Title	App::Basis::ConvertText2
Author	Kevin Mulholland
Last Updated	2015-02-06
Version	6

This document may not be easily readable in this form, try **pdf** or **HTML** as alternatives. These 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!

1 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

```
%TITLE%
```

Output

App::Basis::ConvertText2

2 Table of contents

As documents are processed, the HTML headers (H2..H3) are collected together to make a table of contents. This can be used either in your template or document using the TOC variable.

Example

```
%TOC% will show
```

Contents

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.

2.1 Skipping header {.toc_skip}

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

Example

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

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

3 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.

3.1 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'}}
```

4 Buffering data for later use

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

```
* %SPARK_DATA%
* %GREENSPARK%
```

Output

- 1,4,5,20,4,5,3,1
- 

5 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 full set of optional arguments is

- 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, only things in red blue green orange mono are valid
- size
  - size of image, default 80x20, widthxheight

Example

```
~~~~{.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
~~~~

or we can draw the sparkline using buffered data

~~~~{.sparkline title='blue sparkline' scheme='blue' from_buffer='spark_data'}
~~~~
```

Output

here is a standard sparkline

or we can draw the sparkline using buffered data

6 Charts

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

The various arguments to the code-block are shown in the examples below, hopefully they are self explanatory.

We will buffer some data to start

Example

```
~~~~{.buffer to='chart_data'}
apples,bananas,cake,cabbage,edam,fromage,tomatoes,chips
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
~~~~
```

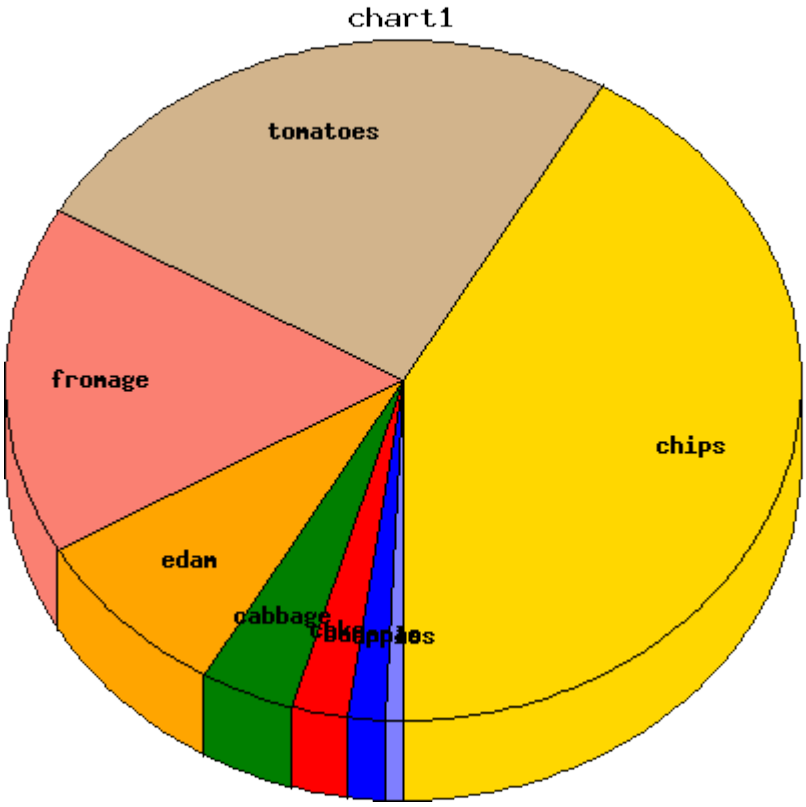
The content comprises a number of lines of comma separated data items. The first line of the content is the legends, the subsequent lines are numbers relating to each of these legends.

6.1 Pie chart

Example

```
~~~~{.chart format='pie' title='chart1' from_buffer='chart_data'
    size='400x400' xaxis='things xways' yaxis='Vertical things'
    legends='a,b,c,d,e,f,g,h' }
~~~~
```

Output

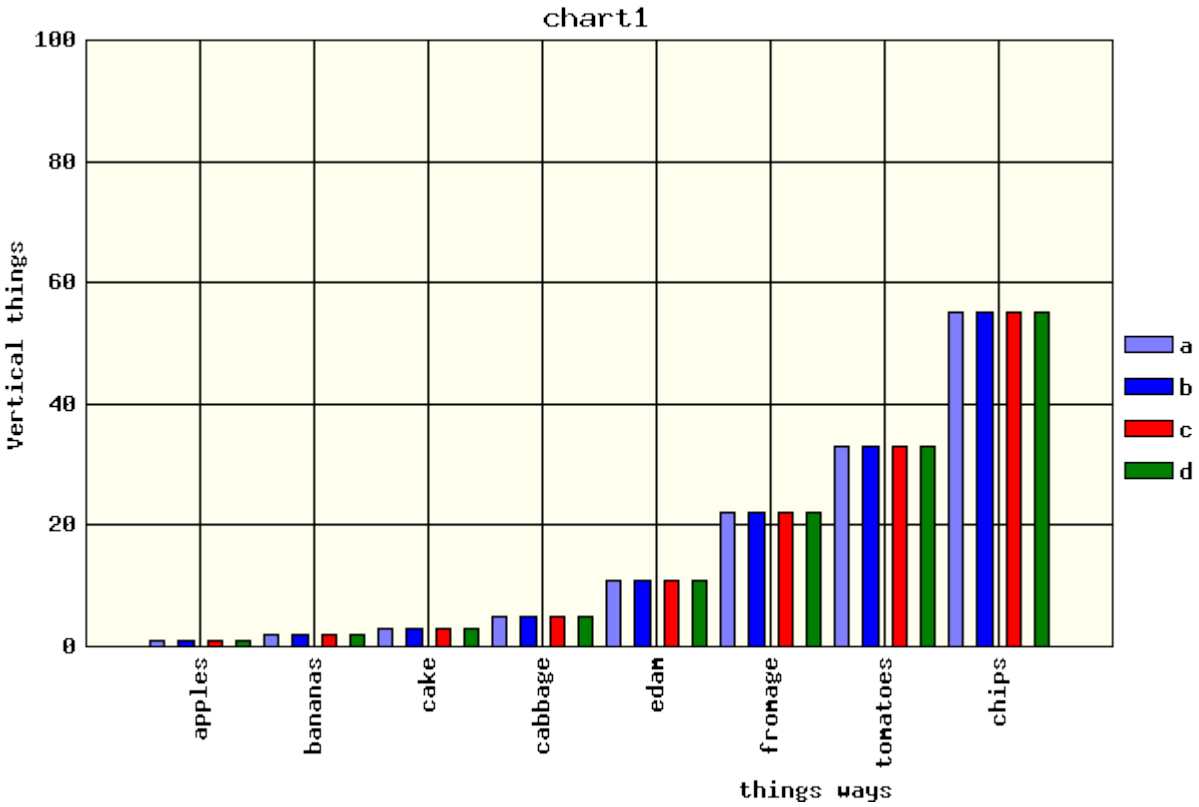


6.2 Bar chart

Example

