Msc-generator

A tool to draw message sequence charts
(version v3.0.0, 2 March 2011)

Zoltan R. Turanyi

This manual is for Msc-generator (version v3.0.0, 2 March 2011), a tool to draw message sequence charts from a textual description.

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

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

This version of msc-generator is 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 Windows version is written using MFC.

1 What's new in Msc-generator 3.0

The improvements added since version 2.5 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.

Language improvements.

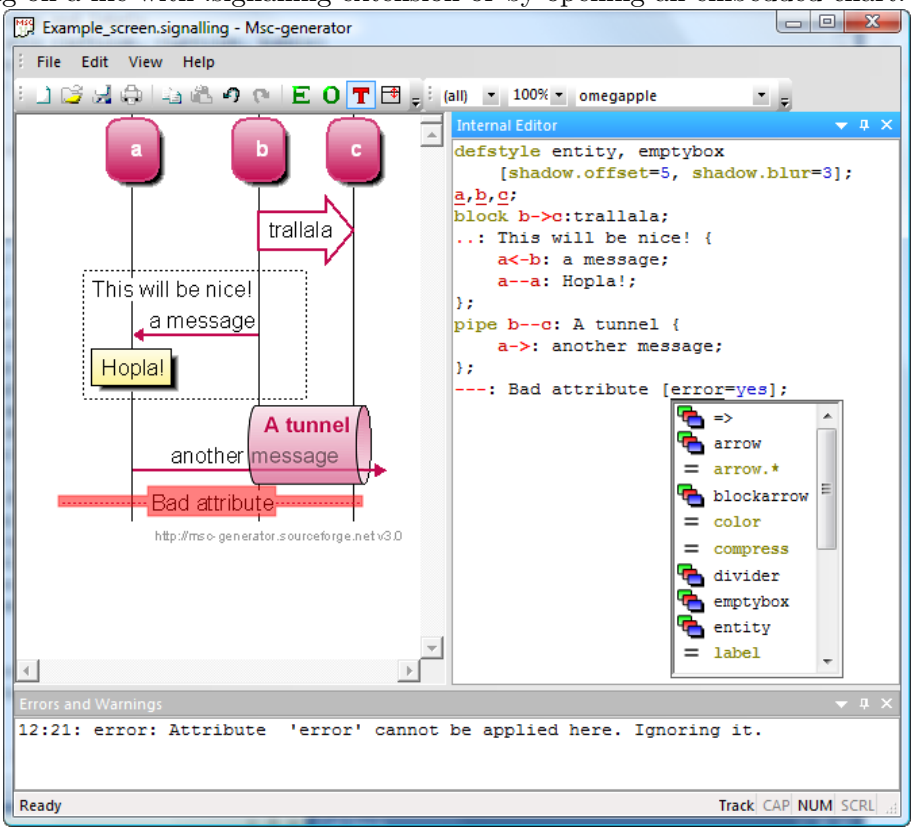
- Improved layout engine. Compression now places also round or triangular elements correctly.
- Rewritten the rendering engine. As one consequence, block arrows can have '`shadows`'. Designs using shadows have been updated.
- Added '`triple`', '`triple_thick`', '`long_dashed`' and '`dash_dotted`' line types.
- Added new corner types in addition to rounded ones. Use the '`line.corner`' attribute with '`bevel`' and '`note`'.
- Added new arrowhead types ('`triple_*`', '`double_*`', '`sharp`' and '`empty_sahrp`').
- Added richer drawing of block arrowhead types: now '`empty_*`' types draw differently. Also added '`empty_inv`' and '`sharp`' type.
- Expanded auto-numbering, with multi-level and roman numbers, letters and formatting options.
- Added '`text.*`' chart option to control the default font.
- Added '`side`' attribute to pipes to control which side the pipe looks toward. Also, the '`readfromleft`' attribute of verticals have been renamed to '`side`'.
- Verticals are not considered experimental any more and will be fully maintained with the rest of the laguage.
- Added '`show`' and '`hide`' keywords. Using them in front of entities will make make the entity to appear or get hidden. This is a shorthand for specifying '`[show=yes]`' or '`no`'.
- Added the '`bye`' keyword. Anything written after this will be ignored.

Interface improvements

- Added automatic typing suggestions (hints) and auto-completion to the internal editor. Pressing Ctrl+Space any place will bring up a list of possible values to continue typing. If there is only one way to complete the word, instead of popping a list, the word will auto-completed. In addition, hints pop up automatically during typing. This can be turned off in the preferences.
- Added auto-split mode (new button on the toolbar) which attempts to keep entity headings on screen by splitting the view so that the user can scroll to the lower part of the chart. This works also in Full Screen mode.
- Added a dialog with scaling options when exporting the chart to a PNG or BMP bitmap.
- Similar options ('`-x`', '`-y`', '`-s`') have been added to the command line tool to control width/height or scaling.
- Supposedly nicer text on Windows XP (on which I have little chance of testing).

2 Getting started

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



The Msc-generator window has the usual elements of a Windows application: menu bar, toolbars and a status bar. We will briefy discuss these here and give a more detailed description in Chapter 4 [Usage Reference], page 27.

You can use the scrollbars or the arrow keys on the keyboard 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 split the chart view into two parts using the small control at the top of the vertical scrollbar. This allows you, e.g., to see the heading of the chart and focus on a later part at the same time.

2.1 Working with Charts

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

You can use the regular Windows File menu operations 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 `File|Export...` menu. In the file menu you also find the usual Print and Print Preview commands.

The Edit menu 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 menu, using the toolbar buttons or by pressing Ctrl+Z or Ctrl+Y. Similar search and replace operations for the text editor can also be accessed from the Edit menu.

2.2 Toolbars

There are two toolbars available. The first one contains a set of usual icons for file new, open, save, print, copy, paste, undo and redo. There are four additional buttons, which can be used to start an external text editor (see Section 4.2 [External Editor], page 27), to zoom to overview mode (see Section 2.3 [Zooming], page 4), to enter tracking mode (see Section 2.4 [Tracking Mode], page 5) or to turn automatic splitting (Section 2.5 [Auto Split], page 5) on or off.

The second toolbar has three drop-down lists. The first drop-down can be used to select which page of the chart is displayed. If '`all`' is selected then pagination is ignored and the whole chart is shown. (See Section 5.11 [Multiple Pages], page 61 for more info on pagination commands.)

The second drop-down list can be used to set the zoom factor of the chart. See Section 2.3 [Zooming], page 4 for more on zooming.

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 5.15 [Chart Designs], page 64 for more info on chart designs.

2.3 Zooming

You can zoom the chart in and out using the commands in the View menu and the associated shortcut keys. The zoom drop-down allows setting a specific zoom value. Finally, the easiest way to zoom is to use the mouse wheel with the Ctrl key pressed.

You can easily set the right zoom factor by selecting certain View menu commands. `View|Zoom to overview` changes the window size to as large as possible and adjusts zoom to fit the entire chart into the window. This is useful to get an overview of a chart. You can get the same effect by pressing the '`O`' shaped button on the toolbar. `View|Adjust width` changes the width of the window to fit the width of the chart at the current zoom factor. Finally, `View|Fit to width` changes the zoom factor to fit the width of the chart to the current window.

You can make Msc-generator apply one of the above three zoom adjustments after every update by selecting the appropriate `View|Keep...` commands.

You can also view the chart in full screen mode, by pressing F11. All shortcut keys and mouse zooming and panning works in full screen mode. You can exit full screen mode by pressing Escape.

2.4 Tracking Mode

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

If you double-click the chart you enter Tracking Mode, where the above behaviour becomes permanent. Visual elements are selected just by hoovering above them. You can enter tracking mode also by the '`T`' button on the toolbar or by pressing Ctrl+T. If you move around in the text editor, the visual element corresponding to the text around the current cursor position is highlighted.

You can leave Tracking Mode by pressing Escape.

2.5 Auto Split

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

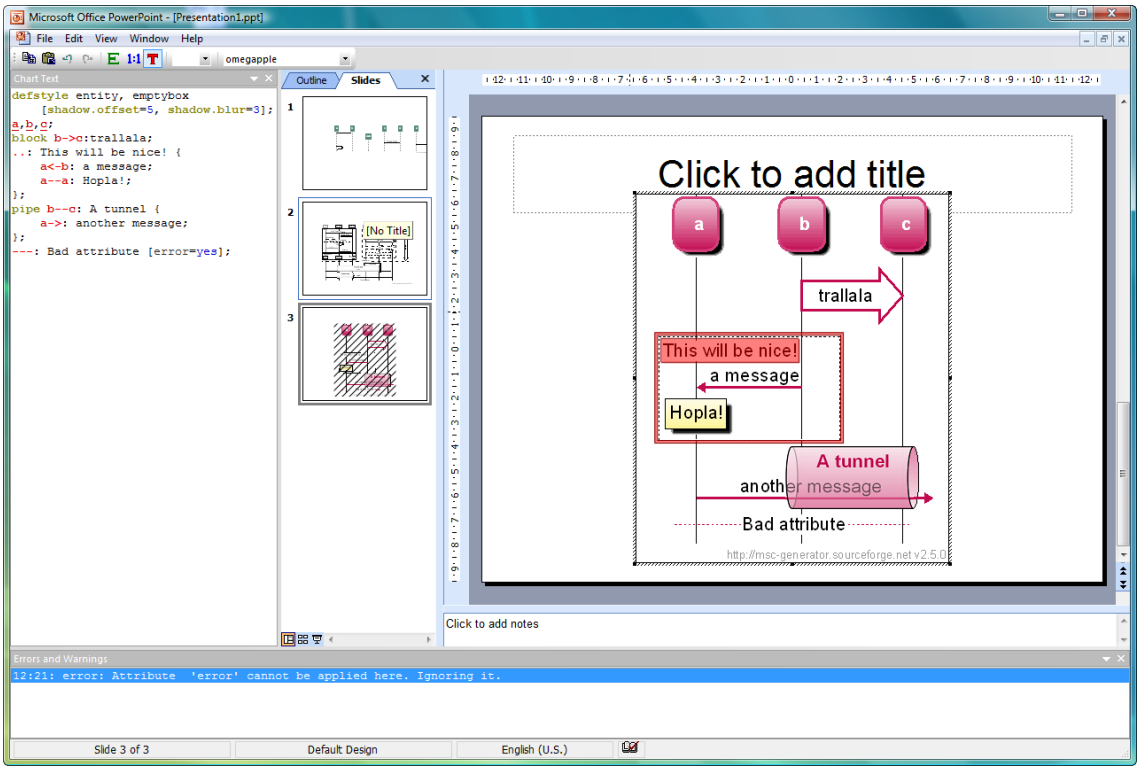
Note that it is possible to define charts where there is no meaningful row of entity headings at the top. In such cases, Msc-generator will get confused and Auto Split is of no use. Similar, in case of multi-page documents Msc-generator will adjust the upper view based on the entity heading on the first page even if a subsequent page is viewed.

Auto Split also works in Full Screen mode.

2.6 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 selecting `Edit|Copy entire chart` or (the corresponding toolbar button) and paste it into the compound document. Later you can edit the chart in-place by double clicking the chart in the document\[1].

[1] In preferences you can set if you want in-place editing when double clicking an embedded object or you want a separate window to open with the chart. The default is set to opening a separate window, due to problems with in-place editing in Windows.



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

Unfortunately, Windows displays embedded objects badly if their aspect ratio is changed. To fix this, an additional button is included on the toolbar during in-place editing to reset the aspect ratio of the chart back to 1:1.

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.

2.7 Command-line Tool

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

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

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

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

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

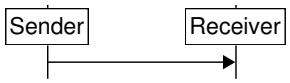
3 Language Tutorial

In this chapter we give a step-by-step introduction into the language of Msc-generator. At the end you will master most of the language to create charts. Further details (mostly on controlling appearance) are provided in Chapter 5 [Language Reference], page 33.

3.1 Defining Arrows

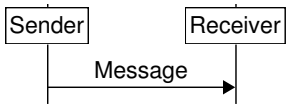
Message sequence charts consits of *entities* and *messages*. The simplest file consists of a single message between two entities: a '`Sender`' and a '`Receiver`'.

```
Sender->Receiver;
```



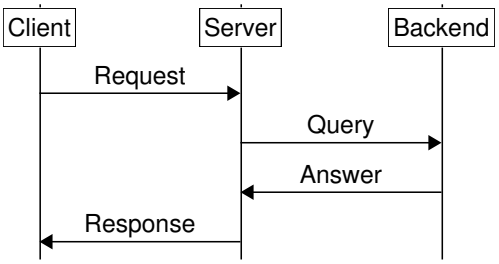
The message may have a label, as well.

