Msc-generator

A small tool to draw message sequence charts
(version v2.3, 12 February 2010)

Zoltan R. Turanyi

This manual is for Msc-generator (version v2.3, 12 February 2010), a small tool to draw message sequence charts from a textual description.

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

1 Getting started

1.1 Overview

Msc-generator is a small program that parses Message Sequence Chart descriptions and produces graphical output in a variety of file formats, or as a Windows OLE embedded object. Message Sequence Charts (MSCs) are a way of representing entities and message interactions between entities over some time period and are often used in combination with SDL. MSCs are popular in telecom an 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 tools, such as Microsoft Visio.

This version of msc-generator is heavily extended and completely rewritten version of the 0.8 version of Michael C McTernan's mscgen. It has a number of enhancements, but does not support ismaps (clickable URLs embedded into the image). The original tool was more geared towards describing interprocess communication, this version is more geared towards networking.

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

1.2 Use on Linux

Msc-generator on Linux is a command-line tool. You specify the text file containing the description of the chart and it gives you a graphics file in return. Msc-generator supports PNG, PDF, EPS, SVG. To start it simply type

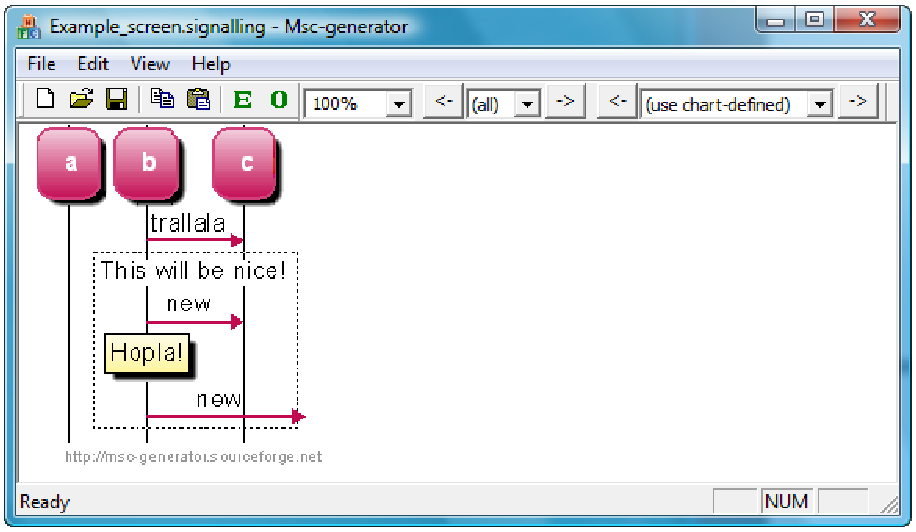
```
mscgen -T pdf inputfile.signalling
```

This will give you '`inputfile.pdf`'. you can change pdf to get the other file formats. If you omit the '`-T`' switch altogether, 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. After discussing Windows use we guide you step-by-step through the process of creating a message sequence chart description that can be given to Msc-generator. The complete description with all the features will come after.

1.3 Use on Windows

On Windows Msc-generator is both a command-line tool and a windowed application. If the command line contains the '`-T`' switch, the command line-version is started that largely works the same as the Linux variant, but also supports EMF (Enhanced Metafile) output. Otherwise Msc-generator starts as a windowed application.



1.3.1 Editing the chart

When you press the '`E`' button on the toolbar, a text editor is started, where you can edit the chart description. If you save in the text editor, the chart drawing is updated, so you can follow your changes. Also, if there were errors or warnings, they are displayed in a separate window. You can close the editor and re-open later by selecting the '`Edit|Edit in text editor...`' menu item or clicking the '`E`' shaped button on the toolbar.

You can select the text editor to start in '`Edit|Preferences...`'.

You can use the regular Windows File and Edit menu operations to load/save the file or do clipboard operations. The file operations use the text description of the chart as file format, but you can also save the file in various graphics formats using the '`File|Export...`' menu.

1.3.2 Toolbar and Zooming

You can zoom in and out using the commands in the View menu and the associated shortcut keys. You can also use the mouse wheel with the Ctrl key pressed to zoom in and out. You can also use the Zoom control on the toolbar to select a zoom factor.

On the toolbar you can find the standard buttons for new, open, save, copy and paste. The '`E`' shaped button is used to start and stop the text editor. The '`O`' shaped button is used to adjust the size of the window and the zoom factor to give an overview of the entire chart. This is automatically performed after a file is opened. The same effect can be achieved by selecting the '`View|Zoom to overview`' command or pressing the '`Ctrl+O`' shortcut.

Another zooming commands can be found in the View menu. '`View|Adjust width`' attempts to set the window width to best accomodate the wuidth of the chart. It does not change the height of the window. '`View|Fit to width`' adjusts the zoom factor to fit the chart to the width of the window. The '`View|Keep...`' commands fix Msc-generator in a particular zoom mode. At every update of the chart (via the text editor or changing deisgn or page) the window size and zoom factor accordingly. Msc-generator remembers the zoom mode between invocations.

The second drop-down on the toolbar can be used to select which page of the chart is displayed. See Section 2.11 [Commands], page 34 for more info on pagination. If '`(all)`' is selected then '`newpage;`' commands are ignored and the whole chart is shown. You can use the arrow keys beside the combo box to cycle through the pages. Of course if the chart has no '`newpage;`' command included and it contains only one page, you cannot select among pages.

The last drop-down on the tool bar is the design selector. By selecting a chart design here you can override the selection in the source file. This is an easy way of reviewing how your chart would look like in a particular design. See Section 2.14 [Chart Designs], page 36 for more info on chart designs.

1.3.3 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 and paste it into the compound document[1]. Later you can edit the chart in-place by double clicking the chart in the document.

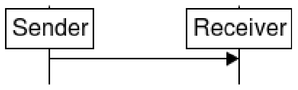
We note that page and chart design settings you select on the toolbar are saved with embedded documents but not when you save the chart into a file.

Now let's see, how to describe your message sequence chart using Msc-generator.

1.4 Defining arrows

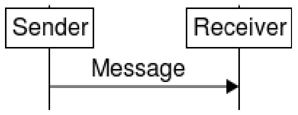
Message sequence charts consits of *entities* and *messages*. The simplest file consists of a single message between two entities: a '`Sender`' and a '`Receiver`'.

```
Sender->Receiver;
```



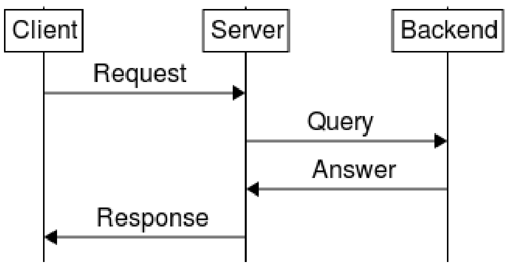
The message may have a label, as well.

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



A more complicated procedure would be to request some information from a server, which, in turn, queries a backend. We can also add comments using '`#`' characters.

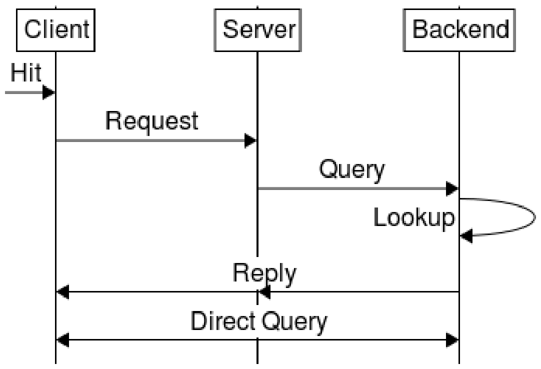
```
#A more complex procedure
Client->Server:  Request;
Server->Backend:  Query;
Server<-Backend:  Answer;
Client<-Server:  Response; #final
```



[1] Only a whole chart can be placed onto the clipboard, and not parts of it.

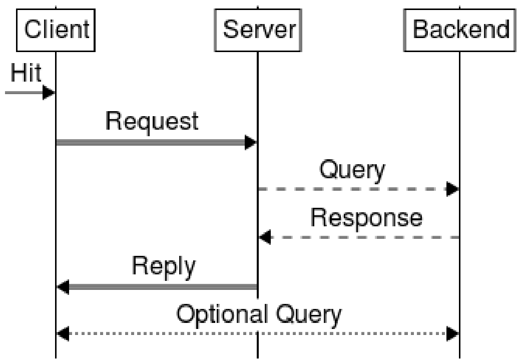
Arrows can take various forms, for example they can be bi-directional, can span multiple entities, start and end at the same entity and can come "from outside"

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



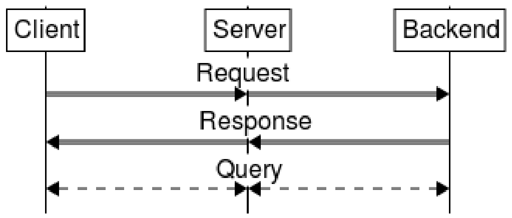
It is also possible to make use of various arrow types, such dotted, dashed and double line. To achieve this the '->' symbol need to be replaced with '>', '>>' and '=>', respectively.

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



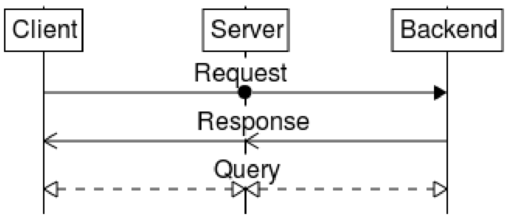
As a shorthand for arrows spanning multiple entitues, the dash '-' symbol can be used in the second and following segments, as seen below.

