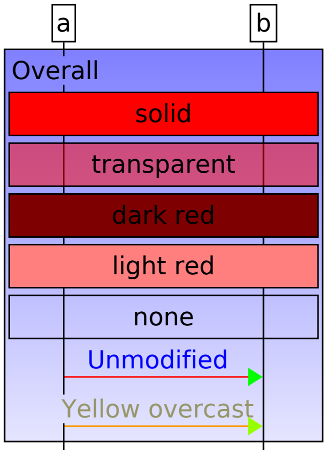


6.2 Specifying Colors

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

Color names in signalling charts 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 specifly 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. For graphviz graphs precede them by hash-marks (`#`) and use hexadecimal numbers, six digits in total.

For signalling charts separate the R, G and B components 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[7]. 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.)

For charts other than graphviz graphs, 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 6.5 [Scoping], page 63 for more on scopes.

6.3 Text Formatting

Any text displayed in the chart can contain *formatting escapes*.[8] 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.

`\n` Inserts a line break.

`\-` Switches to small font.

`\+` Switches to normal (large) font.

`\^` Switches to superscript.

`\_` Switches to subscript.

`\b` Toggles bold font.

`\B` Sets font to bold.

`\i` Toggles italics font.

`\I` Sets font to italics.

`\u` Toggles font underline.

`\U` Sets font to underlined.

[7] 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.)

[8] For graphs only labels specified via the double-colon notation interpret escape sequences this way. If you assign a label via the `label=` attribute, the escape sequences will be interpreted as graphviz escape sequences, see `http://www.graphviz.org/doc/info/attrs.html#k:escString`.

`\f(`*`font face name`*`)`
: Changes the font face. Available font face names depend on the operating system you use. (See Section 5.14 [Fonts], page 50.) If you specify no font, just `f()`, the font used at the beginning of the label is restored.

`\0..\9`
: Inserts the specified number of pixels as line spacing below the current line.

`\c(`*`color definition`*`)`
: Changes the color of the text. Color names or direct rgb definitions can both be used, as described in Section 6.2 [Specifying Colors], page 58. No quotation is needed. You can also omit the color and just use `\c()`, which resets the color back to the one at the beginning of the label.

`\s(`*`style name`*`)`
: Applies the specified style to the text[9]. Naturally only the `text.*` attributes of the style are applied. You can omit the style name and specify only `\s()`, which resets the entire text format to the one at the beginning of the label[10]. See Section 6.1.4 [Styles], page 56 for more information on styles.

`\S(`*`shape|height|fillcolor`*`)`
: Inserts a shape into the text as a single character. Shapes can be arbitrary forms described by polygons or bezier curves and are defined using the `defshape` command, see Section 6.8 [Defining Shapes], page 66 for moreinformation. This escape allows drawing arbitrary things everywhere, where a label is permitted. Within the escape sequence, you first need to specify the name of the shape, then optionally its height (default is the font size) and optionally its fill color (default is transparent fill). The latter option takes effect only for shapes having a fill section, see Section 6.8 [Defining Shapes], page 66. You can omit the height (`\S(`*`shape||fillcolor`*`)`) or the fill color (`\S(`*`shape|height`*`)`) or both (`\S(`*`shape`*`)`).

`\mu(`*`num`*`)`
`\md(`*`num`*`)`
`\ml(`*`num`*`)`
`\mr(`*`num`*`)`
`\mi(`*`num`*`)`
: 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 nonnegative integer and is interpreded 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.

[9] 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.

[10] 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.

`\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.

`\pl \pc \pr`
Changes the identation to left, centered or right. Applying at the beginning of a line (t.i., before any literal character) will apply new identation 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.

`\$xxxx` This escape can be used to insert any unicode character into a label, via its codepoint. Specify the codepoint in hexadecimal using exactly four hex digits. E.g., `\$00A9` will result in the copyright symbol. See more on international characters in Section 5.12 [International Text Support], page 46.

`\|` 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.

`\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.

`\r(`*`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`'.

`\L(`*`link target`*`)`
`\L()` This escape shall be used in pairs to assign a hyperlink to the text in between the two. The target of the link shall be specified in the first escape, whereas the second shall be empty. For example: `a->b: A \L(http://abc.com)link\L() to abc.com.;` The link target cannot contain closing parenthesis. If it contains opening square brackets (`[`), opening curly braces (`{`), semicolons or hash marks, which normally terminates colon labels, use quotation marks around the label,

such as `a->b: "A \L(http://abc.com/#x)link\L() to abc.com/\#x.;` Note that *tag labels* of boxes can also contain links.

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 for signalling chart.

6.4 Links

Hyperlink info can be added labels (or parts of labels). This is useful when the resulting image is embedded in web pages. Hyperlinks are not (yet) exported into a PDF or SVG file. Currently link information can be extracted via the `-T ismap` command-line option, which provides an NCSA formatted ismap file. Such files are used by Doxygen, for example, so this feature is most applicable to Doxygen integration. (If you specify a link target as `\ref Name` then Doxygen will point to a function or class named `Name` in the documentation created by Doxygen.)

On the Windows GUI, links are visible and if they represent a URL they are clickable.

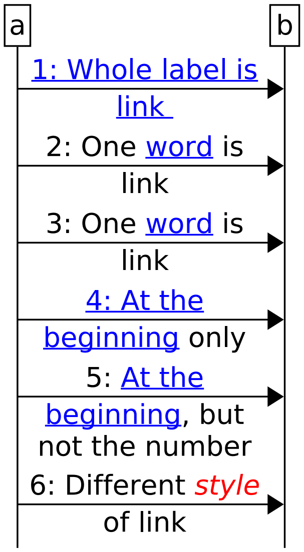
To add a hyperlink to a label, can Use the `\L()` escape in pairs, making the text between them to point to the link target. The target of the link shall be specified as the parameter to the first `\L()` escape. In this case the link target may not contain closing parenthesis[11]. Using this method it is possible to add several links to (different parts of) the same label.

In signalling charts, you can also use the `url` attribute. This makes the whole label a hyperlink. The value of the attribute is the target of the link. You cannot use both mechanisms to the same label. Note that you can only use the first method to add hyperinks to *box tags*.

Links also change formatting. In the plain design, they became blue and underlined. In signalling charts, this is governed by the `text.link_format` chart option and attribute. Any formatting escape sequence you specify as value to this chart option or attribute will be applied at the beginning of the link text and de-applied at the end.

[11] If the link target contains opening brace, hashmark, semicolon or symbols, which terminate a colon label, use quotation marks around the label - it is not possible to use escapes such as `\[` inside a link target.

```
hscale=1.2;
text.wrap=yes;
numbering=yes;
a->b: Whole label is link
    [url="http://abc.com"];
a->b: One \L(http://abc.com)word\L() is link;
a->b: "One \L(http://abc.com/#x)word\L() is link";
a->b: \L(\ref note)At the beginning\L() only;
a->b: \|\L(\ref note)At the beginning\L(), but
      not the number;
a->b: Different \L(target)style\L() of link
    [text.link_format="\c(red)\I"];
```



6.5 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, style or procedure definition 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 6.6 [Defining Styles], page 64.)

In signalling charts 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...
```

Enclosing a set of elements in braces results in exactly the same layout as in case when they are not enclosed in braces (including the handling of `compress`, `vspacing`, `keep_with_next` and `keep_together` attributes and the use of `parallel` and `overlap` keywords)[12] in

[12] This is true only if you do not change the layout of the block, but use the default, see Section 7.9 [Parallel Blocks], page 97.

signalling charts. Thus if you mark an element between the braces with `parallel` elements after the closing brace can be laid out besides it[13].

6.6 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 syle are left unchanged.

In case of graphs, any attribute can be added to a style, while for signalling charts only the ones listed in Section 7.10 [Signalling Chart Attributes and Styles], page 102. 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 signle command. 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.

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

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 or all edges in a graph. This is how chart designs operate: by modifying the built-in styles.

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.

[13] However, you get extra tools, since marking the entire block with `overlap` or `parallel` will make elements after the block to be laid over or besides the whole block, respectively. See Section 7.9.1 [Parallel Keyword], page 100

In addition to default styles, most languages have a number of *refinement styles*. Such styles may contain some attributes set and are applied after and in addition to default styles. For example, after applying the default `arrow` style to a message in a signalling chart, specified as `a=>b;`, an additional refinement style, named `=>` is applied, making the arrow double-lined.

Redefining refinement enables you to quickly define, e.g., various arrow styles and use the various symbols as shorthand for these.

Thus, in summary the actual attributes of an element are set using the following logic. (There are minor variances for each language.)

1. If you specify an attribute directly at the element (perhaps via applying a style), the specified value is used[14].
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.
3. Otherwise, if the attribute is set in the default style of the element, the value there is used.
4. Otherwise (if they exist in the language), the value of the applicable chart option is used, e.g., `text.*` for signalling charts.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.

6.7 Chart Designs

A chart design is a collection of color, style and procedure definitions, and for signalling charts, 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.

Note that color, style and procedure are local to a scope (take effect only within) and can be stored in designs. Shape and design definitions on the other hand are global and as such can not be added to designs.

Except for graphs, we differentiate are *full designs* and *partial designs*. A full design contains a value for all the chart options, default colors and styles. A partial design contails 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 = <design_name>` chart option for signalling charts and the `usedesign <design_name>` for graphs. To apply a partial style use the `msc += <style_name>` chart option. Note that it is best to apply a full design at the first line of the chart, as any chart option specified before it will likely be overwritten by applying a full design.

Currently the following partial designs ship with Msc-generator for signalling charts: `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

[14] If you specify the attribute several times, the last one is used.

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 | usedesign parent design ]
   [ msc+=partial design ]
   options, ...
   color definitions, ...
   style definitions, ...
}
```

First you can name an existing full design to inherit from using the '`msc=`' option or via '`usedesign` ***parent design***'. If specified the design will become a full design, too. Thus in each such design definition the default styles 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` or `.graph` extension and making them available in all charts for use. See Section 5.2 [Design Library], page 37.

6.8 Defining Shapes

This section describes how you can define new shapes that can be used in entity headings. Note that shapes are not (yet) available in graphs.

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 (immaginary) 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 (immaginary) 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 (immaginary) 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 5.6 [Typing Hints and Autocompletion], page 40). 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 5.2 [Design Library], page 37 to see where to find and put your own files defining Shapes.

6.9 Procedures

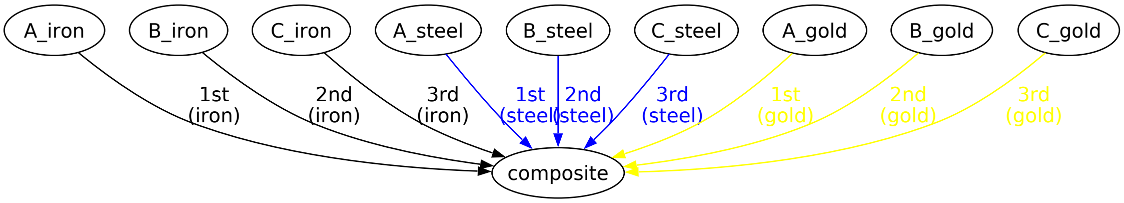
Procedures allow you to store some chart text and replay it later. They are useful to save typing, promote re-use instead of copy/paste. Since procedures can take parameters their result can be customized.

To define a procedure, use the `defproc` command.

```
defproc proc_name {
    ...chart text for procedure...
};

defproc proc_name($param_name[=default value, ...) {
    ...chart text for procedure...
};
```

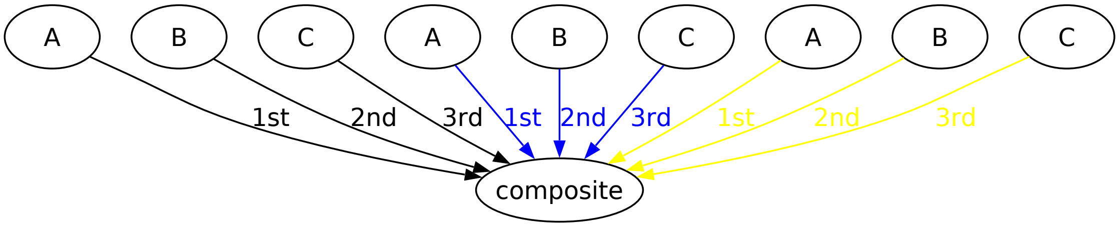
You can list parameters and their (optional) default value, as well. Parameter names should begin with the dollar sign ($). They can be used anywhere a string or number is needed, such as for attribute values, attribute names, entity and node names, etc. You can not invoke a keyword using a parameter.

```
defproc addgroup($group_name, $target, $color=black) {
    edge [color=$color];
    A_~$group_name->$target:: \c($color) 1st\n(\Q($group_name));
    B_~$group_name->$target:: \c($color) 2nd\n(\Q($group_name));
    C_~$group_name->$target:: \c($color) 3rd\n(\Q($group_name));
}

graph {
    composite;
    replay addgroup(iron, composite);
    replay addgroup(steel, composite, blue);
    replay addgroup(gold, composite, yellow);
};
```



The \Q() text formatting escape can be used to substitute a parameter value into a label. The tilde (~) character can be used to append or prepend string constants to a parameter, enabling string composition. In graphviz the + character can also be used. The double-dollar special symbol ($$) can be used to denote an string that is always unique for each procedure replay but when printed it is empty. This can be used, when you want to define a node (or entity or marker for signalling charts) for each time the procedure is replayed, but you do not want to pass a procedure parameter to compose with (such a $group_name above).

```
defproc addgroup($color=black, $target) {
    edge [color=$color];
    A~$$->$target:: \c($color) 1st;
    B~$$->$target:: \c($color) 2nd;
    C~$$->$target:: \c($color) 3rd;
}

graph {
    composite;
    replay addgroup(, composite);
    replay addgroup(blue, composite);
    replay addgroup(yellow, composite);
};
```



As you can see you can fall back to the default value of parameters, even if they are not the last one.