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



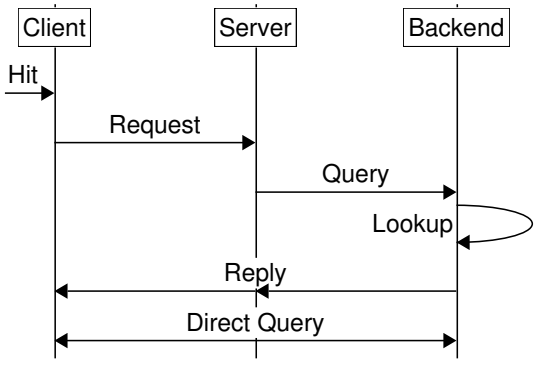
A more complicated procedure would be to request some information from a server, which, in turn, queries a backend. Note that everything in a line after a '`#`' is treated as a comment and is ignored by Msc-generator.

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



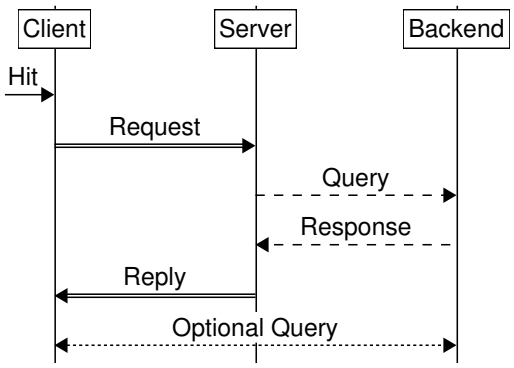
Arrows can take various forms, for example they can be bi-directional or can span multiple entities. They can also start and end at the same entity and can come from or go to "outside"

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



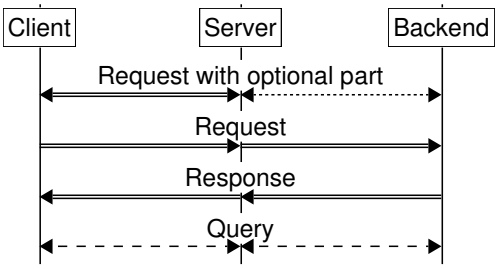
It is also possible to make use of various arrow types, such dotted, dashed and double line. To achieve this the '`->`' symbol need to be replaced with '`>`', '`>>`' and '`=>`', respectively.

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



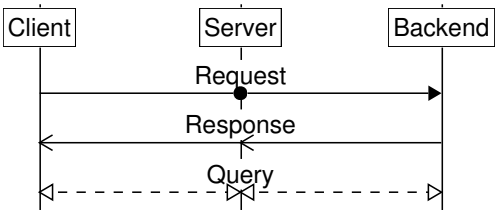
It is also possible to use different line styles for different segments of an arrow - but all must be of the same direction. (That is, it is not possible to write '`a->b<-c`', for example.) In addition, for multi-segment arrows the dash '`-`' symbol can be used in the second and following segments, as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

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



It is possible to change the type of the arrowhead. The arrowhead type is an *attribute* of the arrow. Attributes can be specified between square brackets before or after the label, as shown below. A variety of arrow-head types are available, for a full list of arrow attributes and arrowhead types See Section 5.2 [Specifying Arrows], page 35.

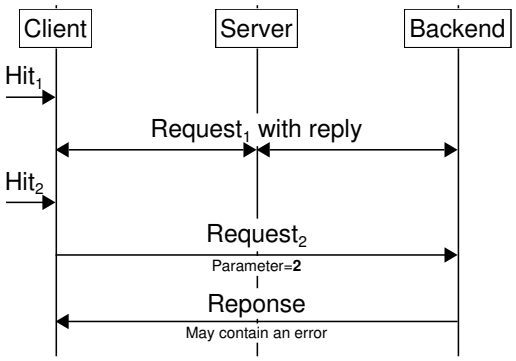
```
Client->Server-Backend: Request
     [arrow.midtype=dot];
Client<-Server-Backend: Response
     [arrow.type=line];
Client<<>>Server-Backend: Query
     [arrow.type=empty];
```



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.

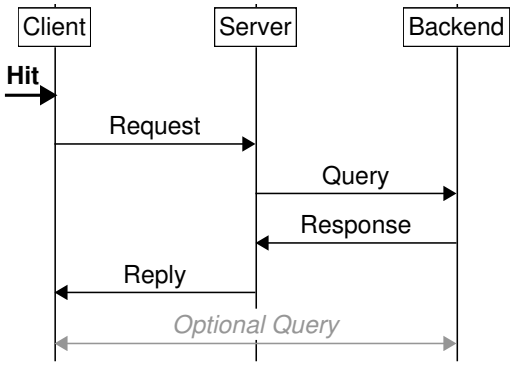
Leading and tailing whitespace will be removed from such lines so you can ident the lines in the source file to look nice.

```
->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[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];
```

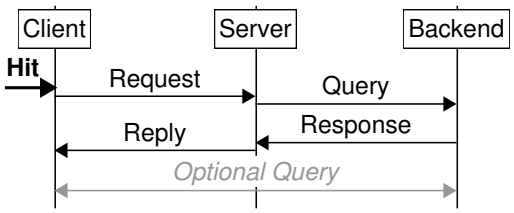


As a final feature of arrows, we note that msc-generator places arrows one-by-one below each other. In case of many arrows, this may result in a lot of vertical space wasted. To reduce the size of the resulting diagram, a *chart option* can be specified, which compresses the diagram, where possible. You can read more on chart options, see Section 5.6.4 [Compression], page 54.

[1] You can define your own styles, as well, see Section 5.14 [Defining Styles], page 62.

[2] For more on chart deisgns Section 5.15 [Chart Designs], page 64.

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

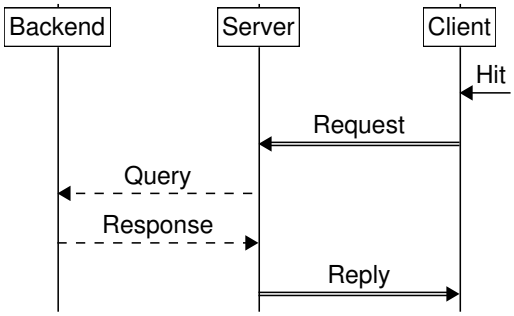


3.2 Defining Entities

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`'.

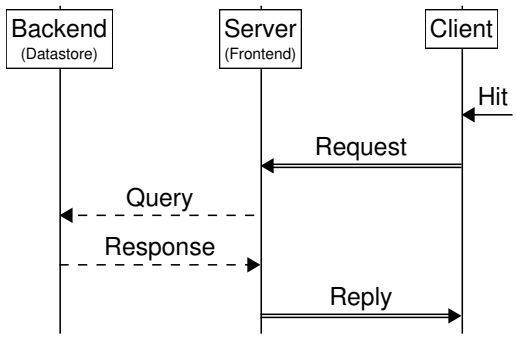
Often one wants to control, in which order entities appear on the chart. This is possible, by listing the entities before actual use. On the example below, the order of the enties are reversed. Note that we have reversed the first arrow to arrive to the '`Client`' from the right.

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



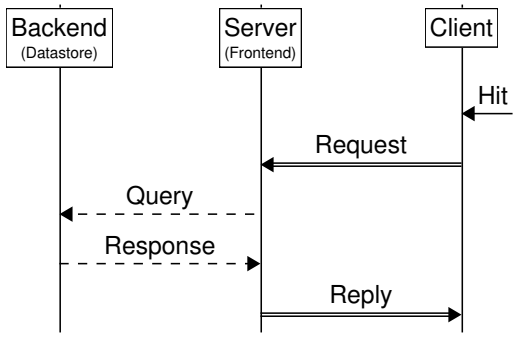
Often the name of the entity need to be multi-line or need to contain formatting characters, or just is too long to type many times. You can overcome this problem by specifying a label for entities. The name of the entity then will be used in the chart description, but on the chart the label of the entity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



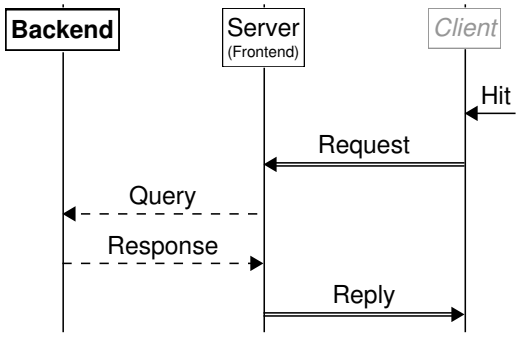
You can also use the colon-notation to specify entity labels, similar to arrows. The above example can thus be written as below. Note that the entity definitions are now terminated by a semicolon – commas are treated as part of the label.

```
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.

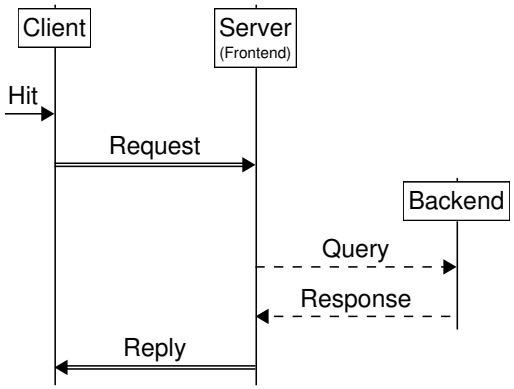
```
B: Backend [strong],
S: Server\n\-(Frontend);
C: Client [weak];
C<-: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C<=S: Reply;
```



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

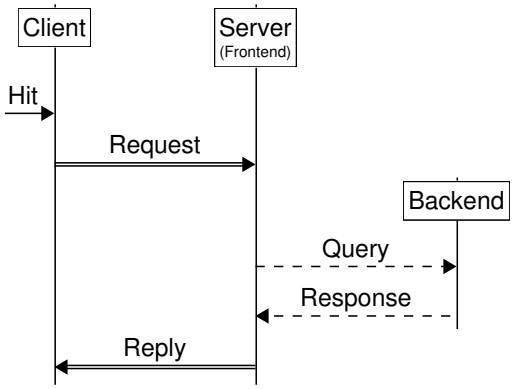
at its definition, it will start hidden and its heading is not drawn at the place of definition. However, when it is later turned on, a heading will be shown.

```
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;
```



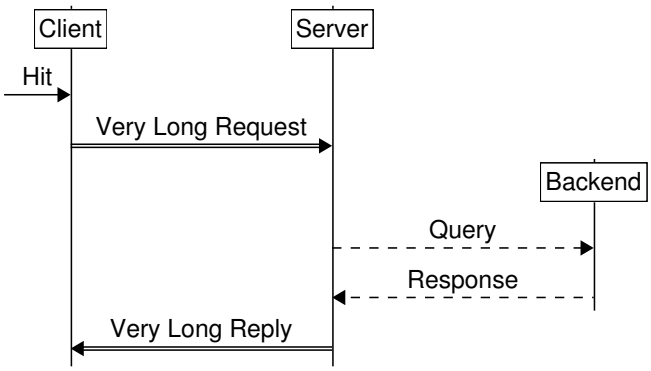
Not showing an entity from the beginning of the chart can also be achieved by simply defining the entity later. Note that this is different from simply starting to use an entity later. When you start using an entity without explicitly defining it first, it will appear at the top of the chart, not only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



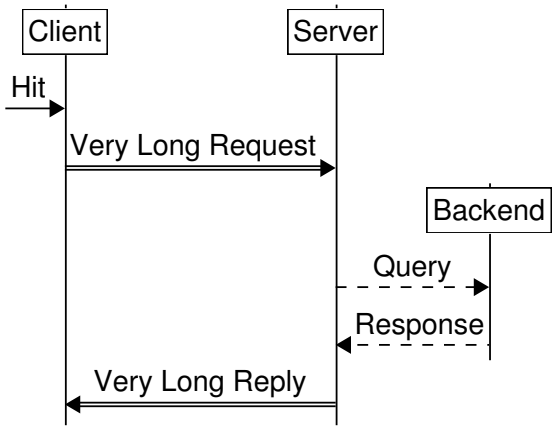
Sometimes the vertical space between entities is just not enough to display a longer label for an arrow. In this case use the '`hscale`' chart option to increase the horizontal spacing. It can be set to a numerical value, 1 being the default.

```
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.