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



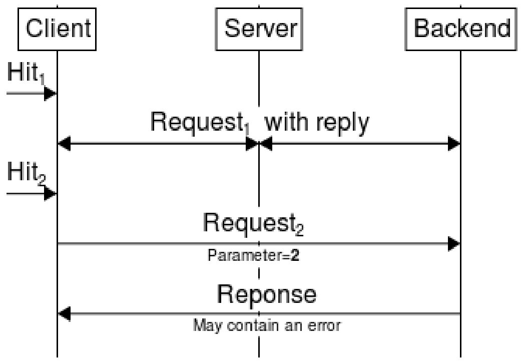
It is possible to change the type of the arrowhead. The arrowhead type is an *attribute* of the arrow. Attributes can be specified between square brackets before or after the label, as shown below. A variety of arrow-head types are available, for a full list of arrow attributes and arrowhead types See Section 2.4 [Arrows], page 26.

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



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

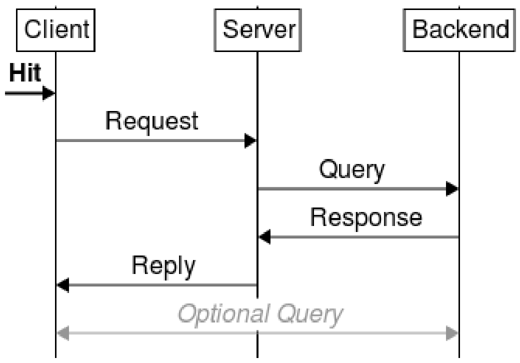


Arrows can further be differentiated by applying *styles* to them. Styles are packages of attributes with a name. They can be specified in square brackets as an attribute that takes no value. Msc-generator has two pre-defined styles '`weak`' and '`strong`', that exits in all chart designs[2]. They will 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[3].

[2] You can define your own styles, as well, Section 2.13 [Styles], page 35.

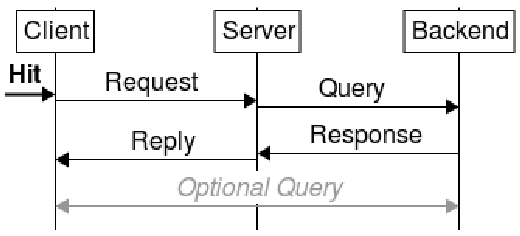
[3] For more on chart deisgns Section 2.14 [Chart Designs], page 36.

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



As a final feature of arrows, we note that 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 2.10 [Chart Options], page 34.

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

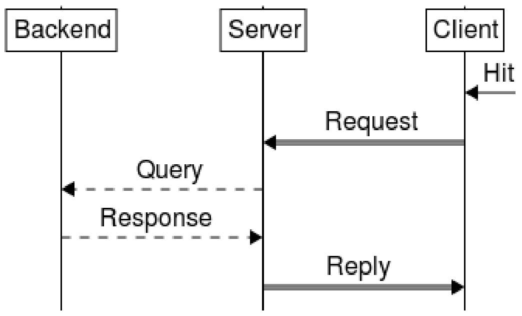


1.5 Defining Entities

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`'.

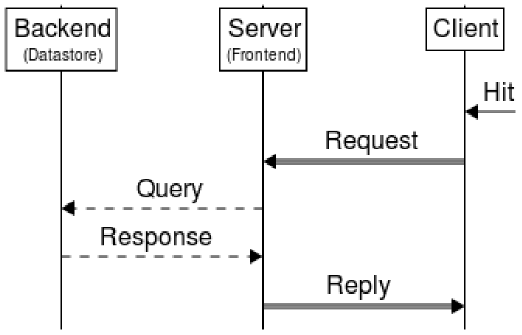
Often one wants to control, in which order entities appear on the chart. This is possible, by listing the entities before actual use. On the example below, the order of the enties are reversed. Note that we have reversed the first arrow to arrive to the '`Client`' from the right.

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



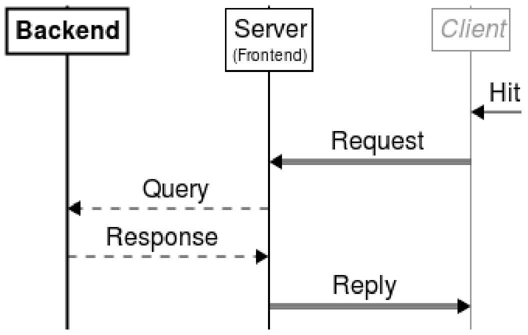
Often the name of the entity need to be multi-line or need to contain formatting characters, or just is too long to type many times. You can overcome this problem by specifying a label for entities. The name of the entity then will be used in the chart description, but on the chart the label of the entity will be displayed. The '`label`' is an attribute of the entity and can be specified between square brackets after the entity name, before the colon, as shown below. (You can specify entity attributes only when explicitly defining an entity.)

```
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;
```



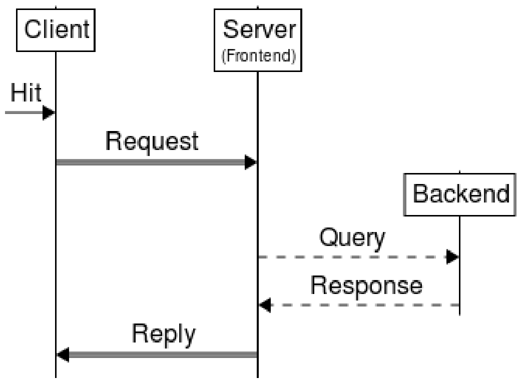
Entities can also be specified as '`weak`' or '`strong`', by adding them as attributes.

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

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



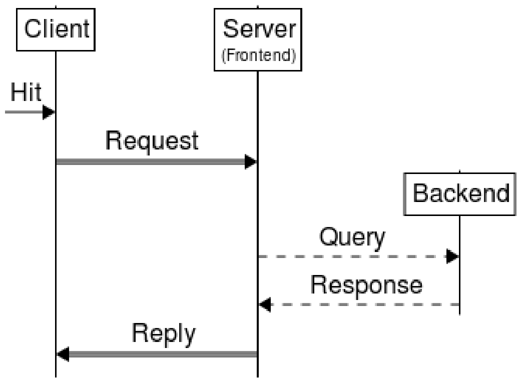
Entities can be turned on and off. An entity that is turned off, will not have its vertical line displayed. This is useful if the chart has many entities, but one is involved only in a small part of the process. An entity can be turned off by seting its '`show`' attribute to '`no`'. When an entity is defined with '`show=no`', its heading is not drawn at the top of the chart. Similar, when it is later turned on, a heading will be shown.

```
C [label="Client"],
S [label="Server\n\-(Frontend)"],
B [label="Backend", show=no];
->C: Hit;
C=>S: Request;
B [show=yes];
S>>B: Query;
S<<B: Response;
B [show=no];
C<=S: Reply;
```



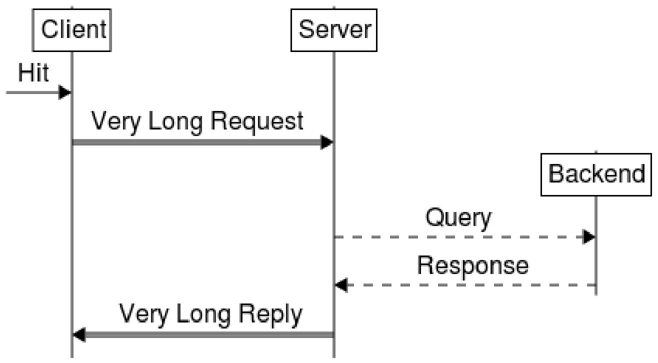
The same can be achieved by simply defining the entity later. Note that this is different from simply starting to use an entity later. In that case the entity will be shown from the top of the chart as in earlier examples.

```
C [label="Client"],
S [label="Server\n\-(Frontend)"];
->C: Hit;
C=>S: Request;
B [label="Backend"];
S>>B: Query;
S<<B: Response;
B [show=no];
C<=S: Reply;
```