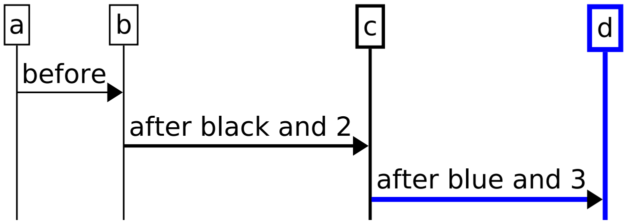
It is possible to replay a procedure within another procedure. Do not use recursion. It is not allowed to define a procedure in another procedure, however.

When you refer to a style, color or procedure name, or an entity, node or marker from within a procedure definition, their value will be substituted at the place of the replay. Thus their existence cannot be verified at the time of definition. A full parsing and verification is performed at replay. In order to avoid repeating error messages as much as possible, if an error is already caught at procedure definition time, the procedure cannot be replayed.

Color Syntax Highlighting is also limited for procedures, for example, styles cannot be determined to exist, so they are never marked for error.

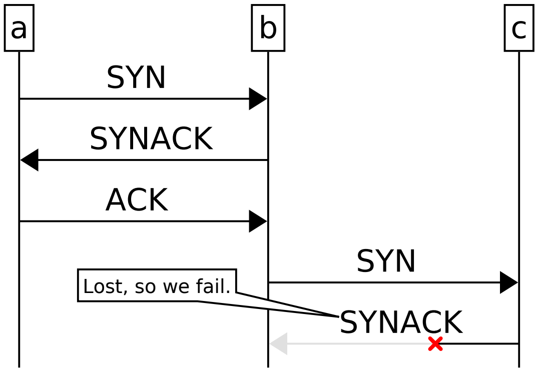
Styles and colors defined inside a procedure are local to that procedure (since the procedure has its own scope). If you want to export these after a replay, you can use the `[export=yes]` attribute at the procedure definition. This enables you to create procedures that define styles based on some parameters.

```
defproc adjust_line($color, $width, $weak_width=1) [export=yes] {
    defstyle arrow, blockarrow, vertical, pipe, box, emptybox, divider,
        symbol, indicator, note, vertical_pointer, vertical_range,
        vertical_bracket, vertical_brace
        [line.width=$width, line.color=$color];
    defstyle box_collapsed, box_collapsed_arrow, entity, entitygroup,
        entitygroup_collapsed
        [line.width=$width, line.color=$color];
    defstyle box, box_collapsed, emptybox
        [tag.line.width=$width, tag.line.color=$color];
    defstyle entity, entitygroup_collapsed
        [vline.width=$width, vline.color=$color];
    defstyle weak [line.width=$weak_width, vline.width=$weak_width];
};
hscale=auto;
a->b: before;
replay adjust_line(black, 2);
b->c: after black and 2;
replay adjust_line(blue, 3);
c->d: after blue and 3;
```



Finally, inside procedures you may use conditional statements that execute or don't based on the value of some parameters. After the `if` statement you can use either a single parameter or two strings (potentially composed of multiple substrings with `~`) compared with one of the following operators: `<=`, `<`, `==`, `>`, `>=` and `<>` (the latter for non-equal). String comparison is UTF-8 and not numeric (thus "2" is larger than "10"). If you use only a single parameter, the condition is true if the parameter is any non-empty string.

```
defproc threeway($e1, $e2, $note="")
{
    $e1->$e2: SYN;
    if $note then {
        $e1<-*$e2: SYNACK;
        note [label=$note];
    } else {
        $e1<-$e2: SYNACK;
        $e1->$e2: ACK;
    };
};

replay threeway(a, b);
replay threeway(b, c, "Lost, so we fail.");
```



Note that you can use braces to include more than one statement for the then or else branches.

6.10 File Inclusion

It is possible to include the text of another file in the current chart, similar to how it works in the C preprocessor. Use the `include` command followed by a filename in quotation marks. Currently the file is searched only in the directory of the file containing the `include` command. You can use relative or absolute paths to include files from other directories.

On Windows, you can use both forward and backward slashes to separate directories, but due to text escape sequences you need to double backslahes. Thus on Windows both are valid syntax:

```
include "../file.signalling";
include "..\\file.signalling";
```

You can include other files from included files, in arbitrary depth.

When you issue an `include` command, parsing continues in the specified file and returns to the original one - just after the include command. Certain errors (such as opening braces without closing pairs) may cause errors in the original file - try including only totally correct files.

Note that Color Syntax Highlighting does not parse include files. Any styles, colors or procedures defined there may be marked as undefined in the original file.

Note that if you want to define a set of procedures, designs or shapes you frequently use, in the Windows GUI it is better to place them in a design library (Section 5.2 [Design Library], page 37) as opposed to including them at the beginning of your file. Design libraries are pre-parsed when the GUI starts up and are not re-parsed again at compilation. In addition any styles or designs defined in them are taken into account by Color Syntax Highlighting.

7 Signalling Chart Language Reference

7.1 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.

7.2 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 litte 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.

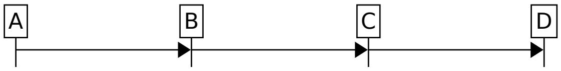
7.2.1 Entity Positioning

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.

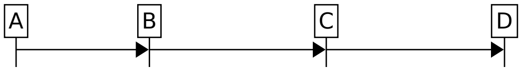
Specifying `pos` will place the entity left or right from its default location. E.g., specifying `pos=-0.25` for entity `B` makes `B` to be 25% closer to its left neighbour. Thus `pos` shall be specified in terms of the unit distance between entities. (Which is 130 points - a historic value kept for 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.

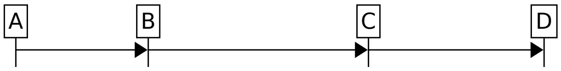
```
A, B, C, D;
A->B-C-D;
```



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

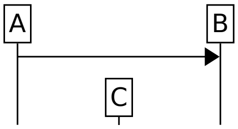


```
A, B [pos=-0.25], C [pos=+0.25], D;
A->B-C-D;
```



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 positining 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 7.11 [Chart Options], page 110. 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.

7.2.2 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[1]. 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 6.5 [Scoping], page 63 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

[1] 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.

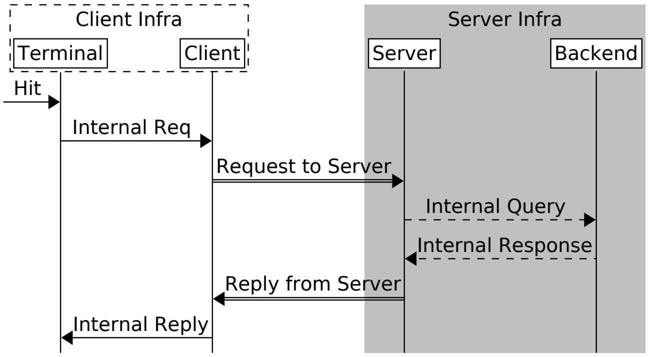
made a group entity later by adding entities to it. Nor can a group be later extended with additional entities.

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.

Setting the `large` attribute makes the group entity span the entire height of the chart in the background as opposed to drawing a group entity heading. See the difference below. This is in effect only if the group entity is not collapsed.

```
hscale=auto;
CI: Client Infra {
    T: Terminal;
    C: Client;
};
SI: Server Infra [large=yes] {
    S: Server;
    B: Backend;
};
->T: Hit;
T->C: Internal Req;
C=>S: Request to Server;
S>>B: Internal Query;
S<<B: Internal Response;
C<=S: Reply from Server;
T<-C: Internal Reply;
```



7.2.3 Entity Attributes

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

`label` This specifies the text to be displayed for the entity. It can contain multiple lines or any text formatting character. See Section 6.3 [Text Formatting], page 59. If the label contains non alphanumeric characters, it must be quoted between double quotation marks. The default is the name of the entity.

`pos` 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 `relative` 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.

`relative` This attribute takes the name of another entity and specifies the horizontal position used as a base for the `pos` attribute. Grouped entities cannot have this attribute.

`shape` This attribute takes the name of the shape you want for the entity headings. See Section 7.2.6 [Entity Shapes], page 75.

`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.

`collapsed`
This attribute can be used to collapse a group entity. Only group entities can have this attribute.

`indicator`
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.

`show` 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 7.2.5 [Entity Headings], page 75.

`active` This is a binary attribute, defaulting to no. If set the entity line becomes a thin long rectange 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. Uisng 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 6.1 [Common Attributes], page 52 for the description of these attributes.

`aline.*`
`atext.*`
`arrow.*` These attributes can be used to set the line, text and arrowhead attributes of arrows starting from this entity. Their use is described in Section 7.3.3 [Arrow Appearance], page 83. Note they do not apply to block arrows.

7.2.4 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 7.11 [Chart Options], page 110. Disabling implicit definition is useful to generate warnings for mis-typed entity names[2].

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

7.2.5 Entity Headings

By default, when an entity is defined, its heading is drawn at that location. If the entity name is preceeded 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 preceeded 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 7.14 [Commands], page 119. However, the 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).

7.2.6 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 6.8 [Defining Shapes], page 66 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.

7.3 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.

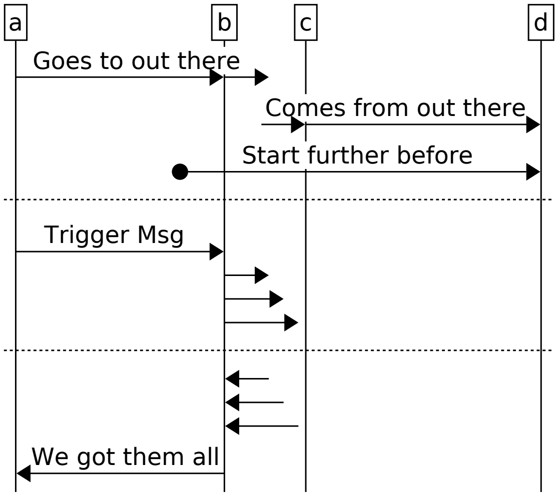
[2] To this end, color syntax highlighting underlines an entity name appearing the first time. This allows quickly realizing if the name of an entity is misspelled.

`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 6.6 [Defining Styles], page 64. There are two more arrow symbols for you to re-define: '`==>`' and '`=>>`' (with reverse and bi-directional variants, too: '`<==>`' and '`<<=>>`'). These default to double lines, with the latter having sharp arrowheads, as well.

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. Using the pipe symbol '`|`' as start or endpoint instead will still make the arrow go to (or come from) no specific entity, but not at the very end of the chart, but immediately after the last entity (or before the first one).

```
hscale=auto;
a, b, c, d;
a->b-|: Goes to out there;
|->c-d: Comes from out there;
->d start before b:
    Start further before
    [arrow.starttype=dot];
---;
a->b: Trigger Msg;
b-> end after b;
b-> end after b +10;
b-> end after b +20;
---;
b<- start before b;
b<- start before b +10;
b<- start before b +20;
b->a: We got them all;
```



You can also use the `start before` and `end after` keywords after the arrow specification to precisely indicate where the arrow should start and/or end. You need to specify an entity afterwards and optionally an offset. The terms '`before`' and '`after`' should be understood in the direction of the arrow, so if an arrow is right-to-left, then `start before X` makes the arrow start right of entity `X`. A positive offset moves the arrow endpoint even further from the entity specified. (In the example above `start before X +10` would move the start of the arrow even more to the right. The offset is specified in pixels. The default distance from the entity (to which offset is added) is 30 pixels times the value of the `hscale` (and exactly 30 for `hscale=auto`).

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 by default, but they can also impact the arrowhead, text or other attributes, see Section 7.3.3 [Arrow Appearance], page 83 below.

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.

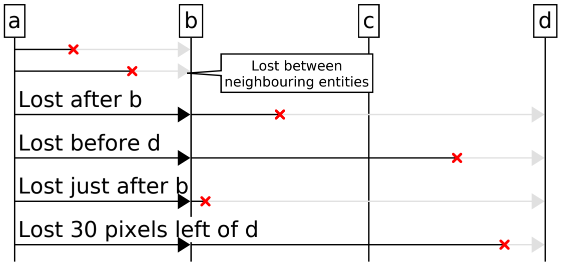
Arrows can also be defined starting and ending at the same entity, e.g., `a->a;`. In this case the arrow will start at the vertical line of the entity and curve back to the very same line. Such arrows cannot be multi-segmented.