```
~~~~{.chart format='bars' title='chart1' from_buffer='chart_data'
    size='600x400' xaxis='things ways' yaxis='Vertical things'
    legends='a,b,c,d,e,f,g,h' }
~~~~
```

Output

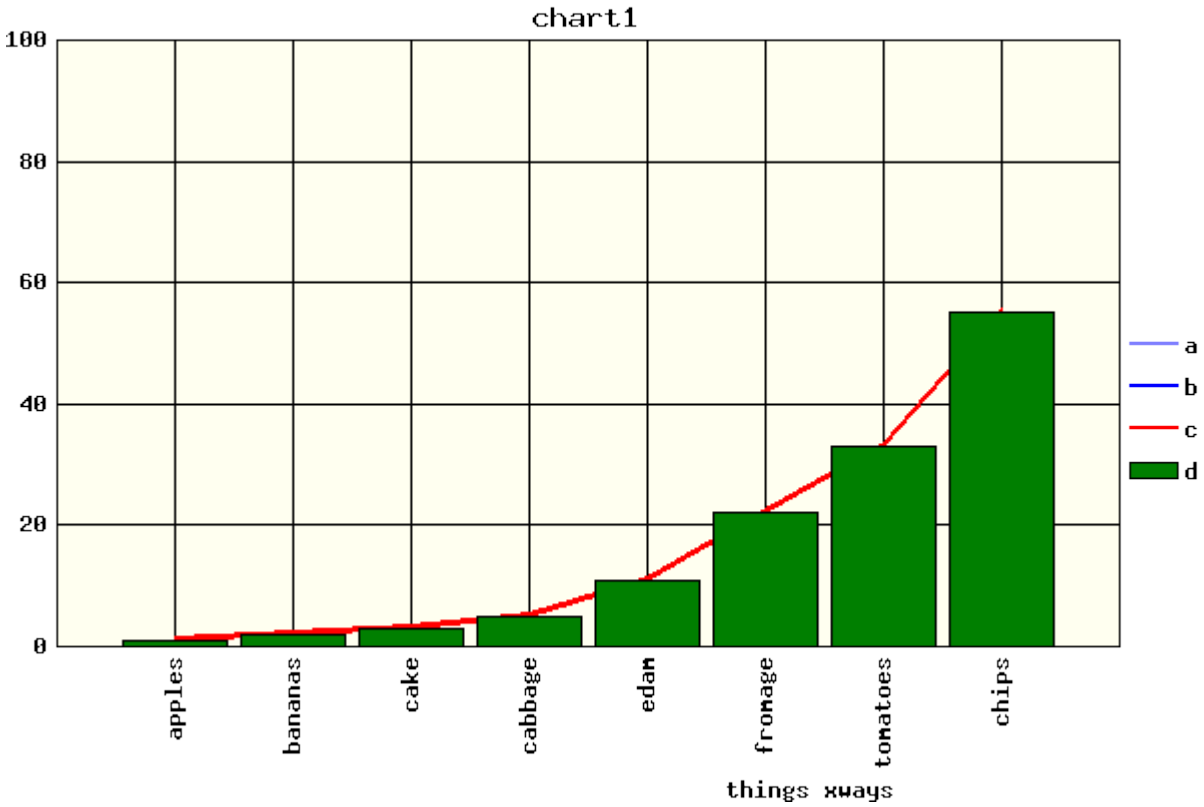


6.3 Mixed chart

Example

