Title	App::Basis::ConvertText2
Author	Kevin Mulholland
Last Updated	2015-05-07
Version	7

Contents

1 Introduction

If you are reading this document as a markdown document you may want to try **README PDF** as an alternative. This have been generated from this file and the software provided by this distribution.

This is a perl module and a script that makes use of App::Basis::ConvertText2

This is a wrapper for **pandoc** implementing extra fenced code-blocks to allow the creation of charts and graphs etc. Documents may be created a variety of formats. If you want to create nice PDFs then it can use **PrinceXML** to generate great looking PDFs or you can use **wkhtmltopdf** to create PDFs that are almost as good, the default is to use pandoc which, for me, does not work as well.

HTML templates can also be used to control the layout of your documents.

The fenced code block handlers are implemented as plugins and it is a simple process to add new ones.

There are plugins to handle

- ditaa
- mscgen
- graphviz
- uml
- gnuplot
- gle
- sparklines
- charts
- barcodes and qrcodes
- and many others

As a perl module you can obtain it from https://metacpan.org/pod/App::Basis::ConvertText2 or install

```
cpanm App::Basis::ConvertText2
```

Alternatively it is available from https://github.com/27escape/App-Basis-ConvertText2

You will then be able to use the **ct2** script to process files

If you are reading this document in PDF form, then note that all the images are created by the various plugins and included in the output, there is no store of pre-built images. That you can read this proves the plugins all work!

Most of the chapters are based around the various plugins that are available and the commands that they expose.

2 Document header and variables

If you are just creating simple things, then you do not need a document header, but to make full use of the templating system, having header information is vital.

Example
`title: App::Basis::ConvertText2` `format: pdf` `date: 2014-05-12` `author: Kevin Mulholland` `keywords: perl, readme` `template: coverpage` `version: 5`

As you can see, we use a series of key value pairs separated with a colon. The keys may be anything you like, except for the following which have special significance.

- *format* shows what output format we should default to.
- *template* shows which template we should use

The keys may be used as variables in your document or in the template, by upper-casing and prefixing and postfixing percent symbols '%'

Example
`version as a variable %VERSION%`

If you want to display the name of a variable without it being interpreted, prefix it with an underscore '_', this underscore will be removed in the final document.

Example	Output
`%TITLE%`	App::Basis::ConvertText2

2.1 Gotchas about variables

- Variables used within the content area of a code-block will be evaluated before processing that block, if a variable has not yet been defined or saved to a buffer then it will only be evaluated at the end of document processing, so output may not be as expected.
- Variables used in markdown tables may not do what you expect if the variable is multi-line.

3 Table of contents

As documents are processed, the HTML headers (H2..H6) are collected together to make a table of contents. This can be used either in your template or document using the TOC variable.

Example
`%TOdC%`

The built table of contents is at the top of this document.

Note that if using a TOC, then the HTML headers are changed to have a number prefixed to them, this helps ensure that all the TOC references are unique.

3.1 Skipping header

If you do not want an item added to the toc add the class ‘toc_skip’ to the header (or skiptoc)

Example
`### Skipping header {.toc_skip}`

Hopefully you can see that the header for this section is not in the TOC

Note
This feature is disabled at the moment while bugs are fixed.

4 Fenced code-blocks

A fenced code-block is a way of showing that some text needs to be handled differently. Often this is used to allow markdown systems (and **pandoc** is no exception) to highlight program code.

code-blocks take the form

Example
`~~~~{.tag argument1='fred' arg2=3}` `contents ...` `~~~~`

code-blocks **ALWAYS** start at the start of a line without any preceding whitespace. The ‘top’ line of the code-block can wrap onto subsequent lines, this line is considered complete when the final ‘}’ is seen. There should be only whitespace after the closing ‘}’ symbol before the next line.

We use this construct to create our own handlers to generate HTML or markdown.

Note that only code-blocks described in this documentation have special handlers and can make use of extra features such as buffering.

If using **pandoc** then you can take advantage of the code blocks for code syntax highlighting

Example

````
~~~~{.perl}
sub process
{
    my $self = shift ;
    my ( $tag, $content, $params, $cachedir ) = @_ ;

    # make sure we have no tabs
    $content =~ s/\t/    /gsm ;
    $content = "ditaa\n$content" ;

    # and process with the normal uml command
    $params->{png} = 1 ;
    return run_block( 'uml', $content, $params, $cachedir ) ;
}
~~~~
````

Output

```perl
sub process
{
    my $self = shift ;
    my ( $tag, $content, $params, $cachedir ) = @_ ;

    # make sure we have no tabs
    $content =~ s/\t/    /gsm ;
    $content = "ditaa\n$content" ;

    # and process with the normal uml command
    $params->{png} = 1 ;
    return run_block( 'uml', $content, $params, $cachedir ) ;
}
```

4.1 Code-block short cuts

Sometimes using a fenced code-block is overkill, especially if the command to be executed does not have any content. So there is a shortcut to this. Additionally this will allow you to use multiple commands on a single line, this may be important in some instances.

Finally note that the shortcut must completely reside on a single line, it cannot span onto a separate next line, the parser will ignore it!

We wrap the command and its arguments with double braces.

Example

````
{{.tag argument1='fred' arg2=3}}
````

Its is possible to add content that would normally be in the fenced code-block, if there is not too much information, by adding it to a *content* attribute.

We can see this in action below and in the barcode examples later on.

Example

```
{{.tag argument1='fred' arg2=3 content='some text'}}
```

5 Buffers

Sometimes you may either want to repeatedly use the same information or may want to use the output from one of the fenced code-blocks .

To store data we use the **to_buffer** argument to any code-block.

Example

```
~~~~{.buffer to_buffer='spark_data'}
1,4,5,20,4,5,3,1
~~~~
```

If the code-block would normally produce some output that we do not want displayed at the current location then we would need to use the **no_output** argument.

Example

```
~~~~{.sparkline title='green sparkline' scheme='green'
    from_buffer='spark_data' to_buffer='greenspark' no_output=1}
~~~~
```

We can also have the content of a code-block replaced with content from a buffer by using the **from_buffer** argument. This is also displayed in the example above.

To use the contents (or output of a buffered code-block) we wrap the name of the buffer once again with percent '%' symbols, once again we force upper case.

Example

```
%SPARK_DATA% has content 1,4,5,20,4,5,3,1
%GREENSPARK% has a generated image
```

Buffering also allows us to add content into markdown constructs like bullets.

Example	Output
* %SPARK_DATA% * %GREENSPARK%	• 1,4,5,20,4,5,3,1 • 

6 Text

The text plugin has lots of simple handlers, they all output text/HTML.

6.1 Yamlasjson

Software engineers often use **JSON** to transfer data between systems, this often is not nice to create for documentation. **YAML** which is a superset of **JSON** is much cleaner so we have a

Example	Output
~~{.yamlasjson } list: - array: [1,2,3,7] channel: BBC3 date: 2013-10-20 time: 20:30 - array: [1,2,3,9] channel: BBC4 date: 2013-11-20 time: 21:00	{ "list" : [{ "channel" : "BBC3", "date" : "2013-10-20", "array" : [1, 2, 3, 7], "time" : "20:30" }, { "array" : [1, 2, 3, 9], "time" : "21:00", "date" : "2013-11-20", "channel" : "BBC4" }] }

6.2 Yamlasxml

Software engineers often use [XML] to transfer data between systems, this often is not nice to create for documentation. We cam create basic XML, we do not allow element attributes. If you want real XML layout use *.xml* in a fenced code block.

Example	Output
~~{.yamlasxml } list: - array: [1,2,3,7] channel: BBC3 date: 2013-10-20 time: 20:30 - array: [1,2,3,9] channel: BBC4 date: 2013-11-20 time: 21:00 ~~~~	<list> <array>1</array> <array>2</array> <array>3</array> <array>7</array> <channel>BBC3</channel> <date>2013-10-20</date> <time>20:30</time> </list> <list> <array>1</array> <array>2</array> <array>3</array> <array>9</array> <channel>BBC4</channel> <date>2013-11-20</date> <time>21:00</time> </list>

6.3 Table

Create a simple table using CSV style data

- class
  - HTML/CSS class name
- id
  - HTML/CSS class
- width
  - width of the table
- style
  - style the table if not doing anything else
- legends
  - if true first line csv as headings for table, these correspond to the data sets
- separator
  - what should be used to separate cells, defaults to ‘,’

Example

```
~~~~{.table separator=',' width='100%' legends=1}
Date,Item,Cost
2015-06-25, Tree, 23.99
2015-04-20, Shed, 400
2015-03-02, Lawn mower, 69.95
2014-12-12, Gnome, 7.95
~~~~
```

Output

Date	Item	Cost
2015-06-25	Tree	23.99
2015-04-20	Shed	400
2015-03-02	Lawn mower	69.95
2014-12-12	Gnome	7.95

6.4 Links

With one code-block we can create a list of links

The code-block contents comprises a number of lines with a reference and a URL. The reference comes first, then a '|' to separate it from the URL.

The reference may then be used elsewhere in your document if you enclose it with square ([]) brackets

There is only one argument

- class
  - CSS class to style the list

These links used in this example are the ones used in this document.

Example

```
~~~~{.links class='weblinks' }
pandoc      | http://johnmacfarlane.net/pandoc
PrinceXML   | http://www.princexml.com
markdown    | http://daringfireball.net/projects/markdown
msc         | http://www.mcternan.me.uk/mscgen/
ditaa       | http://ditaa.sourceforge.net
PlantUML    | http://plantuml.sourceforge.net
Salt        | http://plantuml.sourceforge.net/salt.html
graphviz    | http://graphviz.org
JSON        | https://en.wikipedia.org/wiki/Json
YAML        | https://en.wikipedia.org/wiki/Yaml
wkhtmltopdf | http://wkhtmltopdf.org/
My Github   | https://github.com/27escape/App-Basis-ConvertText2/tree/master/
Brewer      | http://www.graphviz.org/content/color-names#brewer
README PDF  | https://github.com/27escape/App-Basis-ConvertText2/blob/master/
~~~~
```

Output

- **Brewer**
  - http://www.graphviz.org/content/color-names#brewer
- **ditaa**
  - http://ditaa.sourceforge.net
- **graphviz**
  - http://graphviz.org
- **JSON**
  - https://en.wikipedia.org/wiki/Json
- **markdown**
  - http://daringfireball.net/projects/markdown
- **msc**
  - http://www.mcternan.me.uk/mscgen/
- **My Github**
  - https://github.com/27escape/App-Basis-ConvertText2/tree/master/scripts
- **pandoc**
  - http://johnmacfarlane.net/pandoc
- **PlantUML**
  - http://plantuml.sourceforge.net
- **PrinceXML**
  - http://www.princexml.com
- **README PDF**
  - https://github.com/27escape/App-Basis-ConvertText2/blob/master/docs/README.pdf
- **Salt**
  - http://plantuml.sourceforge.net/salt.html
- **wkhtmltopdf**
  - http://wkhtmltopdf.org/
- **YAML**
  - https://en.wikipedia.org/wiki/Yaml

6.5 Version

Documents often need revision history. I use this code-block to create a nice version table of this history.

The content for this code-block comprises a number of sections, each section then makes a row in the generated table.

```
version YYYY-MM-DD
   indented change text
   more changes
```

The version may be any string, YYYY-MM-DD shows the date the change took place. Alternate date formats is DD-MM-YYYY and '/' may also be used as a field separator.

So give proper formatting to the content in the changes column you should indent text after the version/date line with 4 spaces, not a tab character.

- class
  - HTML/CSS class name
- id
  - HTML/CSS class
- width
  - width of the table
- style
  - style the table if not doing anything else
- title
  - create a title for the version table

Example

```
~~~~{.version}
0.1 2014-04-12
  * removed ConvertFile.pm
  * using Path::Tiny
0.006 2014-04-10
  * first release to github
~~~~
```

Output

Version	Date	Changes
0.1	2014-04-12	• removed ConvertFile.pm • using Path::Tiny
0.006	2014-04-10	• first release to github

6.6 Page

There are 2 ways for force the start of a new page, using the **.page** fenced code block or by having 4 '-' signs next to each other, i.e. '—-' on a line on their own

Example
This is start a new page, again using short block form. `{{.page}}` `as will this` `----`
Output
will not be shown as it will mess up the document!