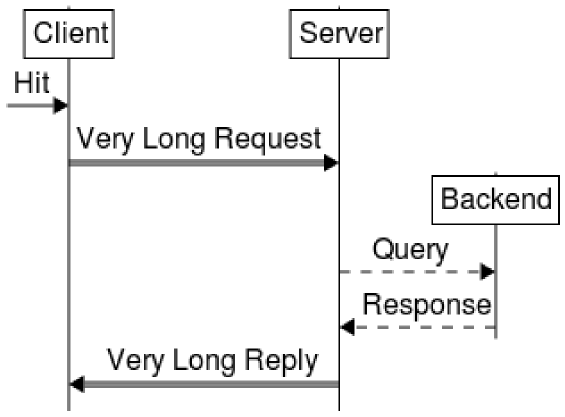
Sometimes the vertical space between entities is just not enough to display a longer label for an arrow. In this case use the '`hscale`' chart option to increase the horizontal spacing. It can be set to a numerical value, one being the default.

```
hscale=1.3;
C [label="Client"],
S [label="Server"];
->C: Hit;
C=>S: Very Long Request;
B [label="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.

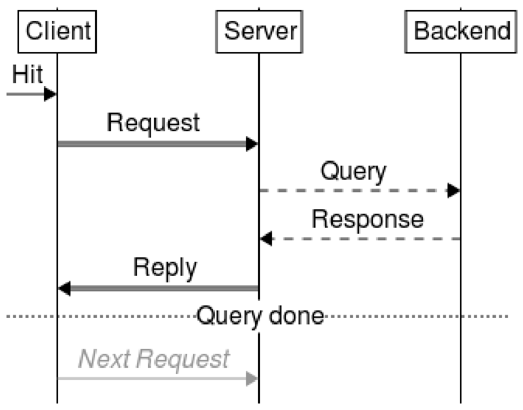
```
hscale=auto;
C [label="Client"],
S [label="Server"];
->C: Hit;
C=>S: Very Long Request;
B [label="Backend"];
S>>B: Query;
S<<B: Response;
B [show=no];
C<=S: Very Long Reply;
```



1.6 Dividers

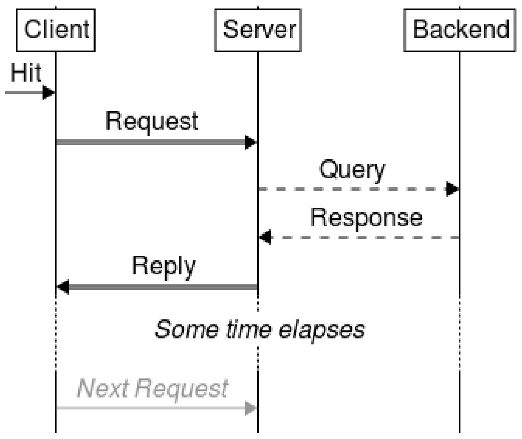
In an MSC it is often important to segment the process into multiple logical parts. You can use the '---' element to draw a horizontal line acorss the chart with some text, e.g., to summarize what have been achieved so far.

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



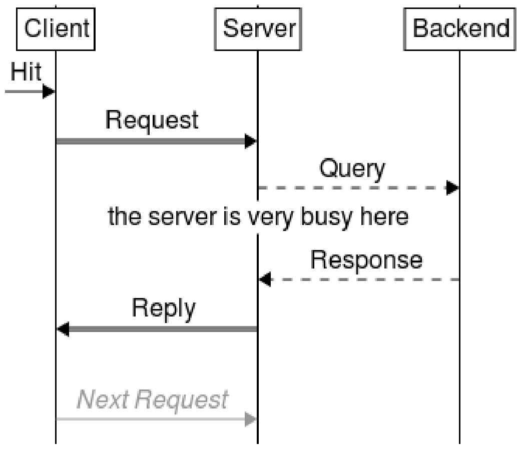
Similar to this, using the '...' element can express the passage of time by making the vertical lines dotted.

```
C [label="Client"],
S [label="Server"],
B [label="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;`’ or like ‘`[]: text;`’. Using ‘`[];`’ will create an empty vertical space;

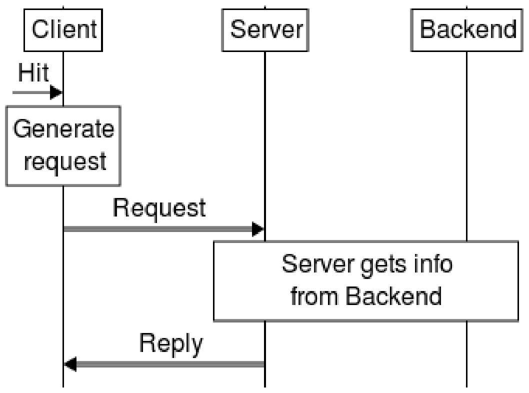
```
C [label="Client"],
S [label="Server"],
B [label="Backend"];
->C: Hit;
C=>S: Request;
S>>B: Query;
:  the server is very busy here;
S<<B: Response;
C<=S: Reply;
[];
C->S [weak]:  Next Request;
```



1.7 Drawing Boxes

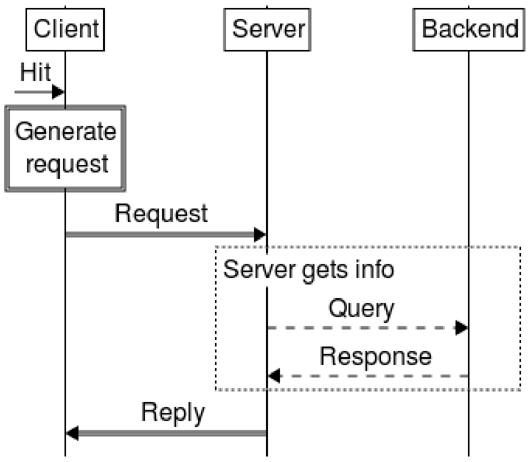
A *box* can be used to express many different thing, such as to add textual comments, group a set of arrows or describe alternative behavior. In their simplest form they only contain text. A box always spans between two entities, or alternatively only one.

```
C [label="Client"],
S [label="Server"],
B [label="Backend"];
->C: Hit;
C--C: Generate\nrequest;
C=>S: Request;
S--B: Server gets info\nfrom Backen
C<=S: Reply;
```



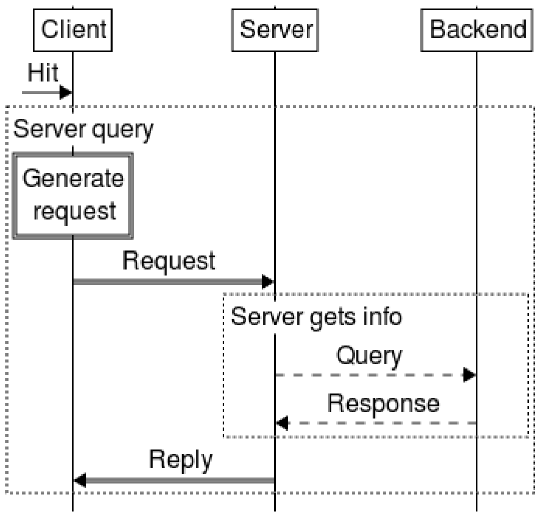
Boxes can be dotted, dashed and double line, too, by using '..', '++' or '==' instead of '--'. Boxes can also be used to group a set of arrows. To do this, simply insert the arrow definitions enclosed in curled braces just before the semicolon terminating the definition of the box.

```
C [label="Client"],
S [label="Server"],
B [label="Backend"];
->C: Hit;
C==C: Generate\nrequest;
C=>S: Request;
S..B: Server gets info
{
    S>>B: Query;
    S<<B: Response;
};
C<=S: Reply;
```



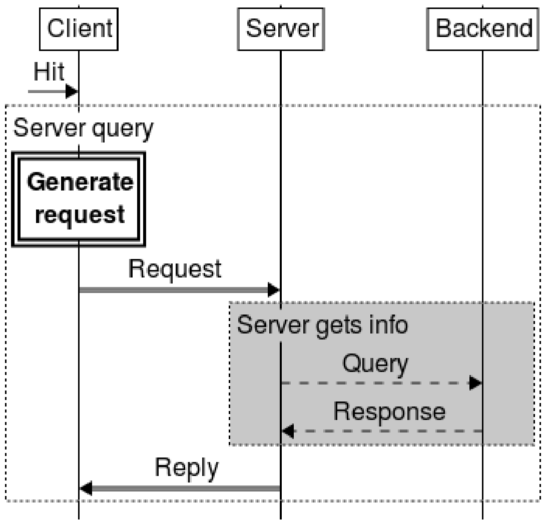
When a box contains arrows, it is not necessary to specify which entities it shall span between, it will be calculated automatically. Also boxes can be nested arbitrarily deep.

```
C [label="Client"],
S [label="Server"],
B [label="Backend"];
->C: Hit;
..:  Server query
{
    C==C: Generate\nrequest;
    C=>S: Request;
    S..B: Server gets info
    {
       S>>B: Query;
       S<<B: Response;
   };
   C<=S: Reply;
};
```



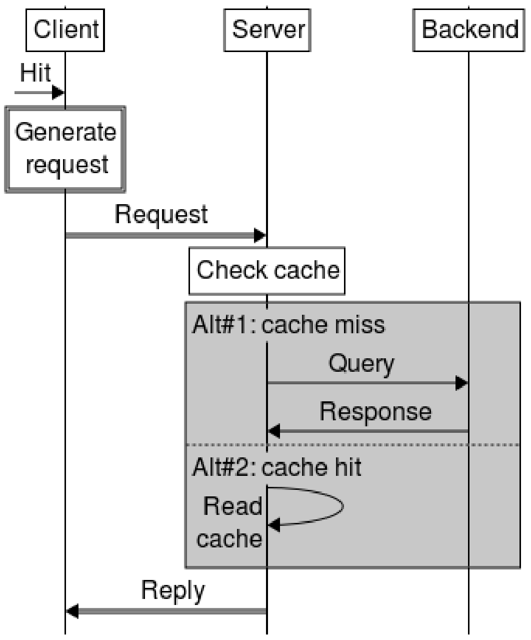
You can shade boxes, by specifying the color attribute. For a full list of box attributes and color definitions, See Section 2.8 [Boxes], page 31 and see Section 2.5 [Color Definition], page 29. It is also possible to make a box ‘weak’ or ‘strong’.