```
~~~~{.chart format='mixed' title='chart1' from_buffer='chart_data'
  size='600x400' xaxis='things xways' axis='Vertical things'
  legends='a,b,c,d,e,f,g,h' types='lines linepoints lines bars' }
~~~~
```

Output



7 Message Sequence Charts - mscgen

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**

There are only optional 2 arguments

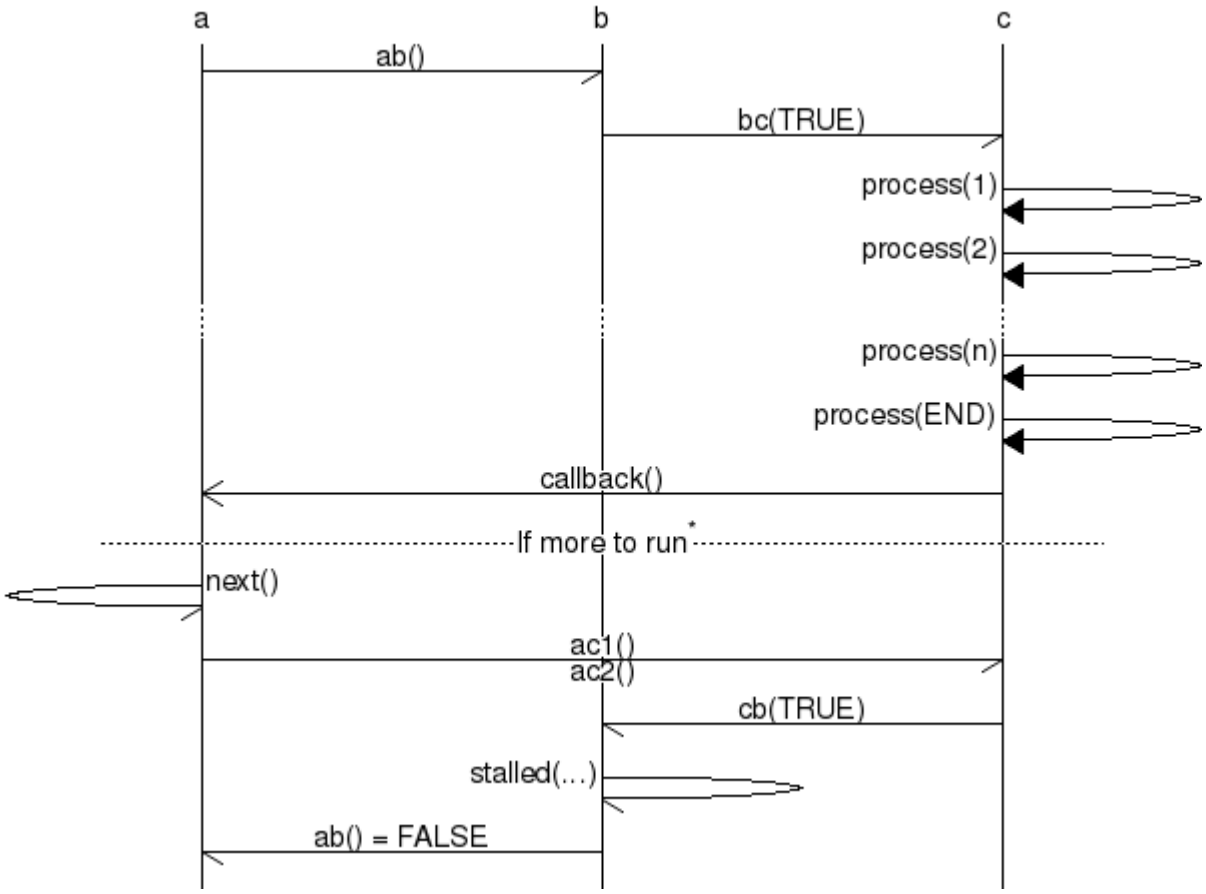
- title
  - used as the generated images 'alt' argument
- size
  - size of image, widthxheight

Example

```
~~~~{.mscgen  title="mscgen1" size="600x400}
# 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", ID="*" ];
  a->a [ label = "next()"];
  a->c [ label = "ac1()\nac2()"];
  b<-c [ label = "cb(TRUE)"];
  b->b [ label = "stalled(...)"];
  a<-b [ label = "ab() = FALSE"];
}
~~~~
```

Output



8 Diagrams Through Ascii Art - ditaa

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. See **ditaa** for reference.

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

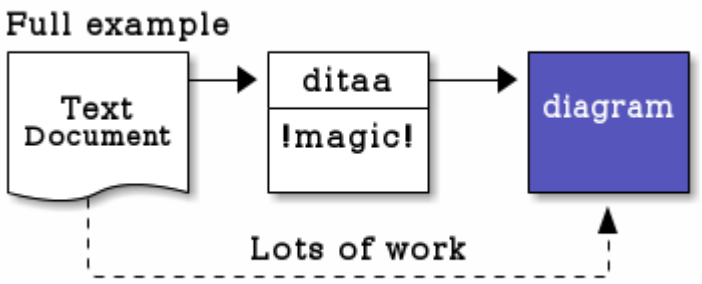
- title
  - used as the generated images 'alt' argument
- size
  - size of image, default 80x20, widthxheight
  - shadow - have shadow, default true

- alias - apply aliasing, default true
- round - round edges, default false
- separation - default false

Example

```
~~~~{.ditaa }
Full example
+--------+   +-------+    +-------+
|        +-->| ditaa +--->|       |
|  Text  |   +-------+    |diagram|
|Document|   |!magic!|    |       |
|     {d}|   |       |    |cBLU   |
+---+----+   +-------+    +-------+
    :                         ^
    |       Lots of work      |
    \-------------------------+
~~~~
```

Output



9 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 component.

Here is mine, it is also available in the scripts directory in the