6.7 Columns

Create a columner layout, like a newspaper. The full text in the content is split into columns, the height of the section is determined by the volume of the text.

The optional arguments are

- count
  - number of columns to split into, defaults to 2
- lines
  - number of lines the section should hold, defaults to 20
- ruler
  - show a line between the columns, defaults to no, options are 1, true or yes to show it
- width
  - how wide should the column area be, defaults to 100%

Example

```
~~~~{.columns count=3 ruler=yes width='95%'}
Flexitarian lo-fi occupy, Echo Park yr chia keffiyeh iPhone pug kale chips
fashion axe PBR&B 90's readymade beard.

McSweeney's Tumblr semiotics
beard, flexitarian artisan bitters twee small batch next level PBR mustache
post-ironic stumptown.

Umami Pinterest mixtape Truffaut, Blue Bottle ugh
artisan whatever blog street art Odd Future crucifix tomato shore invisible
spelling.
~~~~
```

Output

Flexitarian lo-fi occupy, Echo Park yr chia keffiyeh iPhone pug kale chips fashion axe PBR&B 90’s readymade beard.	McSweeney’s Tumblr semiotics beard, flexitarian artisan bitters twee small batch next level PBR mustache post-ironic stumptown.	Umami Pinterest mixtape Truffaut, Blue Bottle ugh artisan whatever blog street art Odd Future crucifix tomato shore invisible spelling.

6.8 Tree

Draw a bulleted list as a directory tree. Bullets are expected to be indented by 4 spaces, we will only process bullets that are * + or -.

Example	Output

```
~~~~{.tree}
* one
    * 1.1
* two
    * two point 1
    * 2.2
* three
    * 3.1
    * 3.2
    * three point 3
        * four
            * five
        * six
    * 3 . seven
~~~~
```

- one
  - 1.1
- two
  - two point 1
  - 2.2
- three
  - 3.1
  - 3.2
  - three point 3
    - four
      - five
    - six
  - 3 . seven

6.9 Badges

Badges (or shields) are a way to display information, often used to show status of an operation on websites such as github.

Examples of shields can be seen at **sheilds.io**

The badges are placed inline, so you can insert text around the fenced codeblock.

Depending on your template the color of the text and the color for the status portion may clash, so take care!

The required argument are

- subject
  - text saying what the button is
- status
  - text status to put at the end of the button

The optional arguments are

- color
  - over ridge default color goldenrod
- size
  - the width of the button

Example
A basic badge ~~~~{.badge subject='test run' status='completed' color='green'} ~~~~ Badges / shields work well as short blocks {{.shield subject='test run' status='pending' size='150'}}
Output
test run completed Badges / shields work well as short blocks test run pending

6.10 Box

Show that something is important by putting it in a box

The optional arguments are

- class
  - HTML/CSS class name
- id
  - HTML/CSS class
- width
  - width of the box (default 98%)
- title
  - optional title for the section
- style
  - style the box if not doing anything else

Example	Output
~~{.box from_buffer=box title='Important Notice' width='80%'} ~~~~	**Important Notice** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque sit amet accumsan est. Nulla facilisi. Nulla lacus augue, gravida sit amet laoreet id, commodo vitae velit. Fusce in nisi mi. Nulla congue nulla ac bibendum semper. In rutrum sem eget purus auctor porttitor. Mauris vel pellentesque lorem. Vestibulum consectetur massa non fermentum dignissim.

6.11 Glossary

Build a glossary of terms or abbreviations as you progress with your document. Show them later on, there is no way (currently) to place the glossary ahead of any definitions.

Example

```
This is a {{.gloss abbr='SMPL' def='short spelling of SAMPLE'}}
There are other things we can do
{{.glossary abbr='test' define='Test long form and arguments'}}

Optionally if there is a link to a item e.g. [Links](#links)
{{.gloss abbr='msc' define='Message Sequence Charts' link=1}}
then this can link to the relevant website, the following link
has not been added to
{{.gloss abbr='JSON' define='JavaScript Object Notation'}},
so no link to the website.

Now finally, show the results
{{.gloss show=1}}
```

Output

This is a SMPL There are other things we can do test

Optionally if there is a link to a item in **Links msc** then this can link to the relevant website, the following link has not been added to JSON, so no link to the website.

Now finally, show the results

Abbreviation	Definition
JSON	JavaScript Object Notation
SMPL	short spelling of SAMPLE
msc	Message Sequence Charts
test	Test long form and arguments

6.12 Quote

Pandoc provides for blockquotes, these are often like

```
> a standard quote
> another line of the block
>
> And a final one
```

as

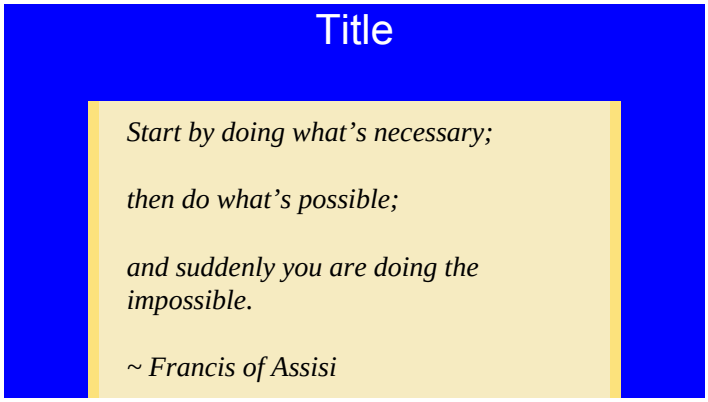
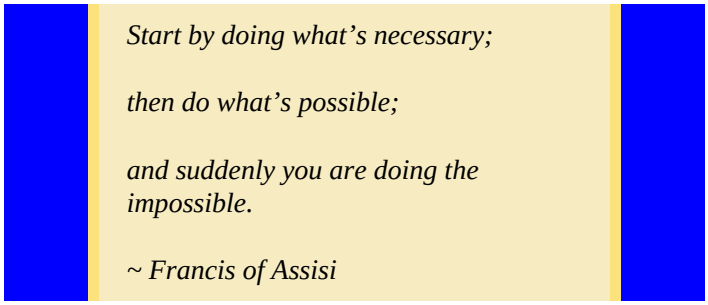
> *a standard quote another line of the block*
> *And a final one*

We want something that can be styled differently and can have a title

The optional arguments are

- class
  - HTML/CSS class name

- id
  - HTML/CSS class
- title
  - optional title for the section

Example	Output
~~{.quote title='Title' width=100%} Start by doing what's necessary; then do what's possible; and suddenly you are doing the impossible. ~ Francis of Assisi ~~~~	
Or without the title ~~~~{.quote title='Title' width=350px} Start by doing what's necessary; then do what's possible; and suddenly you are doing the impossible. ~ Francis of Assisi ~~~~	

7 Sparklines

Sparklines are simple horizontal charts to give an indication of things, sometimes they are barcharts but we have nice smooth lines.

The only valid contents of the code-block is a single line of comma separated numbers.

The optional arguments are

- title
  - used as the generated images 'alt' argument
- bgcolor
  - background color in hex (123456) or transparent

- line
  - color or the line, in hex (abcdef)
- color
  - area under the line, in hex (abcdef)
- scheme
  - color scheme
  - options: red blue green orange mono
- size
  - size of image, default 80x20, widthxheight

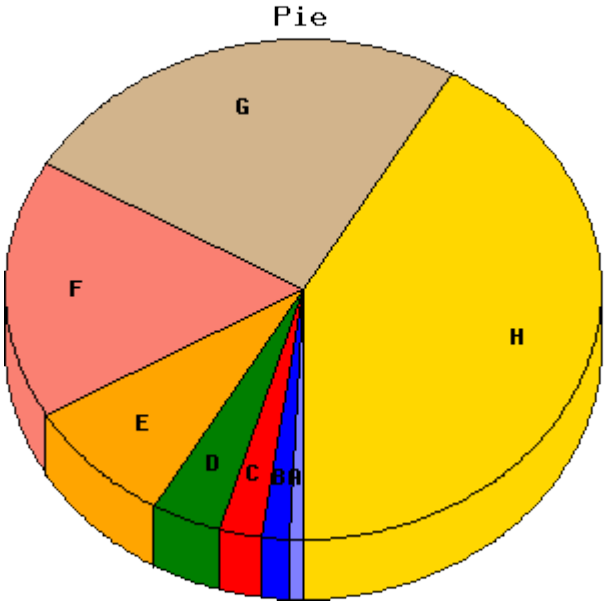
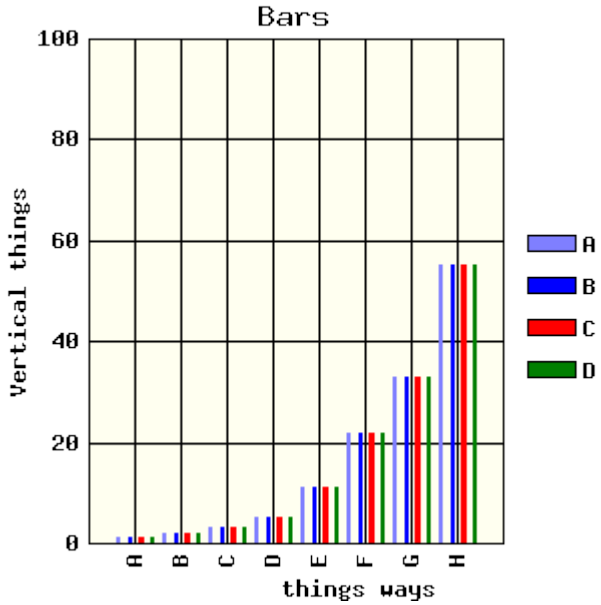
Example	Output
~~{.buffer to_buffer='spark_data'} 1,4,5,20,4,5,3,1 ~~~~ here is a standard sparkline ~~~~{.sparkline title='basic sparkline'} 1,4,5,20,4,5,3,1 ~~~~	
Draw the sparkline using buffered data ~~~~{.sparkline title='blue sparkline' scheme='blue' from_buffer='spark_data'} ~~~~	

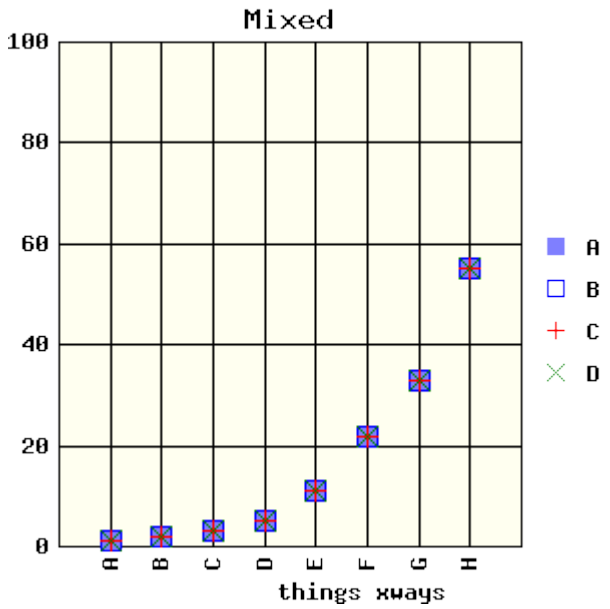
8 Charts

Displaying charts is very important when creating reports, so we have a simple **chart** code-block.