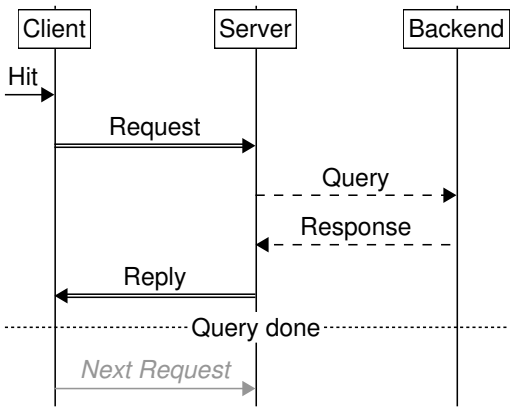
```
hscale=auto;
C: Client;
S: Server;
->C: Hit;
C=>S: Very Long Request;
B: Backend;
S>>B: Query;
S<<B: Response;
B [show=no];
C<=S: Very Long Reply;
```



3.3 Dividers

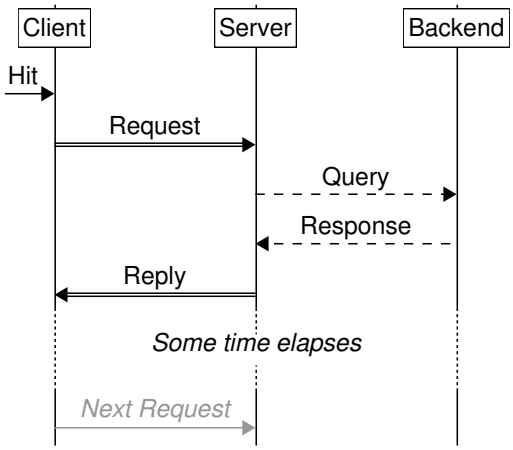
In an message sequence chart it is often important to segment the process into multiple logical parts. You can use the '`---`' element to draw a horizontal line acorss the chart with some text, e.g., to summarize what have been achieved so far.

```
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;
```



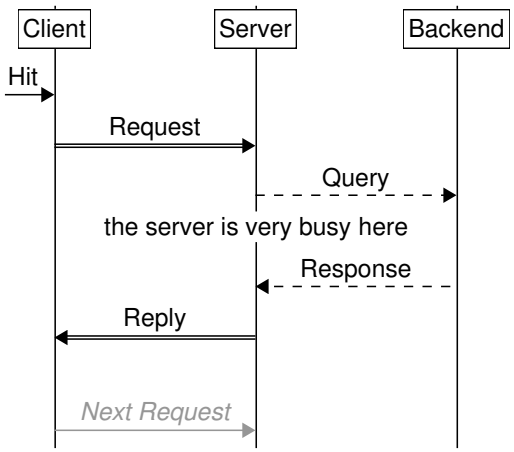
Similar to this, using the '`...`' element can express the passage of time by making the vertical lines dotted.

```
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;`' or like '`[]: text;`'. Using '`[];`' will create an empty vertical space;

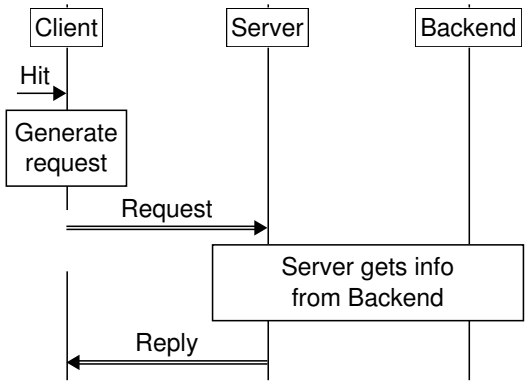
```
C: Client;
S: Server;
B: Backend;
->C: Hit;
C=>S: Request;
S>>B: Query;
: the server is very busy here;
S<<B: Response;
C<=S: Reply;
[];
C->S [weak]: Next Request;
```



3.4 Drawing Boxes

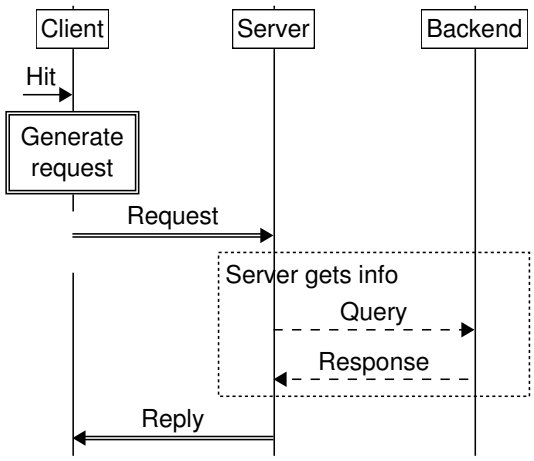
A *box* is a line around one part of the chart. It can be used to add textual comments, group a set of arrows or describe alternative behavior. In their simplest form they only contain text, but they can also encompass arrows. A box spans between two entities, or alternatively around only one.

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



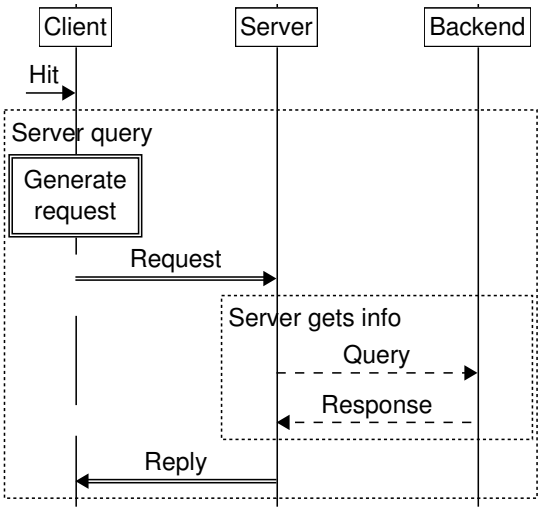
The line around boxes can be dotted, dashed and double line, too, by using '..', '++' or '==' instead of '--'. Boxes can also be used to group a set of arrows. To do this, simply insert the arrow definitions enclosed in curled braces just before the semicolon terminating the definition of the box.

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



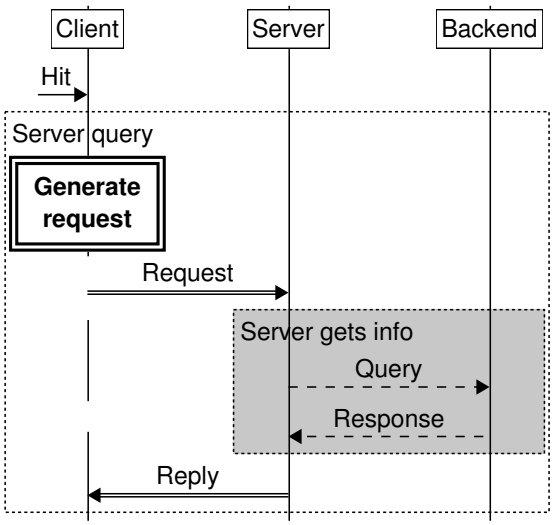
When a box contains arrows, it is not necessary to specify which entities it shall span between, it will be calculated automatically. Also boxes can be nested arbitrarily deep.

```
C: Client;
S: Server;
B: 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;
};
```



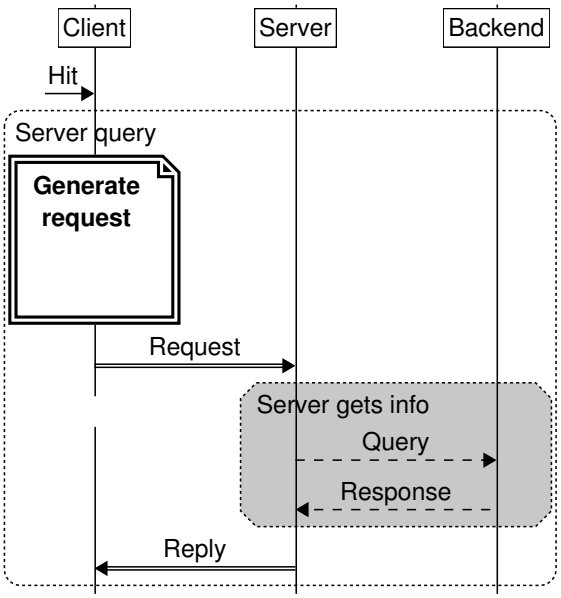
You can shade boxes, by specifying the color attribute. For a full list of box attributes and color definitions, See Section 5.3 [Boxes], page 40, and see ⟨undefined⟩ [Color Definition], page ⟨undefined⟩. It is also possible to make a box '`weak`' or '`strong`'.

```
C: Client;
S: Server;
B: 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;
};
```



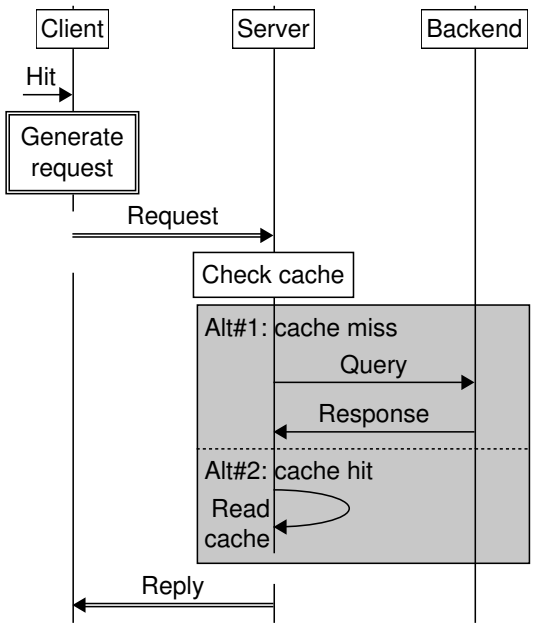
A number of box contours are available via the '`line.corner`' attribute.

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

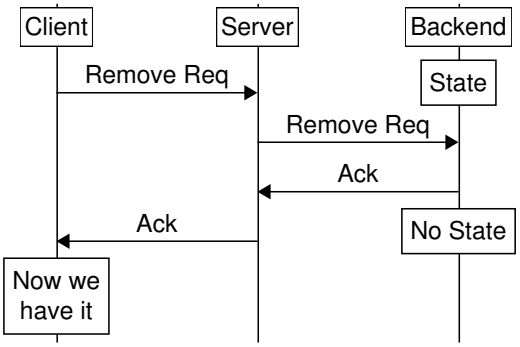


You can observe in the previous example that the '`\#`' sequence inserts a '`#`' character into a label. The '\' is needed to differentiate from a comment.

3.5 Drawing Things in Parallel

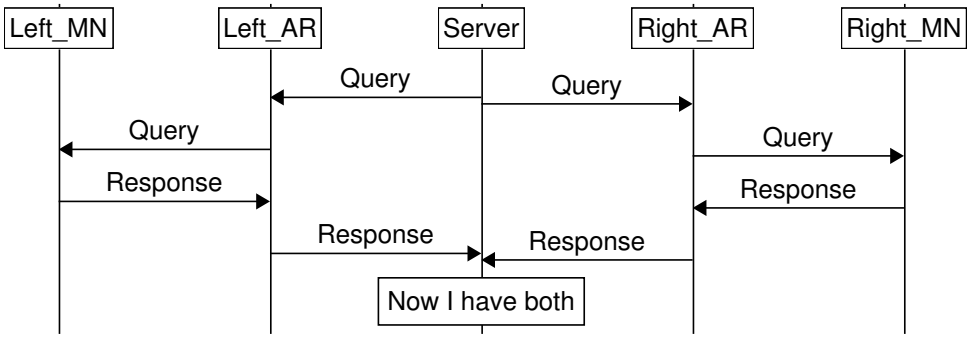
Sometimes it is desired to express that two separate process happen side-by-side. The easiest way to do so is to write '`parallel`' before any arrow, box or other element. As a result the elements after it will be drawn in parallel with it.

```
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;
} {
    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.

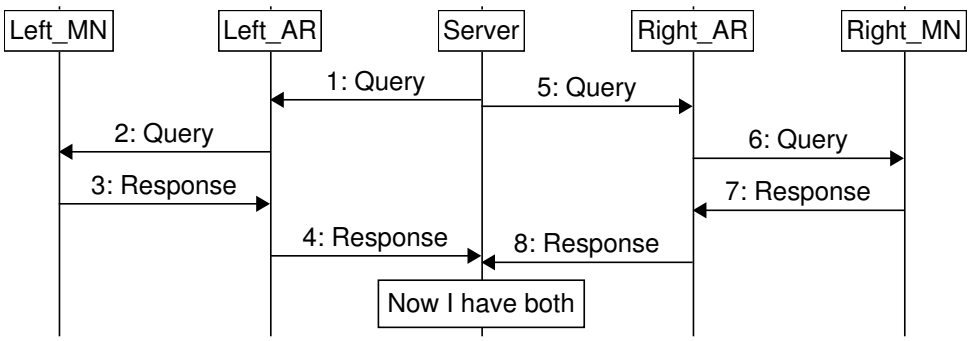
The top of each block will be drawn at the same vertical position. The next element below the series of parallel blocks (the `"Now I have it"` box in our example) will be drawn after the longest of the parallel blocks. Note that this means that in the example the lines `Server->Left_AR: Query;` and `Server->Right_AR: 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 elements in parallel blocks.