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



Finally, 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 [label="Client"],
S [label="Server"],
B [label="Backend"];
->C: Hit;
C==C: Generate\nrequest;
C=>S: Request;
S--S: Check cache;
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;
```

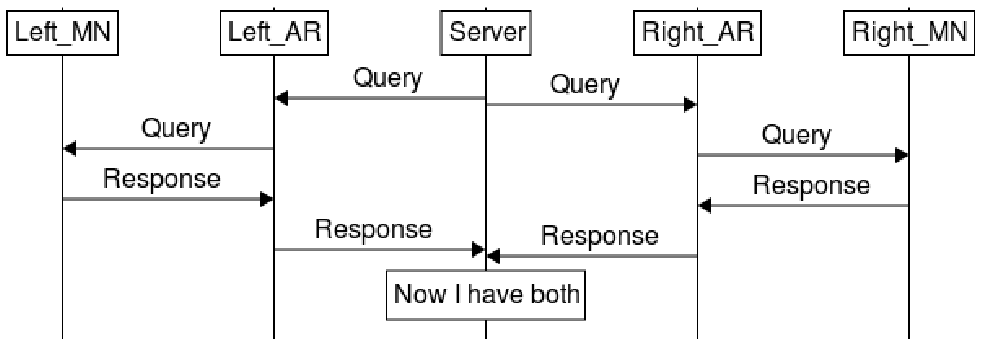


1.8 Parallel Blocks

Sometimes it is desired to express that two separate process happen side-by-side. *Parallel blocks* allow this. See 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;
} {
    nudge;
    Server->Right_AR: Query;
    Right_AR->Right_MN: Query;
    Right_AR<-Right_MN: Response;
    Server<-Right_AR: Response;
};
```

```
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. Note that this means that in the example the lines `Server->Left_AR: Query;` and `Server->Right_AR: Query;` are drawn at exactly the same vertical position and thus the two unidirectional arrows meld into one bidirectional one between ‘`Left_AR`’ and ‘`Right_AR`’. This is prevented by the `nudge;` command in the second block. This command inserts a small vertical space and is most useful in exactly such situations.

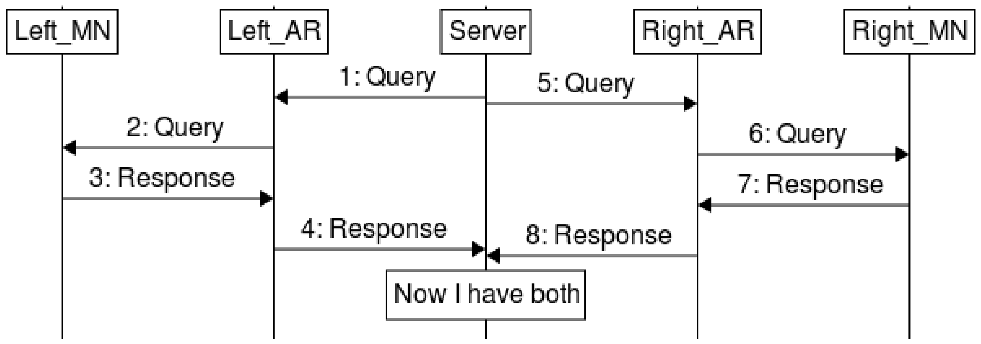
Use parallel blocks with caution. It is easy to specify charts where two such blocks overlap and result in visually unpleasing output. Msc-generator makes no attempt to avoid overlaps between parallel blocks.

1.9 Other features

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

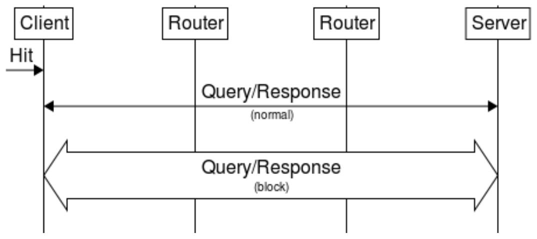
```
numbering=yes;
Left_MN, Left_AR, Server, Right_AR, Right_MN;
{
    Server->Left_AR: Query;
```

```
        Left_AR->Left_MN: Query;
        Left_AR<-Left_MN: Response;
        Server<-Left_AR: Response;
    } {
        nudge;
        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];
```



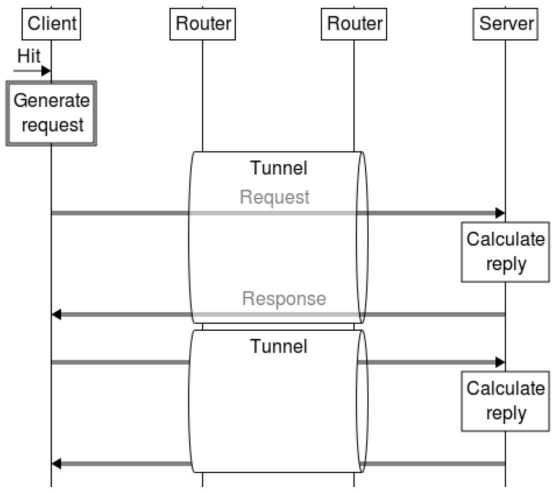
Sometimes a block of actions would be best summarized by a block arrow. This can be achieved by typing ‘`block`’ in front of any arrow declaration.

```
C [label="Client"],
R1 [label="Router"],
R2 [label="Router"],
S [label="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 in it. To achieve this, just type ‘`pipe`’ in front of any box definition. You can add ‘`[solid=255]`’ to make the pipe fully opaque, the default is semi-transparent.

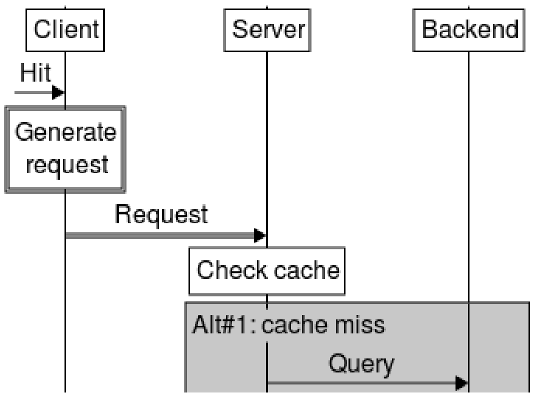
```
C [label="Client"],
R1 [label="Router"],
R2 [label="Router"],
S [label="Server"];

->C: Hit;
C==C: Generate\nrequest;
pipe R1--R2:  Tunnel {
    C=>S: Request;
    S--S: Calculate\nreply;
    C<=S:Response;
};
pipe R1--R2:  Tunnel [solid=255] {
    C=>S: Request;
    S--S: Calculate\nreply;
    C<=S:Response;
};
```



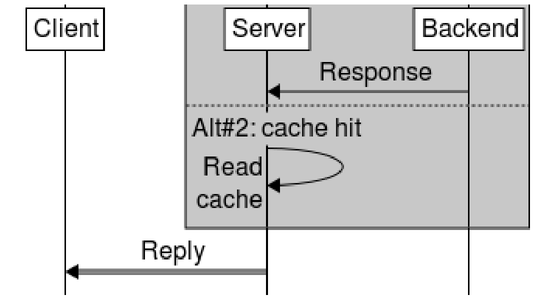
Another useful feature is multi-page support. This is useful when describing a single procedure in a document in multiple chunks. By inserting the '`newpage;`' command, the rest of the chart will be drawn to a separate file. You can specify as many pages, as you want. In order to display the entity headings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

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

Chunk one:



Chunk two:

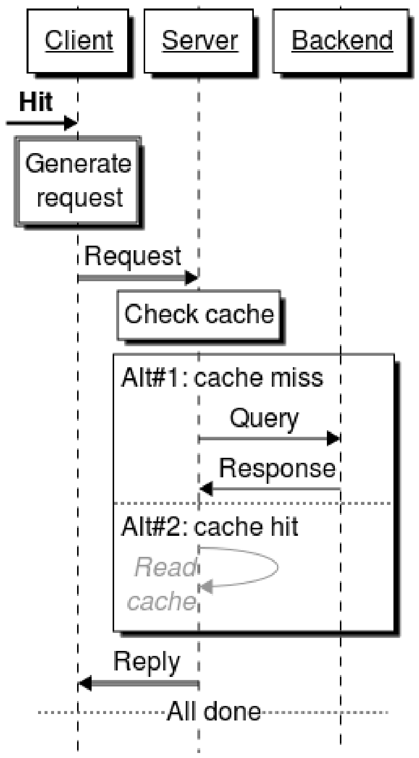


Finally, an easy way to make charts more appealing is through the use of *Chart Designs*. A chart design is a collection of colors and visual style for arrows, boxes, entities and separators. The design can be specified either on the command line after double dashes, or at the first line of the chart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

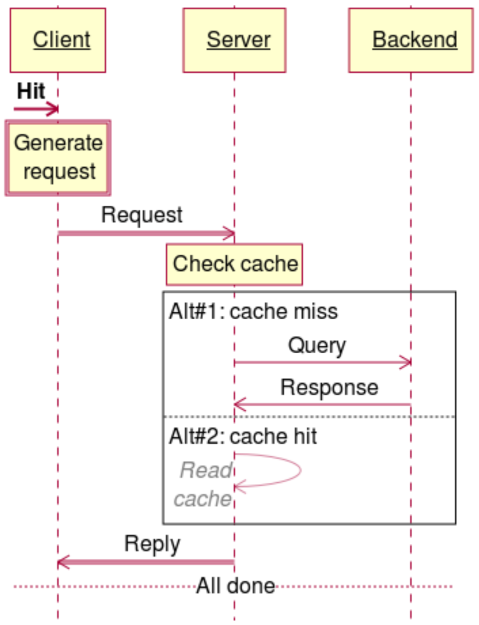
Currently eight designs are supported. '`plain`' was used as demonstration so far. Below we give an example of the other seven.