We will buffer some data to start. The content comprises lines of comma separated data. The first line of the content is the legends; subsequent lines relate to each of these legends.

```
~~~~{.buffer to='chart_data'}
A,B,C,D,E,F,G,H
1,2,3,5,11,22,33,55
1,2,3,5,11,22,33,55
1,2,3,5,11,22,33,55
1,2,3,5,11,22,33,55
~~~~
```

Example	Output
Pie Chart ~~~~{.chart format='pie' title='Pie' from_buffer='chart_data' size='300x300'} ~~~~	
Bar Chart ~~~~{.chart format='bars' title='Bars' from_buffer='chart_data' size='300x300' xaxis='things ways' yaxis='Vertical things' legends='A,B,C,D,E,F,G,H' } ~~~~	

Example	Output
Mixed Chart ~~~~{.chart format='mixed' title='Mixed' from_buffer='chart_data' size='300x300' xaxis='things xways' axis='Vertical things' legends='A,B,C,D,E,F,G,H' } types='lines linepoints lines bars' } ~~~~	

9 mscgen

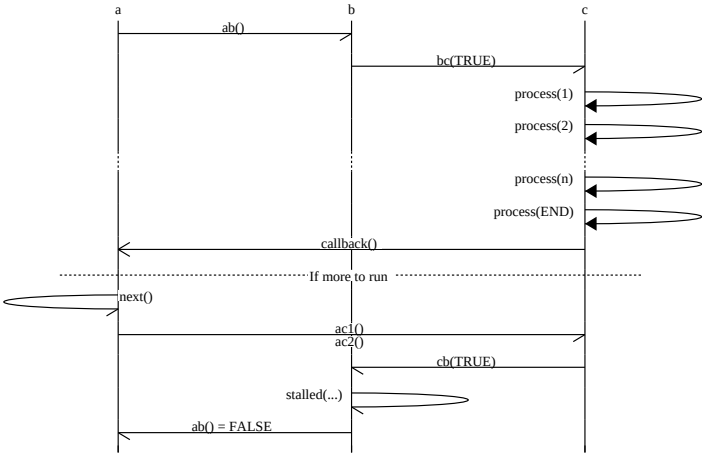
Message Sequence Charts

Software (or process) engineers often want to be able to show the sequence in which a number of events take place. We use the **msc** program for this. This program needs to be installed onto your system to allow this to work

The content for this code-block is EXACTLY the same that you would use as input to **msc**

The optional arguments are

- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image

Example	Output
~~{.mscgen title="mscgen1" width=350} # MSC for some fictional process msc { a,b,c; a->b [label = "ab()"] ; b->c [label = "bc(TRUE)"]; c=>c [label = "process(1)"]; c=>c [label = "process(2)"]; ...; c=>c [label = "process(n)"]; c=>c [label = "process(END)"]; a<<=c [label = "callback()"]; --- [label = "If more to run"]; a->a [label = "next()"]; a->c [label = "ac1()\nac2()"]; b<-c [label = "cb(TRUE)"]; b->b [label = "stalled(...)"]; a<-b [label = "ab() = FALSE"]; } ~~~~	

10 UML Diagrams

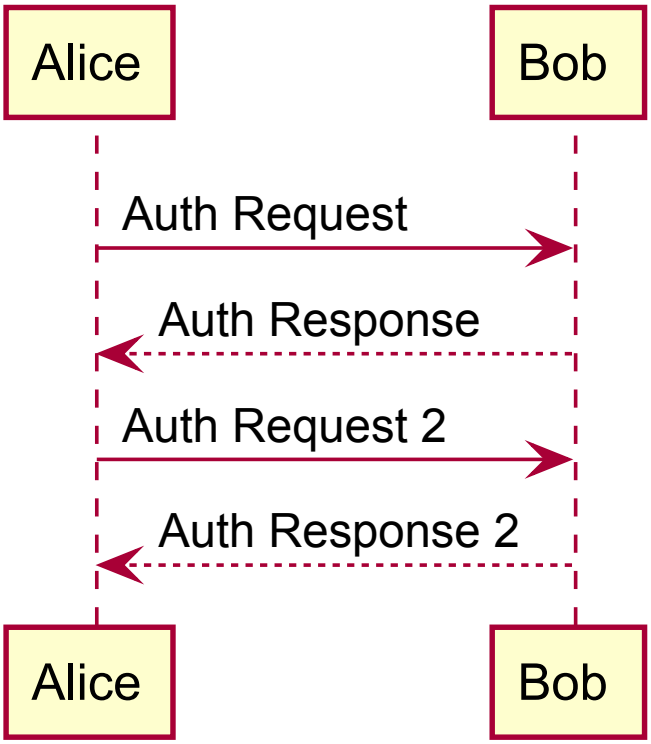
Software engineers love to draw diagrams, **PlantUML** is a java component to make this simple.

You will need to have a script on your system called ‘uml’ that calls java with the plantuml component. Mine is available from **My Github** repo.

The content for this code-block must be the same that you would use to with the **PlantUML** software

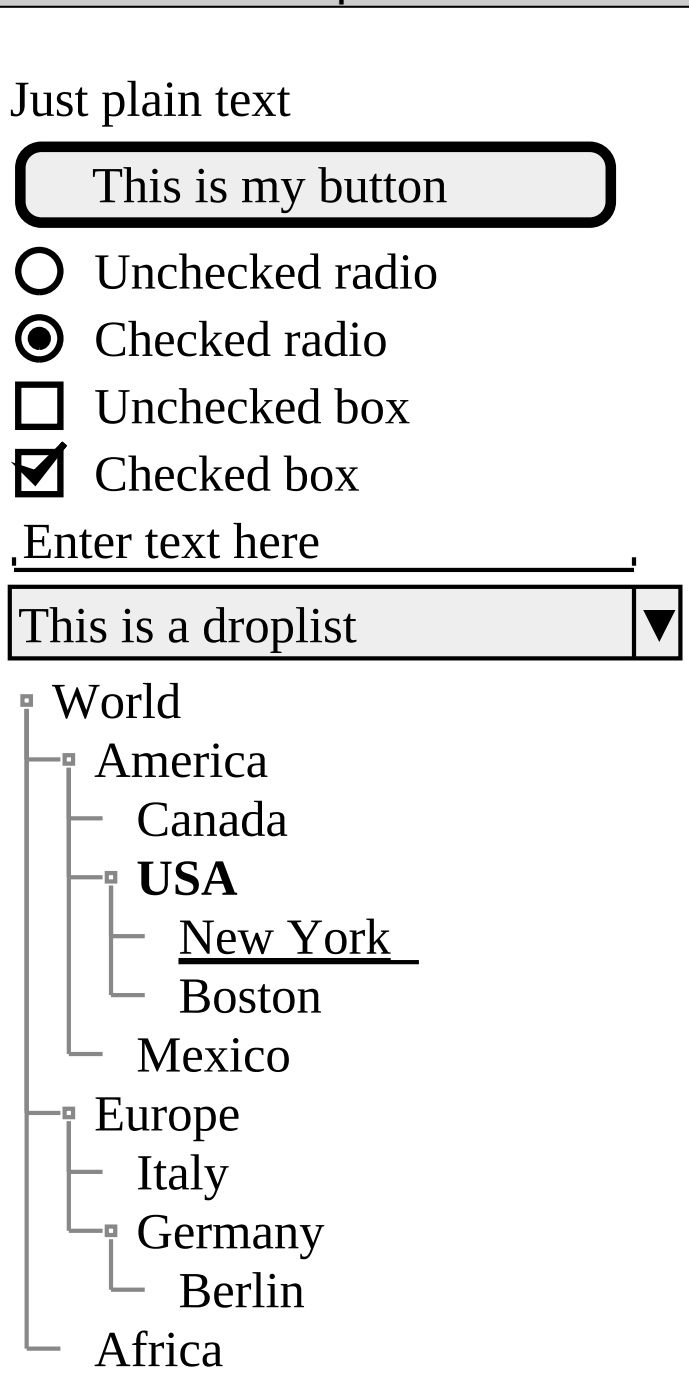
The optional arguments are

- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image

Example	Output
~~{.uml width=350} ' this is a comment on one line /' this is a multi-line comment'/ Alice -> Bob: Auth Request Bob --> Alice: Auth Response Alice -> Bob: Auth Request 2 Alice <-- Bob: Auth Response 2 ~~~~	

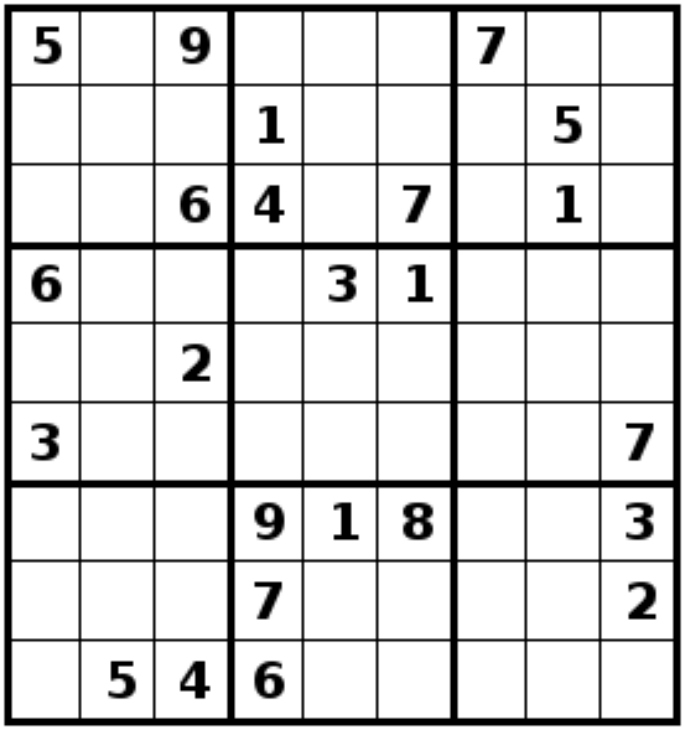
10.1 Salt

PlantUML can also create simple application interfaces See **Salt**

Example	Output
~~{.uml width=350} @startuml salt { Just plain text [This is my button] () Unchecked radio (X) Checked radio [] Unchecked box [X] Checked box "Enter text here " ^This is a droplist^ {T + World ++ America +++ Canada +++ **USA** ++++ __New York__ ++++ Boston +++ Mexico ++ Europe +++ Italy +++ Germany ++++ Berlin ++ Africa } } @enduml ~~~~	

10.2 Sudocku

Plantuml can generate random sudocku patterns

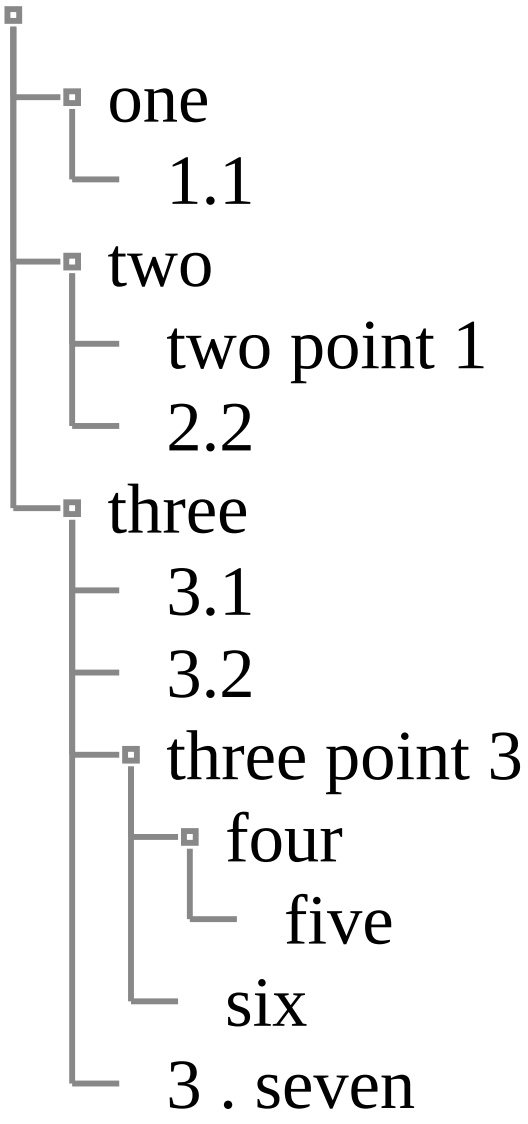
Example	Output
~~{.uml width=350} sudoku ~~~~	

To always generate the same pattern, append a seed value after ‘sudoku’

~~~~{.uml}
sudoku 45azkdf4sqq
~~~~

10.3 Umltree

Draw a bulleted list as a tree using the plantuml salt GUI layout tool. Bullets are expected to be indented by 4 spaces, we will only process bullets that are * + or -.