3.6 Other Features

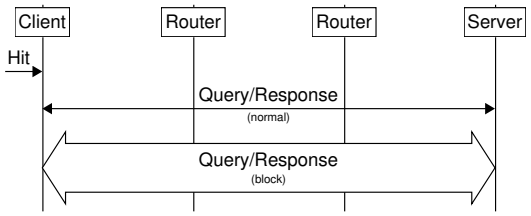
There are a few more features that are easy to use and can help in certain situations. One of them is the numbering of labels. This is useful if you want to insert your chart into some documentation and later refer to individual arrows by number. By specifying the `numbering=yes` chart option all labels 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;
} {
    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];
```



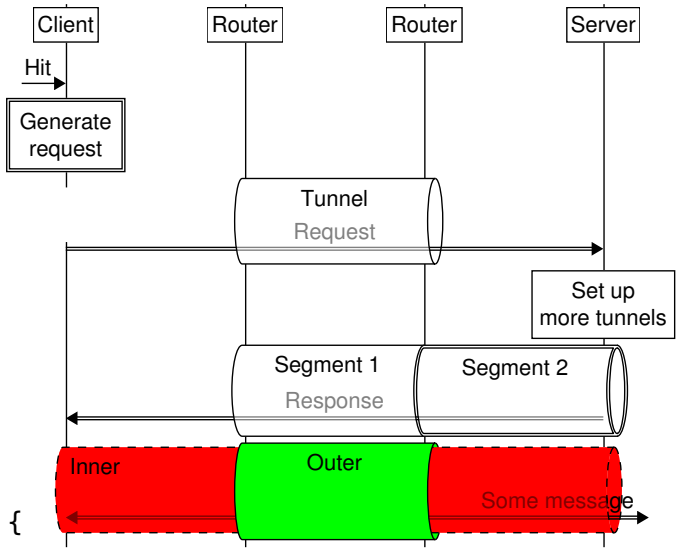
Sometimes a block of actions would be best summarized by a block arrow. This can be achieved by typing '`block`' in front of any arrow declaration.

```
C: Client;
R1: Router;
R2: Router;
S: Server;
->C: Hit;
C<->S: Query/Response\n\-(normal);
block C<->S: Query/Response\n\-(block);
```



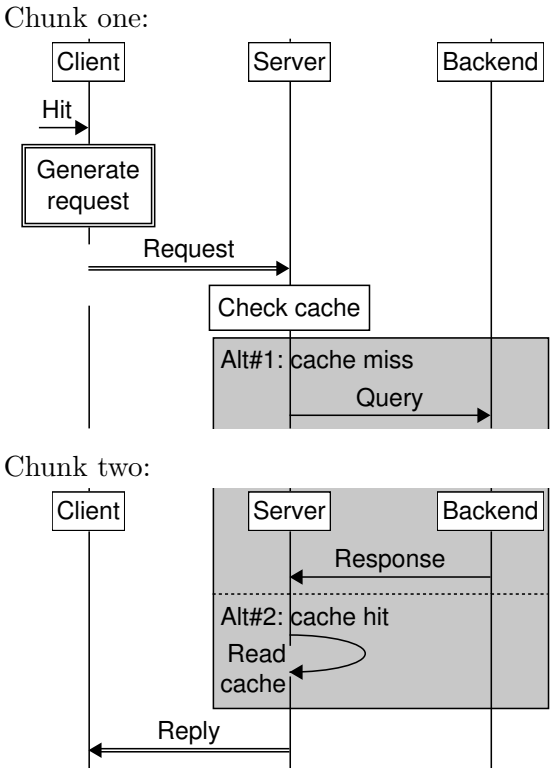
Similar, many cases you want to express a tunnel between two entities and messages travelling through it. To achieve this, just type '`pipe`' in front of any box definition. You can define a series of connected or disconencted pipe segment each with its own visual style or even encapsulate pipes. More on this in Section 5.3.1 [Pipes], page 41.

```
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;
    };
};
```



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 `heading;` command. Breaking a page is possible even in the middle of a box, see the following example.

```
C: Client;
S: Server;
B: 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;
```

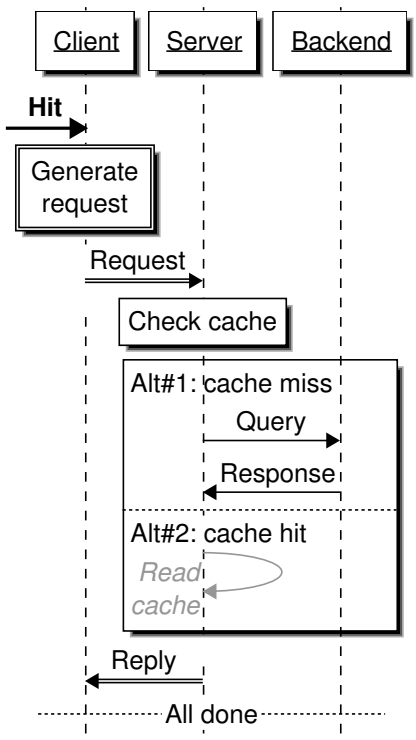


Finally, an easy way to make charts visually more appealing is through the use of *Chart Designs*. A chart design is a collection of colors and visual style for arrows, boxes, entities and dividers. The design can be specified either on the command line after double dashes, or at the beginning of the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

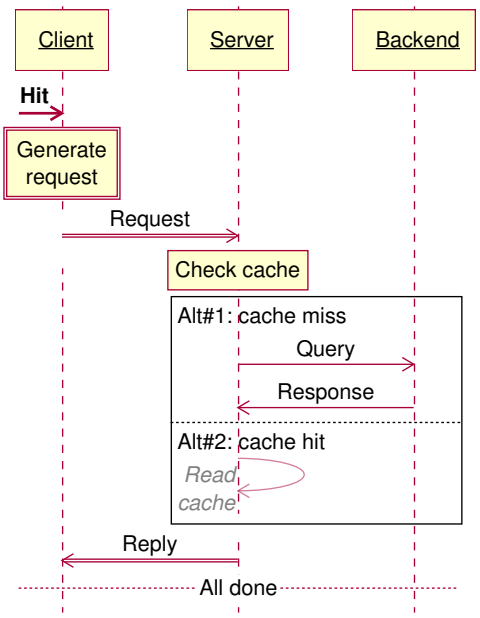
Currently eight designs are supported. '`plain`' was used as demonstration so far. Below we give an example of the other seven.