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).

7.3.1 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 `small` 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, specifly a narrower line or empty arrowheads.

7.3.2 Arrow Attributes

Arrows can have the following attributes.

`label` This is the text associated with the arrow. See Section 6.1.5 [Labels], page 56 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.

`text.*` All text formatting attributes described in Section 6.1 [Common Attributes], page 52 can be used to manipulate the appearance of the label[3].

`number` Can be set to `yes`, `no` or to a number, to turn numbering on or off, or to specify a number, respectively. See Section 7.10.3 [Numbering], page 105.

`refname` 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 `\r(name)` escape in labels to insert the number of the referenced arrow. See Section 7.10.3 [Numbering], page 105.

`compress` Can be set to `yes` or `no` to turn compressing of this arrow on or off. See Section 7.10.4 [Compression and Vertical Spacing], page 108.

`vspacing` Can be set to a number interpreted in pixels or to the string `compress`. Governs how much vertical space is added before the arrow (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`.

`angle` 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 `angle` chart option (or is zero in the absence of such an option, which corresponds to horizontal arrows).

`slant_depth`
This is similar in effect to the `angle` 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 `angle` attribute and `slant_depth` the latter takes effect.

[3] 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.

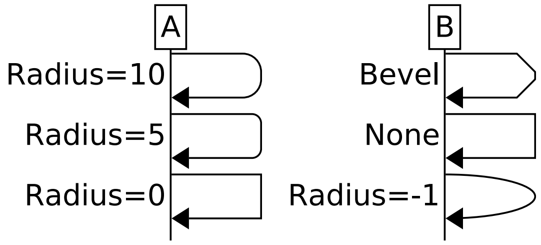
`color` This specifies the color of the text, arrow and arrowheads. It is a shorthand to setting `text.color`, `line.color` and `arrow.color` to the same value.

`line.color, line.width`
Set the color and the width of the line, see Section 6.1 [Common Attributes], page 52.

`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.

```
 {
   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];
 };
hspace A-B 150;
```



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

`arrow.color`
The color of the arrowheads.

`arrow.type`
Specity 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. Even more pointed are `vee` and `empty_vee`. `diamond` and `empty_diamond` draws a filled or empty diamond, while `dot` and `empty_dot` draws a filled or empty circle. The last four has non-symmetric versions, where the dot or diamond is not drawn on the entity line, but before it: `nsdiamond`, `empty_nsdiamond`, `nsdot` and `empty_nsdot`. 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.

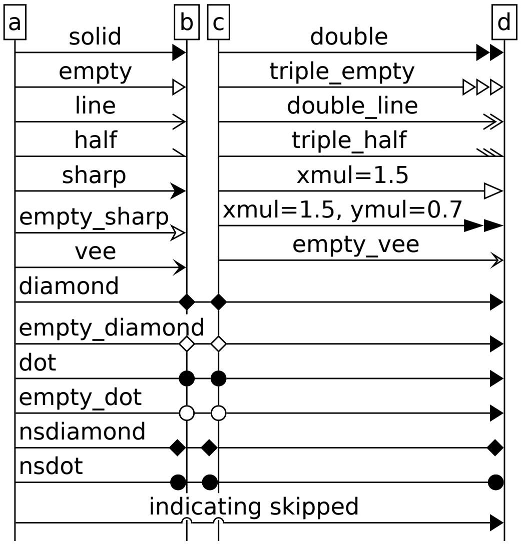
`arrow.midtype`
This attribute sets the arrowhead type used for intermediate entities of a multi-segment arrow. It defaults to a filled triangle.

`arrow.skiptype`
This attribute specifies what to draw for entities the arrow passes over but does not stop at. E.g., if we have three entities ‘`a`’, ‘`b`’ and ‘`c`’, then an arrow specified as `a->c`, will pass over entity ‘`b`’. This attribute defaults to none, but another suitable value is `jumpover`, which draws a small half-circle.

`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 multipiers, 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];
a->b: vee[arrow.type=vee];
} {
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];
c->d: empty_vee[arrow.type=empty_vee];
};
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];
a->b-c-d: \plnsdiamond [arrow.type=nsdiamond];
a->b-c-d: \plnsdot [arrow.type=nsdot];
a->d: indicating skipped [arrow.skiptype=jumpover];
```



`arrow.gvtype`
`arrow.gvendtype`
`arrow.gvmidtype`
`arrow.gvskiptype`
`arrow.gvstarttype`

Use these attributes to set the corresponding arrowhead using the graphviz-style arrowhead specification syntax. This syntax is a superset of what graphviz supports and is not really meaningful for block arrows. Setting any of `arrow.*type` or `arrow.gv*type` will erase the other attribute, since these govern the same property of the arrow.

You have a set of basic forms `normal` (a regular filled triangle), `inv` (filled triangle in inverse direction), `vee` (a v shape), `crow` (v shape in the reverse direction), `tee` (a thin line perpendicular to the arrow line), `box` (a square), `diamond`, `dot`, `curve` (a half circle line) and `icurve` (in the reverse direction). Msc-generator supports the following additional basic forms: `sharp`, `line` and `jumpover`, which are the same as for the '`arrow.type`' attribute above; and `sbox`, `sdiamond` and `sdot`, which are symmetric versions of `box`, `diamond` and `dot`, respectively.[4]

You can combine the basic forms by appending them, such as `normalnormal`, which is a double filled triangle. You can create arbitrary long combinations. The first form specified will be the tip of the arrow closest to the entity line with the remaining forms following behind.

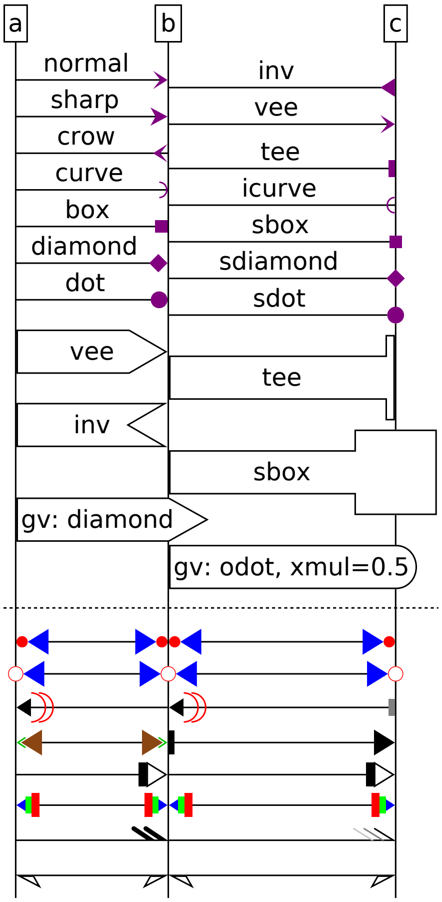
Some of the forms may be prepended by the letter `o` to create a non-filled version, such as `odot` or `onormal`. You can use the `l` or `r` letters to draw only half of the arrowhead which falls on one side of the arrow line, such as `lbox`, `rcurve` or even `olnormal`.

Inserting a size modifier will change the size of the subsequent forms, such as `spotdotsmallnormal`, will draw a smaller dot and a larger triangle. Since the term '`normal`' is both a size and a type, if you mean the size, use `sizenormal`. In addition to size modifiers, you can also insert any color name. Finally, you can also insert one of the following attributes: `lwidth` to change the line width of the arrowhead; `ltype` to change the line type of the arrowhead; `color` to change the color of the arrowhead (e.g., to an rgb value); `xmul`, `ymul` or `mul` to apply a (cumulative) multiplier to the x or y dimension of the arrowhead or to both. These attributes must be terminated with a pipe symbol (`|`) and when using them the attribute value must be enclosed in quotation marks, such as `arrow.gvtype="xmul=0.2|normal|color=12,34,56|dot"`. See the examples below.

Note that you can use these attributes to set arrowhead style for block arrows, but in that case only one form can be given (you can use `stripes` and `triangle_stripes`, but cannot use `line`, `curve`, `icurve`, `jumpover` and `crow`. Also, line/color attributes have no effect. All in all, you are better off not using the graphviz syntax for block arrows.

[4] This is because `box`, `diamond` and `dot` in graphviz are drawn before the target, in conflict with Msc-generator beahviour for '`arrow.type`'. Thus, if you specify '`arrow.type=dot`', you will get a dot centered on the arrow line; while if you use '`arrow.gvtype=dot`', you will get one before the entity line.

```
hscale=auto;
{
    compress=yes;
    defstyle arrow [arrow.color=purple];
    a->b [arrow.gvtype=vee]: normal;
    b->c [arrow.gvtype=inv]: inv;
    a->b [arrow.gvtype=sharp]: sharp;
    b->c [arrow.gvtype=vee]: vee;
    a->b [arrow.gvtype=crow]: crow;
    b->c [arrow.gvtype=tee]: tee;
    a->b [arrow.gvtype=curve]: curve;
    b->c [arrow.gvtype=icurve]: icurve;
    a->b [arrow.gvtype=box]: box;
    b->c [arrow.gvtype=sbox]: sbox;
    a->b [arrow.gvtype=diamond]: diamond;
    b->c [arrow.gvtype=sdiamond]: sdiamond;
    a->b [arrow.gvtype=dot]: dot;
    b->c [arrow.gvtype=sdot]: sdot;
    nudge;
    block a->b [arrow.gvtype=vee]: vee;
    block b->c [arrow.gvtype=tee]: tee;
    block a->b [arrow.gvtype=inv]: inv;
    block b->c [arrow.gvtype=sbox]: sbox;
    block a->b [arrow.gvtype=diamond]:
         gv: diamond;
    block b->c [arrow.gvtype="xmul=0.5|odot"]:
         gv: odot, xmul=0.5;
};


---;

defcolor brown=139,69,19;
a<->b-c [arrow.gvtype=tinyreddotsizenormalbluenormal];
a<->b-c [arrow.gvtype="smallredlwidth=0.5|osdot|lwidth=1|sizenormalbluenormal"];
a<-b-c [arrow.gvtype=smallnormalbigredicurveredicurve,
        arrow.gvstarttype=graytee];
a<->b [arrow.gvendtype="tiny|color=0,192,0|linesizenormalbrownnormal"];
join b->c [arrow.size=normal, arrow.gvstarttype="xmul=0.5|tee"];
a->b-c [arrow.size=normal, arrow.gvtype=onormaltee];
a<->b-c [arrow.gvtype=bluetinynormalgreensmallteeredsizenormaltee];
a->b-c [arrow.gvendtype=llinegrayllinelgraylline,
        arrow.gvmidtype="lwidth=3|llinelline",
        arrow.size=normal];
a<->b-c [arrow.size=normal, arrow.gvendtype=orsharp];
```



`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 6.6 [Defining Styles], page 64.

7.3.3 Arrow Appearance

There are a number of factors influencing how an arrow appears. In this section we describe the order of precedence among them. See Section 6.6 [Defining Styles], page 64 for more information on styles.

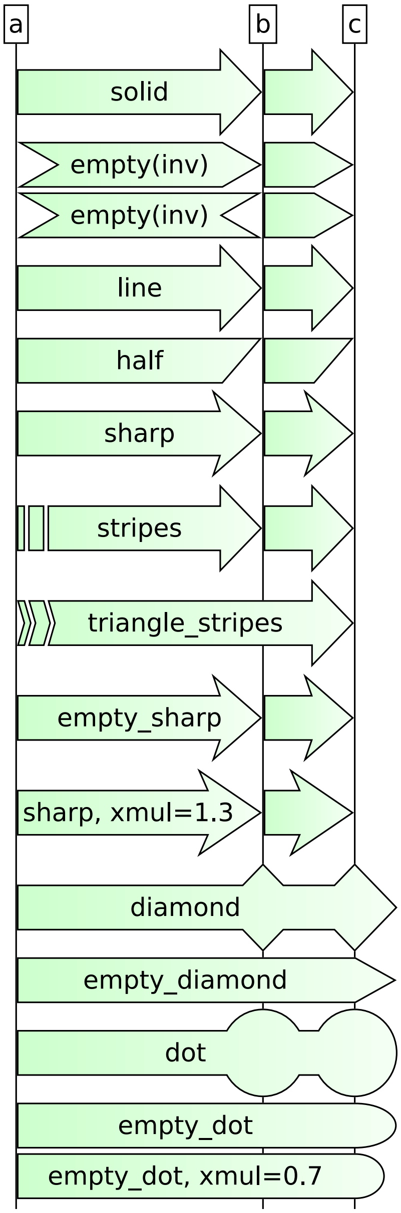
1. Attributes specified after the arrow definition always take precedence.
2. If an attribute is not specified after the arrow definition, it takes its value from a refinement style, such as `->` or `=>`. In case of multi-segment arrows, which use multiple types of arrow symbols, such as `a->b=>c` each refinement style is applied in the order the arrow points. Thus in the case of `a->b=>c` any attribute specified in style `=>` takes precedence over the value (if any) specified in `->` for the same attribute. The same order (the order towards the destination end of the arrow) is applied even if the arrow is specified backwards, as `c<=b<-a`. For bidirectional arrows, the order is as written in the input file. Line attributes are special, because they are also recorded and applied to the respective segments. (This is why normally refinement styles only specify line attributes. Specifying the arrowhead could also make sense, but those are not applied to individual segments, but to the whole arrow.)
3. If a line, text or arrowhead attribute is not specified in any of the refinement styles applied it takes its value from the `aline.*`, `atext.*` or `arrow.*` attribute of the source entity, respectively. (For bidirectional arrows, first entity written in the text file.) Since refinement styles usually overwrite the line type, it makes more sense to set e.g., line color in these attributes, if you want to automatically highlight arrows starting from a specific entity.
4. If an attribute is not specified via any of the ways above, it takes its default value from the `arrow` default style. (Chart designs change this to impact all arrows.) This style is guaranteed a value for all attributes, except `text.*`.
5. Finally, unspecified `text.*` attributes take their values from the value set in the `text.*` chart options (used to globally set the font for example) or, if none set, a default font.

7.3.4 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 that 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.

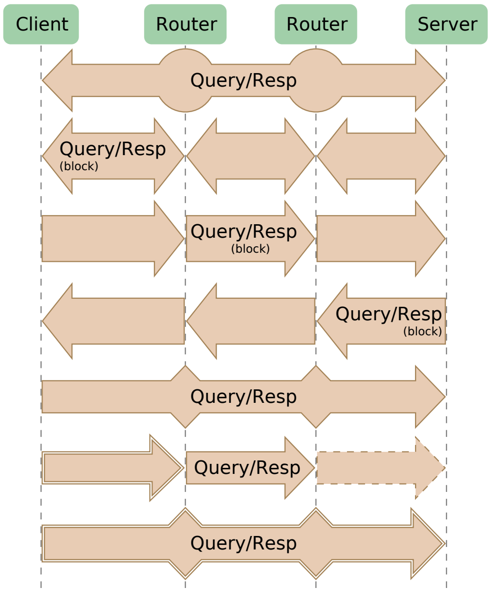
```
hscale=auto;
defstyle blockarrow [fill.color="green+80",
                     fill.gradient=right];
block a->b-c: solid [arrow.type=solid];
block a->b-c: empty(inv)  [arrow.type=empty,
                arrow.starttype=empty_inv];
block a->b-c: empty(inv)[arrow.endtype=empty,
                   arrow.starttype=empty_inv,
                    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 do not accept the `arrow.skiptype` attribute and are not impacted by setting the `aline.*`, `atext.*` or the `arrow` attribute of their source entity. However, they can be slanted using the `angle` attribute.

