

# App::Basis::ConvertText2

## Contents

- [1 Document header and variables](#)
- [2 Table of contents](#)
- [3 Fenced code-blocks](#)
- [4 Buffering data for later use](#)
- [5 Sparklines](#)
- [6 Charts](#)
  - [6.1 Pie chart](#)
  - [6.2 Bar chart](#)
  - [6.3 Mixed chart](#)
- [7 Message Sequence Charts - mscgen](#)
- [8 Diagrams Through Ascii Art - ditaa](#)
- [9 UML Diagrams](#)
- [10 Graphviz](#)
- [11 Venn diagram](#)
- [12 Barcodes](#)
  - [12.1 Code39](#)
  - [12.2 EAN8](#)
  - [12.3 EAN13](#)
  - [12.4 COOP2of5](#)
  - [12.5 IATA2of5](#)
  - [12.6 Industrial2of5](#)
  - [12.7 ITF](#)
  - [12.8 Matrix2of5](#)
  - [12.9 NW7](#)
  - [12.10 QR code](#)
- [13 YAML convert to JSON](#)
- [14 Table](#)
- [15 Links](#)
- [16 Version table](#)
- [17 Start a new page - page](#)
- [18 Gotchas about variables](#)
- [19 HTML templates](#)

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

Markup is my version of [markdown](#) with extra fenced code-blocks to allow the creation of charts and graphs etc. it uses [pandoc](#) to generate documents in a variety of formats and optionally can use [PrinceXML](#) to generate great looking PDFs.

As I do not know Haskell which [pandoc](#) is written in and the way to add filters looks awkward, I decided to create a pre-processing system that will output markdown and HTML to [pandoc](#).

This also allows you to make use of templates.

# 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: 2
```

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, all the HTML headers (H1..H4) 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
```

- [1 Document header and variables](#)
- [2 Table of contents](#)
- [3 Fenced code-blocks](#)
- [4 Buffering data for later use](#)
- [5 Sparklines](#)
- [6 Charts](#)
  - [6.1 Pie chart](#)
  - [6.2 Bar chart](#)
  - [6.3 Mixed chart](#)
- [7 Message Sequence Charts - mscgen](#)
- [8 Diagrams Through Ascii Art - ditaa](#)
- [9 UML Diagrams](#)
- [10 Graphviz](#)
- [11 Venn diagram](#)
- [12 Barcodes](#)
  - [12.1 Code39](#)
  - [12.2 EAN8](#)
  - [12.3 EAN13](#)
  - [12.4 COOP2of5](#)
  - [12.5 IATA2of5](#)
  - [12.6 Industrial2of5](#)
  - [12.7 ITF](#)
  - [12.8 Matrix2of5](#)
  - [12.9 NW7](#)
  - [12.10 QR code](#)
- [13 YAML convert to JSON](#)
- [14 Table](#)
- [15 Links](#)
- [16 Version table](#)
- [17 Start a new page - page](#)
- [18 Gotchas about variables](#)
- [19 HTML templates](#)

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 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.

## 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 of 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
~~~~
```