```
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;
```

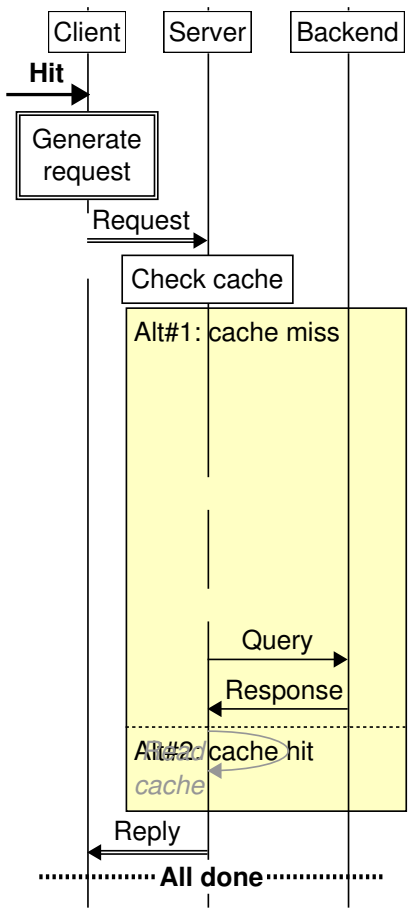
The ‘qsd’ design:



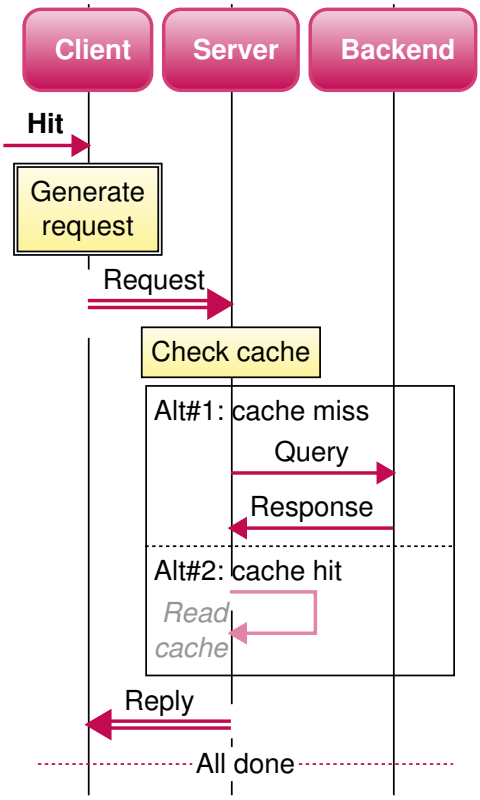
The '`rose`' design:



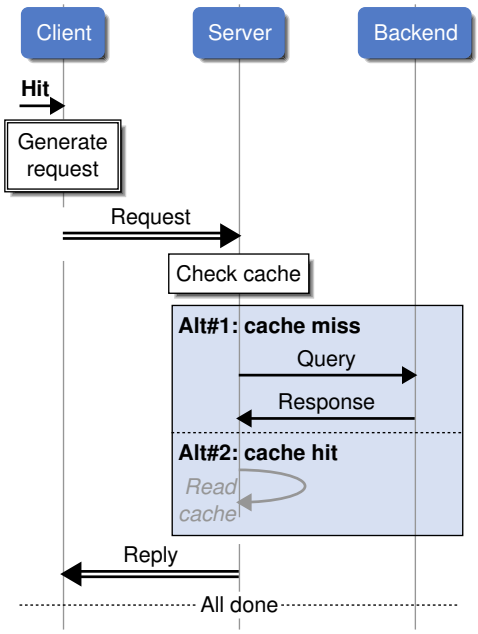
The '`mild_yellow`' design:



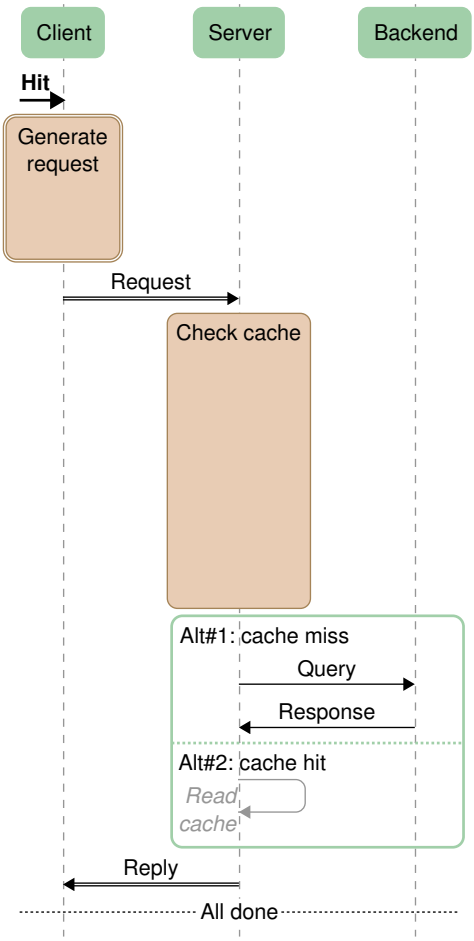
The ‘`omegapple`’ design:



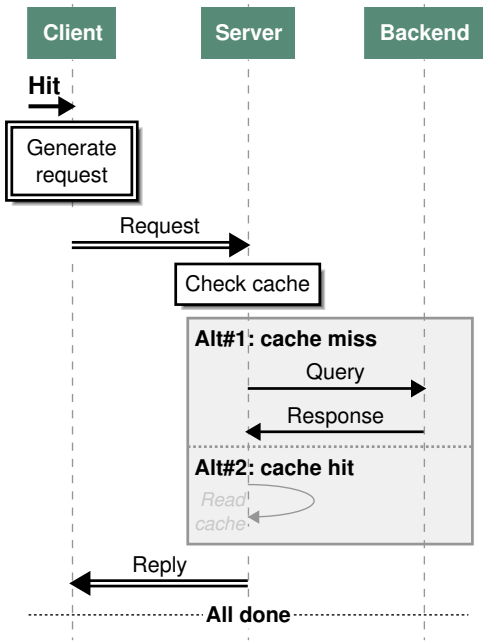
The ‘`modern_blue`’ design:



The '`round_green`' design:



The '`green_earth`' design:



4 Usage Reference

4.1 Design Library

On Windows at startup Msc-generator looks for a file called `designlib.signalling` in the directory where the executable is located (a default one is placed there by the installer.) If found, its content is parsed as regular chart description before any chart. This file is used to define the designs at the end of the previous section. The design selector combo box on the toolbar is also populated with the designs found in `designlib.signalling`.

You are free to modify this file to add or change designs. However, please avoid text in this file that results in warnings, errors or any visual elements.

On Linux, the content of this file is embedded into the executable. This means that you cannot change the definitions and you are limited to the eight ones included. On the other hand there is no need to care for an additional file: the msc-gen executable runs standalone.

4.2 External Editor

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

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

You can select the text editor to start in `Edit|Preferences...`. 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[1].

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

4.3 Smart Ident

The internal editor supports automatic identation for TAB, RETURN and BACKSPACE keys. TAB and Shift+TAB works also with selections as in most programming editors.

In addition (if the related option is turned on) Msc-generator can try to detect the beginning of multi-line labels and align all subsequent lines of the label to that. This also works when you select a block of text and press TAB or Ctrl+TAB. In the below example,

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

Smart Ident would make the second lines of the labels to start exactly aligned with the first character of the first line above.

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

4.4 Color Syntax Highlighting

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

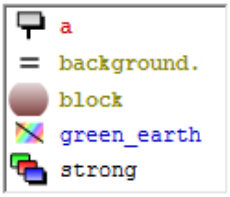
In the preferences it is also possible to select to unerline parse error locations. In this case you get instant feedback on syntax problems. Finally, it is also possible to request error messages for any error that has been underlined in the internal editor. These explanatory messages appear in the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

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

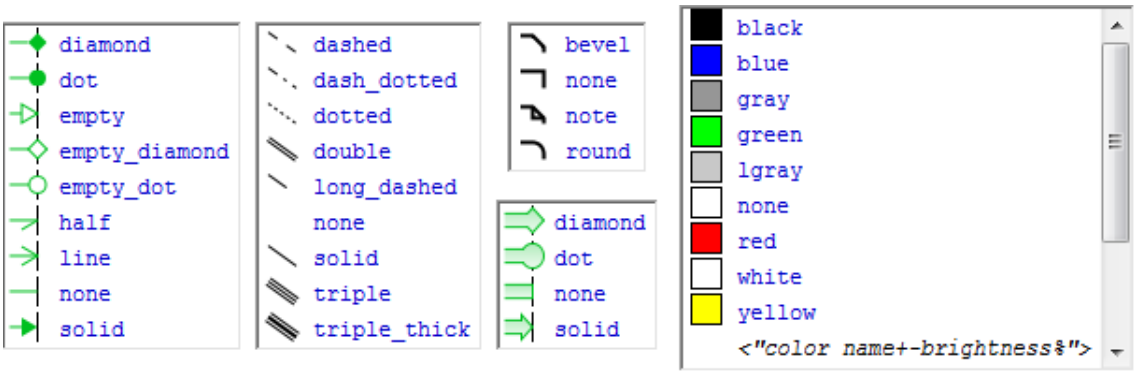
4.5 Typing Hints and Autocompletion

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

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

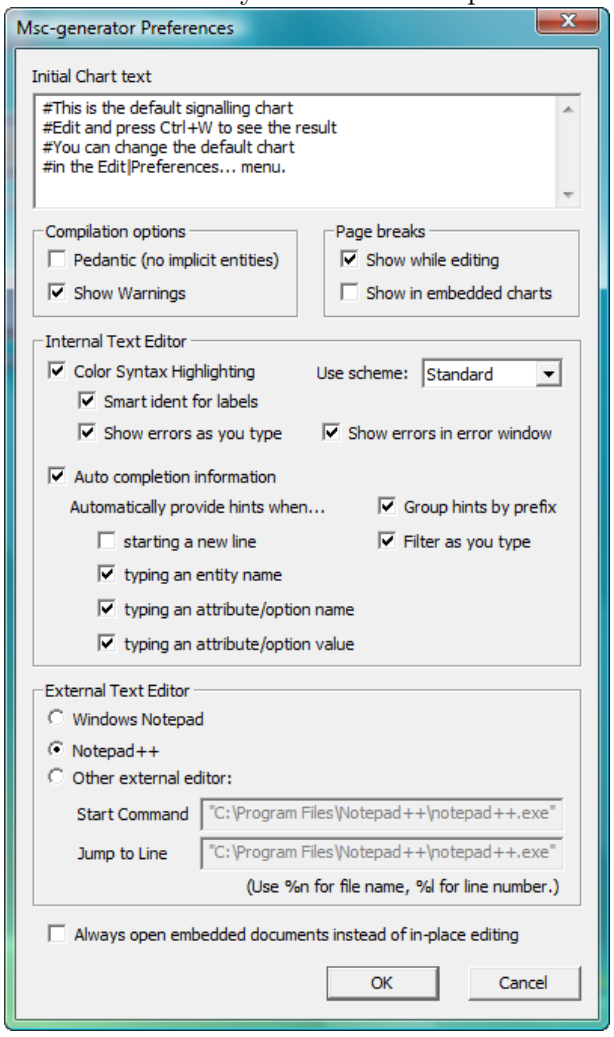


Various attribute values offer a graphic representation to ease selection. The items in italics are not actual text to be inserted into the chart, more like descriptions of what you can write there.



4.6 Options

Selecting `Edit|Preferences...` allows you to set a few options of Msc-generator.



On the top you can 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; or just delete everything here to start real empty.

Under '`Compilation options`' you can set the pedantic and show warning options. When pedantic is set Msc-generator generates a warning if an entity is no explicitly declared before use. Turning the second option of will supress the generation of warning messages altogether (including the ones generated due to the pedantic option).

'`Page breaks`' govern if a dashed line is drawn to show where page breaks are when watching all of the pages. You can select if you want to see such page breaks only while editing the chart or also in embedded charts. See Section 5.11 [Multiple Pages], page 61 for more information.

In the '`Internal Text Editor`' section you can select if you want to use color syntax highlighting in the built-in editor and if yes, which color scheme. There are four pre-defined schemes: Minimal, Standard, Colorful and Error oriented. The first three applies increasing amount of color, while the last is a minimalist scheme but with potential errors heavily highlighted[2]. At the moment you can not customize individual colors in the schemes.

Below you can control smart label identing (Section 4.3 [Smart Ident], page 27) and wether you want to see errors as you type underined and/or in the error window, Section 4.4 [Color Syntax Highlighting], page 28.

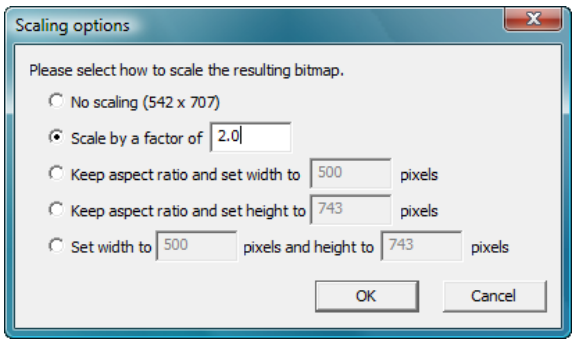
Below there are four checkboxes to control when does the hint box pop up. Ticking them all off prevents the hints to appear in response to typing. (Ctrl+Space still works.) The two checkboxes to the right control how hints are presented. If grouping is on, attributes starting with the same text, such 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.

Next, you can specify which external text editor to use. You can also select any editor using the last 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. 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.

4.7 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.

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



4.8 Command-Line Referece

The syntax of the command-line version is the same on Linux and Windows[3].

```
Usage: msc-gen [-T type] [-o file] [infile] [-Wno]
               [--pedantic] [[-x=width] [-y=height] | [-s=scale]]
               [--chart_option=value] [--chart_design]
       msc-gen -l
```

'`-T type`' Specifies the output file type, which maybe one of '`png`', '`eps`', '`pdf`', '`svg`' or '`emf`' (on Windows only). Default is '`png`'.

'`-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.

'`-l`' Display program licence and exit.

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

'`-x=width`'
Specifies chart width (in pixels). Effective only for PNG and BMP bitmaps.

'`-y=height`'
Specifies chart height (in pixels). If only one of '`-x`' or '`-y`' is specified, the aspect ratio is kept. Effective only for PNG and BMP bitmaps.

'`-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`'. Effective only for PNG and BMP bitmaps.

'`--chart_option=value`'
Any chart option (see Section 5.10 [Chart Options], page 58) 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.

[3] The only two exceptions are in how pathnames are written on the two systems and the fact that the Windows version will look for a designlib.signalling file for design definitions, while the Linux version will not.

‘*--chart_design*’
: The design pattern of the chart can be specified on the command line (see Section 5.15 [Chart Designs], page 64). This will overridde any design specified in the file.

‘-Wno’
: No warnings displayed.

5 Language Reference

5.1 Specifying Entities

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

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

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.

5.1.1 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`.

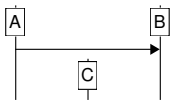
```
A, B, C, D;
A->B-C-D;
```

```
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 5.10 [Chart Options], page 58. In most cases it is simpler to use `hscale=auto`, you need `pos` only to fine-tune a chart, if automatic layout is not doing a good job.

5.1.2 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 5.8 [Text Formatting], page 55. 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 entity 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.

`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 5.1.4 [Entity Headings], page 35.

`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.*`
`text.*`
`shadow.*` See Section 5.6 [Common Attributes], page 45 for the description of these attributes.

5.1.3 Implicit Entity Definition

It is not required to explicitly define an entity before it is used. Just typing the arrow definition `a->b;` will automatically define entities '`a`' and '`b`' if not yet defined. This behaviour can be disabled by specifying the '`--pedantic`' command-line option or specifying `pedantic=yes` chart option. See Section 5.10 [Chart Options], page 58. Disabling implicit definition is useful to generate warnings for mis-typed entity names[1].

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

5.1.4 Entity Headings

By default, when an entity is defined, its heading is drawn at that location. If the entity is preceeded by the `hide` 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 10 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 is useful after a `newpage;` command, see Section 5.12 [Commands], page 61.

5.2 Specifying Arrows

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

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

arrowsymbol can be any of '`->`', '`<-`' or '`<->`', the latter for bidirectional arrows. `a->b` is equivalent to `b<-a`. This produces an arrow between the two entities specified using a solid line. Using '`>`'/'`<>`', '`>>`'/'`<<>>`' or '`=>`'/'`<=>`', will result in dotted, dashed or double line arrows, respectively. These settings can be redefined using styles, see Section 5.14 [Defining Styles], page 62.

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.

[1] 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.

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

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

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

5.2.1 Arrow Attributes

Arrows can have the following attributes.

`label` This is the text associated with the arrow. See Section 5.6.2 [Labels], page 50 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 5.6 [Common Attributes], page 45 can be used to manipulate the appearance of the label.

`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 5.6.3 [Numbering], page 51.

`compress` Can be set to `yes` or `no` to turn compressing of this arrow on or off. See Section 5.6.4 [Compression], page 54.

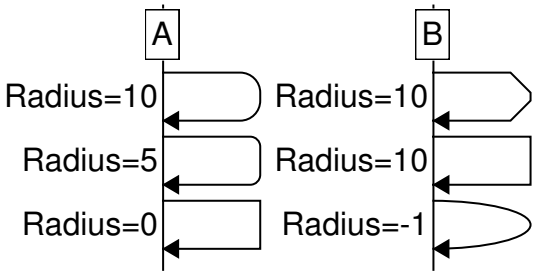
`color` This specifies the color of the text, arrow and arrowheads. It is a shorthand to setting `text.color`, `line.color` and `arrow.color` to the same value.

`line.color, line.width`
Set the color and the width of the line, see Section 5.6 [Common Attributes], page 45.

`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` will result in a round corner.

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



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

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

`arrow.type`
: 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. `diamond` and `empty_diamond` draws a filled or empty diamond, while `dot` and `empty_dot` draws a filled or empty circle. Specifying `none` will result in no arrowhead at all. This attribute sets both the `endtype` and `midtype`, see below.