```
#!/bin/bash
# run plantuml
# moodfarm@cpan.org

# we assume that the plantuml.jar file is in the same directory as this executa
EXEC_DIR=`dirname $0`
PLANTUML="$EXEC_DIR/plantuml.jar"
```

```
INPUT=$1
OUPUT=$2
function show_usage  {
    arg=$1
    err=$2
    if [ "$err" == "" ] ; then
        err=1
    fi
      "Create a UML diagram from an input text file
(see http://plantuml.sourceforge.net/ for reference)
    usage: $0 inputfile outputfile.png
"
    if [ "$arg" != "" ] ; then
        echo "$arg
"
    fi
    exit $err
}
if [ "$INPUT" == "-help" ] ; then
    show_usage "" 0
fi
if [ ! -f "$INPUT" ] ; then
    show_usage "ERROR: Could not find input file $1"
fi
if [ "$OUPUT" == "" ] ; then
    show_usage "ERROR: No output file specified"
fi
# we use the pipe option to control output into the file we want
cat "$INPUT" | java -jar $PLANTUML -nbthread auto -pipe >$OUPUT
# exit 0
```

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

The arguments allowed are

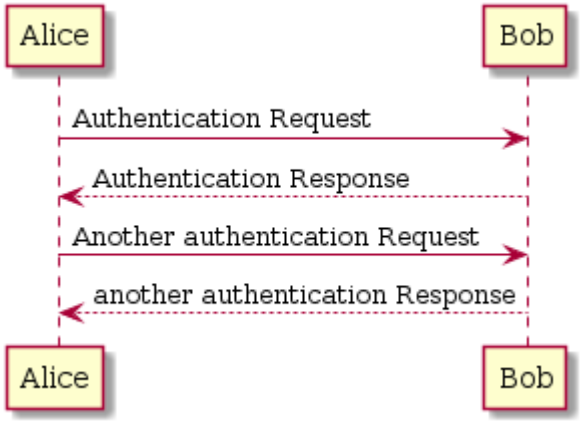
- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, default 80x20, widthxheight

Example

```
~~~~{.uml }
' this is a comment on one line
/' this is a
multi-line
comment'/
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
```

```
Alice -> Bob: Another authentication Request
Alice <-- Bob: another authentication Response
~~~~
```

Output



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

Example

```
~~~~{.uml }
@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
~~~~
```

Output

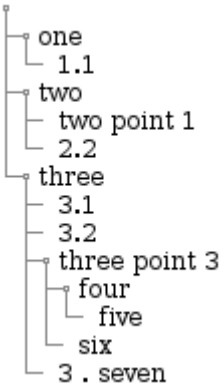
10 Umltree

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

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

Output



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 arguments allowed are

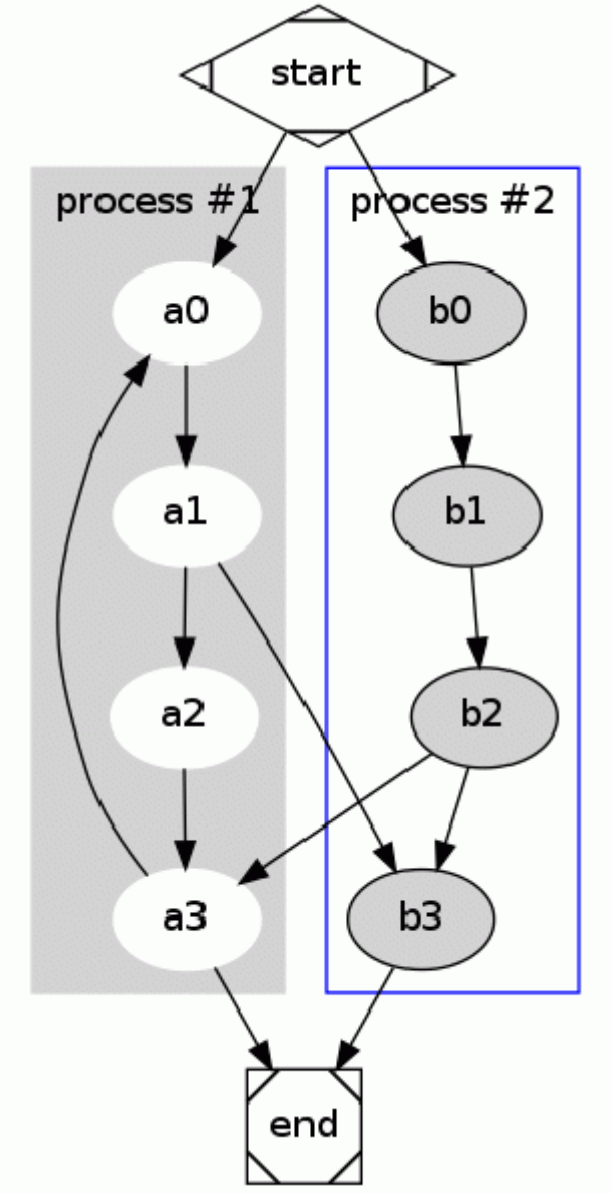
- title
  - used as the generated images 'alt' argument
- size
  - size of image, default 80x20, widthxheight

Example