7.4 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*.

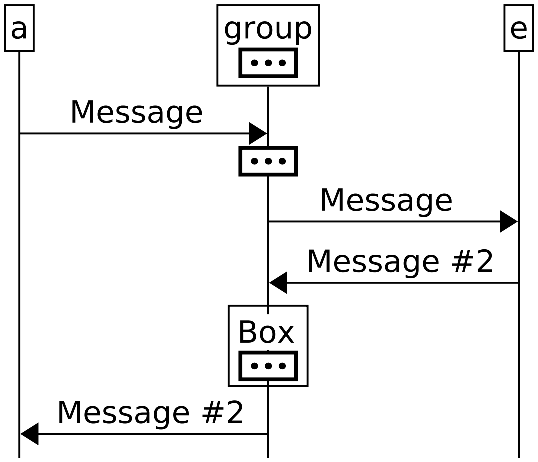
The syntax definition for boxes is as follows.

```
[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;
```



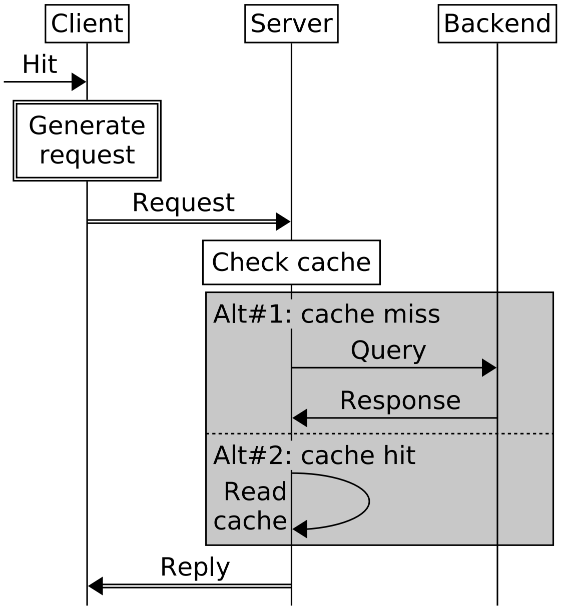
Boxes take attributes, controlling colors, numbering, text identation quite similar to arrows. Specifically boxes also have a `label` attribute that can also be shorthanded, as for arrows. For example: `..: Auto-adjusting empty box;` is a valid definition. The valid box attributes are `label`, `number`, `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 identation 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.

7.4.1 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\ncache;
};
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.

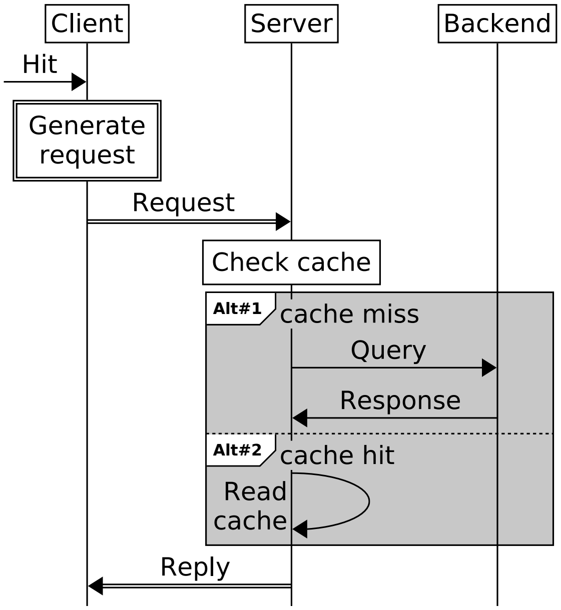
Boxes can be collapsed, similar to group entities. The ‘`indicator`’ attribute governs if collapsed boxes show a small indicator to indicate that there is hidden content inside.

7.4.2 Box Tags

Boxes can also have *tags*, which are smal l 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\ncache;
};
C<=S: Reply;
```



7.5 Pipes

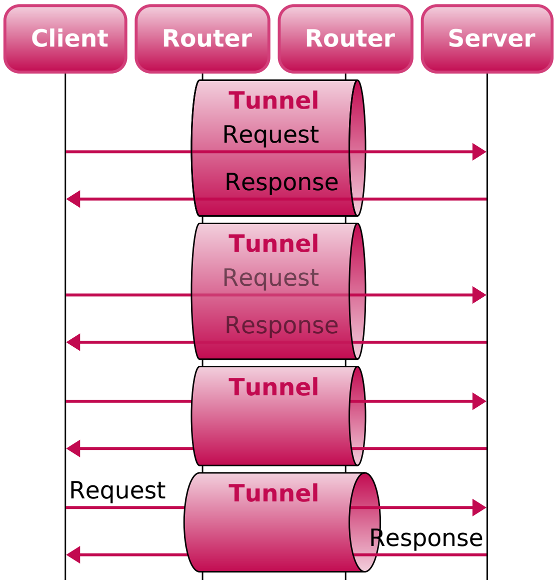
By typing **pipe** in front of a box definition, it is turned into a pipe. Pipes can represent tunnels, encapsulation or other associations (e.g., encryption) in networking technologies. Using them one can visually express as messages travel within the tunnels or along other associations.

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[5].

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.

[5] 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;
};
```

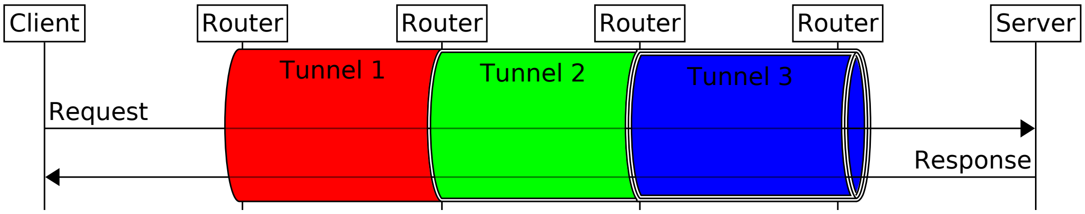


On the example above one can observe, that the last two pipes are smaller than the first two, even though they have exactly the same two arrows within. This is because in case of the first two arrows the label of the pipe itself is visible at together with the two arrows within. In contrast, the last 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 6.6 [Defining Styles], page 64.

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;
};
```



7.6 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).

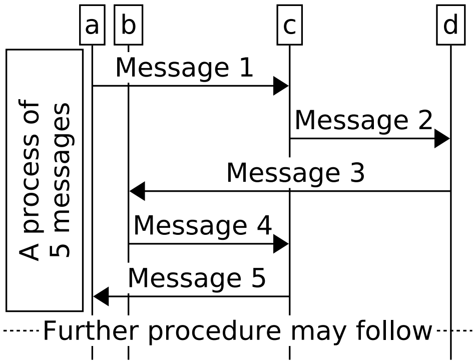
```
vertical [shape | [from] symbol [to]] [at position] [attributes...];
```

The *shape* keyword can be one of `box`, `block 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. (See more on marking at the end of this section.) The third line of the example below places a marker named `top` just below the enitiy 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 top the top and bottom of the entire chart, respectively.

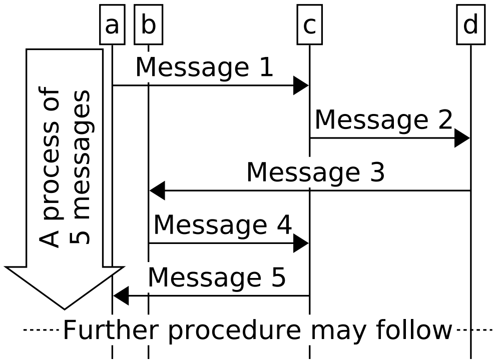
```
hscale=auto;
a, b, c, d;
mark top;
a->c: Message 1;
c->d: Message 2;
d->b: Message 3;
b->c: Message 4;
c->a: Message 5;
vertical top-- at a-:
        A process of\n5 messages;
---: Further procedure may follow;
```



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 `->`[6]. 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;
```

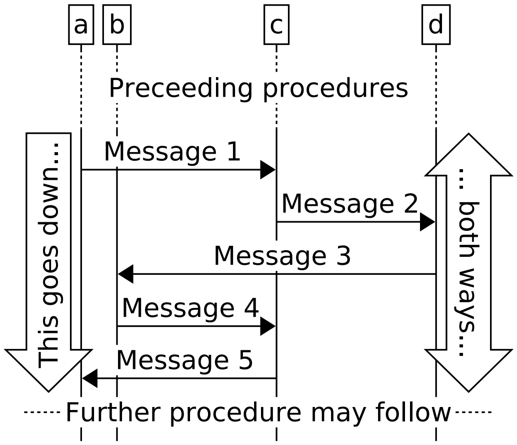


Verticals can contain a label, which can be rotated 90 degrees compared to other elements. This can be set via the `side` attribute, which specifies from which direction the text is readable from (`left`, `right` or `end`, which means the regular horizontal typeset). If you sent `side=end` to typeset 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.