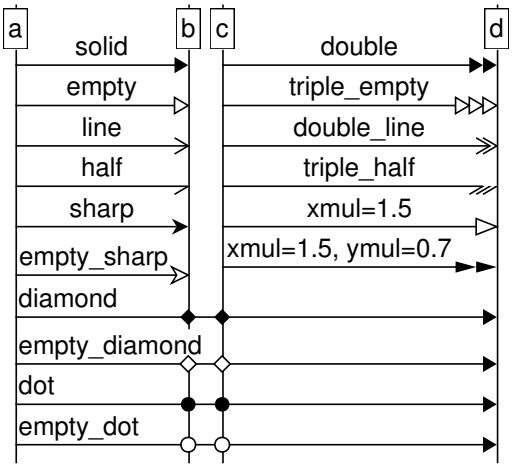
`arrow.endtype`
: Sets the arrow type for arrow endings only. This refers to the end of the arrow, where it points to. In case of bidirectional arrows, both ends are drawn with this type. It defaults to a filled triangle.

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

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

`arrow.xmul`
`arrow.ymul`
: These attributes change the width or the height of the arrowhead. The default value is '`1`'. They are 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];
} {
c->d: double[arrow.type=double];
c->d: triple_empty[arrow.type=triple_empty];
c->d: double_line[arrow.type=double_line];
c->d: triple_half[arrow.type=triple_half];
c->d: xmul=1.5 [arrow.type=empty, arrow.xmul=1.5];
c->d: xmul=1.5, ymul=0.7 [arrow.type=double,
           arrow.xmul=1.5, arrow.ymul=0.7];
};
a->b-c-d: \pldiamond [arrow.midtype=diamond];
a->b-c-d: \plempty_diamond [arrow.midtype=empty_diamond];
a->b-c-d: \pldot [arrow.midtype=dot];
a->b-c-d: \plempty_dot [arrow.midtype=empty_dot];
```



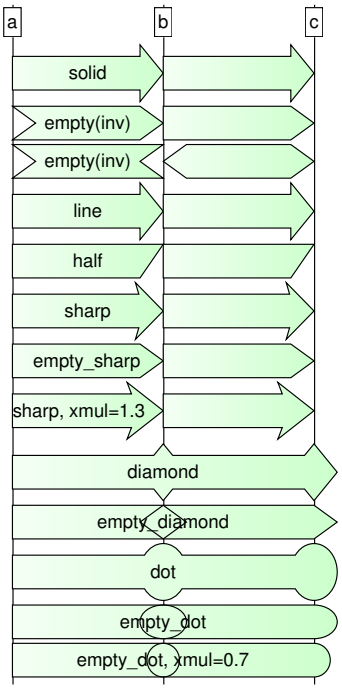
Note that default values can be changed using styles, see Section 5.14 [Defining Styles], page 62.

5.2.2 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. An additional type `empty_inv` is supported, as well (and more can be expected in future releases). See the example below for a detailed list of all types supported for block arrows.

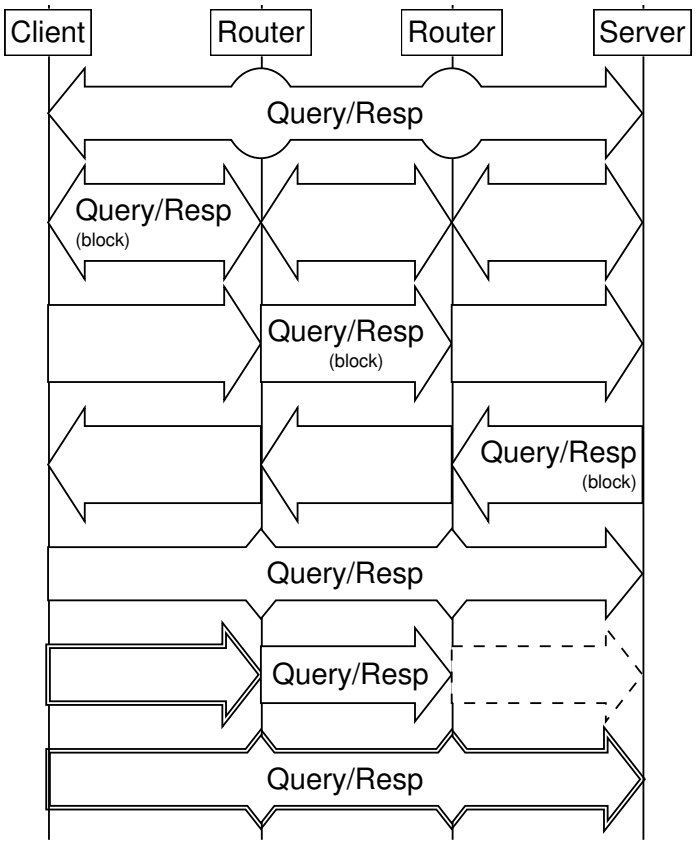
```
defstyle blockarrow [fill.color="green+80",
                     fill.gradient=right];
block a->b-c: solid [arrow.type=solid];
block a->b-c: empty(inv) [arrow.type=empty,
                          arrow.starttype=empty_inv];
block a->b-c: empty(inv)[arrow.endtype=empty,
                         arrow.starttype=empty_inv,
                         arrow.midtype=empty_inv];
block a->b-c: line [arrow.type=line];
block a->b-c: half [arrow.type=half];
block a->b-c: sharp [arrow.type=sharp];
block a->b-c: 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.label` 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.

```
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];
```



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

The syntax definition for boxes is as follows.

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

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.

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

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`, `compress`, `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.

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 Section 3.4 [Drawing Boxes], page 15 for an example.

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.

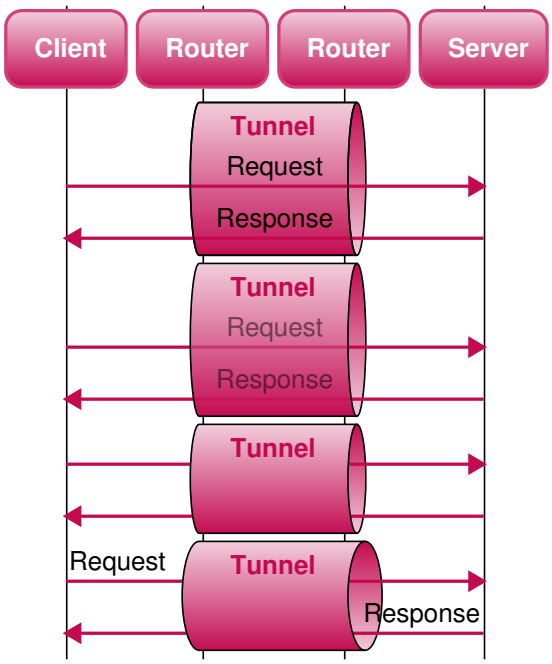
5.3.1 Pipes

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.

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.

```
msc=omegapple;
C: Client;
R1: Router;
R2: Router;
S: Server;
defstyle pipe [fill.color=rose];
defstyle pipe [fill.gradient=down];
pipe R1--R2: Tunnel [solid=0] {
    C->S: Request;
    C<-S: Response;
};
pipe R1--R2: Tunnel [solid=0.5] {
    C->S: Request;
    C<-S: Response;
};
pipe R1--R2: Tunnel [solid=1] {
    C->S: Request;
    C<-S: Response;
};
pipe R1--R2: Tunnel
     [solid=1, line.radius=10] {
    C->S: \plRequest;
    C<-S: \prResponse;
};
```

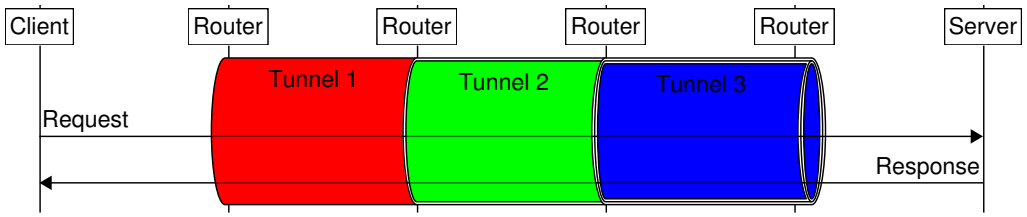


On the example above one can observe, that the last two pipes are smaller than the first two, even though they have exactly the same two arrows within. This is because in case of the first two arrows the label of the pipe itself is visible at together with the two arrows within. In contrast, the last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

Note the two `defstyle` commands before the pipes, as well. They are re-defining the default fill for pipes. You can read more about this in Section 5.14 [Defining Styles], page 62.

Similar to boxes multiple definitions can be placed after each other without a semicolon. In case of boxes this results in a series of connected boxes. In case of pipes this results in different pipe segments besides each other. However, contrary to boxes only one set of content can be specified.

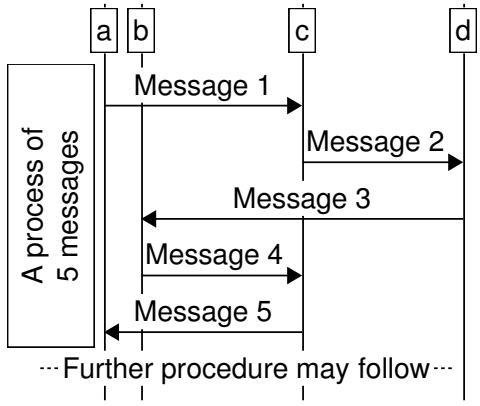
```
C: Client;  R1: Router;
R2: Router; R3: Router;
R4: Router; S: Server;
pipe R1--R2: Tunnel 1 [color=red]
     R2==R3: Tunnel 2 [color=green]
     R3==R4: Tunnel 3 [color=blue, line.type=triple]
{
    C->S: \plRequest;
    C<-S: \prResponse;
};
```



5.4 Verticals

A *vertical* is a block arrow or box with a general direction of up and down as opposed to regular block arrows or boxes, which go from left to right or back. Verticals can contain text, which is rotated 90 degrees compared to other elements. They are useful to comment on a procedure going on besides, or to indicate one message triggering another one below. Consider the example below.

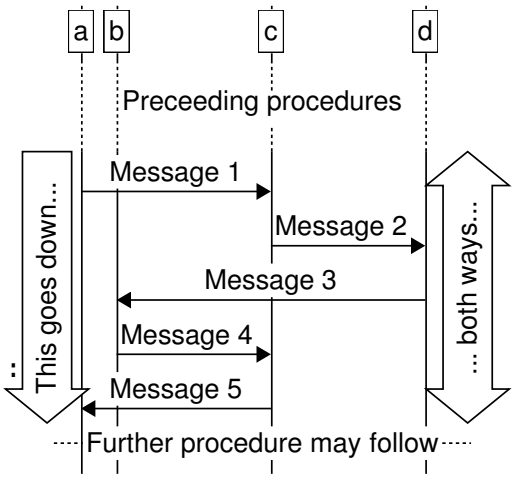
```
hscale=auto;
a, b, c, d;
mark top;
a->c: Message 1;
c->d: Message 2;
d->b: Message 3;
b->c: Message 4;
c->a: Message 5;
vertical top-- at a- [makeroom=yes] :
       A process of\n5 messages;
---: Further procedure may follow;
```



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 via vertical *markers*. Markers can be placed with the `mark` command. The 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 '`<entity>`', '`<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=yes] :
       ... both ways...;
---: Further procedure may follow;
```



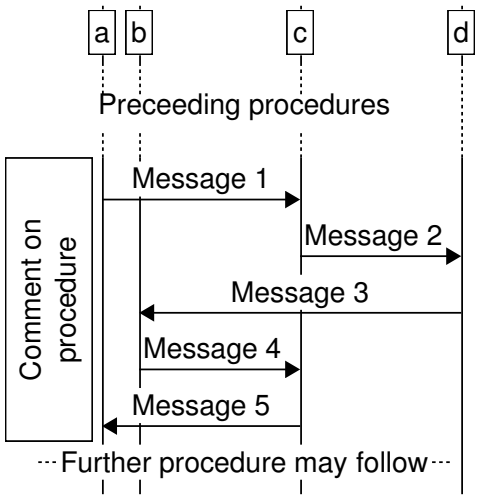
In the second vertical arrow, the horizontal position is specified as '`<entity>++`'. This avoids the effect with the first one, where the tip of the arrow overlaps the entity line of entity '`a`'.

Both the mark command and the vertical element can have an `offset` attribute. It takes a number and shifts the position down and right by that many pixels for mark commands and vertical elements, respectively. In case of a negative `offset` the position is shifted up or left.

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 verticals can still overlap with other elements.

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 not specified this way in the second or later 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;
```



Verticals are drawn over elements specified before and under elements specified after. You can somewhat influence this (the z-order) by specifying the vertical earlier or later in the file. E.g., if you specify the vertical at the end of the file, it will be drawn on top of any other element. Note that markers can be forward referred to before they are defined (unlike any other construct in the language). This allows a vertical to be specified at the beginning of the file referring to markers defined later.