Example	Output
~~{.umltree width=350} * one * 1.1 * two * two point 1 * 2.2 * three * 3.1 * 3.2 * three point 3 * four * five * six * 3 . seven ~~~~	

10.4 Ditaa

Diagrams Through Ascii Art

This is a special system to turn ASCII art into pretty pictures, nice to render diagrams. You do need to make sure that you are using a proper monospaced font with your editor otherwise things will go awry with spaces.

Rather than use the **ditaa** application and to reduce the number of applications that need to be installed on the system, we use the ditaa component of plantuml. This does have some limitations, for example there is no way to switch spaces or shadows off. However this is a useful tradeoff

The content for this code-block must be the same that you would use to with the ditaa software.

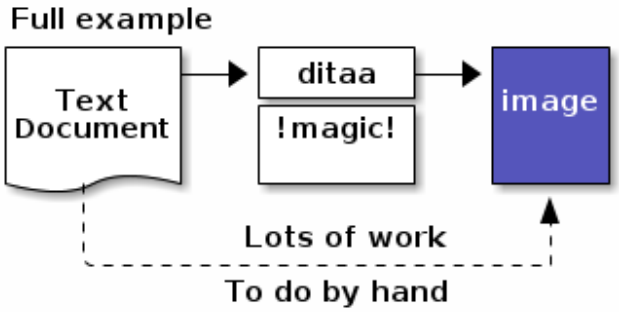
The optional arguments are

- title

- used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image

Example	Output

```
~~~~{.ditaa width=350}
Full example
+--------+   +-------+   +-----+
|        +-->| ditaa +-->|     |
|  Text  |   +-------+   |image|
|Document|   |!magic!|   |     |
|     {d}|   |       |   |cBLU |
+---+----+   +-------+   +-----+
    :                       ^
    |       Lots of work    |
    \-----------------------+
            To do by hand
~~~~
```



11 Graphviz

graphviz allows you to draw connected graphs using text descriptions.

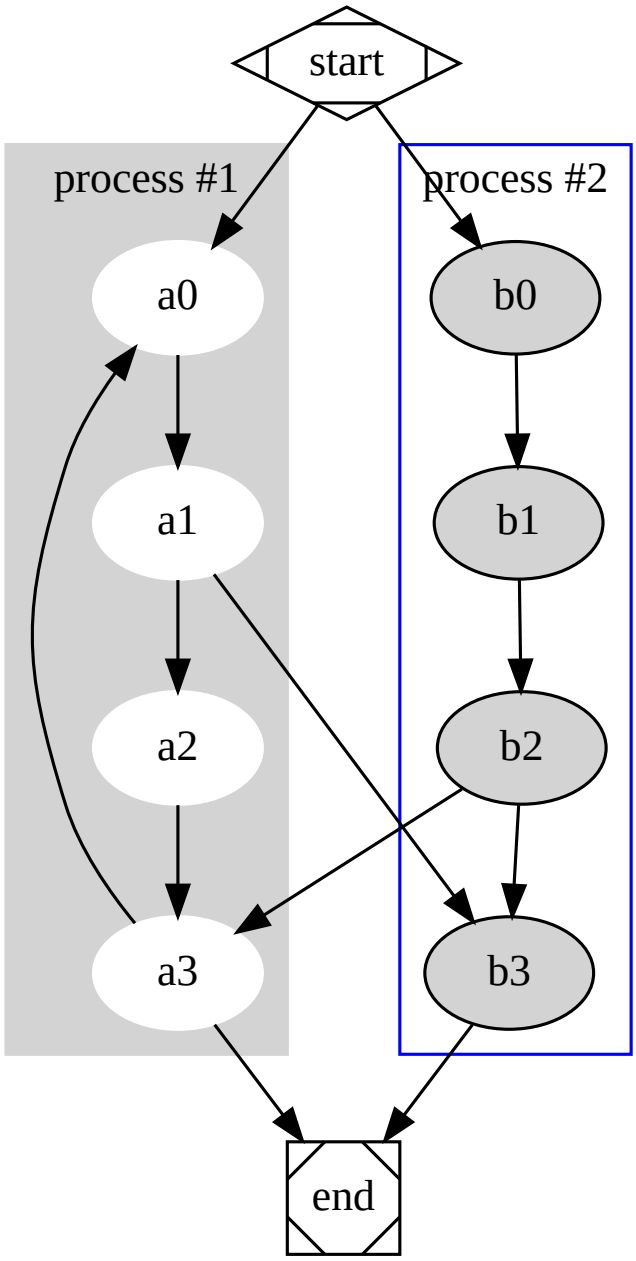
The content for this code-block must be the same that you would use to with the **graphviz** software

The optional arguments are

- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image
- command
  - command used to draw the graph, defaults to dot
  - options are dot, neato, twopi, fdp, sfdp, circo, osage

Example	Output

```
~~~~{.graphviz  title="graphviz1"
  size=width=350}
  digraph G {

  subgraph cluster_0 {
    style=filled;
    color=lightgrey;
    node [style=filled,color=white];
    a0 -> a1 -> a2 -> a3;
    label = "process #1";
  }

  subgraph cluster_1 {
    node [style=filled];
    b0 -> b1 -> b2 -> b3;
    label = "process #2";
    color=blue
  }
  start -> a0;
  start -> b0;
  a1 -> b3;
  b2 -> a3;
  a3 -> a0;
  a3 -> end;
  b3 -> end;

  start [shape=Mdiamond];
  end [shape=Msquare];
}
~~~~
```



11.1 Mindmap

There is no nice way to convert plain text to mindmaps, the only way normally would be to use graphviz or something similar.

However, converting a bulleted list into a simple mindmap would be ideal!

Each bulleted item would be a node in the mindmap.

Bullets can be '*','+'or'-' and should be indented 4 spaces to indicate a child. Poor indenting can be managed to a degree.

There should be a single top level bullet, which is used as the root of the map, see the example below. Multiple top level bullets will result in a badly organised mindmap.

Markdown style bold and italics markers can be used, as can ‘’ to start a new line in a node.

Comments can be added and will be ignored when rendering the mindmap, anything following ‘:’ will be considered as a comment.

The optional arguments are

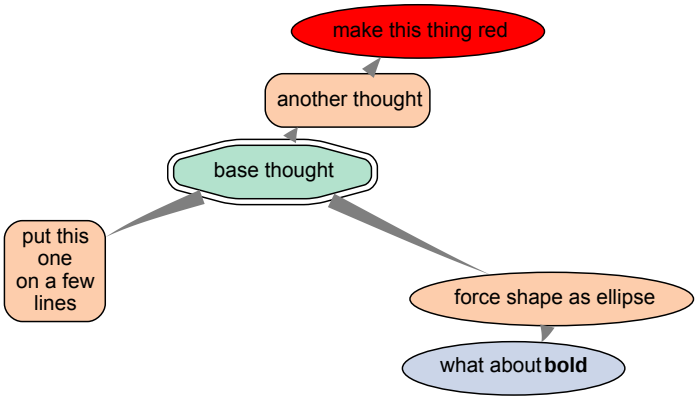
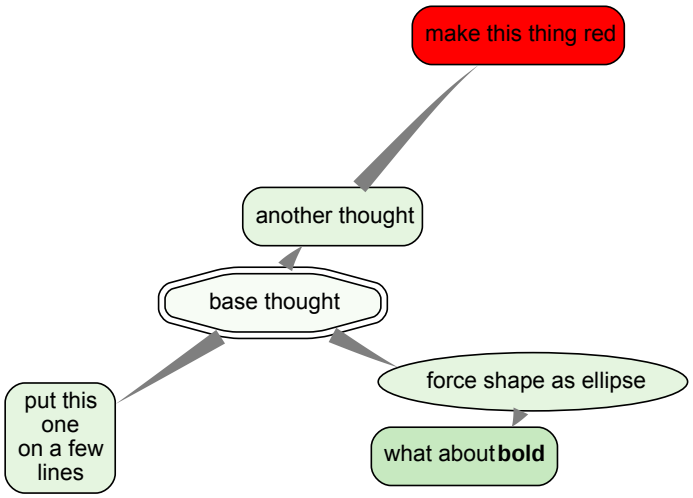
- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image
- command
  - command used to draw the graph, defaults to dot
    - options are dot neato twopi fdp sfdp circo osage
- scheme - color scheme to use - optional
  - default pastel28, schemes taken from **Brewer**
  - blue purple green grey mono orange red brown are shortcuts
- shapes - list of shapes to use - optional
  - default box ellipse hexagon octagon

Bullet text can override some shapes

- wrap in the following to get the required shape
- [] shape=box
- () shape=ellipse
- <> shape=diamond
- {} shape=octagon
- include a shape=
  - shape=box3d

Bullet text can include a color override

- #red or #123456 or #ff00ff

Example	Output
~~{.mindmap size=350x250} * base thought + (force shape as ellipse) + what about **bold** + another thought :comment to ignore + make this thing red #red * put this\none\non a few\nlines ~~~~	
Change the node style and the color scheme. ~~~~{.mindmap shapes='box' scheme=green size=350x250} * base thought + (force shape as ellipse) + what about **bold** + another thought :comment to ignore + make this thing red #red * put this\none\non a few\nlines ~~~~	

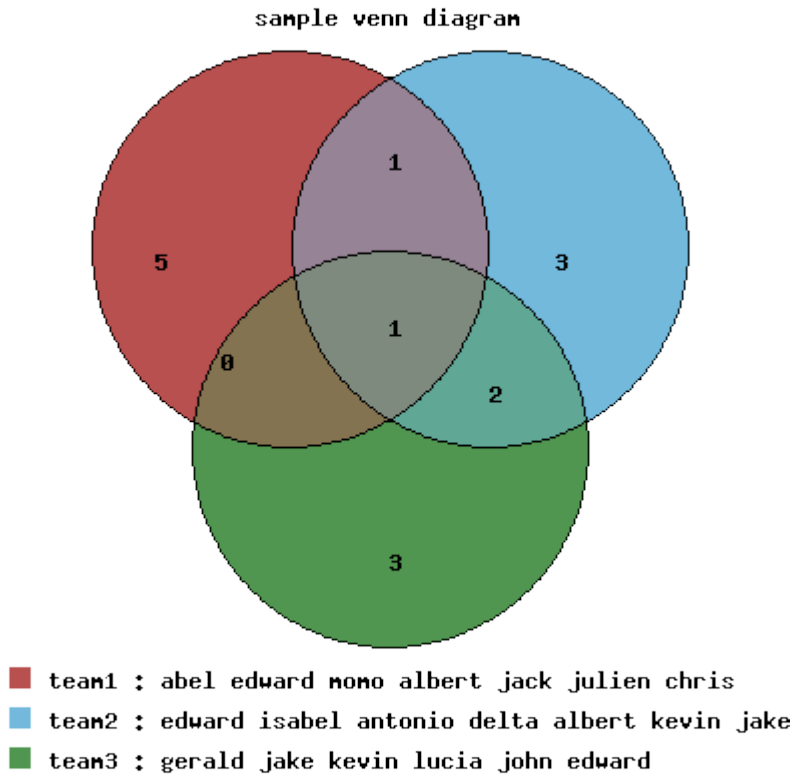
12 Venn diagram

Creating venn diagrams may sometimes be useful, though to be honest this implementation is not great, if I could find a better way to do this then I would!

Example

```
~~~~{.venn  title="sample venn diagram"
    legends="team1 team2 team3" scheme="rgb" explain='1'}
abel edward momo albert jack julien chris
edward isabel antonio delta albert kevin jake
gerald jake kevin lucia john edward
~~~~
```

Output



- only in team1 : julien abel momo jack chris
  - only in team2 : delta isabel antonio
  - team1 and team2 share : albert
- only in team3 : lucia john gerald
  - team1 and team3 share :
  - team2 and team3 share : jake kevin
  - team1, team2 and team3 share : edward

13 Barcodes

Sometimes having barcodes in your document may be useful, certainly qrcodes are popular.

The code-block only allows a single line of content. Some of the barcode types need content of a specific length, warnings will be generated if the length is incorrect.