[6] 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;
...: Preceeding 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;
```

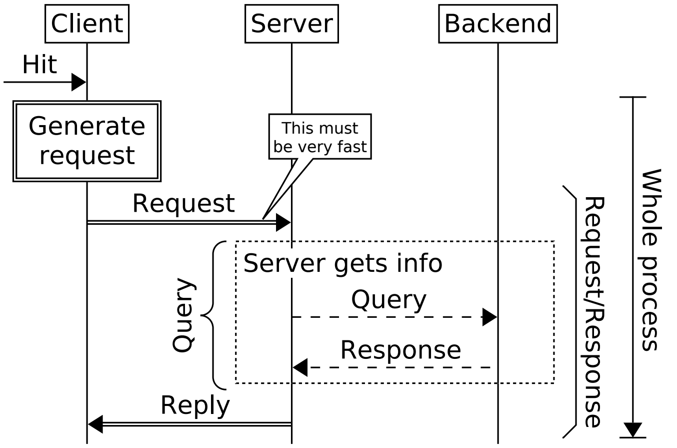


You can also omit the `at` clause, which results in the vertical being placed besides the entities it spans (by default on the right side of them using '`<entity>+`'). If, however, the `side` attribute is set to `right` (to mean that the text is readable from the right direction), the vertical is placed left of the entities is 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)[7]. 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).

[7] Note that Msc-generator does not lay out verticals entirely correctly in relation to parallel blocks.

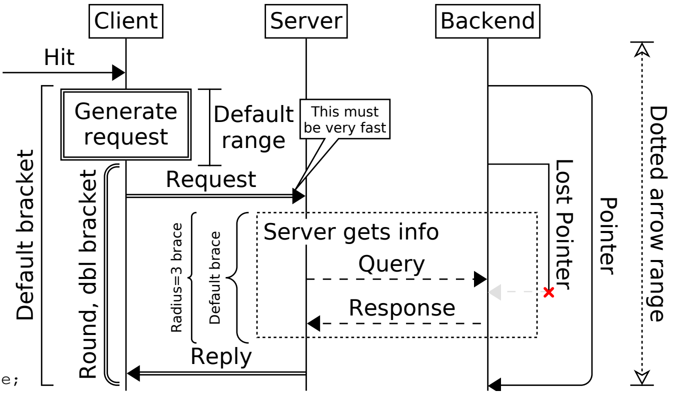
```
C: Client;
S: Server;
B: Backend;
->C: Hit;
{
    box C==C: Generate\nrequest;
    {
        C=>S: Request;
        note: This must\nbe very fast;
        box S..B: Server gets info
        {
            S>>B: Query;
            S<<B: Response;
        };
        vertical brace at S-: Query;
        C<=S: Reply;
    };
    vertical bracket: Request/Response
        [line.corner=bevel];
};
vertical range: Whole process;
```



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 attribuet. 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.

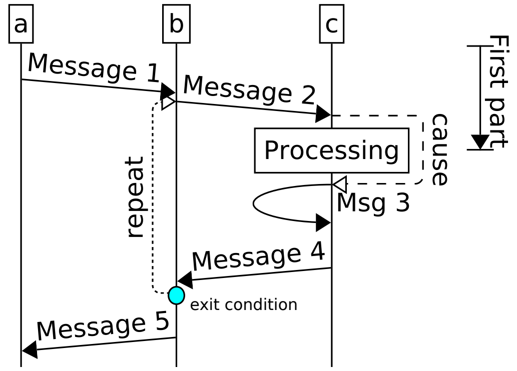
```
C: Client;
S: Server;
B: Backend;
{
    ->C: Hit;
    {
        box C==C: Generate\nrequest;
        vertical range -- [side=end]:
            Default\nrange;
        mark top2;
        C=>S: Request;
        note: This must\nbe very fast;
        mark q_top;
        box S..B: Server gets info
        {
            S>>B: Query;
            mark middle;
            S<<B: Response;
        };
        vertical brace at S-: \-Default brace;
        vertical brace at S-: \-Radius=3 brace
            [line.radius=3];
        mark bottom;
        C<=S: Reply;
    };
    vertical bracket top2=> at C-:
        Round, dbl bracket [line.corner=round];
    vertical bracket at C-: Default bracket;
    vertical pointer top2->*middle:
        Lost Pointer [lost.line.type=dashed];
    vertical pointer: Pointer [line.corner=round];
};
vertical range <>: Dotted arrow range
    [arrow.endtype = empty_sharp];
```



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 7.3.1 [Lost Messages], page 77) 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.

Finally, about the `mark` command. You can mark a position between chart elements or the centerline of an arrow or box. This can be expressed via the `centerline` attribute (the `yes` and `destination` values are equivalent.) In order for this to work, the marker must be immediately after an arrow or box. Note that other elements do not have centerline. Distinguishing source and destination centerlines makes sense only for slanted arrows or arrows starting and ending at the same entity. You can also add the `offset` attrinute to specify certain number of pixels above (negative) or below (positive) the position. This can be used in combination with centerline, if needed.

```
hscale=auto;
compress=yes;
angle = 5;
a, b, c;

mark top1;
a->b: Message 1;
b->c: Message 2;
mark src2 [centerline=source];
mark dst2 [centerline=destination];
box c--c: Processing;
mark center_box [centerline=yes];
vspace 5;
c->c: Msg 3 [side=left];
mark src3 [centerline=source];
b<-c: Message 4;
mark dst3 [offset=+5];
parallel symbol arc at b
    [fill.color=aqua, draw_time=after_default];
text at b+ +6: exit condition;
a<-b: Message 5;

vertical pointer dst2>>src3:cause
     [line.radius=5, arrow.endType=empty];
vertical pointer dst3>src2 at b-:repeat
     [line.radius=5, arrow.endType=empty];
vertical range top1->center_box at c+:First part;
```



Note that in these complex cases the layout engine makes its best attempt at automatic horizontal spacing and compression, but sometimes it fails. Please report especially annoying cases.

7.7 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.

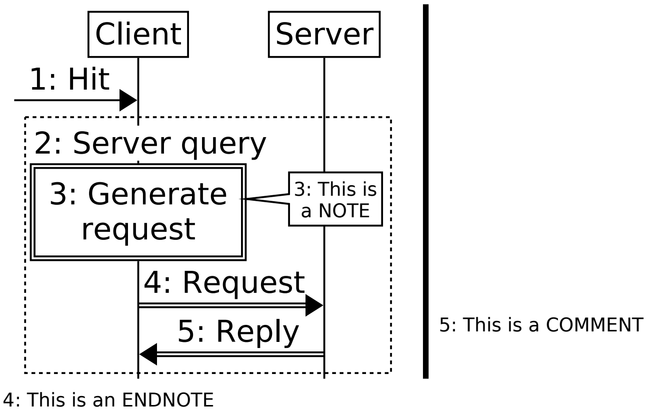
The third type of divider '`|||`' represents a simple vertical space. This can also be specified by entering just attributes in square brackets. The extreme '`[];`' simply inserts a lines worth of vertical space. You can add text, too by specifying a label. See Section 3.3 [Dividers], page 18 for examples.

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 6.6 [Defining Styles], page 64.

7.8 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.

```
msc += hcn;
C: Client;
S: Server;
->C: Hit;
box..: Server query
{
    box C==C: Generate\nrequest;
    note: This is\na NOTE;
    C=>S: Request;
    endnote: This is an ENDNOTE;
    C<=S: Reply;
    comment: This is a COMMENT;
};
```



Each note, commend and endnote has a *target element*. The target element is the element preceeding the '`note`', '`comment`' or '`endnote`' command[8]. In case of notes the tip of the callout will point to the target element, whereas side notes will be typeset beside 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.

[8] Note that some elements cannot be targets, such as chart options. In this case the preceeding element becomes the target.

7.8.1 Notes

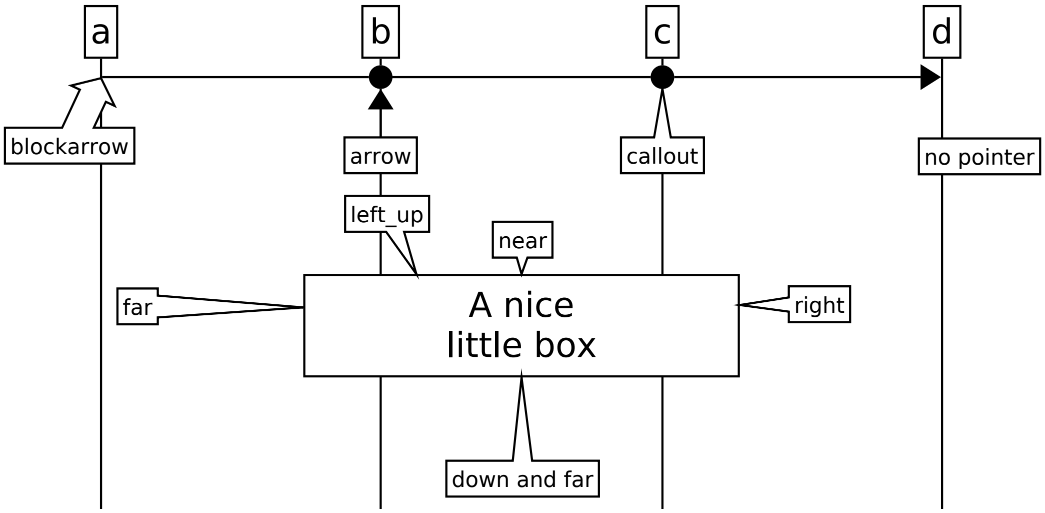
For notes the tip of the callout can be guided using the `at` keyword. After it you can spacify either an entity or a marker. This is useful if you want to make a note to a specific part of an arrow.

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 attirbutes 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;
```



7.8.2 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 `side` attribute of 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[9].

```
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 7.13.1 [Spacing], page 114.

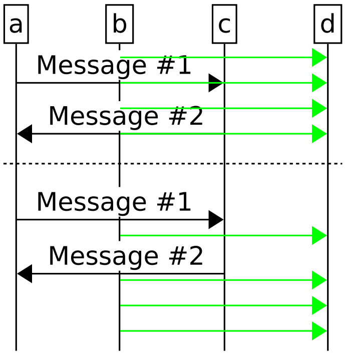
7.9 Parallel Blocks

Sometimes it is desired to express that two (or more) separate processes 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 24. You can specify as many parallel blocks as you want. The last (and only the las) parallel block shall be followed by a semicolon. The order of the blocks is not much relevant, 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. Below the series of parallel blocks the next element will be drawn after the longest of the parallel blocks.

There are two ways to lay out parallel blocks. They differ in how they handle cases when elements from the individual blocks would overlap. For non-overlapping cases they function the same way. The first algorithm, called `one-by-one` places elements from blocks one by one always taking the next element from the block which is currently the shortest (has its bottom end the highest). Elements are placed so as to avoid overlap between them. The other algorithm, called `overlap` lays out the blocks independently and allows overlap. The algorithm to use can be selected by the `layout` attribute that can be specified for the entire parallel block series before the first block.

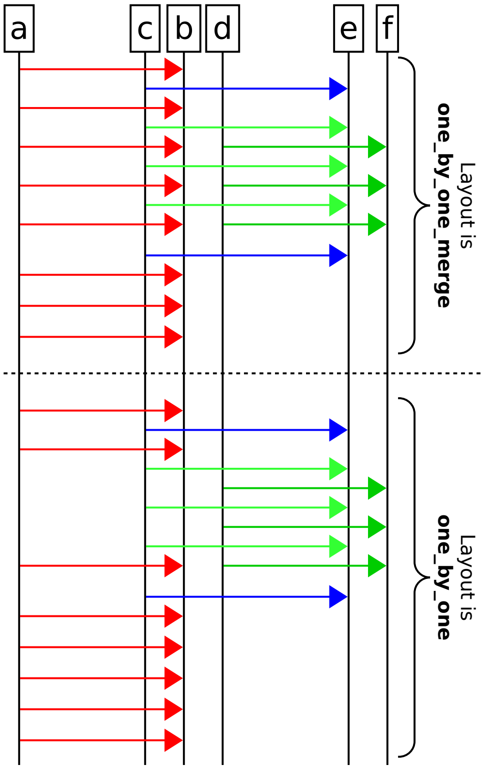
[9] These are equivalent to changing the '`comment`' style. There is no such shortcut for endnotes, yet.

```
hscale=0.5;
a, b, c, d;
[layout=overlap] {
    a->c: Message \#1;
    a<-c: Message \#2;
} {
    defstyle arrow [color=green];
    b->d; b->d; b->d; b->d;
};
---;
[layout=one_by_one] {
    a->c: Message \#1;
    a<-c: Message \#2;
} {
    defstyle arrow [color=green];
    b->d; b->d; b->d; b->d;
};
```



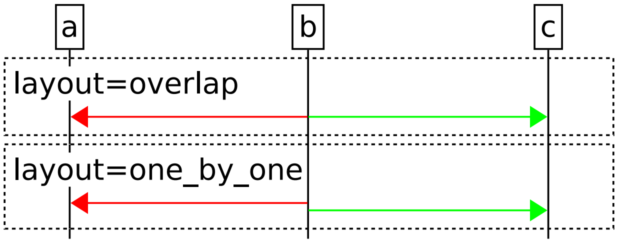
The `one_by_one` algorithm has a variant, called `one_by_one_merge`, which behaves differently in case of nested parallel blocks. If you apply this algorithm to the inner parallel block, they will be merged with the outer blocks. In contrast, `one_by_one` results in laying out the inner parallel blocks on their own as if they were a single element.

```
hscale=0.5;
a, c, b[pos=-0.7], d[pos=-0.7], e, f[pos=-0.7];

{
    defstyle arrow [color=red];
    a->b; a->b; a->b; a->b;
    a->b; a->b; a->b; a->b;
} {
    c->e [color=blue];
    [layout=one_by_one_merge] {
        defstyle arrow [color=green+20];
        c->e; c->e; c->e;
    } {
        defstyle arrow [color=green-20];
        d->f; d->f; d->f;
    };
    c->e[color=blue];
};
vertical brace: \-Layout is\n\bone_by_one_merge;
---;
{
    defstyle arrow [color=red];
    a->b; a->b; a->b; a->b;
    a->b; a->b; a->b; a->b;
} {
    c->e [color=blue];
    [layout=one_by_one] {
        defstyle arrow [color=green+20];
        c->e; c->e; c->e;
    } {
        defstyle arrow [color=green-20];
        d->f; d->f; d->f;
    };
    c->e[color=blue];
};
vertical brace: \-Layout is\n\bone_by_one;
```



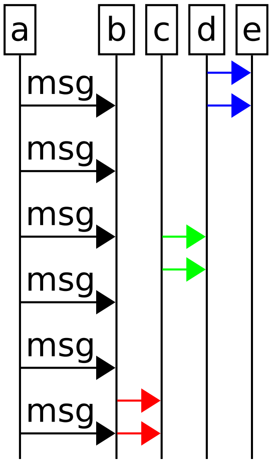
Note that with `one_by_one` and `one_by_one_merge` Msc-generator not only avoids overlap between elements, but in addition keeps a minimum distance between two elements. This means that arrows to/from the same entity cannot be drawn completely besides each other, since in that case they would touch. Thus one of them is drawn a little lower. If this is not intended, use `overlap`.