5.5 Dividers

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

The third type of divider is a simple vertical space. This can be specified by entering just attributes in square brackets. The extreme '`[];`' simply inserts a lines worth of vertical space. You can add text, too by specifying a label. See Section 3.3 [Dividers], page 13 for examples.

Dividerss take the `label`, `color`, `text.*`, `line.*`, `compress` and `number` attributes with the same meaning as for arrows. In addition, the type of the vertical line can be specified with `vline.*`, with `vline.type` defaulting to `dotted` for '`...`' dividers and to `solid` for '`---`' dividers. Other values are `dashed`, `none` and `double`. Again, note that the default values can be changed by using styles, see Section 5.14 [Defining Styles], page 62.

5.6 Common Attributes

As discussed earlier, attributes can influence how chart elements look like and how they are placed. There is a set of attributes that apply to multiple 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[2]. 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 5.7 [Specifying Colors], page 54.

`line.color`
: Specifies the color of the line for the element. For arrows and dividers this is the horizontal line. For block arrows, boxes, pipes and entities this is the line around the element. Unless you use a single color name you must quote the color specification, see Section 5.7 [Specifying Colors], page 54 for the syntax of colors.

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

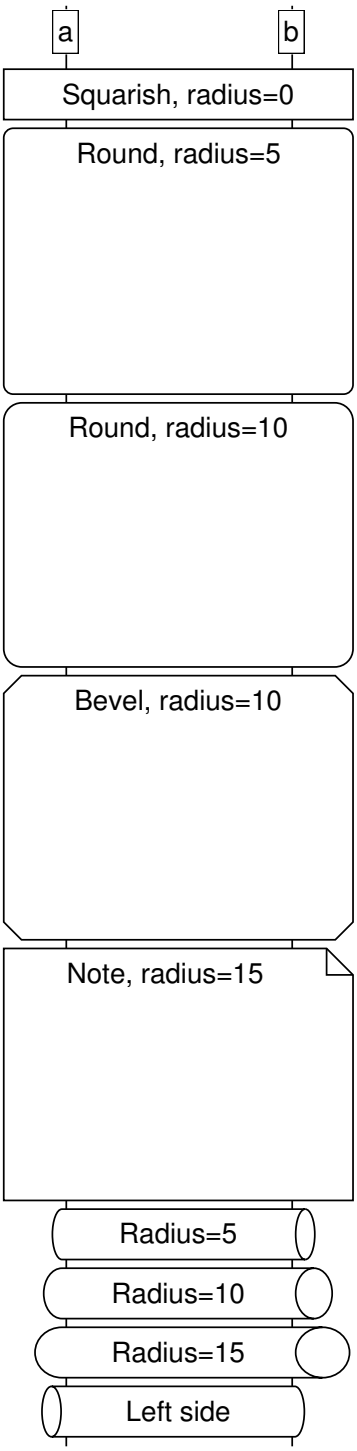
`line.type`
: Specifies the type of the line. Its value can be `solid`, `dashed`, `dotted`, `double` or `none`.

`line.radius`
: For arrows it has effects only on arrows starting and ending in the same entity (see Section 5.2.1 [Arrow Attributes], page 36). For entities and boxes, this specifies the size of the corners. 0 is fully sharp, values are meant in pixels. If no `line.corner` is specified setting radius to a positive value will result in round corners. For pipes, it specifies the width of the oval, in other words from how left we look at the pipe.

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

[2] Specifically strings that contain characters other than letters, numbers, underscores or dots, must be quoted. If the string starts with a number or a dot or it it ends with a dot, it must also be quoted. The only exception to this are built-in style names, see Section 5.14 [Defining Styles], page 62.

```
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];
```



`vline.*` Specifies the color, width or type of the vertical line stemming from entities. This is useful to indicate some change of state for the entity. `vline.radius`

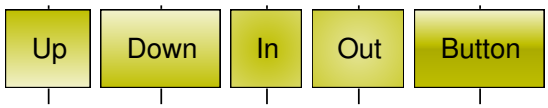
and `vline.corner` has no effect. These attributes can be used for entities and dividers.

`fill.color`
: Defines the background color of the box, entity, block arrow or pipe. Specifying `none` results in no fill at all. Unless you use a single color name you must quote the color specification, see Section 5.7 [Specifying Colors], page 54 for the syntax of colors.

`fill.color2`
: If this attribute is specified then the fill gradient will not be between `fill.color` and a lighter variant, but between `fill.color` and the value specified here. If no gradient specified or `button` is used, this attribute has no effect.

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

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

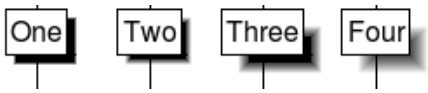


`shadow.offset`
: If not set to zero, then the entity or box will have a shadow (default is 0). The value of this attribute then determines, how much the shadow is offset (in pixels), in other words how "deep" the shadow is below the entity or box.

`shadow.color`
: The color of the shadow. This attribute is ignored if shadow.offset is 0. Unless you use a single color name you must quote the color specification, see Section 5.7 [Specifying Colors], page 54 for the syntax of colors.

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

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



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

`text.color`
: Sets the color of the label. Unless you use a single color name you must quote the color specification, see Section 5.7 [Specifying Colors], page 54 for the syntax of colors.

`text.format`
: Takes a (quoted) string as its value. Here you can specify any of the text formatting escapes that will govern the style of the label, see Section 5.8 [Text Formatting], page 55. Specifying them here or directly at the beginning of the label has the same effect, so having this attribute is more useful for styles.

`arrow.*`
: Styles can also contain arrow formatting attribues. These are described in Section 5.2.1 [Arrow Attributes], page 36.

`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. It has no effect on any other elements.

`solid`
: This attribute can be used to set the transparency of a pipe. See Section 5.3.1 [Pipes], page 41 for more information.

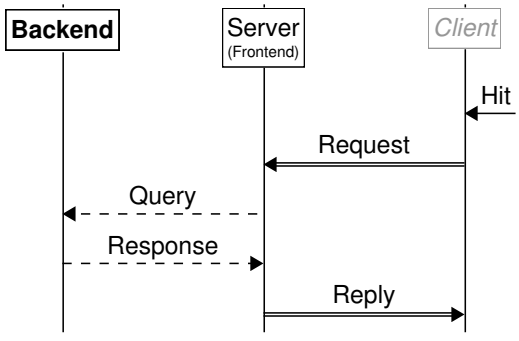
`number`
: This attribute giverns if the arrow, box, etc. is numbered or not. See Section 5.6.3 [Numbering], page 51 for details.

`compress`
: If this attribute is set to `yes`, the element is drawn as close to the ones above it as possible without touching those. It is useful to save space, see Section 5.6.4 [Compression], page 54 for a detailed description.

5.6.1 Styles

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

```
B: Backend [strong],
S: Server\n\-(Frontend);
C: Client [weak];
C<-: Hit;
C=>S: Request;
S>>B: Query;
S<<B: Response;
C<=S: Reply;
```



Styles can contain any of the attributes listed in the above section. If a style contains an attribute not applicable for the element that you apply the style to, that attribute is

simply ignored. For example, applying a style with `fill.color=red` attribute setting to an arrow, will ignore this attribute since arrows take no fill attributes.

You can define your own styles or redefine existing ones. See Section 5.14 [Defining Styles], page 62 for more on this.

5.6.2 Labels

Entities, arrows, boxes, pipes and dividers have a `label` attribute, which specifies the text to be displayed for the element. Each element displays it at a different place, but the syntax to describe a label is the same for all. For entities the label defaults to the name of the entity, while for the rest it defaults to the empty string. Labels have to be quoted if they contain any character other than letters, numbers, underscores and the dot, or if they start with a dot or number or end with a dot. You can use all character formatting features in labels, see Section 5.8 [Text Formatting], page 55.

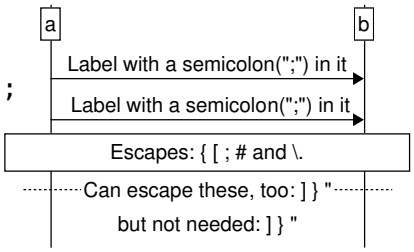
To avoid typing `[label="..."]` many times it is possible to specify the label attribute in a simpler way. After the definition of the element, just type a colon, the text of the label unquoted and terminate with a semicolon (or opening brace '`{`' or bracket '`[`'). You can write attributes before or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

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

If the label needs to contain a opening bracket ('`[`'), opening brace ('`{`'), hashmark ('`#`') or a semicolon ('`;`') use quotations or preceed these characters by a backslash '`\`'[3]. 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 '`"`'[4].

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

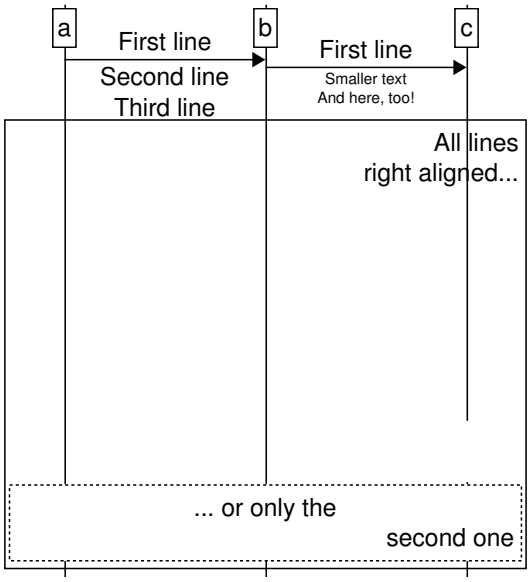


Labels can span multiple lines. You can insert a line break by adding the '`\n`' escape sequence. Alternatively you can simply break a label and continue in the next line. In this case leading and trailing whitespace is removed from each line.

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

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

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



5.6.3 Numbering

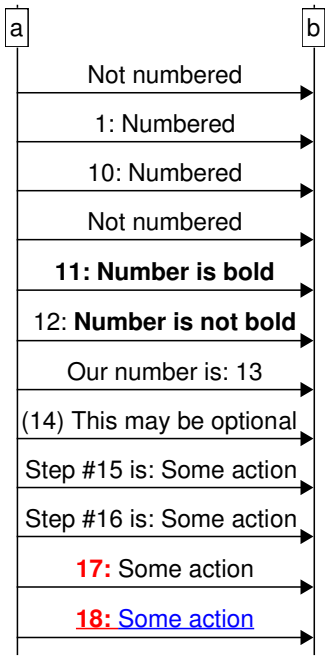
Arrows, boxes and dividers (any element with a label, except entities) can be auto-numbered. It is a useful feature that allows easier reference to certain steps in a procedure from explanatory text. To assign a number to an element, simply set its `number` attribute to `yes`. You can also assign a specific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

Styles can also control numbering. If a style has its `number` attribute set to `yes` or `no`, any element that you assign the style to will have its attribute set likewise. See Section 5.6.1 [Styles], page 49 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 5.13 [Scoping], page 62.