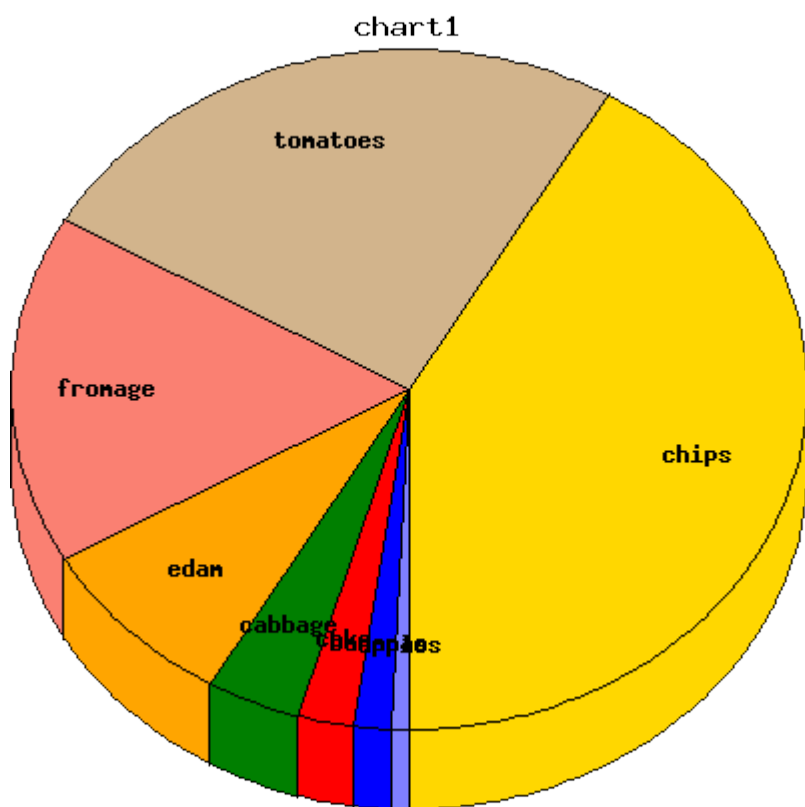
The content comprises a number of lines of comma separated data items. The first line of the content is the legends, the subsequent lines are numbers relating to each of these legends.