```
a,b,c;

..: layout=overlap {
    [layout=overlap]{
        a<-b [color=red];
    } {
        b->c [color=green];
    };
};
..: layout=one_by_one {
    [layout=one_by_one]{
        a<-b [color=red];
    } {
        b->c [color=green];
    };
};
```



If you use the `overlap` algorithm, you can also specify vertical alignment of the individual blocks via the `vertical_ident` attribute, which can be set to `top`, `middle` or `bottom`.

```
hscale=auto;
 [layout=overlap]{
    a->b: msg; a->b: msg;
    a->b: msg; a->b: msg;
    a->b: msg; a->b: msg;
} [vertical_ident=bottom] {
    b->c [color=red];
    b->c [color=red];
} [vertical_ident=middle] {
    c->d [color=green];
    c->d [color=green];
} [vertical_ident=top] {
    d->e [color=blue];
    d->e [color=blue];
};
```



The default behaviour is `one_by_one_merge`, which allows fine parallelism, but avoids ugly overlaps[10].

You can mark entire parallel block series, with the `parallel` and `overlap` keywords (and attributes). This results in elements after the entire parallel block series to be laid out besides and over the elements in the parallel blocks, respectively. In addition, you can set the `keep_with_next` and `keep_together` attributes to influence automatic pagination.

> Parallel block serieses also have `compress` and `vspacing` attributes. These govern, how they are laid out under the previous element. For block series with `layout=overlap` and `layout=one_by_one`, first

[10] Setting the `classic_parallel_layout` chart option to `yes` causes the default to be `overlap`, because prior v3.6 the only algorithm available was `overlap`. This chart option is now deprecated and will be removed in future releases. Use `layout=overlap` instead.

the entire block series is laid out without regard to already placed elements. Then, if compress is on (or `vspace=compress` is set, which is equivalent) the whole block series is moved upwards as one until some parts of it bump into an already placed element. If a nonzero vertical spacing is used, the whole block series is shifted down such that the requested spacing appears between the top of the block series and the prior element.

For `layout=one_by_one_merge` with `compress=yes` the first elements of the parallel blocks are individually moved upwards until they hit one of the elements above. In case a positive `vspacing` is specified, the behaviour is the same as for `layout=one_by_one`. The `compress` and the `vspacing` attribute of the first element in each parallel block can modify this behaviour (e.g., by adding vertical spacing).

Note that if the `vspacing` chart option is set to a positive number, the `vspacing` attribute of parallel blocks is set to zero instead of this number by default. This is to avoid having this vertical space twice: once for the parallel block series and once for the first elements in the blocks. You can nevertheless assign a nonzero vertical space for the block series by manually specifying the `vspacing` attribute for the parallel block. If case of `vspacing=compress` or `compress=yes` chart options, the corresponding attribute of parallel block series is set according to the value of the chart option.

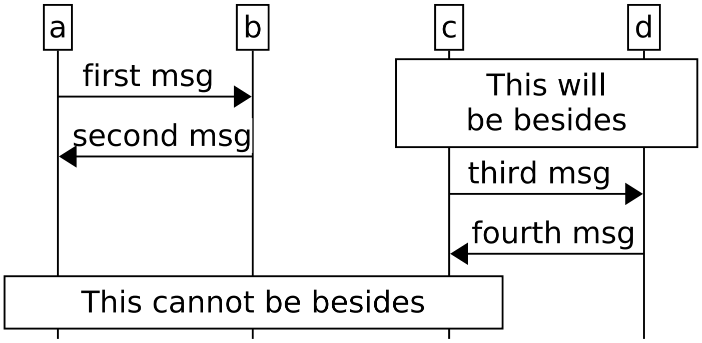
One design goal with `layout=one_by_one_merge` was that in case the parallel block series contains just one block, it should get laid out exactly as if its content were not enclosed between '`{}`' marks. This allows you to put '`{}`' marks around any set of elements, creating a new scope (see Section 6.5 [Scoping], page 63), where any changes to styles or options take effect only inside the scope.

7.9.1 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[11].

You can place `parallel` in front of really any element, including entity definitions or even series of 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;
```



[11] 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.

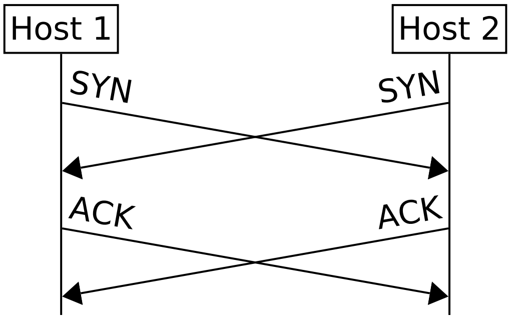
7.9.2 Overlap Keyword

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 via '`layout=overlap`' but that is a bit cumbersome for overlapping short parts. A shorthand 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;
```



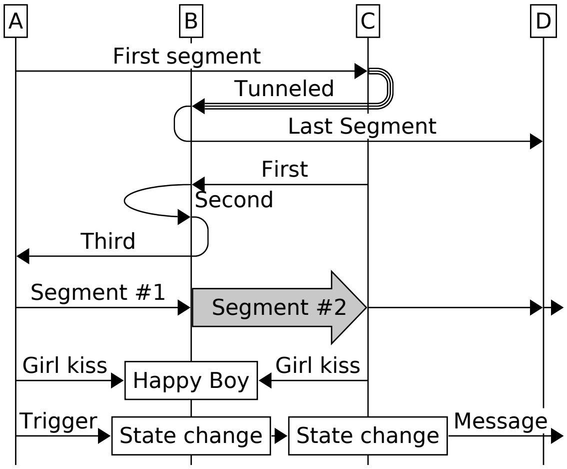
7.9.3 Joining Arrows and Boxes

The `join` keyword can be used to connect arrows to other arrows or boxes. In its simplest form one arrow can be joined to another. If the second arrow is a continuation of the first arrow and they overlap, then a curly connecting segment is added as in case of the first example below. Note that this is done only if the two arrows have the same directionality; either the second comes after the first one or they are both bi-directional; they have a common end; and they overlap. Note that the first arrow can be an arrow starting and ending at the same entity, but the second cannot. If you want that use the `side` attribute, as illustrated on the second example below.

If the two arrows do not overlap, they will be simply laid out aligned by their centerline. It is possible to mix any type of arrow, including block arrows, irrespective of directionality. The arrows do not have to connect (but cannot overlap). This construct enables you to have a different label for each joined arrow part. Note that each joined arrow may be multi-segment itself.

Finally, you can also include boxes in a join series. You can leave the end of the arrows towards the box unspecified. In this case the arrow will connect to the side of the box. Note that you cannot join a box directly to another, an arrow must come in between.

```
A, B, C, D;
A->C: First segment;
join C->B: Tunneled [line.type=triple];
join B->D: Last Segment;

C->B: First;
join B->B: Second [side=left];
join B->A: Third;

A->B: Segment \#1;
join block ->: Segment \#2 [color=lgray];
join C->D->;

A->: Girl kiss;
join box B--B: Happy Boy;
join <-C: Girl kiss;

A->: Trigger;
join box B--B: State change;
join ->;
join box C--C: State change;
join ->: Message;
```



You cannot specify `parallel` or `overlap` for an element if you specify `join`. Any `parallel` or `overlap` keyword specified for a join series, on the other hand, will apply to all elements of the join series.

7.10 Signalling Chart Attributes and Styles

7.10.1 Appearance

`line.*` Specifies the apprarance of the line for the element. For arrows and dividers it is the horizontal line. For block arrows, boxes, pipes and entities this is the line around the element. `line.radius` effects only on arrows starting and ending in the same entity (see Section 7.3.2 [Arrow Attributes], page 78); for entities and boxes, this specifies the size of the corners. For pipes, it specifies the width of the oval, in other words from how left we look at the pipe.

`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.*` Defines the fill of the box, entity, block arrow or pipe.

`shadow.*` Defines the shadow for boxes, entities, block arrows and pipes. Currently no shadow can be specified for simple arrows.

`arrow.*` Arrowhead formatting attribues, described in detail in Section 7.3.2 [Arrow Attributes], page 78.

`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 7.3.1 [Lost Messages], page 77. Only applicable for arrows.

`x.size`
`x.line.*` The controls the appearance of the loss symbol for lost messages, see Section 7.3.1 [Lost Messages], page 77.

`tag.line.*`
`tag.fill.*`
`tag.text.*`
These attribues applie 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 7.2.6 [Entity Shapes], page 75. They can be made part of style but have effect only on enities.

`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 7.8 [Notes and Comments], page 95 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 7.5 [Pipes], page 88 for more information.

`number` This attribute giverns if the arrow, box, etc. is numbered or not. See Section 7.10.3 [Numbering], page 105 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 7.10.4 [Compression and Vertical Spacing], page 108 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 6.1.5 [Labels], page 56.

`url` This assigns a link target to the label, such as an URL or a Doxygen target. Note that *box tags* cannot be turned into a link using this attribute, use the `\L()` escape instead. See Section 6.4 [Links], page 62 for more info.

`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 earier or later and thereby control how they overlap. See more in Section 7.13.2 [Symbols], page 115.

`parallel` This can take a `yes` or a `no` and is equivalent to prepending the element with the `parallel` keyword, see Section 7.9 [Parallel Blocks], page 97.

7.10.2 Word Wrappiog and Long Labels

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 detemined by the spacing of entities, a new `width` attribute is inserted, which can specify the width of the note making word wrapping meaningful.

You have effective 3 easy way to typeset long labels.

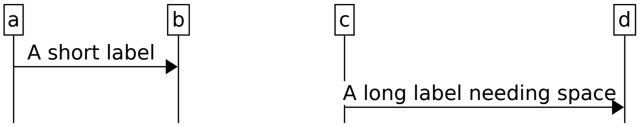
- Word wrapping: Use `text.wrap=yes` (and perhaps a fixed `hscale`), in this case the long labels wrap into multiple lines.
- Automatic scaling: Use `hscale=auto` and no word wrapping, in this case entities are spaced apart, so that there is enough space for all labels.
- None: No word wrappig or automatic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

```
hscale=1;
a, b, c, d;
a->b: A short label;
c->d: A long label needing space;
hspace c-d: A long label needing space;
```



7.10.3 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.

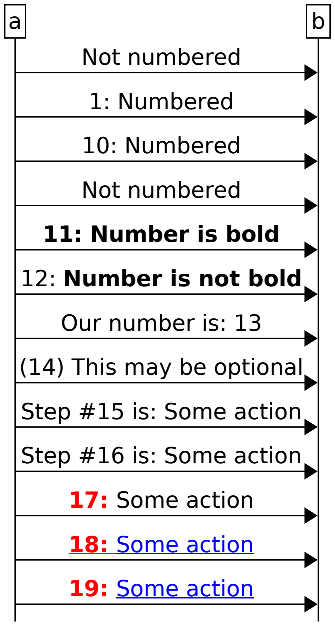
Notes and comments will not increase numbering, instead they carry the number of the element they are referring to. If the target element had no number comments will have none, even if numbering is turned on for them.

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 6.1.4 [Styles], page 56 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 numberig for only blocks of the chart, see Section 6.5 [Scoping], page 63.

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 6.3 [Text Formatting], page 59)[12]. 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 6.3 [Text Formatting], page 59 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[13]. 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.

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

[13] Using '`arabic`', '`letters`' or '`roman`' is also valid (both uppercase or lowercase).

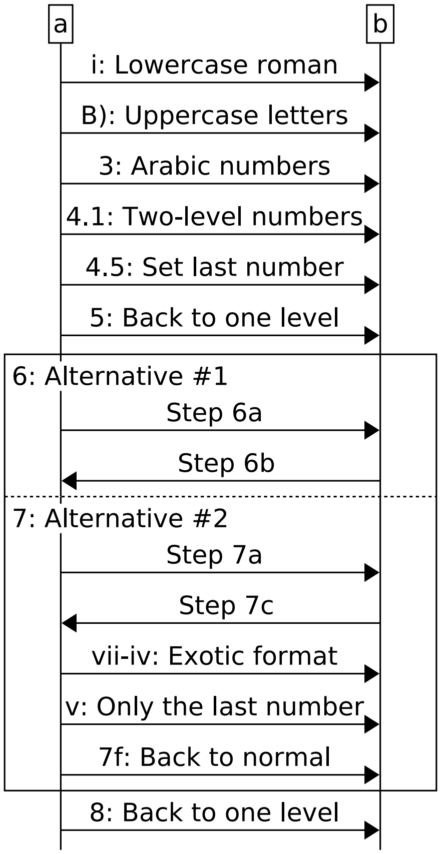
Note also, that when using roman numbers or letters, you can use such numbers as the value of the `number` attribute, as shown below for ‘`7c`’.

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`’.

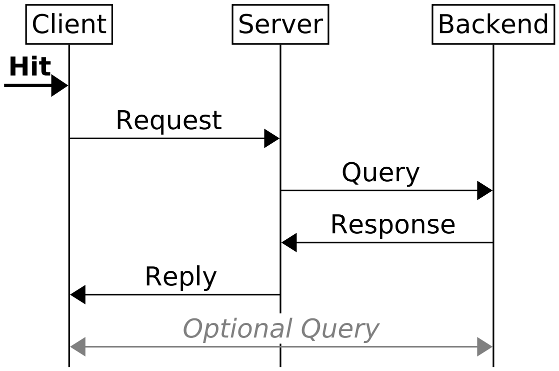
```
hscale=auto, numbering=yes;