```
~~~~{.graphviz  title="graphviz1" size='600x600'}
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];
}
~~~~
```

Output



12 Venn diagram

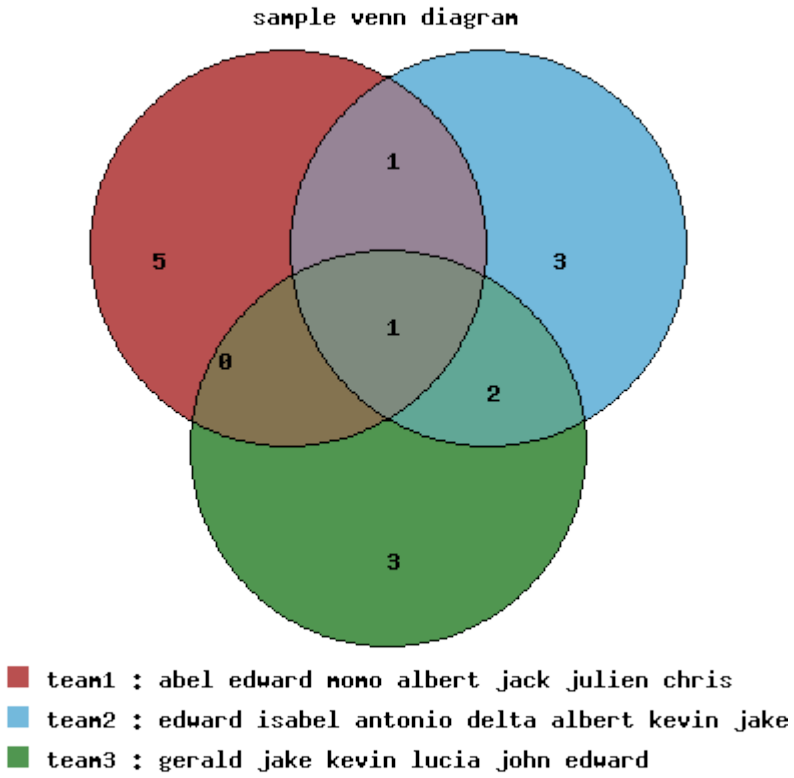
Creating venn diagrams may sometimes be useful, though to be honest this implementation is not great, if I could find a better way to do this then I would!

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

13.1 Code39

Example

```
{{.barcode type='code39' content=123456789}}
```

Output

13.2 EAN8

Only allows 8 characters

Example

```
~~~~{.barcode type='ean8'}
12345678
~~~~
```

Output

13.3 EAN13

Only allows 13 characters

Example

```
~~~~{.barcode type='EAN13'}
1234567890123
~~~~
```

Output

13.4 COOP2of5

Example

```
~~~~{.barcode type='COOP2of5'}
12345678
~~~~
```

Output

13.5 IATA2of5

Example

```
~~~~{.barcode type='IATA2of5'}
12345678
~~~~
```

Output

13.6 Industrial2of5

Example

```
~~~~{.barcode type='Industrial2of5'}
12345678
~~~~
```

Output

13.7 ITF

Example

```
~~~~{.barcode type='ITF'}
12345678
~~~~
```

Output

13.8 Matrix2of5

Example

```
~~~~{.barcode type='Matrix2of5'}
12345678
~~~~
```

Output

13.9 NW7

Example

```
~~~~{.barcode type='NW7'}
12345678
~~~~
```

Output

13.10 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

```
~~~~{.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..
~~~~
```

Output

To change the size of the barcode

To use version 1

Version 1 only allows 15 characters

To change pixel size

14 YAML convert to JSON

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

```
~~~~{.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

~~~~
```

Output

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

15 YAML convert to XML

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

```
~~~~{.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

~~~~
```

Output

```
  <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>
```

16 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
  - csv of 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
    from_buffer='chart_data'}
~~~~
```

Output

apples	bananas	cake	cabbage	edam	fromage	tomatoes	chips
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

17 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

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
See 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/
~~~~
```

Output

- **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/
- **pandoc**
  - http://johnmacfarlane.net/pandoc
- **PlantUML**
  - http://plantuml.sourceforge.net
- **PrinceXML**
  - http://www.princexml.com
- **See Salt**
  - http://plantuml.sourceforge.net/salt.html
- **wkhtmltopdf**
  - http://wkhtmltopdf.org/
- **YAML**
  - https://en.wikipedia.org/wiki/Yaml

18 Version table

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

Example

```
~~~~{.version class='versiontable' width='100%'}
0.1 2014-04-12
    * removed ConvertFile.pm
```

```
    * using Path::Tiny rather than other things
    * changed to use pandoc fences
   ~~~~{.tag} rather than xml format <tag>
0.006 2014-04-10
    * first release to github
~~~~
```

Output

19 Document Revision History

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

20 Start a new page - 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

I will not show the output as it will mess up the document!

21 Columns

Create a columner layout, like a newspaper.

The full set of optional arguments is

- 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='75%'}
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.  Slow-carb Tumblr
actually fashion axe, kitsch Williamsburg Austin bicycle rights forage
Carles occupy.  Aesthetic High Life cray seitan.  Mumblecore butcher
biodiesel mixtape Bushwick fanny pack.  Tofu twee typewriter Truffaut.