### 6.1 Pie chart

#### Example

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

#### Output

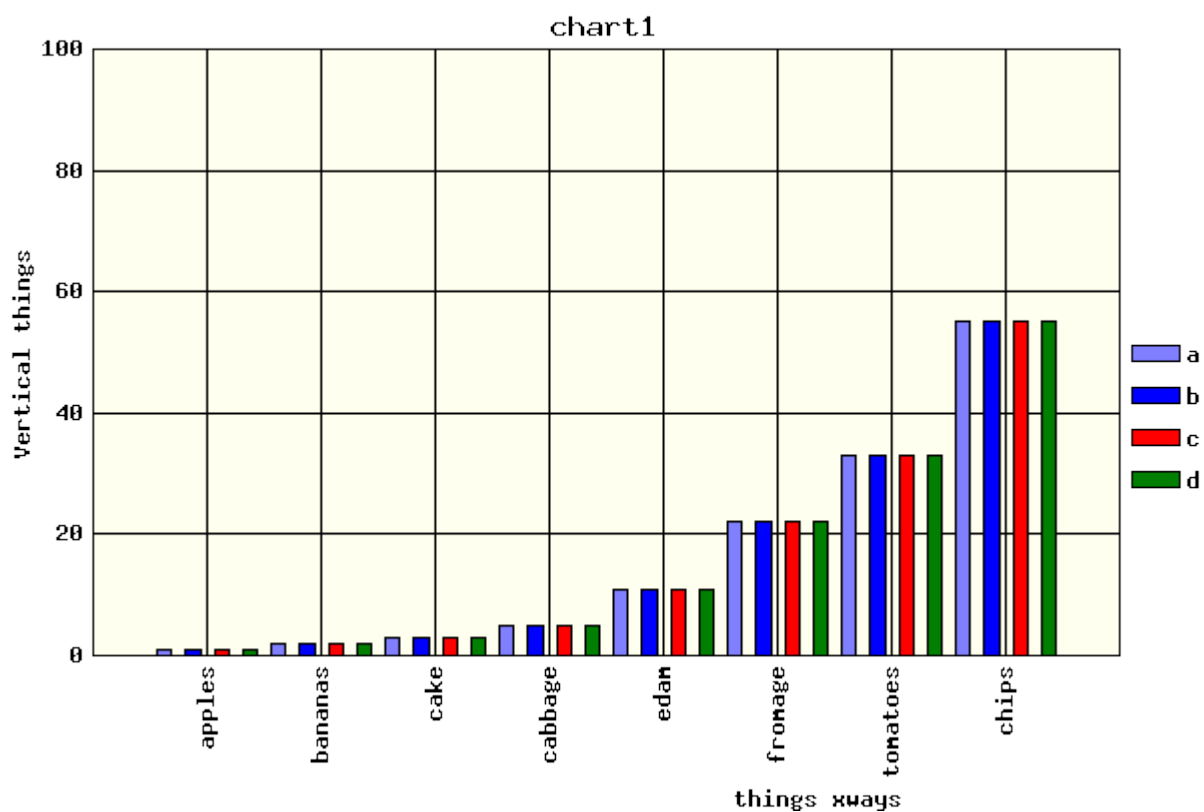


## 6.2 Bar chart

### Example

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

### Output



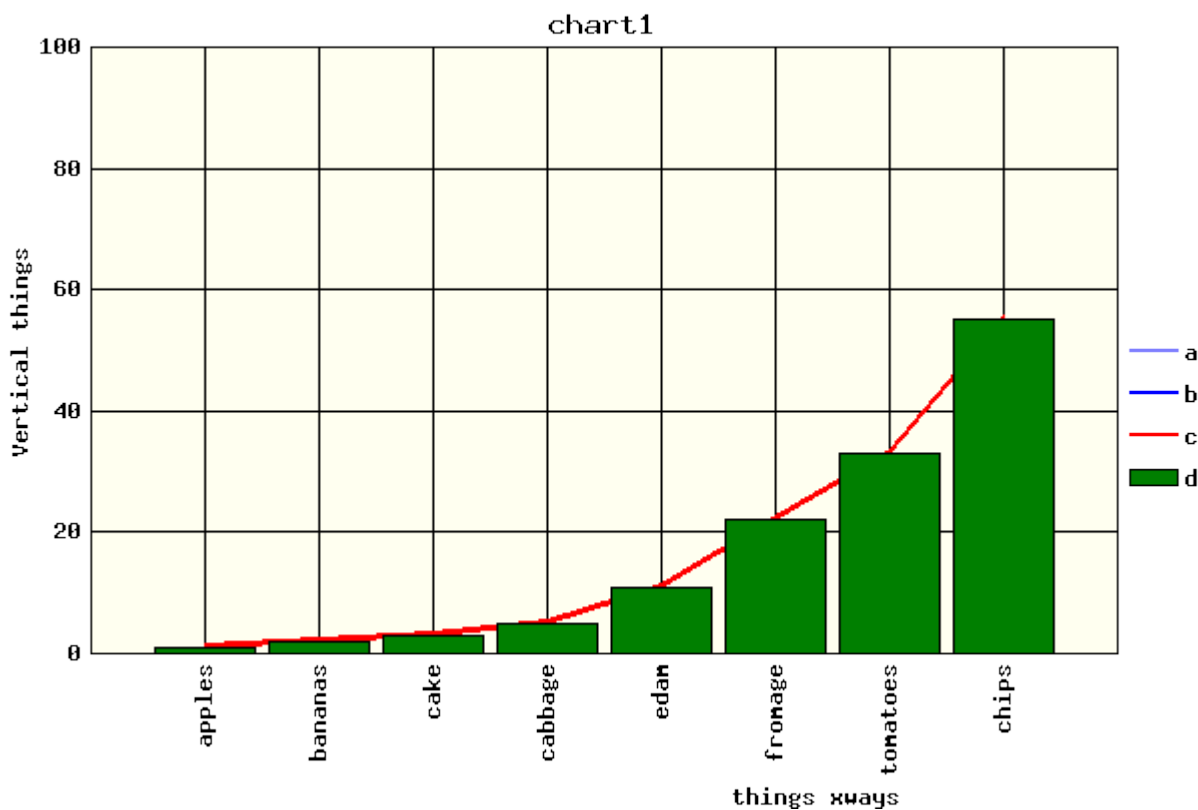
## 6.3 Mixed chart

### Example

```
~~~~{.chart title="chart1" from_buffer='chart_data' size="600x400"
  xaxis='things xways' yaxis='Vertical things' format='mixed'
  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;