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

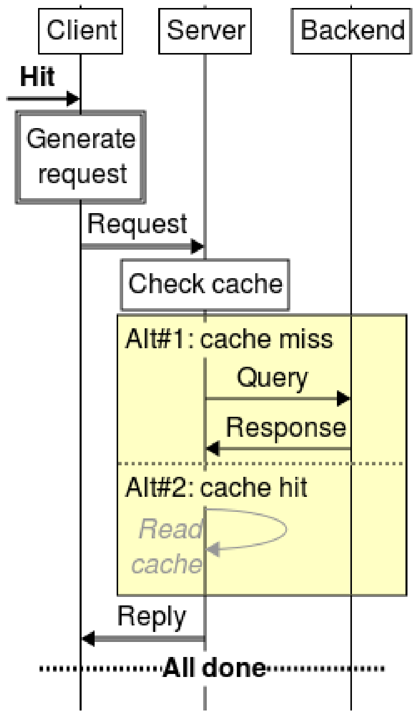
The ‘qsd’ design:



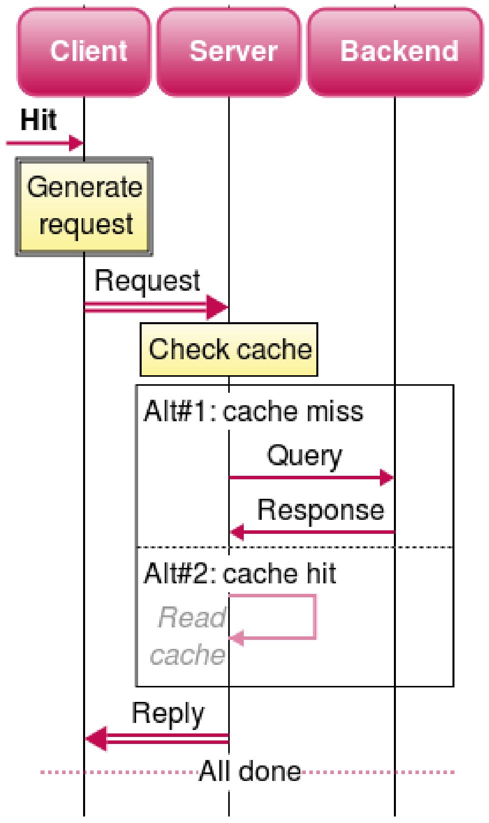
The ‘rose’ design:



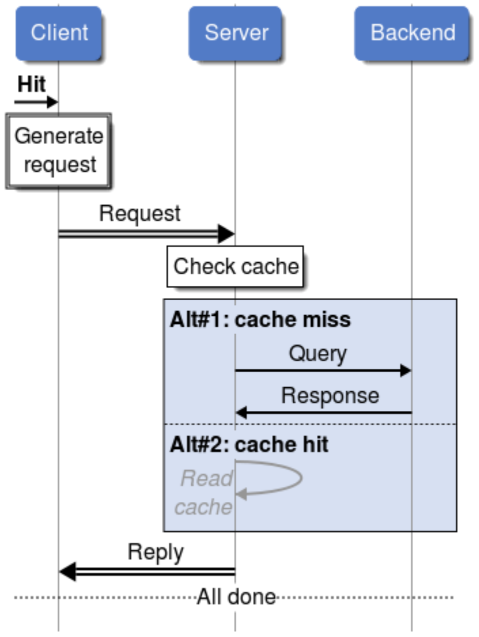
The ‘mild_yellow’ design:



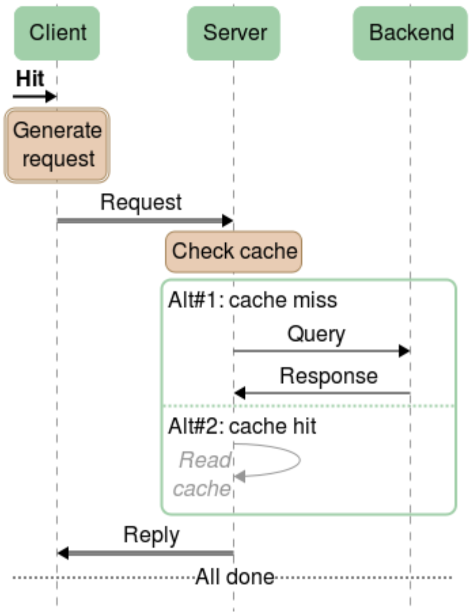
The ‘omegapple’ design:



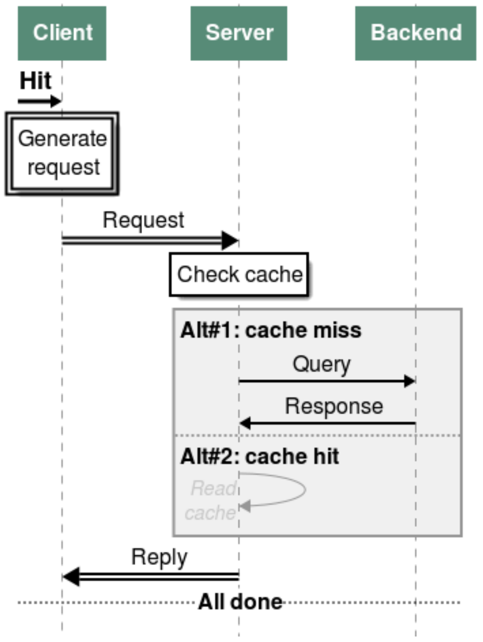
The ‘modern_blue’ design:



The ‘round_green’ design:



The ‘green_earth’ design:



2 Detailed description

2.1 Command-Line Invocation

```
Usage: mscgen [-p] [-T type] [-o file] [infile] [-Wno] [-Werror]
              [--pedantic] [--strict] [--chart option] [--chart design]
       mscgen -l
```

‘`-T type`’ Specifies the output file type, which maybe one of ‘`png`’, ‘`eps`’, ‘`pdf`’ or ‘`svg`’. Default is ‘`png`’.

‘`-o file`’ Write output to the named file. If omitted the input filename will be appended by the appropriate extension. If neither input nor output file is given, ‘`mscgen_out.{png,eps}`’ will be used.

‘`infile`’ The file from which to read input. If omitted or specified as ‘`-`’, input will be read from stdin.

‘`-l`’ Display program licence and exit.

‘`-p`’ Print parsed msc output (for parser debug).

‘`--pedantic`’
When used all entities must be declared before being used.

‘`--strict`’
Requires strict adherence to syntax rules, will bail out on errors

‘`--chart option`’
Any chart option (see Section 2.10 [Chart Options], page 34) can be specified on the command line. These may be overridden by options in the file.

‘`--chart design`’
The design pattern of the chart can be specified on the command line. This will be overridden by a design specified in the file.

‘`-Wno`’ No warnings displayed.

‘`-Werror`’ Warnings result in errors and no output. Also program exit code will indicate error.

The Windows version only supports the ‘`-T`’, ‘`-o`’ and ‘`-l`’ switches on the command line. The ‘`-T`’ swicth must be present otherwise Msc-generator starts in windowed mode. The effect of ‘`--pedantic`’ and ‘`-Wno`’ swicthes can be achieved by setting these among the options of the windowed mode: settings saved there are also valid for the command-line mode. Also on Windows chart design definitions are placed in an external file, see below. That file is also used for command-line invocations.

2.2 Windowed-Mode Invocation (Windows only)

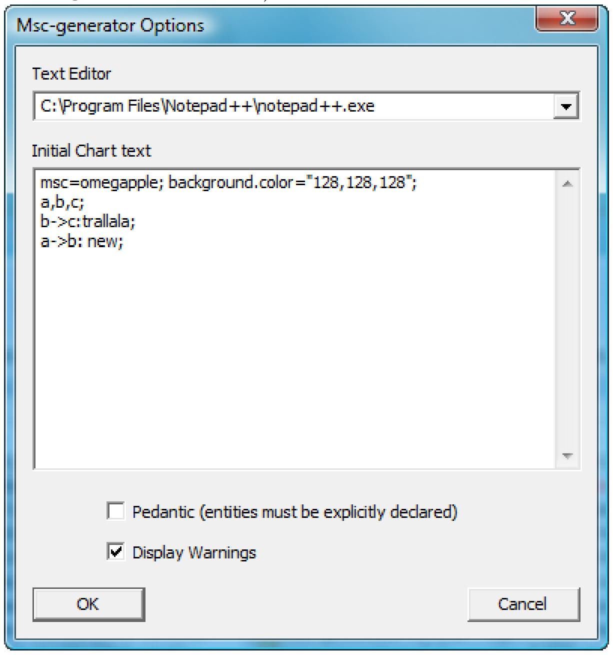
```
Usage: MscGen2 /register
Usage: MscGen2 [infile]
```

In the first form Msc-generator registers itself in the registry, so that OLE requests find it. You may need to run this mode as administrator once, form the final location of the executable, before any work with Msc-generator.

In the second form, the program opens the specified file (text chart description). if no file is given, the default chart is used. The text editor is also immediately opened, so you can start editing right away.

On Windows at startup Msc-generator looks for a file called `designlib.signalling` in the directory where the executable is located. If found its content is parsed as regular chart description before any chart. This is used to define the designs at the end of the previous section. You are free to modify this file to add or change designs. Please avoid text that results in warnings or errors. The design selector combo box on the toolbar is also populated with the designs found in `designlib.signalling`.

In the file menu you can use the Export command to save the chart in various graphics formats. Note that these are usually higher quality than inserting the chart object into a document. The latter is done using an old format Windows Metafile, which does not support color gradients, alpha channel (transparency) and line dashing, so these are emulated. Alpha colors are emulated using a fallback image, gradients are emulated using many smaller fill elements and dashing is done by drawing small dashes, this looks a bit different from other surfaces. Here we note that on PNG drawings a white background is added before drawing the charts. On all other output chart background is transparent (unless specified to be otherwise, using the "background" option).



Text Editor: C:\Program Files\Notepad++\notepad++.exe

```
msc=omegapple; background.color="128,128,128";
a,b,c;
b->c:trallala;
a->b: new;
```

☐ Pedantic (entities must be explicitly declared)

☑ Display Warnings

You can specify a better text editor than notepad (notably one which shows you line numbers). The author suggests using notepad++. You can also specify what is the chart that pops up when a new chart is started. You can place your frequently used constructs here to be readily available when you start a new chart. Finally you can set the pedantic and warning options, which require that you specify all your entities before use and supress warning messages, respectively.