Leggings church-key ethical banjo twee.  Jean shorts messenger bag vinyl,
pork belly blog aesthetic Pinterest ennui mustache lo-fi hella.  Yr blog
hoodie, iPhone whatever twee deep v sriracha polaroid occupy pickled food
truck.  Letterpress Austin kale chips pop-up mixtape vinyl.  Drinking
vinegar slow-carb mlkshk chia sriracha, shabby chic pour-over.  Mlkshk
brunch bespoke Kickstarter fingerstache deep v.  Vegan letterpress
sustainable, squid quinoa organic asymmetrical XOXO.
~~~~
```

Output

Before this section I forced a new page with

```
---
```

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. Slow-carb Tumblr actually fashion axe, kitsch Williamsburg Austin bicycle rights forage Carles occupy. Aesthetic High Life cray seitan. Mumblecore butcher biodiesel mixtape Bushwick fanny pack. Tofu twee typewriter Truffaut

Leggings church-key ethical banjo twee. Jean shorts messenger bag vinyl, pork belly blog aesthetic Pinterest ennui mustache lo-fi hella. Yr blog hoodie, iPhone whatever twee deep v sriracha polaroid occupy pickled food truck. Letterpress Austin kale chips pop-up mixtape vinyl. Drinking vinegar slow-carb mlkshk chia sriracha, shabby chic pour-over. Mlkshk brunch bespoke Kickstarter fingerstache deep v. Vegan letterpress sustainable, squid quinoa organic asymmetrical XOXO.

22 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

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

```
    * 3 . seven
~~~~
```

Output

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

23 Gle / glx

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

The full set of optional arguments is

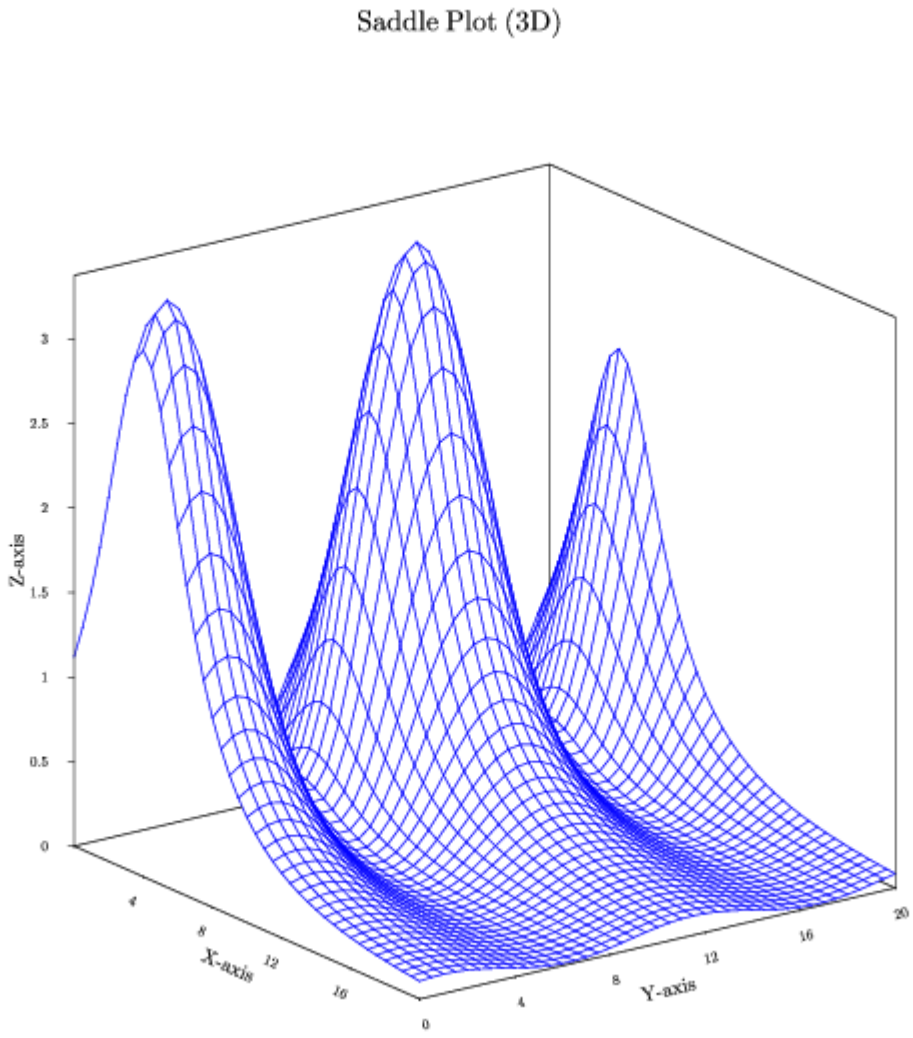
- title
  - used as the generated images ‘alt’ argument
- size
  - size of image, default 720x512, widthxheight, size is approximate
- transparent
  - flag to use a transparent background

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



24 Gnuplot

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

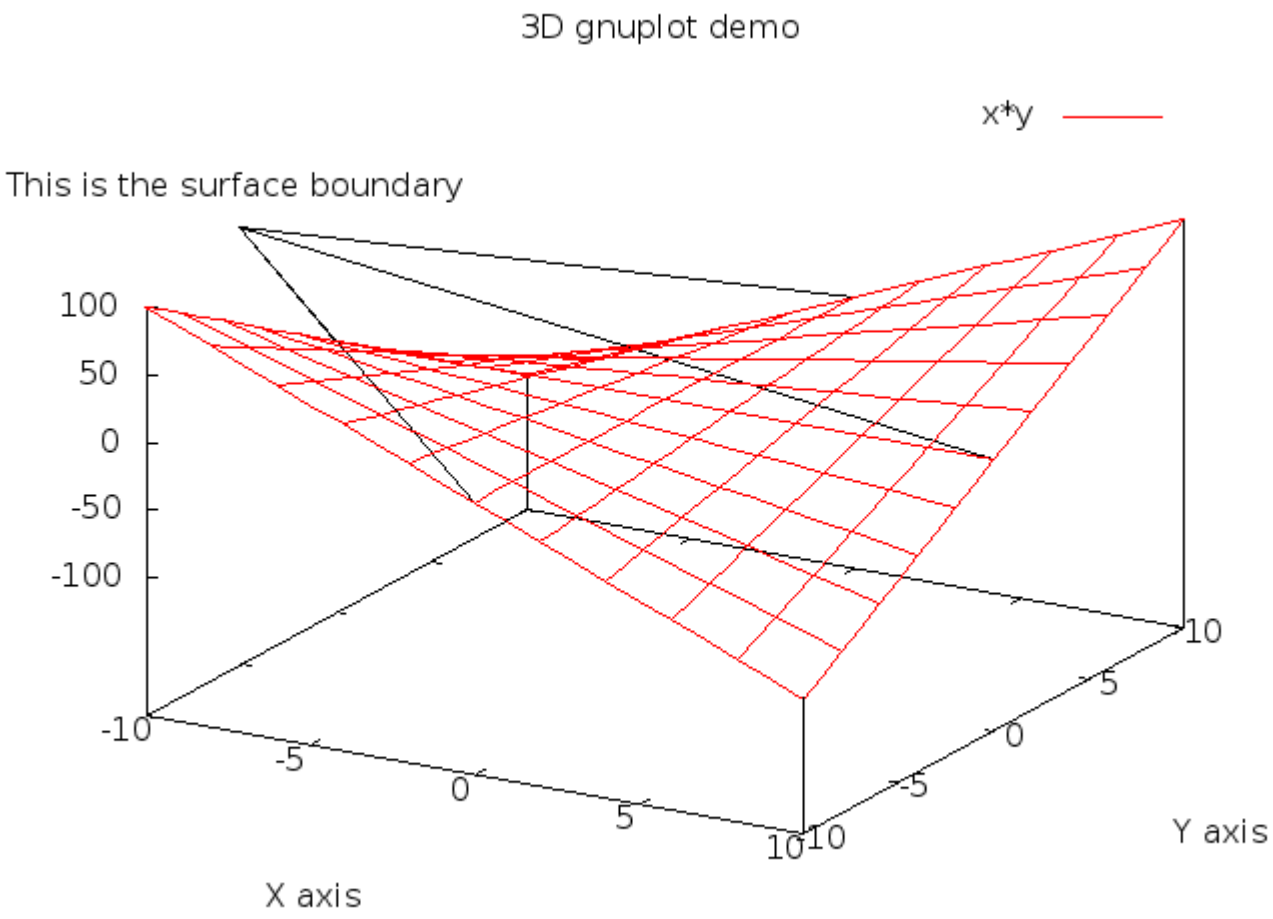
The full set of optional arguments is

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

Example