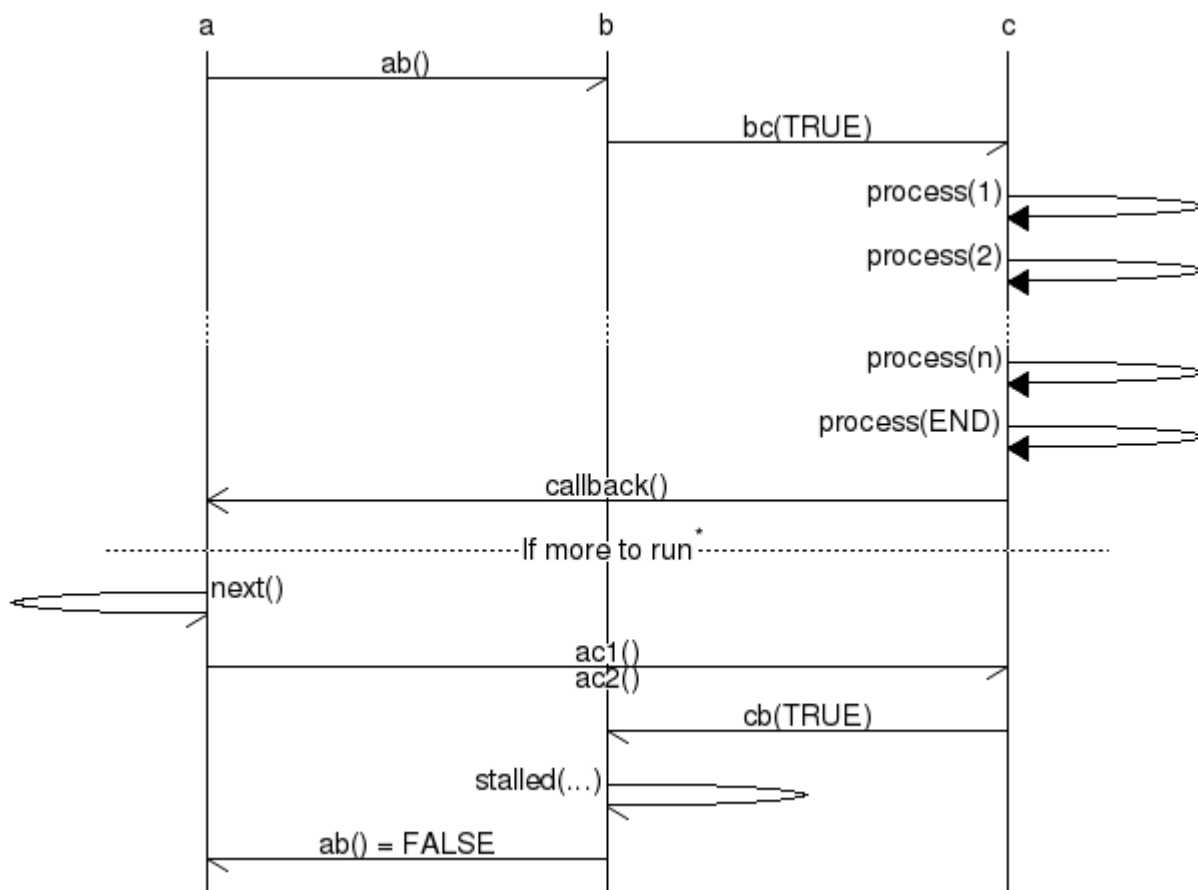
  a->b [ label = "ab()" ] ;
  b->c [ label = "bc(TRUE)" ];
  c=>c [ label = "process(1)" ];
  c=>c [ label = "process(2)" ];
  ...;
}
```

```

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

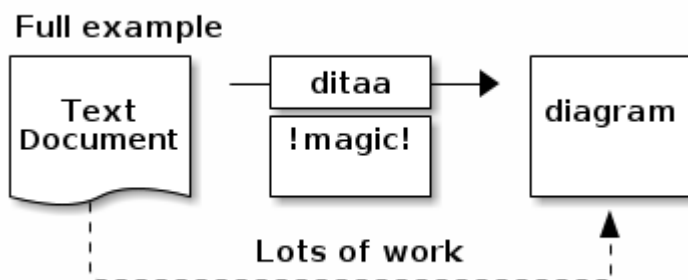
- title
  - used as the generated images 'alt' argument