The arguments allowed are

- title
  - used as the generated images ‘alt’ argument
- height
  - height of image
- notext
  - flag to show we do not want the content text printed underneath the barcode.
- version
  - version of qrcode, defaults to ‘2’
- pixels
  - number of pixels that is a ‘bit’ in a qrcode, defaults to ‘2’
- type
  - the type of the barcode
  - code39
  - coop2of5
  - ean8 - 8 characters allowed in content
  - ean13 - 13 characters allowed in content
  - iata20f5
  - industrial20f5
  - itf
  - matrix2of5
  - nw7
  - qrcode

Example	Output
Code 39 `~~~~{.barcode type='code39'}` `123456789}` `~~~~`	
EAN8 `~~~~{.barcode type='ean8'}` `12345678` `~~~~`	
IATA2of5 `~~~~{.barcode type='IATA2of5'}` `12345678` `~~~~`	

13.1 QR code

As qrcodes are now quite so prevalent, they have their own code-block type.

We can do qr codes, just put in anything you like, this is a URL for bbc news

Example	Output
~~{.qrcode } http://news.bbc.co.uk ~~~~	
To change the size of the barcode ~~~~{.qrcode height='80'} http://news.bbc.co.uk ~~~~	
To use version 1 Version 1 only allows 15 characters ~~~~{.qrcode height=60 version=1} smaller text.. ~~~~	
To change pixel size ~~~~{.qrcode pixels=5} smaller text.. ~~~~	

14 Gle / glx

This is a complex graph/chart drawing package available from http://glx.sourceforge.net/

The optional arguments are

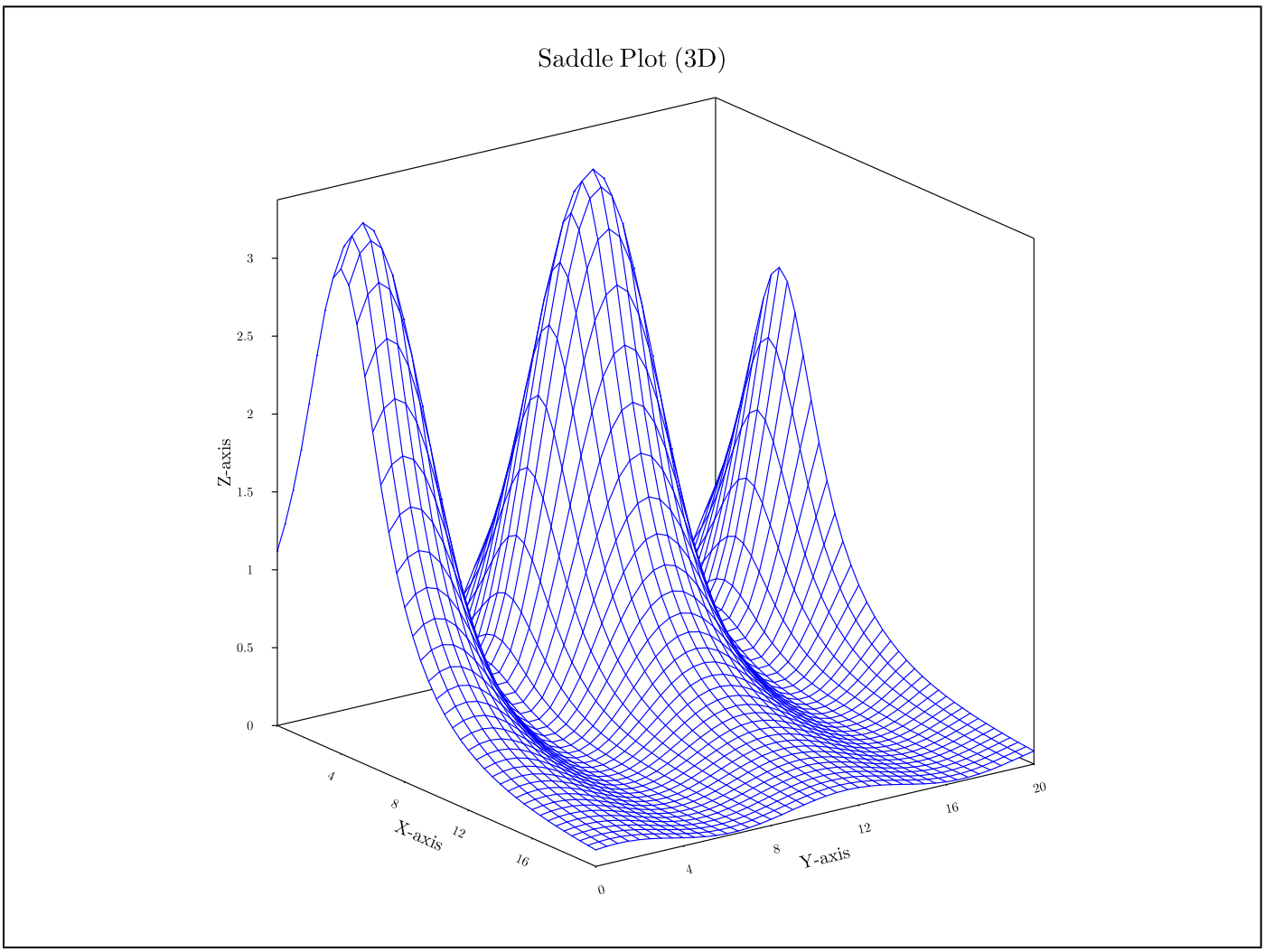
- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, default 720x512, widthxheight, size is approximate
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image
- title

  - used as the generated images ‘alt’ argument
- transparent
  - flag to use a transparent background

Example

```
~~~~{.gle}
set font texcmr hei 0.5 just tc

begin letz
   data "saddle.z"
   z = 3/2*(cos(3/5*(y-1))+5/4)/(1+(((x-4)/3)^2))
   x from 0 to 20 step 0.5
   y from 0 to 20 step 0.5
end letz

amove pagewidth()/2 pageheight()-0.1
write "Saddle Plot (3D)"

begin object saddle
   begin surface
      size 10 9
      data "saddle.z"
      xtitle "X-axis" hei 0.35 dist 0.7
      ytitle "Y-axis" hei 0.35 dist 0.7
      ztitle "Z-axis" hei 0.35 dist 0.9
      top color blue
      zaxis ticklen 0.1 min 0 hei 0.25
      xaxis hei 0.25 dticks 4 nolast nofirst
      yaxis hei 0.25 dticks 4
   end surface
end object

amove pagewidth()/2 0.2
draw "saddle.bc"
~~~~
```

Output



15 Gnuplot

This is the granddaddy of charting/plotting programs, available from http://gnuplot.sourceforge.net/.

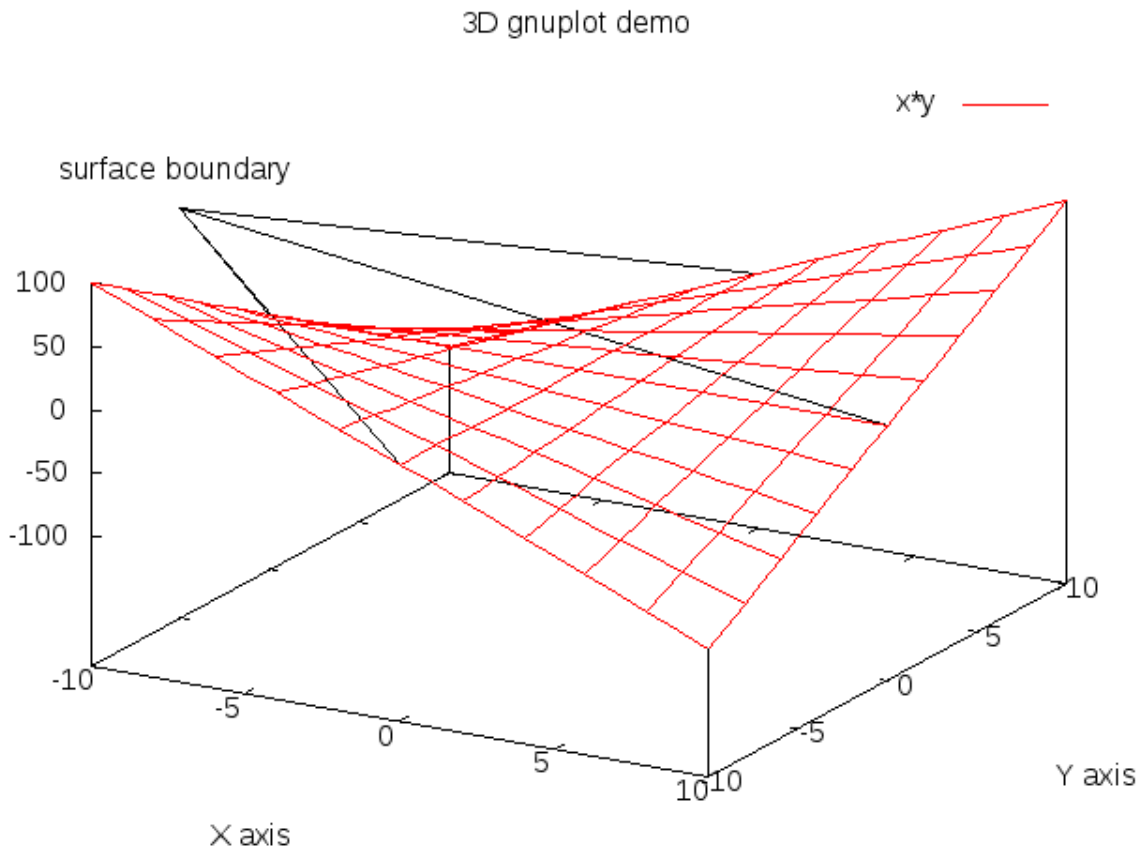
The optional arguments are

- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, default 720x512, widthxheight

Example

~~~~{.gnuplot}
```
set samples 21
set isosample 11
set xlabel "X axis" offset -3,-2
set ylabel "Y axis" offset 3,-2
set zlabel "Z axis" offset -5
set title "3D gnuplot demo"
set label 1 "surface boundary" at -10,-5,150 center
set arrow 1 from -10,-5,120 to -10,0,0 nohead
set arrow 2 from -10,-5,120 to 10,0,0 nohead
set arrow 3 from -10,-5,120 to 0,10,0 nohead
set arrow 4 from -10,-5,120 to 0,-10,0 nohead
set xrange [-10:10]
set yrange [-10:10]
splot x*y
```
~~~~

Output



16 Ploticus

This is a rather old school charting applitcation, though it can create some graphs and charts that the other plugins cannot, e.g. Timelines.

The optional arguments are

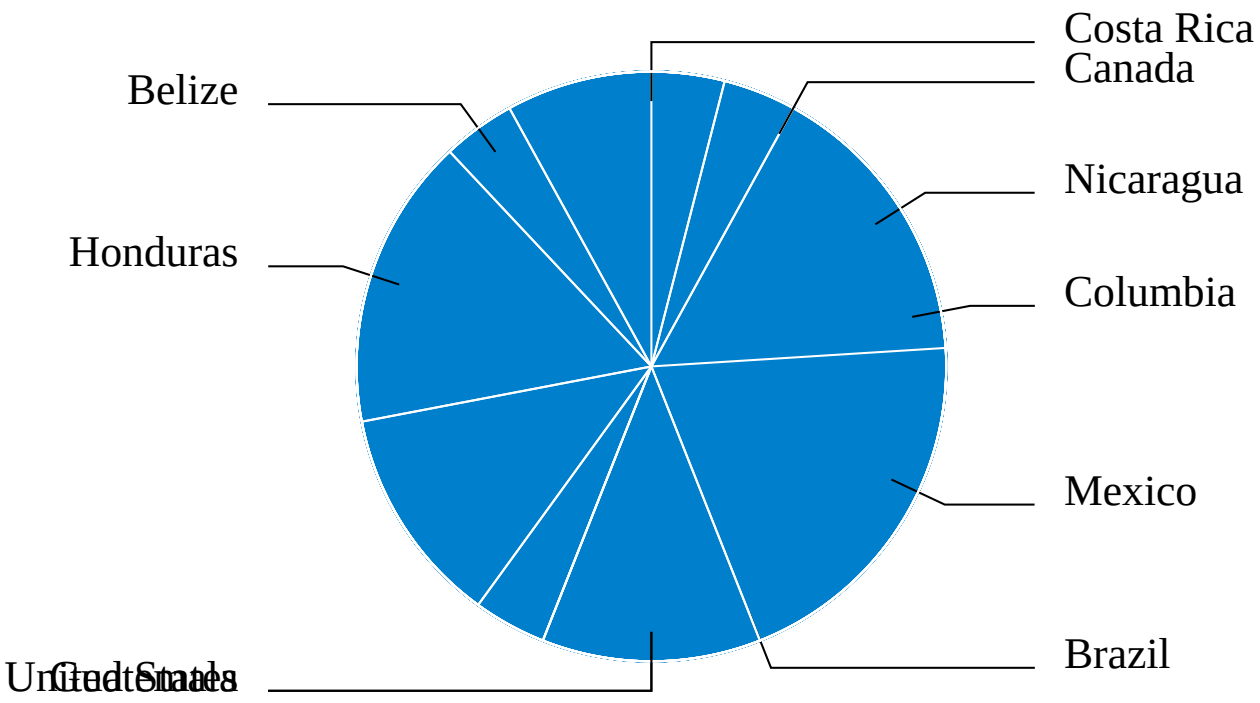
- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image

Its best to let ploticus control the size of the generated images, you may need some trial and error with the ploticus ‘pagesize:’ directive.