```
~~~~{.gnuplot}
# $Id: surface1.dem,v 1.11 2004/09/17 05:01:12 sfeam Exp $
#
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 "This is the 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



25 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 full set of optional arguments is

- title
  - used as the generated images ‘alt’ argument

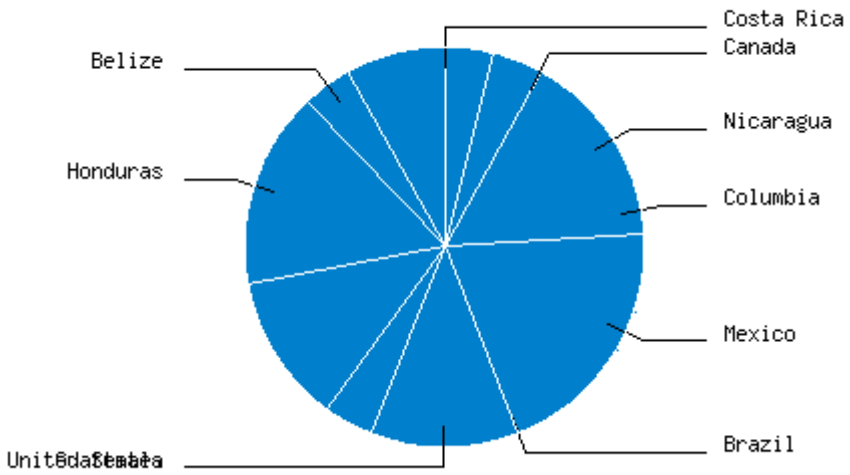
Its best to let ploticus control the size of the generated images, you may need some trial and error with the ploticus ‘pagesize:’ directive.

Example

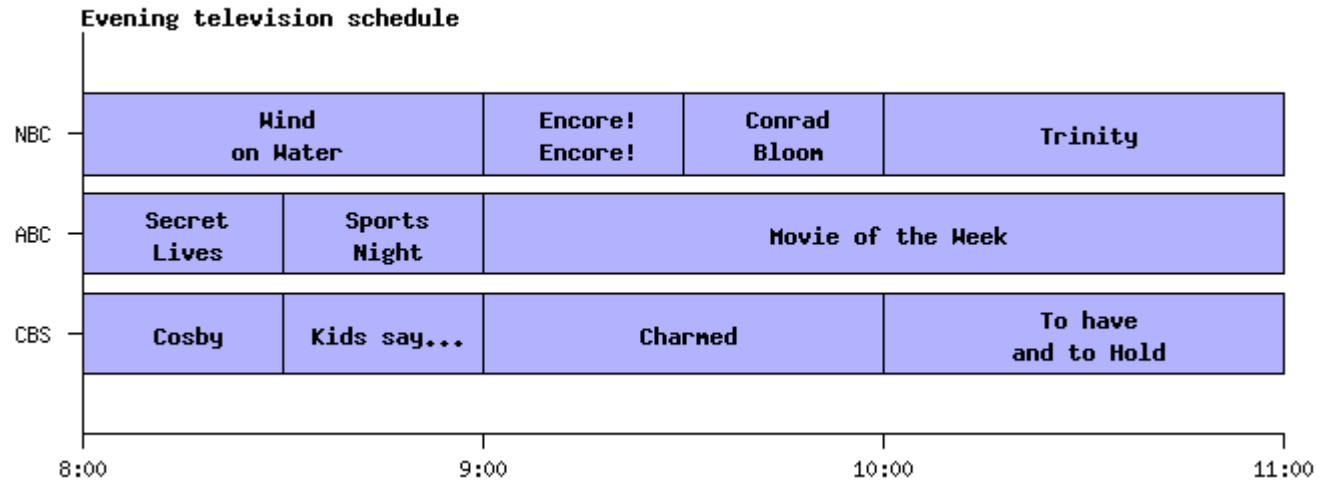
```
~~~~{.ploticus}
//  specify data using proc getdata
#proc getdata
data: Brazil 22
  Columbia 17
  "Costa Rica" 22
  Guatemala 3
  Honduras 12
```

```
  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)



26 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!

Example