2.3 Specifying Entities

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

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

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 after the definition of the entity. The syntax is the same as for definition — obviously the name identifies an already defined entity. Entity names can contain upper or lowercase characters, numbers or underscores only. Entity names are case sensitive, or else you have to put them between quotation marks every time mentioned. (Use labels to display non-alphanumeric entities instead.)

Formatting styles are explained in Section 2.13 [Styles], page 35.

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.

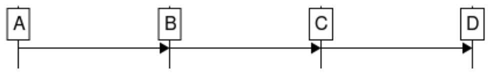
2.3.1 Entity positioning

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.

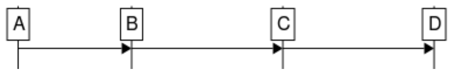
Specifying '`pos`' will place the entity left or right from its default location. E.g., specifying `pos=-0.25` for entity '`B`' makes '`B`' to be 25% closer to its left neighbour. Thus '`pos`' shall be specified in terms of the unit distance between entities.

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`'.

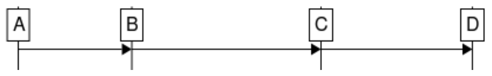
```
A, B, C, D;
A->B-C-D;
```



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

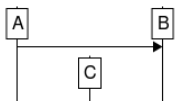


```
A, B [pos=-0.25], C [pos=+0.25], D;
A->B-C-D;
```



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 2.10 [Chart Options], page 34. 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.

2.3.2 Entity attributes

Entity attribute names are case-insensitive. Attributes can take string, number, boolean or color values. String values shall be quoted in double quotes ('`"`') if they contain non-literal characters or spaces (with the exception of built-in style names for convenience). 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 2.5 [Color Definition], page 29.

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 2.6 [Text Formatting], page 30. 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 the previous section for an example.

`relative` This attribute takes the name of another attribute and specifies the horizontal position used as a base for the '`pos`' attribute.

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

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

`fill.color`
Defines the background color of the box of the entity. Specifying '`none`' results in no fill at all.

`fill.gradient`
Defines the background gradient of the box of the entity. It can take five values '`up`', '`down`', '`in`', '`out`' and '`button`'. The first two results in linear gradients getting darker in the direction indicated. The second two results in circular

gradients with darker shades towards the center or edge of the entity box, respectively. The last one mimics light on a button.

`shadow.offset`
: If not set to zero, then the entity 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 box.

`shadow.color`
: The color of the shadow. This attribute is ignored if shadow.offset is 0.

`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.

`line.color`
: Specifies the color of the line around the entity text.

`line.width`
: Specifies the width of the line around the entity text.

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

`line.radius`
: Specifies the roundedness of the box around the entity text. 0 is fully sharp, non-zero values are meant in pixels.

`vline.*`
: Specifies the color, width or type of the vertical line stemming from the entity. This is useful to indicate some change of state for the entity. `vline.radius` have no effect.

`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 2.3.4 [Entity headings], page 25.

2.3.3 Implicit entity definition

It is not required to explicitly define an entity before it is used. Just by 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 2.10 [Chart Options], page 34. Disabling implicit definition is useful to generate warnings for mis-typed entity names.

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

2.3.4 Entity headings

By default when an entity is defined, its heading is drawn at that location. If the '`show=no`' attribute is specified, then the entity heading is not drawn at the location of definition only later, where the entity is turned on again by using '`show=yes`'. Mentioning an entity after its definition with '`show=yes`' will cause an entity heading to be drawn into the chart even if the entity is shown. This can be useful for long charts.

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 is useful after a `newpage;` command, see Section 2.11 [Commands], page 34.

2.4 Specifying Arrows

Arrows can be specified using the following syntax.

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

arrowsymbol can be any of '`->`', '`<-`' or '`<->`', the latter for bidirectional arrows. `a->b` is equivalent to `b<-a`. This produces an arrow between the two entities specified using a solid line. Using '`>`'/'`<>`', '`>>`'/'`<<>>`' or '`=>`'/'`<=>`', will result in dotted, dashed or double line arrows, respectively. These settings can be redefined using styles, see Section 2.13 [Styles], page 35.

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

It is possible to specify multi-segment arrows, such as `a->b->c` in which case the the arrow will expand from '`a`' to '`c`', but an arrow head will be drawn at '`b`', as well. This is used to indicate that '`b`' also processes the message indicated by the arrow. The arrow may contain any number of segments, and may also start or 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.

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.

Arrows have a '`label`' attribute, which specifies the text to be displayed for the arrow. It defaults to the empty string. You can use all character formatting features in labels, see Section 2.6 [Text Formatting], page 30. As a shorthand, it is possible to specify the label attribute after the arrow definition and a colon. Thus the first three lines below result in the same text.

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

In the second and third version heading and trailing spaces are removed from the label. If these are needed, or the label needs to contain a opening bracket ('`[`') or semicolon ('`;`') then use quotations, as in the fourth line above.

2.4.1 Arrow Numbering

Arrows can be auto-numbered. It is a useful feature that allows easier reference to certain steps in a procedure from explanatory text. For this the '`numbering=yes`' chart option must be set. That attaches a number to every arrow (and box, separator, too) automatically. You can use scoping to enable or disable numberig for blocks of the chart, see Section 2.12 [Scoping], page 35. Numbering can also be individually turned on or off separately for each element using the `number` attribute. To achive this, use a boolean value to the '`number`' arrow attribute. Styles can also control numbering.

In addition, the number of each individual arrow can also be specified using `number` by assigning a specific number. This automatically turns numbering on for that arrow. Subsequent elements (if they have numbering on) continue to be numbered from the specified number. This allows re-starting the numbering from a fixed value, e.g., at the beginning of a logical block. This avoids renumbering the whole chart when one element is inserted to a previous block, and so less numbers have to be rewritten in the explanatory text. Currently there is no support for multi-level numbering. (such as `1.1`, `1.2`, etc.)

2.4.2 Arrow Attributes

Arrows can have the following attributes.

`label` This is the text associated with the arrow.

`number` Can be set to `yes`, `no` or to a number, to turn numbering on or off, or to specify a number, respectively.

`compress` Can be set to `yes` or `no` to turn compressing of this arrow on or off. Compression for individual arrows means that the arrow is moved upwards until bumping into someting. Thus turning compression on or off for individual arrows impact the spacing above the arrow, and not below it.

`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.

`text.ident`
This can be '`left`', '`center`' or '`right`' and specifies text alignment. The default is centering. It can be abbreviated as `ident`.

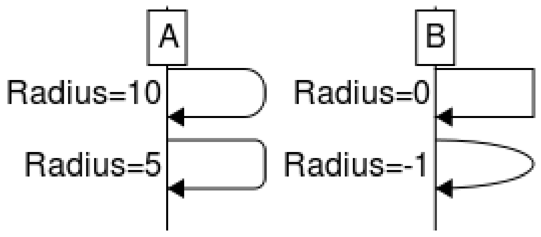
`text.color`
Sets the color of the text.

`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 text, see Section 2.6 [Text Formatting], page 30

`line.color`
`line.width`
Set the color and the width of the line.

`line.radius`
For arrows starting and ending at the same entity, this specifies the roundness of the arrow. 0 is fully sharp, positive values are meant in pixels, a negative value will result in a single arc.

```
hscale=auto;
{
  A->A: Radius=10 [line.radius=10];
  A->A: Radius=5 [line.radius=5];
} {
  B->B: Radius=0 [line.radius=0];
  B->B: Radius=-1 [line.radius=-1];
};
```



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

`arrow.color`
: The color of the arrowheads.

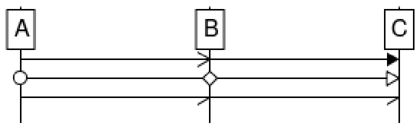
`arrow.type`
: Specity the arrowhead type. The values can be '`half`', '`line`', '`empty`', '`solid`', which draw a single line, a two-line arrow, an empty triangle and a filled triangle, respectively. '`diamond`' and '`empty_diamond`' draws a filled or empty diamond, while '`dot`' and '`empty_dot`' draws a filled or empty circle. This attribute sets both the '`endtype`' and '`midtype`', see below.

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

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

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

```
A->B->C: [arrow.midtype=line];
A->B->C: [arrow.type=empty,
          arrow.midtype=empty_diamond
          arrow.starttype=empty_dot];
A->B->C: [arrow.type=half];
```



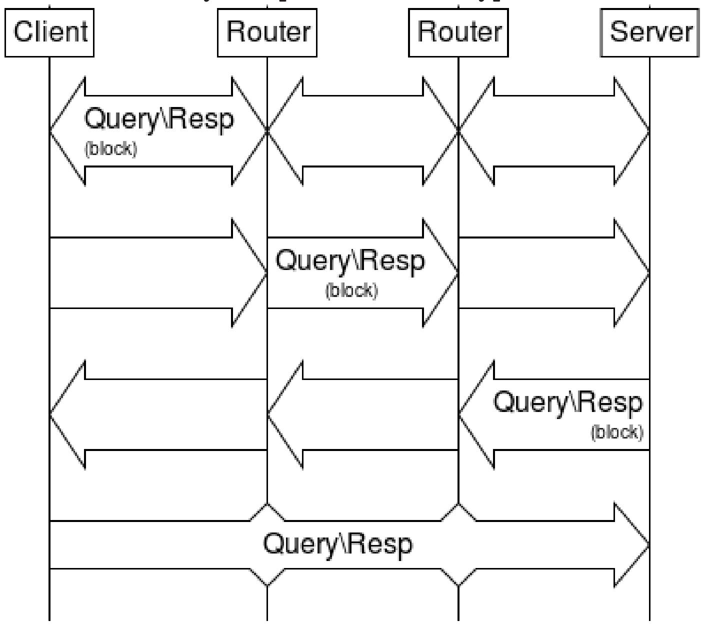
Note that default values can be changed using styles, see Section 2.13 [Styles], page 35.