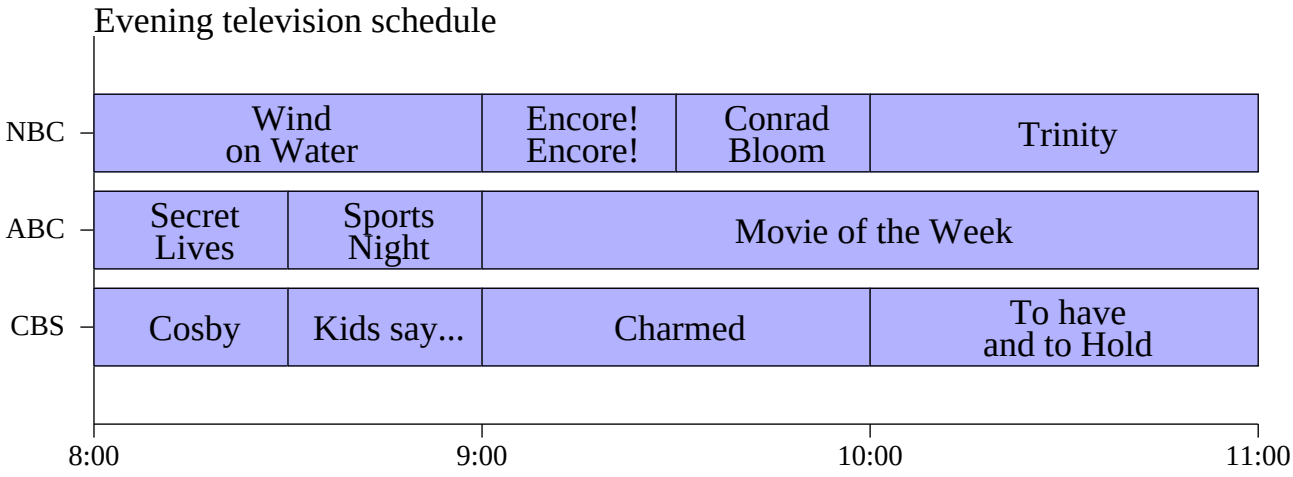
Example

```
~~~~{.ploticus}
//  specify data using proc getdata
#proc getdata
data: Brazil 22
  Columbia 17
  "Costa Rica" 22
  Guatemala 3
  Honduras 12
  Mexico 14
  Nicaragua 28
  Belize 9
  "United States" 21
  Canada 8

//  render the pie graph using proc pie
#proc pie
datafield: 2
labelfield: 1
labelmode: line+label
center: 4 3
radius: 1
colors: oceanblue
outlinedetails: color=white
labelfarout: 1.3
total: 256
~~~~
```

Output



And here is that timeline (from http://ploticus.sourceforge.net/gallery/clickmap_time2.htm)



17 Polaroid

Display an image with a bounding box so it looks like a polaroid snap.

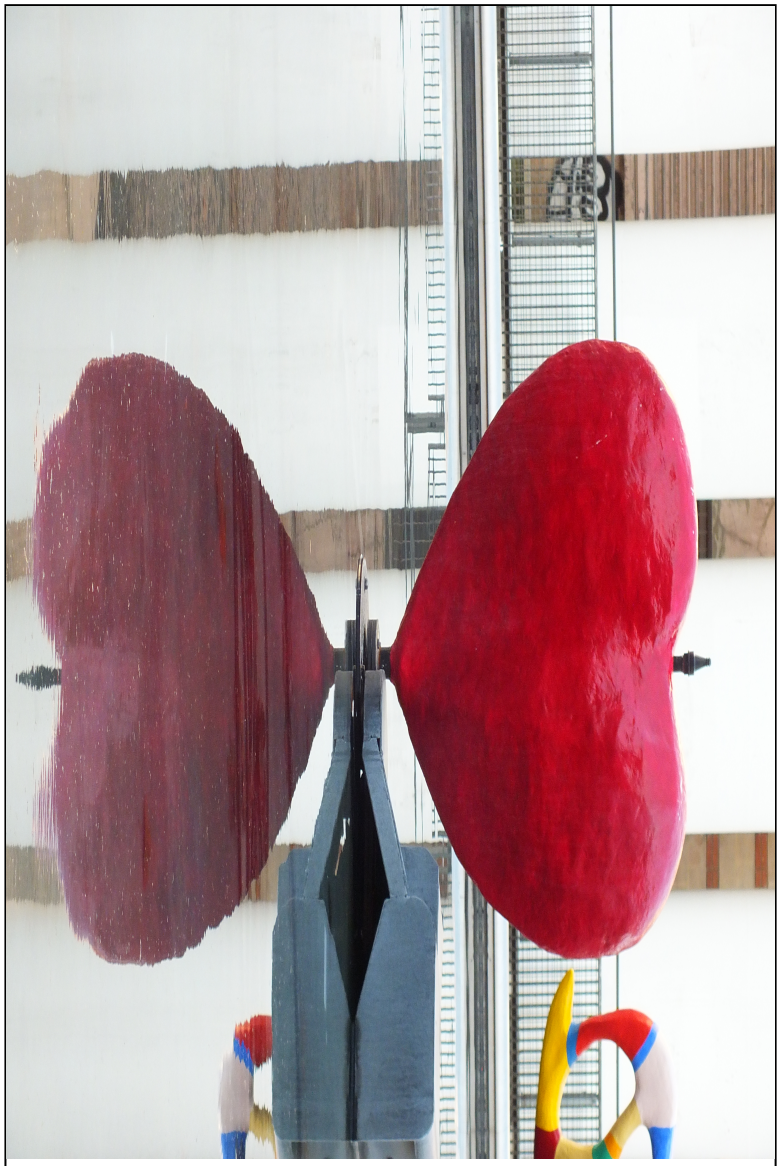
reate a polaroid style image with space underneath for writing on. Image may be automatically rotated depending upon the exif information

The required arguments are

- src
  - filename to convert to a polaroid

The optional arguments are

- title
  - title for the photo
- date
  - optional date for the photo

Example	Output
~~{.polaroid src='heartp.jpg' title='The heart of Paris' date='2015-02-06'} ~~~~	 The heart of Paris 2015-02-06

18 Blockdiag

Blockdiag provides a number of ways to create nice diagrams. See **http://blockdiag.com/** for more information, how to install and examples.

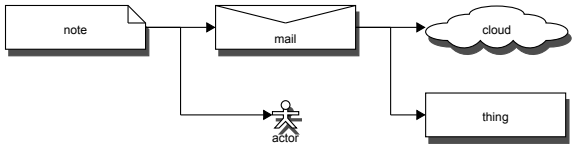
To keep things simple, we add the correct wrapper around the fenced codeblock, so that there is no need to add **blockdiag {** etc to the start and end of the blocks.

The optional arguments are

- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image
- transparent
  - should the background be transparent, default true

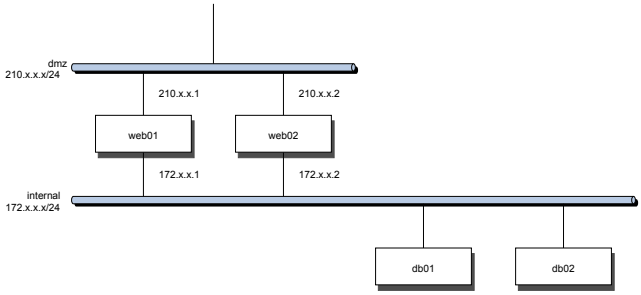
18.1 blockdiag

Similar to graphviz, layout is more grid based.

Example	Output
~~{.blockdiag width=350} note [shape = note]; mail [shape = mail]; cloud [shape = cloud]; actor [shape = actor]; note -> mail ; mail -> cloud ; mail -> thing ; note -> actor; ~~~~	

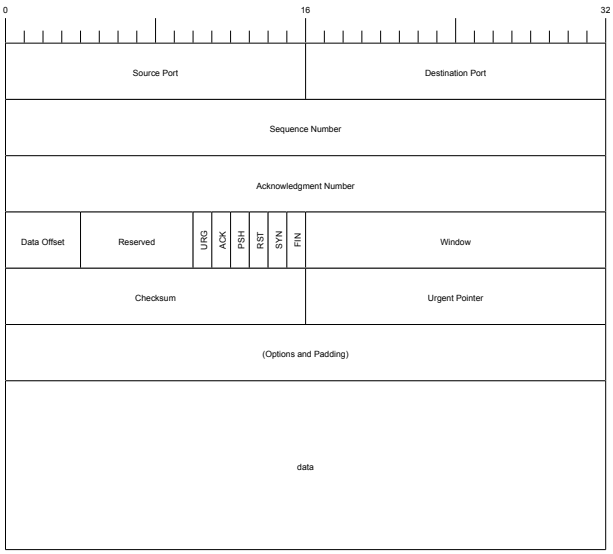
18.2 nwdiag

One for the system administrators, draw your network nicely with the various networks and systems connected to them.

Example	Output
~~{.nwdiag width=350} network dmz { address = "210.x.x.x/24" web01 [address = "210.x.x.1"]; web02 [address = "210.x.x.2"]; } network internal { address = "172.x.x.x/24"; web01 [address = "172.x.x.1"]; web02 [address = "172.x.x.2"]; db01; db02; } ~~~~	

18.3 packetdiag

Useful to describe packets of data, e.g. binary data passed between networks or stored to files.

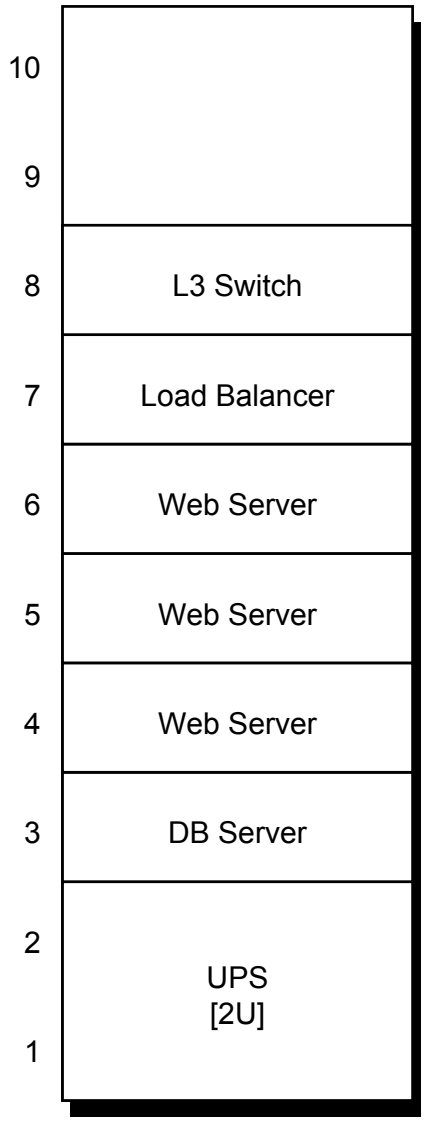
Example	Output
~~{.packetdiag width=350} colwidth = 32 node_height = 72 0-15: Source Port 16-31: Destination Port 32-63: Sequence Number 64-95: Acknowledgment Number 96-99: Data Offset 100-105: Reserved 106: URG [rotate = 270] 107: ACK [rotate = 270] 108: PSH [rotate = 270] 109: RST [rotate = 270] 110: SYN [rotate = 270] 111: FIN [rotate = 270] 112-127: Window 128-143: Checksum 144-159: Urgent Pointer 160-191: (Options and Padding) 192-223: data [colheight = 3] ~~~~	

18.4 rackdiag

Useful for system administrators to keep track of their server rooms.