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;
```

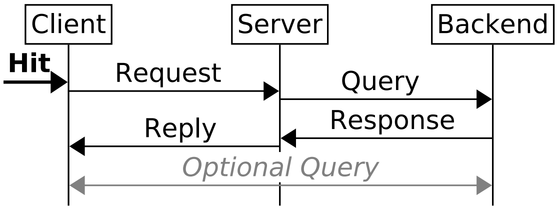


Finally, if an element is named using the `refname` attribute, you can reference the number of that element in another label using the `\r(name)` escape sequence. Note that the value of the `numbering.pre` and `numbering.post` options are ignored when inserting the number of a referenced element, similar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

7.10.4 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` attribuite 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 7.13.1 [Spacing], page 114.

7.10.5 Default and Refinement Styles for Signalling Charts

For signalling charts styles can contain any of the attributes listed in Section 7.10 [Signalling Chart Attributes and Styles], page 102. Styles are explained more in Section 6.6 [Defining Styles], page 64.

There is a built-in default style for each signalling chart element: `arrow`, `box`, `emptybox`, `divider`, `blockarrow`, `pipe entity`, `entitygroup`, `symbol`, `indicator`[14], `title`, `subtitle`, `note`, `comment`, `endnote`, `vertical`[15], `vertical_brace`, `vertical_bracket`, `vertical_range`, `vertical_pointer`, `symbol`[16] and `text`[17].

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. Attributes of large entitygroups default to the `entitygroup_large` style.

The following refinement styles are defined further governing the appearance of elements

- for arrows: '`arrow_self`', '`->`', '`=>`', '`>`', '`>>`', '`==>`' and '`=>>`'.[18].
- 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 refinement styles 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];
```

Finally there are two more pre-defined styles for signalling charts: `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.

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.

[14] The style `indicator` determines the appearance of the small symbols that indicate elements hidden due to a collapsed box or entity group.

[15] referring to vertical boxes and block arrows

[16] referring to all symbols

[17] referring to `text at` commands

[18] '`arrow_self`' is applied to arrows starting and ending at the same entity after and in addition to style '`arrow`'. The others are applied (potentially after '`arrow_style`' to line segments, also for bi-directional arrows. Thus there is no separate '`<->`' style, for example.

1. If you specify an attribute directly at the element (perhaps via applying a style), the specified value is used[19].
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. For arrows starting and ending at the same entity, two refinement styles are applied, first '`arrow_self`', then the one based on the arrow symbol. (In the latter 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.

7.11 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.

`msc` 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 '`plain`' 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 '`=`' symbol, partial designs shall use '`+=`'. See Section 6.7 [Chart Designs], page 65 for more on chart designs.

`hscale` This option takes a number or `auto`, and specifies the default horizontal distance between entities. The default is 1, so to space entities wider apart, use a larger value. When specifying `auto` entity positions will be automatically set according to the spacing needs of elements. In this case the `pos` 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 7.10.3 [Numbering], page 105.

`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 7.10.4 [Compression and Vertical Spacing], page 108.

[19] If you specify the attribute several times, the last one is used.

`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 7.10.3 [Numbering], page 105 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. Describin 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`

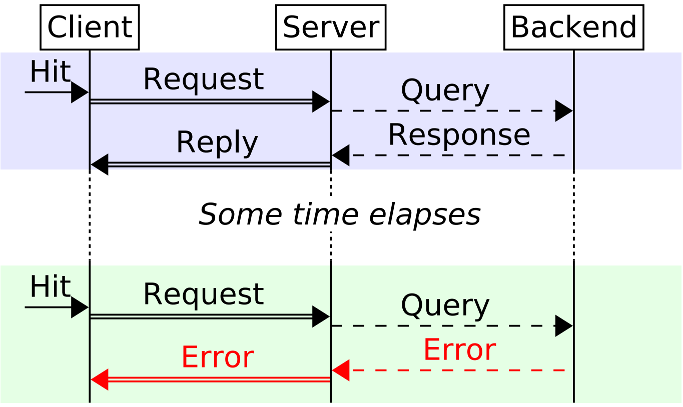
These are similar to `fill.*` attributes and specify the background color of the chart. By default the background is transparent. The only exception The only exceptions are PNG images, which cannot have transparency, so the default background color is white. You can change the background color multiple times, each change taking effect at the place where you issue the background chart option. This is usefult 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 specifythis 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 7.8 [Notes and Comments], page 95 for more information on comments.

7.12 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 sinlge multi-page file is created using the page size, orientation, margins and alignment selected in Print Preview.

7.13 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.

7.13.1 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 secod 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 7.2.1 [Entity Positioning], page 71 and Section 7.11 [Chart Options], page 110.

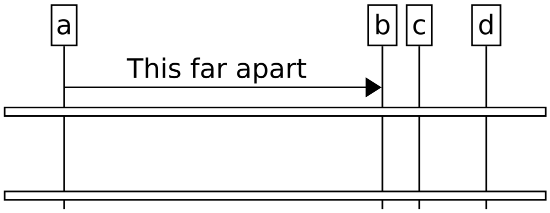
The `hspace` command is useful in the latter case to force a certain horizontal distance between two (not necessarily neighbouring) entity. The space can be larger than the one specified with `hspace` if the layout requires so, but never smaller.

```
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[20]. 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.

```
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;
```



[20] 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;`.

7.13.2 Symbols

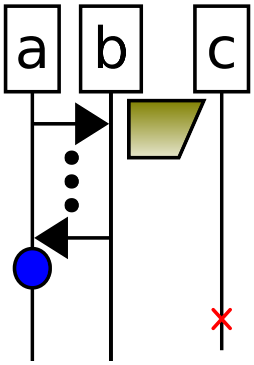
Currently Msc-generator can draw circles (ellipses), ellipses (three dots), cross symbols (a big ‘`X`’), arbitrary shapes, rectangles (optionally with text) or just plain text. We call these *symbols*.

```
symbol symbol_type at entity [attributes];
symbol symbol_type marker-marker hpos1 hpos2 [attributes];
```

By specifying either `arc`, `rectangle`, `...`, `text`, `cross` or `shape` after the `symbol` keyword one instructs Msc-generator to draw one circle/ellipsis, rectangle, ellipses, plain text[21], a cross symbol or a shape, 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, use the first, simpler syntax or the second one without markers (and the dash in-between).

```
hscale=auto;
a, b, c;
parallel a->b;
parallel symbol shape at b++
     [shape=def.trapezoid, xsize=20,
      fill.color=olive, fill.gradient=up];
symbol ... at a-b [fill.color=black];
a<-b;
symbol arc at a
     [xsize=10, ysize=10, fill.color=blue,
      compress=yes];
symbol cross at c [line.color=red];
hide c;
```



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 7.6 [Verticals], page 90. In this case however, the layout engine will ignore the symbol when placing other elements and the symbol may end up drawn overlapping other elements (this may be your intention).

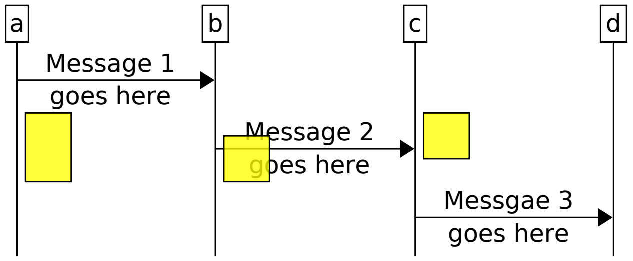
The vertical size of the object can be specified two ways. Either you specify two markers (as above), in which case the symbol will vertically span from one to the other; or you omit one of the markers, in which case the `ysize` attribute specifies the height (in points)[22]. If the dash is in front 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.

[21] We have to note that `text` is just syntactic sugar for a `rectangle` with no line or fill. Rectangles can also contain text.

[22] 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. If you specify a shape, it is enough to specify one of the sizes, the other will be calculated to keep the original aspect ratio of the shape.

```
a, b, c, d;
a->b: Message 1
      goes here;
mark m1;
b->c: Message 2
      goes here;
mark m2;
c->d: Messgae 3
      goes here;
defstyle symbol [fill.color="yellow,200"];
symbol rectangle m1-m2 left at a+ [xsize=30];
symbol rectangle   -m2 left at b+ [xsize=30, ysize=30];
symbol rectangle m1-   left at c+ [xsize=30, ysize=30];
```



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

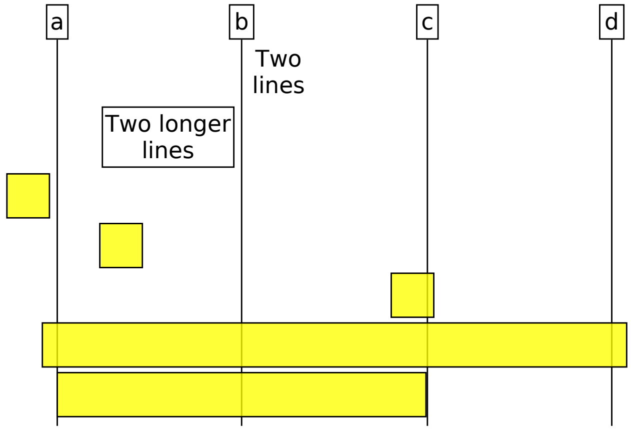
```
left|center|right at entity-entity [number]
left|center|right at entity--
left|center|right at entity-
left|center|right at entity [number]
left|center|right at entity+
left|center|right at entity++
```

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[23]. 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];
```

[23] 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_background`
: Elements with this property will be drawn before the background. This has effect only if the background is transparent (to some degree or part).

`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.

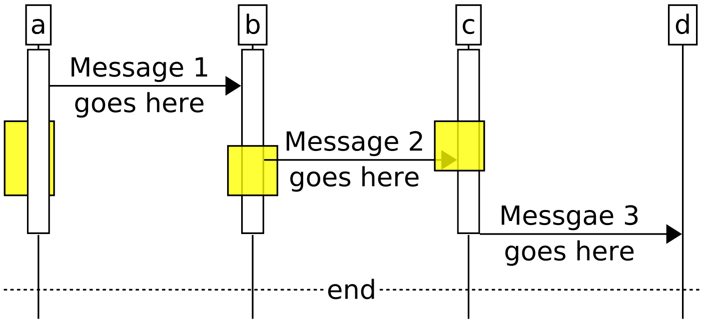
`after_entity_lines`
: Elements with this property will be drawn just after the entity lines are laid out, but before regular elements are drawn.

`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 thet layout only the drawing order (what is called the *z-order*).

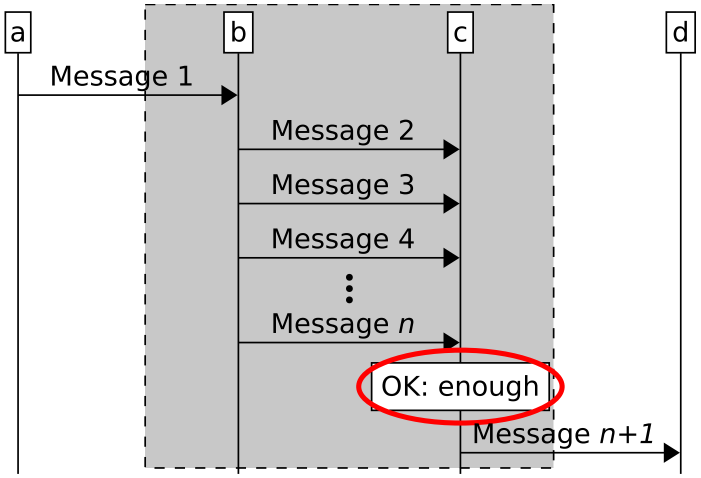
```
a, b, c, d;
vspace 0;
activate a, b, c;
a->b: Message 1
      goes here;
mark m1;
b->c: Message 2
      goes here;
mark m2;
c->d: Messgae 3
      goes here;
deactivate a, b, c;
---: end;
defstyle symbol [fill.color="yellow,200"];
symbol rectangle m1-m2 center at a- [xsize=30, draw_time=before_entity_lines];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



7.13.3 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.

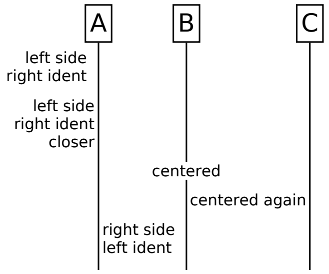
`text at` *pos* `[`*attributes*`]: label;`

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.