```
First time
~~~~{.badge subject='test run' status='completed' color='green'}
~~~~
 followed by {{.badge subject='test run' status='failed' color='red'}}

Finally, badges / shields work well as short blocks
{{.shield subject='test run' status='pending' size='150'}}
{{.shield subject='build' status='passing' color='#33aa00' size='75'}}
```

Output

First time test run completed

followed by test run failed

Finally, badges / shields work well as short blocks test run pending
build passing

27 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

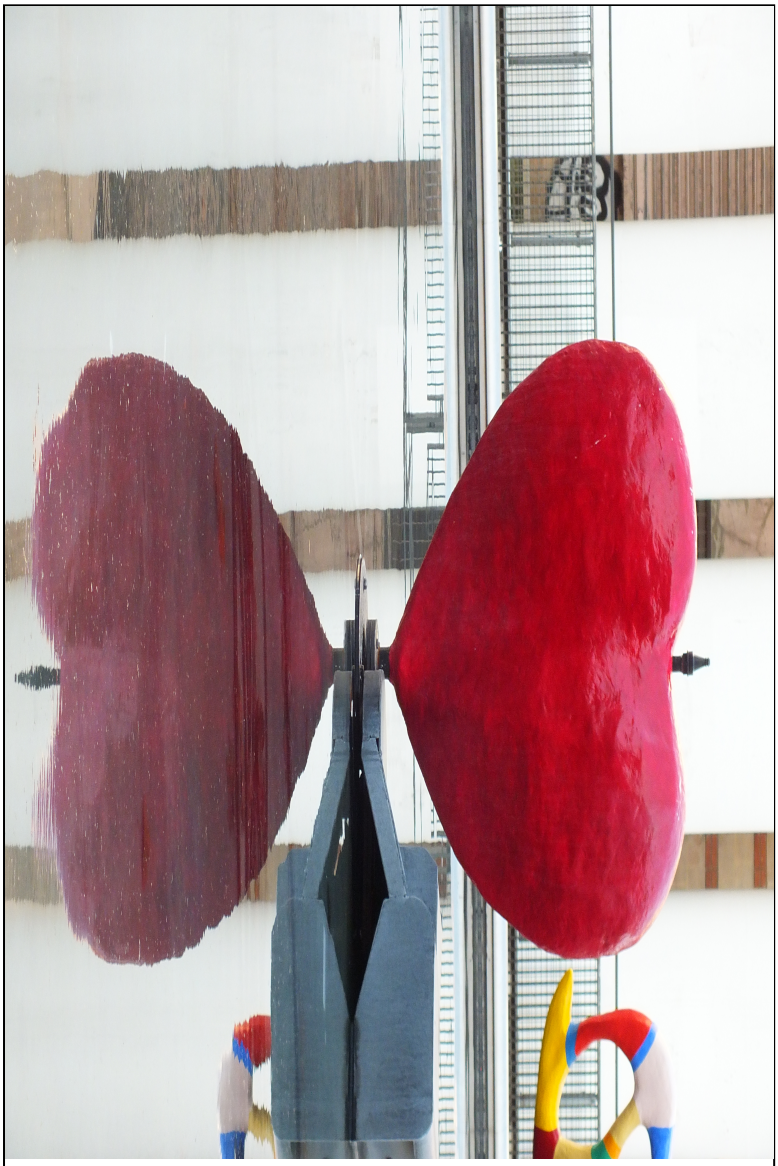
The full set of optional arguments is

- src
  - filename to convert to a polaroid
- title
  - optional title for the photo
- date
  - optional date for the photo

Example

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

Output



The heart of Paris 2015-02-06

28 Box

Show that something is important by putting it in a box

The full set of optional arguments is

- 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

```
~~~~{.box title='Lorem Ipsum - Important Notice' width='80%'}
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque sit amet
~~~~
```

Output

Lorem Ipsum - 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. Aliquam mauris erat, bibendum at mi imperdiet, molestie placerat sem. In fermentum sapien at vulputate mollis. Nulla nec ultrices nulla, ut scelerisque justo. Maecenas a nibh id ligula faucibus fringilla non in nisl.

29 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

30 Smilies

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 |

31 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.

32 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.