- size
  - size of image, default 80x20, widthxheight

### Example

```
~~~~{.ditaa }
Full example
+-----+ +-----+ +-----+
|         | --+ ditaa +--> |         | | |
|  Text   | +-----+ |diagram|
|Document| |!magic!| |         |
|    {d}  | |         | |         |
+-----+ +-----+ +-----+
      :                               ^
      |      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 executable
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

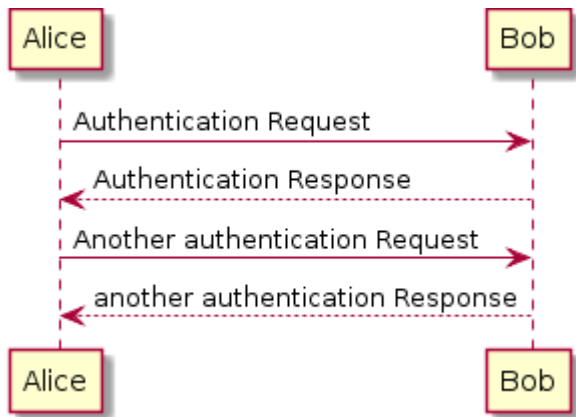
```

~~~~{.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

Just plain text

## 10 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 {

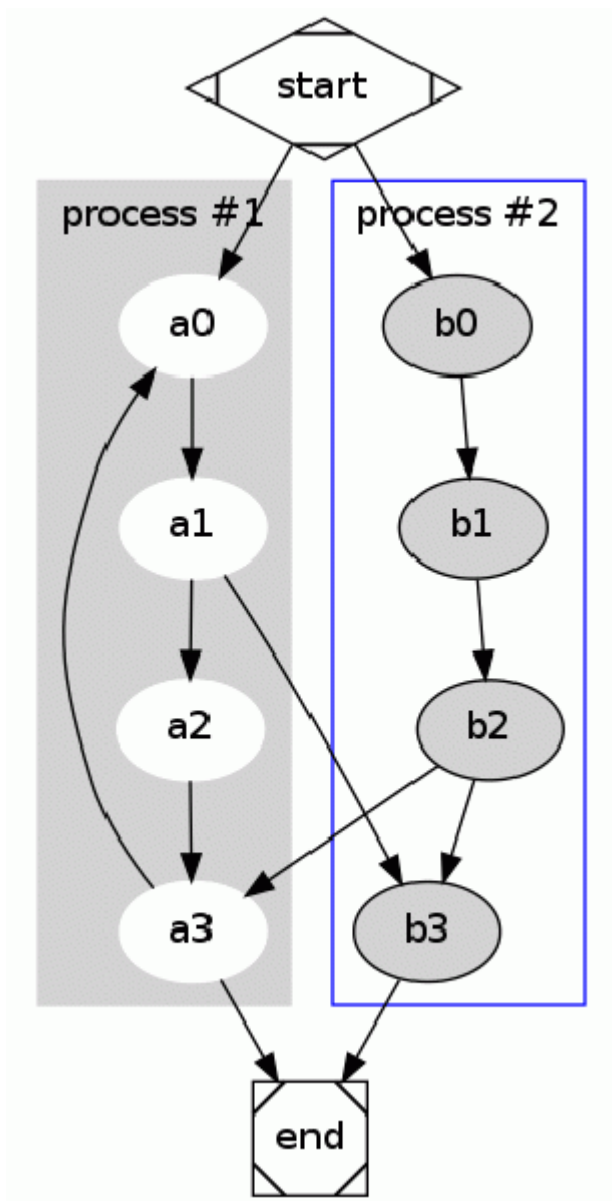
  subgraph cluster_0 {
    style=filled;
    color=lightgrey;
    node [style=filled,color=white];
    a0 -> a1 -> a2 -> a3;
    label = "process #1";
  }
}

```