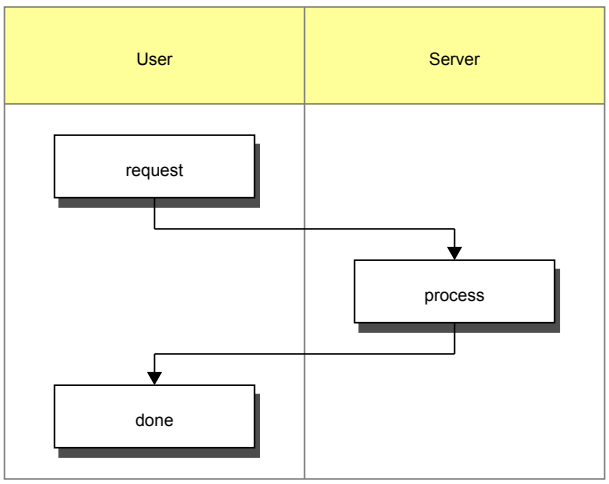
Example	Output

```
~~~~{.rackdiag width=350}
// define height of rack
10U;

// define rack items
1: UPS [2U];
3: DB Server
4: Web Server
5: Web Server
6: Web Server
7: Load Balancer
8: L3 Switch
~~~~
```



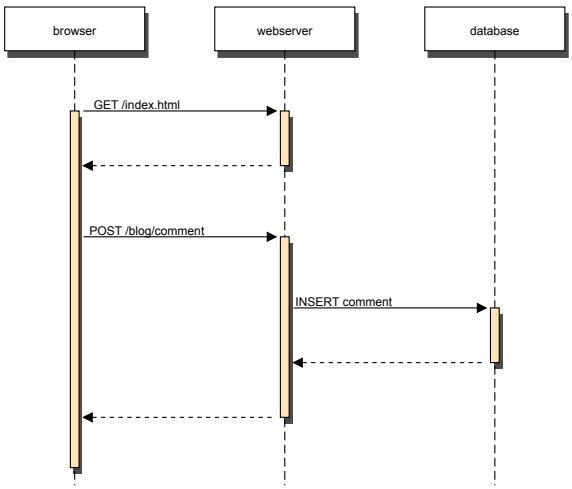
18.5 actdiag

The classic swim lanes.

Example	Output
~~{.actdiag} write -> convert -> image lane user { label = "User" write [label = "request"]; image [label = "done"]; } lane server { label = "Server" convert [label = "process"]; } ~~~~	

18.6 seqdiag

Very similar to the output of mscgen and uml tags.

Example	Output
~~{.seqdiag} browser -> webserver [label = "GET /index.html"]; browser <-- webserver; browser -> webserver [label = "POST /blog/comment"]; webserver -> database [label = "INSERT comment"]; webserver <-- database; browser <-- webserver; ~~~~	

19 Dataflow

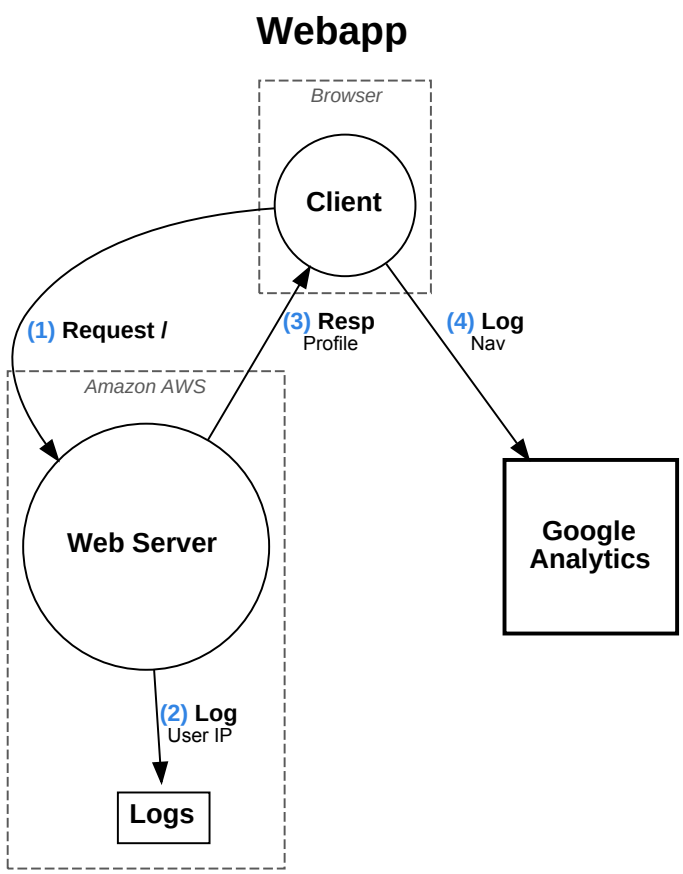
Dataflow diagrams allow multiple outputs from a single workflow description.

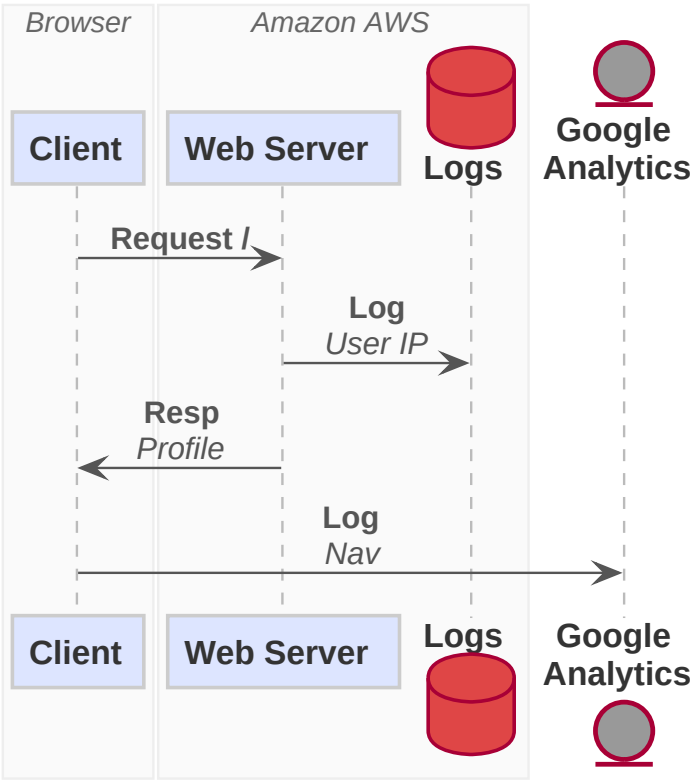
See **https://github.com/sonyxperiadev/dataflow/blob/master/USAGE.md** for more information, how to install and examples.

The optional arguments are

- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image
- format
  - either dfd (default) or seq

The example images are a bit compressed to fit in the table, normally they look a lot better than this!

Example	Output
Dataflow ~~~~{.dataflow format=dfd} diagram 'Webapp' { boundary 'Browser' { function client 'Client' } boundary 'Amazon AWS' { function server 'Web Server' database logs 'Logs' } io analytics 'Google Analytics' client -> server 'Request /' '' server -> logs 'Log' 'User IP' server -> client 'Resp' 'Profile' client -> analytics 'Log' 'Nav' } ~~~~	

Example	Output
As a sequence diagram ~~~~{.dataflow format=seq} diagram 'Webapp' { boundary 'Browser' { function client 'Client' } boundary 'Amazon AWS' { function server 'Web Server' database logs 'Logs' } io analytics 'Google Analytics' client -> server 'Request /' '' server -> logs 'Log' 'User IP' server -> client 'Resp' 'Profile' client -> analytics 'Log' 'Nav' } ~~~~	

20 Google charts

Some of the charts from https://developers.google.com/chart/ have been implemented.

This plugin requires **phantomjs** to be installed on your system.

All the charts have some optional arguments in common

- size
  - default 700x700
- height
  - height of the chart, defaults to 700
- width
  - width of the chart, defaults to 700
- class
  - add to the class that the chart generates
  - initial class is named after the chart type (timeline, barchart, sankey etc)

20.1 Timeline chart

Timeline charts are useful for project management, as an alternative to gantt charts

The optional arguments are

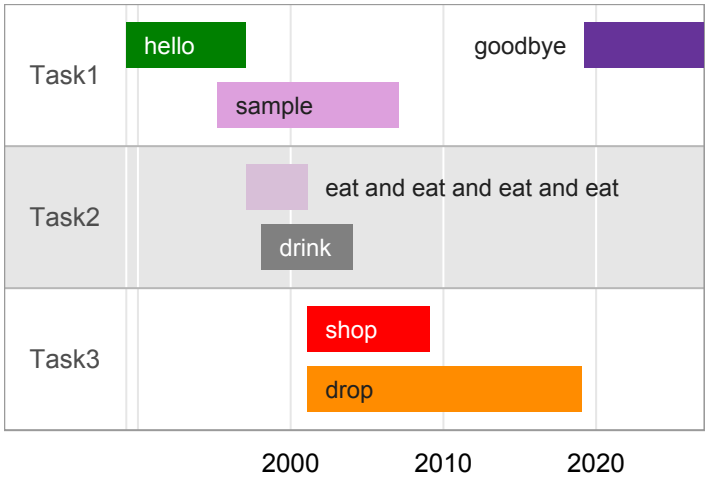
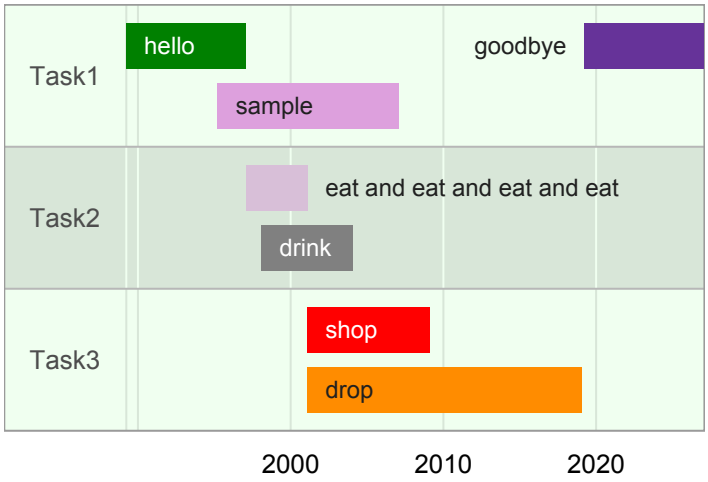
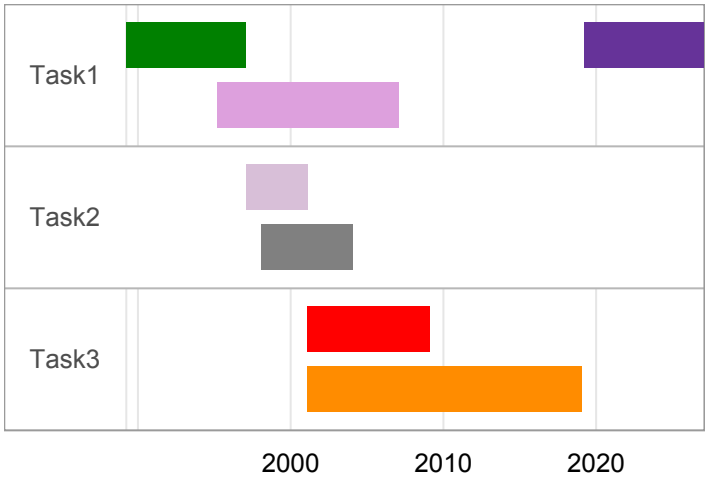
- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, widthxheight
- width
  - just constrain the width
- height
  - just constrain the height
- class
  - add this class to the image

The timeline content consists of rows of lines with a task name, a task item, start and end times. Optionally a #color note at the end of the line (HTML color name or triplet) will set the color of the bar.