2.4.3 Block Arrows

When typing '`block`' in front of any arrow definition, it will become a *block arrow*. The label of a block arrow is displayed inside the block. In addition to the attributes above, block arrows also have fill attributes, similar to entities. Currently the ‘shadow’ attributes are not supported for block arrows.

All arrowheads are supported, but ‘half’, ‘line’, ‘empty’ and ‘solid’ all result in the same output: an arrow. ‘diamond’ and ‘empty_diamond’ also results in the same figure just as ‘dot’ and ‘empty_dot’. If the arrow has multiple segments and the type of the inner arrowheads is either of ‘half’, ‘line’, ‘empty’ ‘solid’, the block arrow is split into multiple smaller arrows. In this case the arrow label is placed into the leftmost, rightmost or middle smaller arrows, depending on the value of the ‘text.label’ attribute.

```
hscale=auto;
C [label="Client"], R1 [label="Router"],
R2 [label="Router"], S [label="Server"];
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];
```



2.5 Color Definition

Colors can be defined by specifying red, green and blue components separated by commas. An optional fourth value can be added for the alpha channel. Values can be either between zero and 1.0 or between 0 and 255. If all values are less or equal to 1, the former range is assumed. If any value is negative or above 255 the definition is invalid.

If a color definition is assigned to an attribute or option, it must be quoted, e.g., color="255,0,0" for full red color.

Color aliases can be defined using the defcolor command as below.

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

Alias names are case-sensitive and can only contain letters, numbers and underscores. An alias can be used any place where a color definition can be used, including the definition of an alias. (Aliases do not have to be quoted.) Aliases can be re-defined using the '`defcolor`' command. Finally, any place, where an alias can be specified, an additional numeric value can be specified overriding the alpha channel (quotation is needed in this case).

The following aliases are defined by default: '`none`', '`white`', '`black`', '`red`', '`green`', '`blue`', '`gray`' and '`lgray`', the first for completly transparent color, and the latter for light gray. The definition of these can be overridden.

Msc-generator honors scoping. Color definitions are valid only until the next closing brace '`}`'. This makes it possible to override a color only for parts of the file, returning to the default later. Note that you can start a new scope any time by placing an opening brace. See Section 2.12 [Scoping], page 35 for more on scopes.

2.6 Text Formatting

Entity, divider, arrow and emphasis box labels can contain formatting characters. Each formatting character is escaped by the backslash '\' character. To insert a literal backslash, type double backslahses '\\'. The following formatting characters are 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.

'`\f(`*font face name*`)`'
Changes the font face. Available font face names depend on the backend.

'`\0..\9`' Inserts the specified number of points as line spacing below the current line.

'`\c(`*color definition*`)`'
Changes the color of the text. Aliases or direct rgb definitions can both be used, no quotation is needed. You can omit the color and just use '`\c()`', which resets the color to the one at the beginning of the label.

'`\s(`*style name*`)`'
Applies the specified style to the text. Naturally only the '`text.*`' attributes of the style are used. You can omit the style name and specify only '`\s()`', which resets the text style to the one at the beginning of the label.[1]

[1] Note that the '`\s`' formatting escape was used to switch to small font in 1.x versions of Msc-generator (in 2.x '`\-`' 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 use small font.

'`\m{u,d,l,r,i}(`*num*`)`'
: Changes 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 non-negative integer. Intra-line spacing comes in addition to the line-specific spacing inserted by '`\0..\9`'. Defaults are: '`\mu(2)\md(2)\ml(4)\mr(2)\mi(0)`'.

'`\m{n,s}(`*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)`'.

'`\p{l,c,r}`'
: Changes the identation to left, centered or right. Applying at the beginning of a line will apply new identation to that line and all following line within the label. Applying at the beginning of the first line will change identation of the whole label. This latter use is identical to specifying the '`text.ident`' attribute.

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 character is displayed literally without warning, except for color definitions, which, if unrecognized, will be removed from displayed text with a warning or error depending on the value of option '`strict`'.

2.7 Separators

Two types of separators are defined. '`---`' draws a horizontal line across the entire chart with potentially some text across it. '`...`', on the other hand, draws no horizontal line, but makes all vertical entity lines dotted, thereby indicating the elapse of time.

A further type of separator is a simple vertical space. This can be specified by simply omitting arc definition and having only attributes. '`[];`' simply inserts a lines worth of vertical space. You can add text, too by specifying a label.

Separators take the `label`, `color`, `text.ident`, `text.color`, `line.type`, `line.width`, `line.color`, `compress` and `number` attributes with the same meaning as for arrows. In addition, the type of the vertical line can be specified with `line.type`, which defaults to '`dotted`' for '`...`' separators and to '`solid`' for '`---`' separators. Other values are `dashed`, `none` and `double`. Again, note that the default values can be changed by using styles, see Section 2.13 [Styles], page 35.

2.8 Emphasis Boxes

Emphasis 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 emphais box, while in the third case no such arrows are added, making the emphasis box *empty*.

The syntax definition for emphasis boxes is as follows.

```
entityname emphboxsymbol entityname [attr = value | style, ...]
{ arcdefinition; ... };
```

As with arrows the two entity names specify the horizontal span of the emphasis box. These can be omitted (even both of them), making the emphasis box auto-adjusting to the size to cover all the arcs within. Specifying the entity names is useful if you want a deliberately larger box, or if you specify an *empty* emphasis box.

The *emphboxsymbol* can be '`..`', '`++`', '`--`' or '`==`' for dotted, dashed, solid and double line emphasis boxes, respectively.

Emphasis boxes take attributes, controlling colors, numbering, text identation quite similar to arrows. Specifically emphasis 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 emphasis box attributes are `label`, `number`, `compress`, `color`, `text.*`, `line.*`, `shadow.*` and `fill.*`. The latter specifies the background color of the emphasis box, while `line.*` specifies the attributes of the line around. Note that `color` for emphasis boxes is equivalent to `fill.color`. `text.ident` defaults to centering for empty emphasis boxes and to left identation for ones having content.

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

If an emphasis box definition is not followed by a semicolon, but another emphasis box definition, then the second emphasis box will be drawn directly below the first one. This is useful to express alternatives, see Section 1.7 [Drawing Boxes], page 11 for an example.

The subsequent emphasis 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.

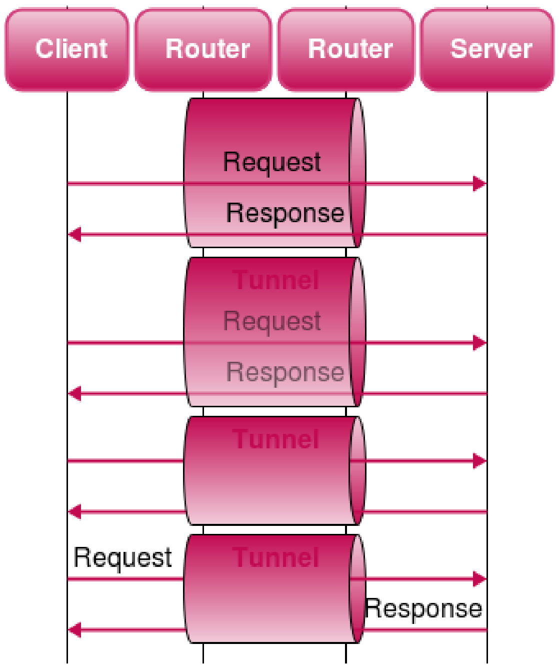
The horizontal size of the combined emphasis box is determined by the first definition, entity names in subsequent boxes are ignored.

2.8.1 Pipes

By typing '`pipe`' in front of a box definition, it is turned into a pipe. Pipes take an extra attribute, called '`solid`' that 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[2].

[2] On Windows this means that embedded OLE objects having semi-transparent pipes will be drawn using a fallback image. This increases object size and somewhat decreases quality and stems from the limitations of the metafile format.

```
msc=omegapple;
C [label="Client"],
R1 [label="Router"],
R2 [label="Router"],
S [label="Server"];

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

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=1.0] {
    C->S: Request;
    C<-S: Response;
};
pipe R1--R2:  Tunnel [solid=1.0] {
    C->S: \plRequest;
    C<-S: \prResponse;
};
```



2.9 Parallel Blocks

Sometimes it is desired to express that two separate process happen side-by-side. *Parallel blocks* allow this. Simply place the each of the parallel blocks between '`{}`' marks and write them one after the other. You can specify as many parallel blocks as you want. The last parallel block shall be termiated with a semicolon. The order of the blocks is irrelevant, with the exception of numbering, which goes in the order the blocks are specified in the source file. It is possible to place anything in a parallel block, arrows, emphasis boxes, or other parallel blocks, as well. However, entity commands may result in unpredictable results, similar to the '`heading;`' command, so use with caution.

The top of each block will be drawn at the same vertical position. If you start with two arrows, they may be aligned and appear as a single arrow. To avoid this use the `nudge;` command in one of the blocks which inserts a small vertical space top mis-align accidentally aligned arrows.

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

Use parallel blocks with caution. It is easy to specify charts where two such blocks overlap and result in visually unpleasing output. Msc-generator makes no attempt to avoid overlaps between parallel blocks.

2.10 Chart Options

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 must be specified as the first thing in the file before any other chart option. It can only be specified once.

‘`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 arcs. The ‘`pos`’ attribute of entities will be ignored except when influencing the order of the entities. This attribute can only be specified at the beginning of the file, before any entites are defined (implicitly or explicitly).