```
hscale=auto;
A, B, C;
text at A--: left side\nright ident;
text at A-: left side\nright ident\ncloser;
text at B: centered;
text at B-C: centered again;
text at A+: right side\nleft ident;
```



7.14 Commands

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

`nudge` This command inserts a small vertical space useful to misalinging two arrows in parallel blocks, see Section 7.9 [Parallel Blocks], page 97.

`hspace` This command forces horizontal distance between two (not necessarily negighbouring) entity. See Section 7.13.1 [Spacing], page 114.

`vspace` This command inserts an arbitrary size vertical space, see Section 7.13.1 [Spacing], page 114.

`newpage` This command starts a new page, see Section 7.12 [Multiple Pages], page 113.

`heading` 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 preceeding or following entity definition commands before or after `heading`, only one copy of the entity headings is drawn.

`show`
`hide` 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 addtion, 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 7.6 [Verticals], page 90 for more information.

`note`
`comment`
`endnote` These commens are useful to annotate the chart, see Section 7.8 [Notes and Comments], page 95.

`symbol`
`text` These commands can be used to draw arbitrary graphics to the chart, see Section 7.13 [Free Drawing], page 114.

`defcolor` This command is used to define or re-define color names, see Section 6.2 [Specifying Colors], page 58.

`defstyle` This command is used to define or re-define styles, see Section 6.6 [Defining Styles], page 64.

`defdesign`
This command is used to define new designs, see Section 6.7 [Chart Designs], page 65.

`defshape` This command is used to define new shapes, see Section 6.8 [Defining Shapes], page 66.

7.15 Mscgen Backwards Compatibility

Msc-generator was forked (and completely re-written) from mscgen version 0.08 and is a superset of the language of that version even today. During the years the original tool has developed quite a lot and now contains features that are not present or not compatible with that of Msc-generator. From version 4.5 Msc-generator aims to support all the features and syntax of mscgen to provide a true superset of features. Since there are conflicts in the syntax, full support is possible only in a special *mscgen compatibility mode* of Msc-generator.

By default, Msc-generator enters this mode, if the chart text starts with `msc {`. This is because mscgen requires you to start your chart like this, but Msc-generator does not[24]. You can force or prevent mscgen compatibility mode using appropriate command-line switches (`--force-mscgen` and `--prevent-mscgen`, respecively) or preference settings in the GUI. Also, on the GUI, the `mscgen-compat` indicator of the status bar turns to red in compatibility mode.

Msc-generator attempts to be liberal in what it accepts in both modes. That is, Msc-generator attributes are available also in mscgen mode (So you can, for example, use vertical constructs, hide or show entities, let entities be defined later or use the style mechanism.)

[24] If you specify a design, like `msc=qsd {` Msc-generator will NOT enter compatibility mode.

Similar, mscgen attribute names and chart options are understood even when not in mscgen compatibility mode. This includes the hashmark syntax for color specifications, like `text.color="#c0ffff"`. For attributes or other syntax I intend to deprecate a warning is given.

A major tool for backwards compatibility with mscgen is the `mscgen` partial design, which contains new styles for arrowhead symbols. E.g., the symbol `->` represents a filled triangle arrowhead in Msc-generator, while it represents a half line arrowhead in mscgen. Using the `mscgen` design via the `msc+=mscgen;` command sets (when not in mscgen compatibility mode) the styles as used by mscgen. In mscgen compatibility mode, such styles are automatically applied.

Some mscgen attributes are interpreted by Msc-generator not exactly the same way as in the mscgen tool due to the different internal workings. The below hold for both the compatibility and regular modes of Msc-generator.

- The `textbgcolor` attribute of mscgen in empty boxes is interpreted as `fill.color`, as in mscgen. It is not possible to have a different fill color and text background for an empty box.
- The `wordwraparcs` chart option is interpreted as `text.wrap`, resulting in word wrapping for all text. Since in mscgen boxes are word wrapped by default, the `mscgen` design sets word wrappin in the `emptybox` style. This command sets it everywhere (as intended in mscgen).
- The `arcskip` attribute of mscgen makes an arrow slant down to the vertical position of the next arrow. This is approximated with a fixed `slant_depth`.
- Line breaks in quoted strings are accepted by mscgen (and by Msc-generator in mscgen mode). This is not the case when not in mscgen mode. In general I think it is a bad idea. Use the `\n` text formatting escape or use colon labels.

There are a few conflicting syntax though that must be interpreted differently in the two modes.

- The arrows symbols `::` and `:>` exist only in mscgen mode. These conflict with colon label syntax. You can use `==>` instead of the latter one in both mscgen mode and outside (to represent a double line arrow with a filled triangle arrowhead).
- The `-x` arrow symbol is used to indicate loss in mscgen mode. This arrowsymbol is not understood outside the mscgen mode. Replace it with `->*` instead.
- The `->*` broadcast arrow construct is interpreted in mscgen mode only, since in Msc-generator it means a lost arrow. Even in mscgen mode it is interpreted as a broadcast arrow only if there is no target entity after the asterisk. If there is one (which is not valid mscgen syntax), it is interpreted as a lost arrow according to Msc-generator syntax.
- The `hscale` attribute in mscgen mode is interpreted as setting the width of the chart to 600 pixels times the specified value. In Msc-generator it is interpreted as setting the inter-entity distance to 130 pixels times the specified value (and can of course be set to `auto` to automatically select the spacing between the entities). Due to user feedback the `hscale` attribute is interpreted the Msc-generator way even in compatibility mode[25].

[25] After all the point of the compatibility mode is to enable migration towards Msc-generator features.

- Since `--`, `..` and `==` are arrow symbols (representing headless arrows, essentially just lines), they cannot be used to define empty boxes in mscgen mode using syntax like this: `a==b [label="label"];`, because in mscgen this is a line, not an empty box. So you need to prepend the `box` keyword to indicate that this is supposed to be a box.

8 Graph Language Reference

Msc-generator supports an extended version of the DOT language, which is fully backwards compatible with the original saving two exceptions.

- New keywords cannot be used as node names. These are `defstyle`, `defproc`, `replay`, `defdesign`, `usedesign`, `include`, `cluster`, `if`, `then` and `else`. If you want these as nodes put them in-between quotation marks.
- Msc-generator does not support graphviz HTML labels. They get parsed, but interpreted only as text.

The language of the graphs (the DOT language) is documented well on the graphviz pages at http://www.graphviz.org/. Below we just list the additions by Msc-generator.

8.1 Graph Attributes

Msc-generator allows you to specify node, edge and subgraph labels via a colon syntax similar to signalling charts (see Section 6.1.5 [Labels], page 56). However, since colons are already used in the DOT language to express node ports, two colons shall be used to start labels, like `node::label;`. Such colon labels are terminated by semicolons and opening square brackets or braces - even if in DOT, nodes can be separated via commas or even spaces. When a label is specified via the colon notation all text formatting escapes used by signalling charts can be used (see Section 6.3 [Text Formatting], page 59). Note that in this case it is not possible to use the escape sequences of graphviz (such as `\N` to insert the name of the node).

The format of labels (sepecified either via the `label` attribute or the colon notation above) can be influenced via the `label_format` attribute. Assigning a series of text formatting escapes to this attribute (see Section 6.3 [Text Formatting], page 59) will apply their effect to the label. Note that the formatting is only applied to the label of the node, edge or cluster subgraph, not to any of headlabel, taillabel or xlabel.

Msc-generator supports shadows for nodes and cluster subgraphs, via the `shadow_offset`, `shadow_blur` and `shadow_color` attributes. Read more on these in Section 6.1 [Common Attributes], page 52.

Since the style attribute of graphviz can take multiple values separated by commas, like `style="dotted,filled"` it is hard to work with it using styles. For example in the style you define `defstyle node [style=filled];`, then later you define a node as `a [style=dotted];`, which would normally completely overwrite the style removing the fill. So in Msc-generator, the style attribute is applied specially. Any assignment to this attibute will *add* a new value in a comma separated way, so `a [style=dotted, style=filled];` is equivalent to `a [code="dotted,filled"];`. If you want to *remove* a previously added component, use the minus sign, such as `style="-filled"`. If you want to drop previous values to style and have the current value be the only one, use an exclamation mark like `style="!bold"`.

Finally, there are a few rules on how attributes can be specified, which are somewhat against the logic of other Msc-generator languages.

- You can use a comma separated list of node names to define nodes, but be aware that any attribute you specify will apply to all nodes in the list. So in `a, b, c::Label;` all three nodes will have the same label. (This is from the original DOT language.)

- Attributes after an edge will apply to the edge and not to the last node. So in `a->b [color=red];` the edge will be red and not node `b`. (This is from the original DOT language.)
- You can assign attributes to subgraphs in the subgraph heading via square brackets. For example: '`subgraph cluster_one [style=rounded] {...};`'. (This is an Msc-generator extension, in the original language you can only specify subgraph attributes inside the subgraph.)

8.2 Default and Refinement Styles for Graphviz Graphs

You can define and redefine *styles*. As with signalling charts a style is a package of attributes with a name (see Section 6.6 [Defining Styles], page 64). Any attribute can be assigned to styles (even `label`).

To apply a style to a node, edge or subgraph, use its name alone enclosed in square brackets, like `a->b::Label [style];`. You can have another square bracket after or before with additional attributes, like `a->b::Label [color=red][style][arrowhead=onormal];`.

Graphviz has a default style for edges, nodes, cluster subgraphs and graphs named `edge`, `node`, `cluster` and `graph`, respectively. The default style for graph, node and edge can be set using the `graph`, `node` and `edge` commands as well, such as

```
node [shape=record];
```

Msc-generator allows you to collapse cluster subgraphs. Such collapsed subgraphs show up as nodes, but will have `cluster_collapsed` as the default style (boxes with double lines in the plain design). Similar edges that were pointed to nodes inside the collapsed cluster, will have the style `edge_collapsed`.

If you want to adjust an attribute for all versions of an arrow symbol (forward, backward, bidir and arrowless) list all of them in the `defstyle` command, such as in

```
defstyle ->>, <<-, <<->>, --- [color=red];
```

Note that no quotation marks are needed for these special style names. (Otherwise non-alphanumeric names need to be enclosed in quotation marks.) Specifically, Msc-generator has the following refinement styles for arrows: '`->`', '`=>`', '`>`', '`>>`', '`->>`' and '`==>`', to represent single, double, dotted, dashed line; double arrowhead and double line, respectively. The last arrow type is provided for you to re-define, by default it is the same as '`=>`'. You can also use them backwards (such as '`<-`') or bidirectionally ('`<->`'). There is also a directionless variant for both, whithout arrowheads: '`--`', '`==`', '`..`', '`++`', '`---`' and '`===`'.

Msc-generator supports design definitions, for graphs as well. Use the `defdesign` command to define a named design, like below.

```
defdesign omegapple {
    defstyle node [color="#d23f7a", style=filled,
        fillcolor="#c20a50:#f3cedc", gradientangle=270,
        fontcolor=white, penwidth=2, label_format="\b"];
    defstyle cluster [color="#c20a50", style=dashed,
        fontcolor="#c20a50"];
    defstyle edge [penwidth=2, fontcolor="#270210"];
}
```

You can invoke such designs via the `usedesign` command like `usedesign opegapple`. You can also select among the designs on the GUI.

Thus, in summary the actual attributes of an element are set using the following logic. (There are minor variances for each language.)

1. If you specify an attribute directly at the element (perhaps via applying a style), the specified value is used[1].
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.
3. Otherwise, if the attribute is set in the default style of the element in any of the designs applied, the value in the last design that has this attribute is used.

8.3 Clusters

When using the `dot` layout algorithm, graphviz makes it possible to make subgraphs visible by drawing a rectangle around the nodes inside the subgraph. By convention this feature is activated if the name of the subgraph starts with '`cluster`'. In Msc-generator, you can use the `cluster` keyword instead of `subgraph`, to create a visible subgraph. You do not need to append the '`cluster_`' prefix to its name. Also, whatever name you specify will also become the default label. Omitting the name will result in an empty label and an auto-generated name, like '`cluster_xxx`', where '`xxx`' is an auto-incremented number.

Msc-generator also supports cluster subgraph collapsing/expansion. When in the GUI, you hoover over a cluster subgraph, a collapse icon (red minus sign) appears at the top-right corner of the cluster. (Similar to how it happens with boxes and group entities in signalling charts.) Clicking that (or double clicking anywhere in the cluser) collapses the subgraph into a single, double-lined node of shape box. In the GUI there are buttons to collapse or expand all subgraphs with one click.

When collapsing a subgraph multiple edges from within the subgraph to the same node outside will be kept if the graph is not `strict`. Use the `collapse_strict=yes;` chart option to remove duplicate edges on collapsing a subgraph. In this case `edge_collapsed` style is applied to the remaining edge. Similar, nodes that result after the collapse of a subgraph are applied the `cluster_collapsed` style instead of `cluster`.

[1] If you specify the attribute several times, the last one is used.