Dates should be in yyyy-mm-dd format.

We will set some data into a buffer for ease of use

```
~~~~{.buffer to_buffer=timeline}
Task1, hello,   1989-03-29, 1997-02-03 #green
Task1, sample,   1995-03-29, 2007-02-03   #663399
Task1, goodbye,   2019-03-29, 2027-02-03 #plum
Task2, eat and eat and eat and eat,  1997-02-03,  2001-02-03 #thistle
Task2, drink,  1998-02-03,  2004-02-03 #grey
Task3, shop,  2001-02-03,  2009-02-03 #red
Task3, drop,  2001-02-03,  2019-02-03 #darkorange
~~~~
```

Example	Output
Default ~~~~{.timeline from_buffer=timeline size=350x300} ~~~~	
Setting a background color Use the 'background' parameter to set a color, this can be a HTML color name or hex triplet, 'none' or 'transparent' will remove the background. ~~~~{.timeline from_buffer=timeline background='GhostWhite' size=350x300} ~~~~	
Removing bar labels This is useful if you only really want to show an approximation of how long things will take without specifying which tasks are which. Set the 'labels' parameter to 'false' or 0. ~~~~{.timeline from_buffer=timeline background='none' labels=false size=350x300} ~~~~	

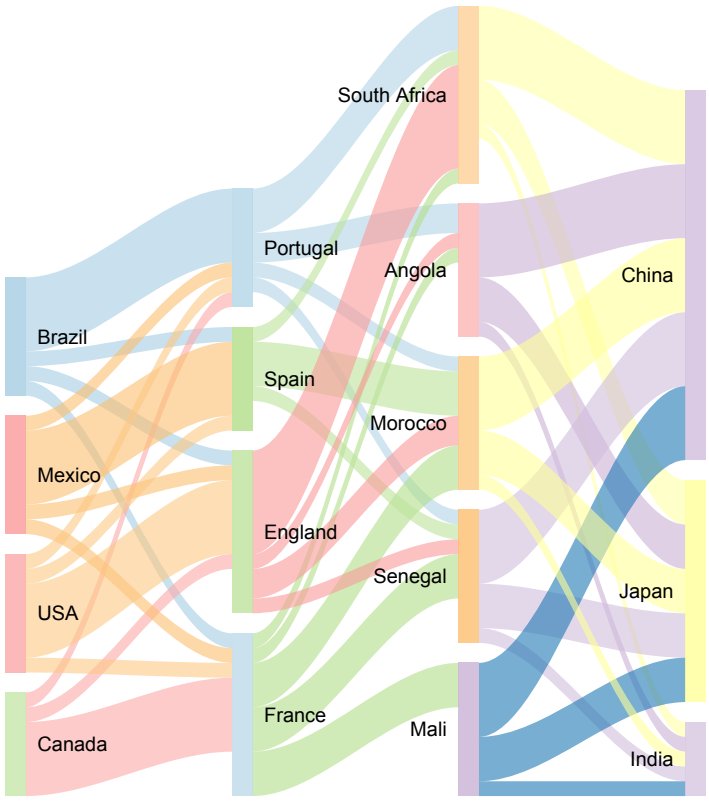
20.2 Sankey Chart

From google charts: A sankey diagram is a visualization used to depict a flow from one set of values to another. The things being connected are called nodes and the connections are called links. Sankeys are best used when you want to show a many-to-many mapping between two domains (e.g., universities and majors) or multiple paths through a set of stages (for instance, Google Analytics uses sankeys to show how traffic flows from pages to other pages on your web site).

For the curious, they're named after Captain Sankey, who created a diagram of steam engine efficiency that used arrows having widths proportional to heat loss.

The optional arguments are

- colors
  - comma separated list of HTML color names (or triplets), to be used for the nodes and links
- mode
  - how the link color is chosen
  - source - link color is the same as the source node
  - target - link color is the same as the target node
  - gradient - link color starts as node color and ends as target color

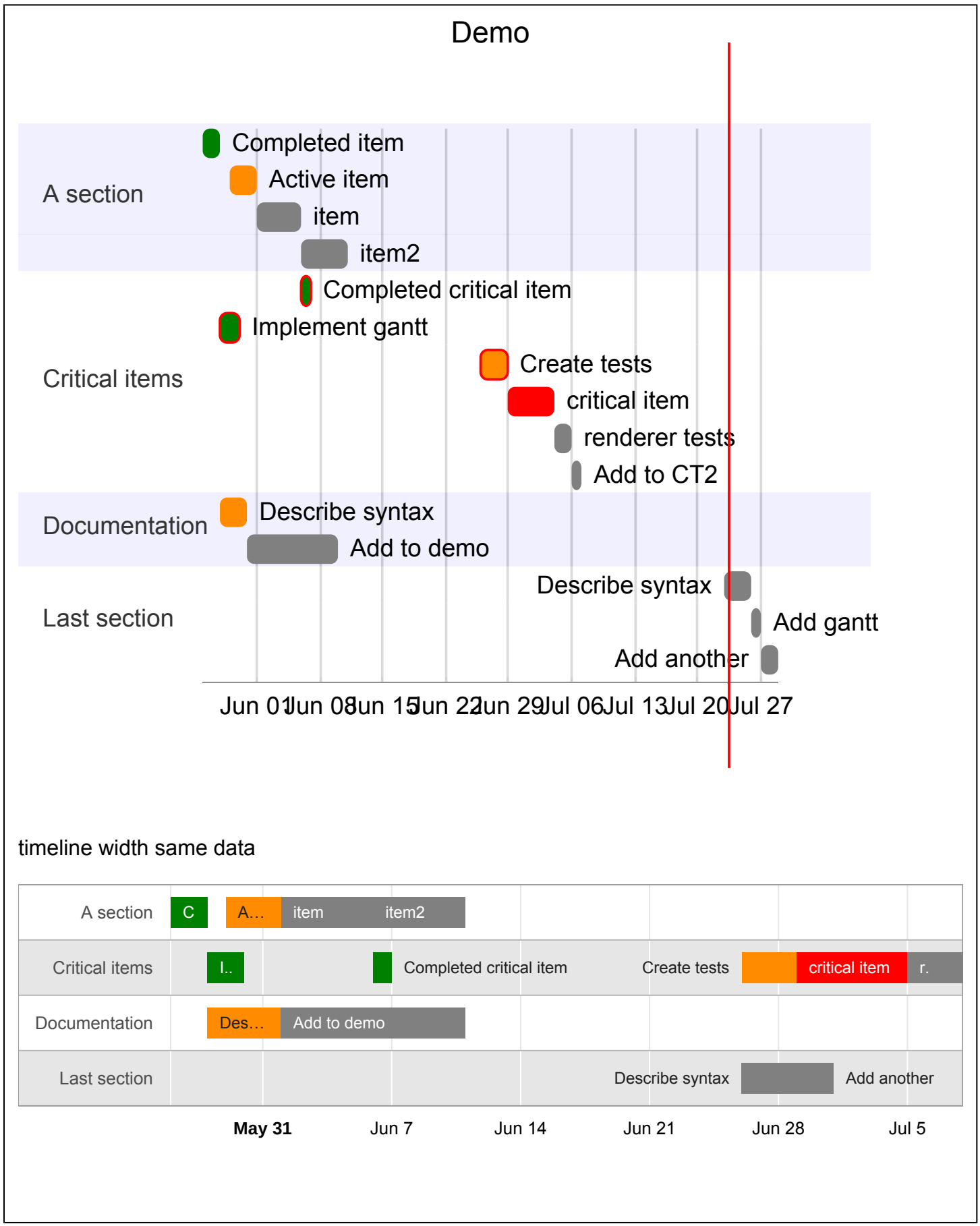
Example	Output
A Simple set ~~~~{.sankey size=350x200 mode=target} A, X, 5 A, Y, 7 A, Z, 6 B, X, 2 B, Y, 9 B, Z, 4 ~~~~	
A Complex set - only a small portion of the data is being shown ~~~~{.sankey size=350x400 mode='source' colors="#a6cee3, #b2df8a, #fb9a99,#fdbf6f, #cab2d6, #ffff99 #1f78b4"}} Brazil, Portugal, 5 , Brazil, France, 1 , Brazil, Spain, 1 , Brazil, England, 1 , Canada, Portugal, 1 , Canada, France, 5 , Canada, England, 1 , Mexico, Portugal, 1 , Mexico, England, 1 , USA, Portugal, 1 , Portugal, Angola, 2 , ... ~~~~	

21 Gantt

Example

```
~~~~{.gantt title=Demo}
section A section
Completed item           :done,    des1, 2015-05-26,2015-05-28
Active item              :active,  des2, 2015-05-29, 3d
item                     :         des3, after des2, 5d
item2                    :         des4, after des3, 5d

section Critical items
Completed critical item  :crit, done, 2015-06-06,24h
Implement gantt          :crit, done, after des1, 2d
Create tests             :crit, active, 2015-06-26, 3d
critical item            :crit, 5d
renderer tests           : 2d
Add to CT2               : 1d

section Documentation
Describe syntax          :active, a1, after des1, 3d
Add to demo              : after a1  , 10d

section Last section
Describe  syntax         : after doc1, 3d
Add gantt                : 1d
Add another              : 2d
~~~~
```

Output



timeline width same data

22 Smilies

THERE ARE ISSUES WITH THE SMILIES AT THE MOMENT!

Conversion of some smilies to unicode characters. This is tricky to show as however I change things the processor will make smilies of these ? ♥ ? .

Just try some of your favourite smilies and see what comes out!

There are a range of smilies that are words pre/post fixed with a colon

smilie	symbol
♥	heart
?	smile
?	grin
?	cool
?	tongue
☹	cry
☹	sad
?	wink
?	halo
?	devil horns
©	c, copyright
®	r, registered
™	tm, trademark
?	email
✔	tick
✘	cross
?	beer
?	wine wine_glass
?	cake
?	star
?	ok, thumbsup
?	bad, thumbsdown
?	ghost
?	skull
?	hourglass
?	watch face
?	sleep

23 Using ct2 script to process files

Included in the distribution is a script to make use of all of the above code-blocks to alter **markdown** into nicely formatted documents.

Here is the help

```
$ ct2 --help

Syntax: ct2 [options] filename

About:  Convert my modified markdown text files into other formats, by
    default will create HTML in same directory as the input file, will only
    process .md files.
    If there is no output option used the output will be to file of same
    name
    as the input filename but with an extension (if provided) from the
    document, use format: keyword (pdf html doc).

[options]
    -h, -?, --help        Show help
    -c, --clean           Clean up the cache before use
    -e, --embed           Embed images into HTML, do not use this if
    converting to doc/odt
    -o, --output          Filename to store the output as, extension will
    control conversion
    -p, --prince          Convert to PDF using princexml, can handle
    embedded images
    --templates           list available templates
    -t, --template        name of template to use
    -v, --verbose         verbose mode
    -w, --wkhtmltopdf     Convert to PDF using wkhtmltopdf, can handle
    embedded images
```

On the first time you run **ct2** a default template will be created in **~/.ct2/templates/default/template.html**, a config file to accompany this will be created in *~/.ct2/templates/default/template.html**

Create new templates in *~/.ct2/templates*, one directory for each template, follow the example in the default directory. If you are creating HTML documents to send out in emails or share in other ways, and use locally referenced images, then it is best to make use of the **–embed** option to pack these images into the HTML file.

If you are using **PrinceXML** remember that it is only free for non-commercial use, it also adds a purple **P** to the top right of the first page of your document, though this does not appear when you print out the document.