‘`numbering`’
: This option can take ‘`yes`’ or ‘`no`’ value, the default is no. If turned on, arrows, emphasis boxes and dividers will be numbered. Only elements with a non-empty label are assigned a number, so in order to number an arrow without label, assign at least a space or a formatting character as label.

‘`compress`’
: This option can take a boolean value, and defaults to off. If turned on, arcs are vertically spaced as close, as possible, to conserve vertical space.

‘`pedantic`’
: This option can take a boolean value. It defaults to no, but once set to yes (either from the command line or from within the input), it cannot be set to no again. When turned on, then all entities must be defined before being used. If an entity name is not recognized in an arrow or emphasis box definition an error or warning is generated, depending on the value of ‘`strict`’.

‘`strict`’
: If this option is set to yes, any syntactic or schemantic error will prevent drawing. In case it is set to no, msc-generator will try to interpret the input file, ignoring erroneous or unrecognized attributes, options or arcs, giving up only at fatal errors. This attribute defaults to no, but once set to yes (either from the command line or from within the input), it cannot be set to no again.

2.11 Commands

Besides entity definitions, arcs, separators, emphasis boxes, parallel block definitions and options, msc-generator also has a few commands. These have been mentioned elsewhere, in this section we collect them for completeness.

‘`nudge;`’
: This command inserts a small vertical space useful to misaling two arrows in parallel blocks, see Section 2.9 [Parallel Blocks], page 33.

‘`newpage;`’
: This command starts a new page. The graph will be split into two (or more) graphics output files with the page number appended to the file name.

‘`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.

‘`defcolor`’
: This command is used to define color aliases, see Section 2.5 [Color Definition], page 29.

‘`defstyle`’
: This command is used to define styles, see Section 2.13 [Styles], page 35.

2.12 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 chart option setting within a scope has its effect only within the scope. Thus, if you set, e.g., the ‘`numbering=yes`’ chart option just after an opening brace, only elements in that block will be numbered (if otherwise numbering was off previously).

Scoping also applies to defining new color aliases and styles and also to redefining them. (See the sections below on these topics.) Note that you can nest scopes arbitrarily deep and can also use the parallel block syntax with a single parallel block to manually open a new scope, such as below;

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

2.13 Styles

It is possible to define a group of attributes as a style and later apply collectively. Styles can be defined using the ‘`defstyle`’ command, as below.

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

Multiple styles can be defined in a single command. Any ‘`line.*`’, ‘`fill.*`’, ‘`text.*`’, ‘`arrow.*`’, ‘`vline.*`’, ‘`shadow.*`’, ‘`solid`’, ‘`number`’ or ‘`compress`’ attribute can be part of a style. Styles can be extended later. Attributes not set in a style will be unaffected when applying the style to an element. Scoping is also applied to style definitions, similar to color definitions, thus any effect terminates at the next closing brace.

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, simply define a style for them.

You can use a style name anywhere where attributes are allowed. Styles can simply be applied to arrows, emphasis boxes, separators an entities (and in defining other styles) by naming the style, such as below.

```
defstyle weak [line.width=0.5, line.type=dotted, arrow.type=line];
defstyle strong [line.width=2];
a->b [weak]: A tentative message;
a<-b [strong]: A definitive answer;
```

There are a number of pre-defined styles. First there is a default style for each element: '`arrow`', '`emphasis`', '`emptyemphasis`', '`divider`', '`blockarrow`', '`pipe`' and '`entity`'. They represent the default settings for the respective elements. By changing them, you can effectively change the appearance of the entire chart. If you want to change a set of attributes for multiple elements (such as both for arrows and separators) simply list these separated by commas before the opening square brackets.

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

It will apply to both.

Then there are further styles defined for each arrow, emphasis box and separator symbol. Four for arrows: '`->`', '`=>`', '`>`' and '`>>`'[3]; another four for block arrows: '`block->`', '`block=>`', '`block>`' and '`block>>`'; four for emphasis boxes: '`--`', '`==`', '`++`' and '`..`'; another four for pipes: '`pipe--`', '`pipe==`', '`pipe++`' and '`pipe..`'; and two for separators: '`---`' and '`...`'. Redefining enables you to quickly define, e.g., various arrow styles and use the various symbols as shorthand for these. When redefining these, use quotation mark, such as

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

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

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

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

2.14 Chart Designs

A chart design is a collection of color definitions, styles and the value of the '`hscale`', '`numbering`' and '`compress`' attributes. You can define or re-define chart designs by using the syntax below.

```
defdesign designname {
   [ msc=parent design ]
   color definitions, ...
```

[3] These are also applied to bi-directional arrows and arrows pointing from an entity back to itself. Thus there is no separate '`<->`' style, for example.

```
        style definitions, ...
        options, ...
    }
```

First you can name an existing design to inherit from. If omitted it is always the '`plain`' style. Thus in each design definition the styles mentioned in the previous section are always present and fully specified. Then you can define colors, styles in any order and/or set one or more of the three attributes mentioned above.

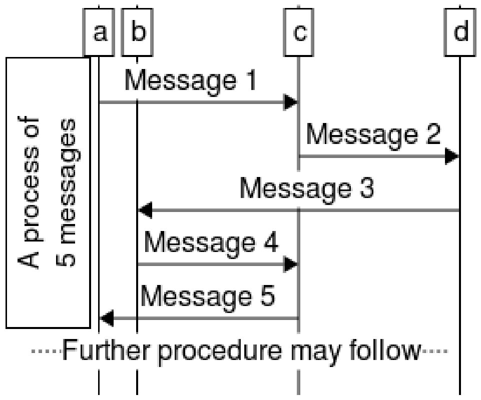
2.15 Experimental Features

The features described in this section are only available in v2.3.x of Msc-generator and not in v2.2.x for which the rest of this document is prepared.

2.15.1 Verticals

A new design element is added, which is a vertical block arrow or box, collectively called a *vertical*. It is useful to comment on a procedure going on besides, or to indicate one message triggering another one below. Consider the example below.

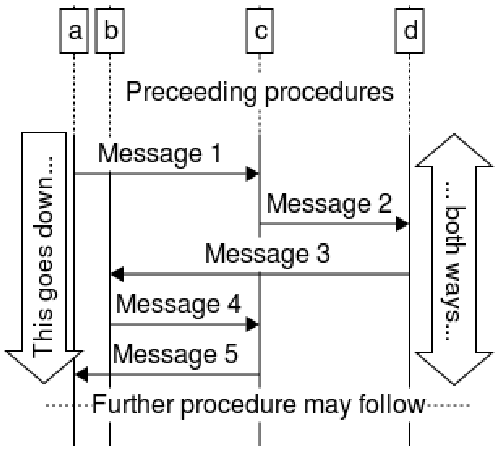
```
hscale=auto;
a, b, c, d;
mark top;
a->c:  Message 1;
c->d:  Message 2;
d->b:  Message 3;
b->c:  Message 4;
c->a:  Message 5;
vertical top-- at a- [makeroom=yes]
       A process of\n5 messages;
---:  Further procedure may follow;
```



The one before the last line contains the new element. The vertical position of the vertical arrow or box is specified after the '`vertical`' keyword. It is defined in terms of vertical *markers*. Markers can be placed with the '`mark`' command. Tthe third line of the example 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. Between the two positions, one of the entity symbols or arrow symbols can be used: '`--`', '`..`', '`++`', '`==`', '`->`', '`=>`', '`>`' or '`>>`'. The arrow symbols can be used also in bidirectional or reverse variants and draw a vertical arrow.

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 '`<entyity>`', '`<entity>-`', '`<entity>+`' or '`<entity1>-<entity2>`', respectively.

```
hscale=auto;
a, b, c, d;
...:  Preceeding procedures;
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- [makeroom=yes]
       This goes down...;
vertical top<-> at d++ [makeroom=ye
       ...  both ways...;
---:  Further procedure may follow;
```



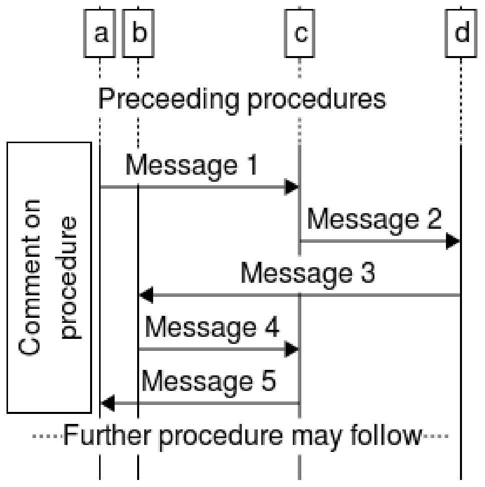
In the second vertical arrow, the horizontal position is specified as '`<entity>++`'. This avoids the effect with the first one, where the tip of the arrow overlaps the entity line of entity '`a`'.

Both the mark command and the vertical element can have an '`offset`' attribute which takes a number shifts the position by that many pixels. In case of markers a positive '`offset`' shifts the position downwards, whereas in case of verticals it shifts to the right.

Verticals have two additional attributes. '`readfrom`' can be either '`left`' or '`right`' and specifies which direction the vertical text can be read. The other '`makeroom`' is a boolean value defaulting to no. 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. It is not perfect, as it does not take many visual elements into account, so beware.

It is also possible to omit both markers from a vertical but only if it is specified inside a parallel block. In this case it will span from the current location to the bottom of the longest of the previous blocks. Msc-generator gives an error if the vertical is specified this way in the first block of a series of parallel blocks.

```
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- [makeroom=yes]
         Comment on\nprocedure;
};
---:  Further procedure may follow;
```



Finally, when a vertical is specified at the beginning of the file it will be drawn under other elements. When specified at the end, the will be drawn over them.