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;
```



If numbering is turned on for a label, the number is inserted at the beginning of the label and is followed by a semicolon and a space by default. More precisely, the number is inserted after any initial text formatting sequences, so that it has the same formatting as the label itself (see Section 5.8 [Text Formatting], page 55)[5]. The above default can be changed by inserting the '`\N`' escape sequence into a label. This causes the number appear where the '`\N`' is inserted, as opposed to the beginning of the label. In this case, the colon and the space is omitted, only the number itself is inserted.

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

Note that for the last two arrows formatting escapes were added to the '`numbering.pre`' option. These are reversed by the '`\s()`' escape in the '`numbering.post`' option. See Section 5.8 [Text Formatting], page 55 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[6]. 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.

---

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

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

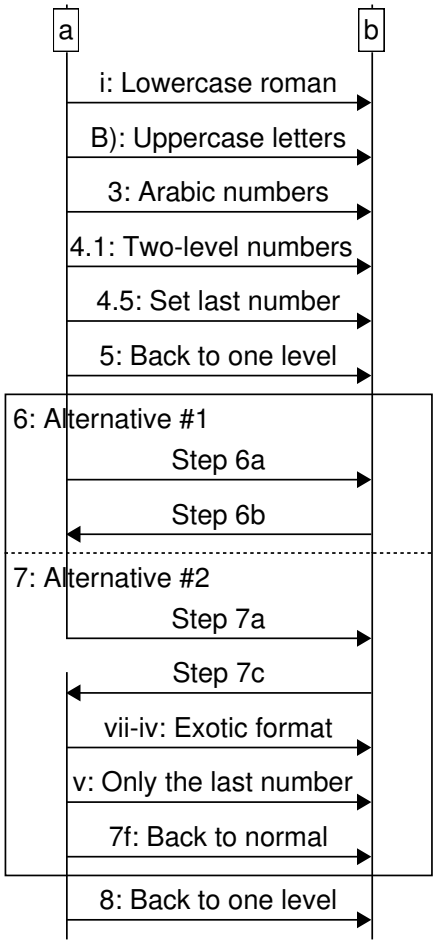
Note also, that when using roman numbers or letters, you can use such numbers as the value of the '`number`' attribute, as shown below for '`7c`'.

Finally, 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.

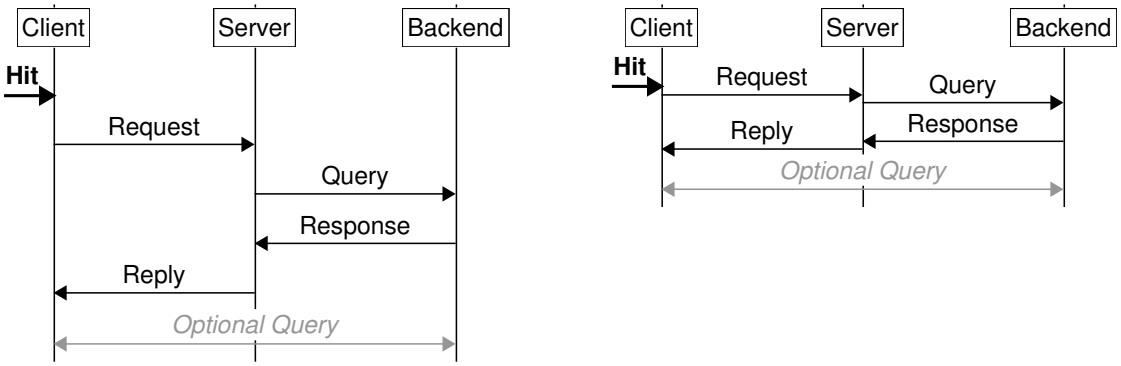
The '`numbering.append`' option can only be used to add levels. There is no explicit way to decrease the number of levels, you have to use scoping to achieve that. On the example above, the second level appended in the scope of '`Alternative #1`' is cancelled at the end of the scope, so we need to append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



5.6.4 Compression

The *compression* mechanism of Msc-generator 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 7. They differ only in that the second begins with `compress=yes`.



Each element (except entities) has a `compress` attribute. When 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.

Compression can be set individually for each element, but to save typing by setting the `compress` chart option, you can effectively set the `compress` attribute of all elements after. This is similar, how the `numbering` chart option effects the `number` attribute.

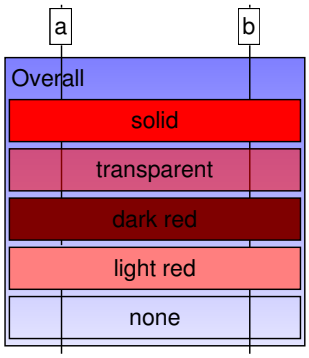
Styles can also influence compression the same way as numbering, that is you can set the compress option for a style, which will effect compression of elements you assign the style to.

5.7 Specifying Colors

Msc-generator has the following color names defined initially: `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 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;
};
```



You can specify colors giving the red, green and blue components separated by commas. An optional fourth value can be added for the alpha channel to control transparency. Values can be either between zero and 1.0 or between 0 and 255. If all values are less than or equal to 1, the former range is assumed[7]. 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.

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

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

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

Msc-generator honors scoping. Color definitions (or re-definitions) are valid only until the next closing brace '}'. This makes it possible to override a color only for parts of the chart, returning to the default later. Note that you can start a new scope any time by placing an opening brace. See Section 5.13 [Scoping], page 62 for more on scopes.

5.8 Text Formatting

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

`\n`	Inserts a line break.
`\-`	Switches to small font.
`\+`	Switches to normal (large) font.
`\^`	Switches to superscript.
`\_`	Switches to subscript.
`\b`	Toggles bold font.

[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.)

`\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 operating system you use. On Windows, you can use all the fonts available, but only OpenType and TrueType fonts provide correct alignment. On Linux you can use whatever font backend your cairo library was compiled for. This typically includes FreeType. If you specify no font, just `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 5.7 [Specifying Colors], page 54. 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[8]. 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[9]. See Section 5.6.1 [Styles], page 49 for more information on styles.

`\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.

`\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)`.

[8] 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.

[9] 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.

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 the escaped version is also there for ease of use.

`\|`
: 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.

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

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

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

5.9 Parallel Blocks

Sometimes it is desired to express that two separate process happen side-by-side. *Parallel blocks* allow this. Simply place the the parallel blocks between '`{}`' marks and write them one after the other, as in Section 3.5 [Drawing Things in Parallel], page 18. 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, boxes, or other parallel blocks, as well. However, elements spanning the entire width of the chart will likely cause overlap (such as the `heading;` command and dividers) so these trigger a warning.

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

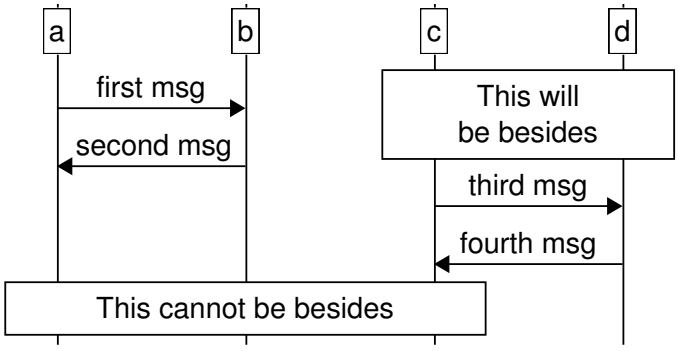
No checks are performed when drawing parallel blocks, so they can actually overlap with each other. You will have to take care of avoiding this when specifying your chart. This is unlike in the case of the `parallel` keyword, where overlap is avoided by Msc-generator. (At the cost of potentially showing elements sequentially even if they were intended to be shown in parallel.)

5.9.1 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[10].

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

```
hscale = 0.8;
parallel {
    a->b: first msg;
    a<-b: second msg;
};
c--d: This will
      be besides;
parallel {
    c->d: third msg;
    c<-d: fourth msg;
};
a--c: This cannot be besides;
```



5.10 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. The whole chart is always drawn using a single design. If you

[10] 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.

specify this attribute multiple times, the last one will be used. See Section 5.15 [Chart Designs], page 64 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 10 for examples. Similar to `msc`, if you specify this attribute multiple times, the last one takes precedence.

`numbering`
: This option takes `yes` or `no` value, the default is no. Any element you define will take the default value of its `number` attribute from this option. See more on numbering in Section 5.6.3 [Numbering], page 51.

`compress` This option takes a boolean value, and defaults to off. Any element you define will take the default value of its `compress` attribute from this option. See more on numbering in Section 5.6.4 [Compression], page 54.

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

`numbering.pre`
`numbering.post`
: These options specify what shall be prepended and appended to label numbers. Their default value is the empty string and a semicolon followed by a space, respectively. The value of these options are ignored when a label number is inserted due to the '`\N`' escape sequence. See Section 5.6.3 [Numbering], page 51 for more.

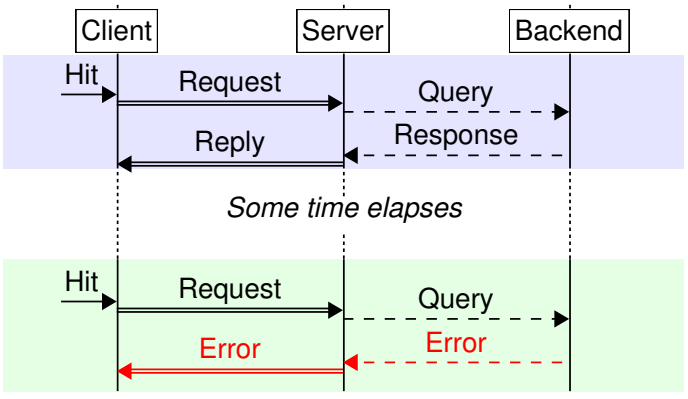
`numbering.format`
: Specifies the format of automatic numbering for labels. Can be an arbitrary string (usually quoted) and may also contain formatting escapes. Any occurrence of '`123`', '`arabic`', '`iii`', '`roman`', '`abc`', '`letters`' (or uppercase versions) will be replaced to the actual number 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. Thus either set the background to a solid color or to fully transparent. The latter is in fact achieved by simply not drawing any background at all.

```
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];
```



5.11 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[11].

To start a new page, use the '`newpage`' command. You may want to add '`heading`' afterwards to display all entity headings at the top of the new page. You can have as many pages in a document as you want.

When editing in Windows, you can select on the toolbar, 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 line and also prints page numbers to the left. This behaviour can be turned off in the options, or can also be turned on for embedded charts. (See Section 4.6 [Options], page 29.)

Page breaks are honored when printing out a chart. This means that Msc-generator will not break a page even if it does not fit onto the paper. Page breaks are only inserted where you added `newpage`.

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. Currently there is no way to generate a single file from a chart containing multiple pages.

5.12 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 5.9 [Parallel Blocks], page 57.

'`newpage`' This command starts a new page, see Section 5.11 [Multiple Pages], page 61.

'`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 or re-define color names, see Section 5.7 [Specifying Colors], page 54.

'`defstyle`'
This command is used to define or re-define styles, see Section 5.14 [Defining Styles], page 62.

[11] Future versions of Msc-generator may also support object linking and thus allowing you to insert links into a compound document pointing to an embedded chart in the very same document. It would be possible to set the lins to show different pages. This arrangement would enable explaining a complicated chart in several steps in a document, but still have uniform numbering throught the pages and a single point for editing all the pages.

'`defdesign`'
: This command is used to define new designs, see Section 5.15 [Chart Designs], page 64.

5.13 Scoping

Each time an opening brace is put into the file, a new *scope* begins. Scopes behave similar as in programming languages, meaning that any color name or style definitions take their effect only within the scope, up to the closing brace. Thus if you redefine a style just after an opening brace, the style returns to its original definition after the closing brace. (See Section 5.14 [Defining Styles], page 62.)

Scoping also applies to the `numbering` (including pre, post, format and append) and `compress` chart options. Any changes to these take effect only until the next closing brace.

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

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

5.14 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.

Any of the attributes listed in Section 5.6 [Common Attributes], page 45 can be added to a style. You can also enlist styles among the attributes. In this case the newly defined style inherits all the attributes specified in that style. If you apply a style to an element, those attributes of the style, which not applicable to that particular element type are simply ignored. For example, applying a style including `fill.color` to an arrow will silently ignore the value of the `fill.color` attribute.

The same syntax above can be used to extend and modify styles. You can add new attributes to an existing style or modify existing attributes. This is when listing multiple

styles comes in handy. You can set attributes to the same value in multiple styles in a signle command.

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

5.14.1 Pre-defined Styles

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

First there is a built-in style for each element: `arrow`, `box`, `emptybox`, `divider`, `blockarrow`, `pipe` and `entity`. If you want to change a set of attributes for multiple elements (such as both for arrows and dividers) simply list these separated by commas before the attributes.

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

It will apply to both.

Then there are further styles defined for each arrow, box and divider symbol.

- for arrows: '`->`', '`=>`', '`>`' and '`>>`'[12].
- for block arrows: '`block->`', '`block=>`', '`block>`' and '`block>>`'.
- for boxes: '`--`', '`==`', '`++`' and '`..`'
- for pipes: '`pipe--`', '`pipe==`', '`pipe++`' and '`pipe..`'
- for dividers: '`---`' and '`...`'

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

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

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

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

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.

[12] 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.

5.15 Chart Designs

A chart design is a collection of color and style definitions, 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 ]
   options, ...
   color definitions, ...
   style definitions, ...
}
```

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 Section 5.14.1 [Pre-defined Styles], page 63 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.

On Windows, it is possible to add your design definitions to the `designlib.signalling` file. These will appear also in the design drop-down list and can also be used as arguments to the `msc` attribute. See that file for example design definitions. (Installed in `C:\Program Files\Msc-generator` by default.)