```
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



## 11 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

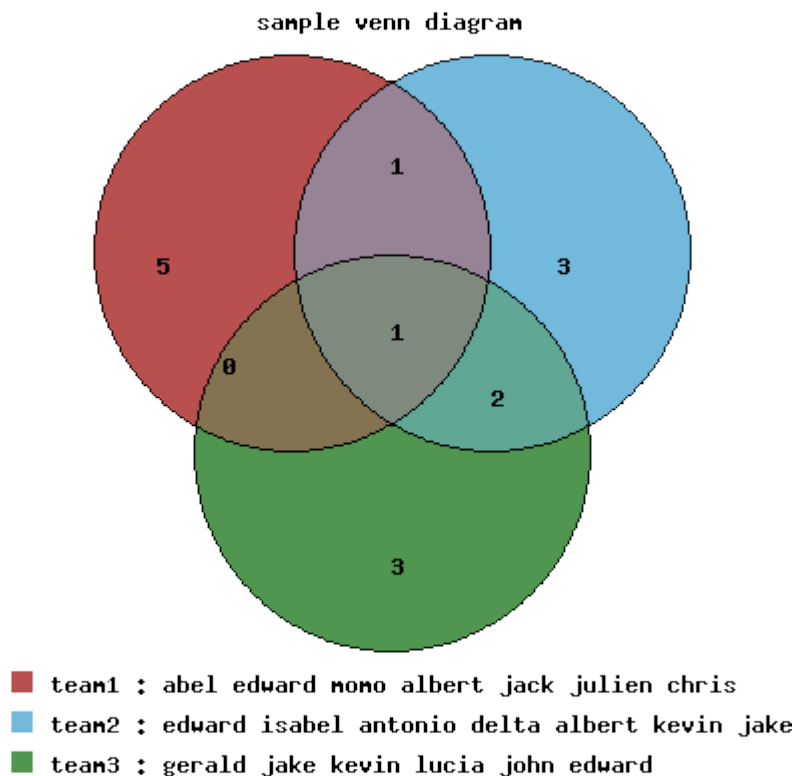
```

~~~~{.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 : chris jack momo julien abel
  - only in team2 : delta antonio isabel
  - 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

## 12 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'

## 12.1 Code39

### Example

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

### Output



## 12.2 EAN8

Only allows 8 characters

### Example

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

### Output



## 12.3 EAN13

Only allows 13 characters

### Example

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

### Output



## 12.4 COOP2of5

### Example

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

### Output



## 12.5 IATA2of5

### Example

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

### Output



## 12.6 Industrial2of5

### Example

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

### Output



## 12.7 ITF

### Example

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

### Output



## 12.8 Matrix2of5

### Example

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

### Output



## 12.9 NW7

### Example

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

### Output



## 12.10 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



## 13 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" : [
    {
      "time" : "20:30",
      "channel" : "BBC3",
      "date" : "2013-10-20",
      "array" : [
        "1",
        "2",
        "3",
        "7"
      ]
    },
    {
      "time" : "21:00",
      "date" : "2013-11-20",
      "channel" : "BBC4",
      "array" : [
        "1",
        "2",
        "3",
        "9"
      ]
    }
  ]
}
```

## 14 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

## 15 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
~~~~
```

## Output

- [ditaa](http://ditaa.sourceforge.net)
  - <http://ditaa.sourceforge.net>
- [graphviz](http://graphviz.org)
  - <http://graphviz.org>
- [JSON](https://en.wikipedia.org/wiki/Json)
  - <https://en.wikipedia.org/wiki/Json>
- [markdown](http://daringfireball.net/projects/markdown)
  - <http://daringfireball.net/projects/markdown>
- [msc](http://www.mcternan.me.uk/mscgen/)
  - <http://www.mcternan.me.uk/mscgen/>
- [pandoc](http://johnmacfarlane.net/pandoc)
  - <http://johnmacfarlane.net/pandoc>
- [PlantUML](http://plantuml.sourceforge.net)
  - <http://plantuml.sourceforge.net>
- [PrinceXML](http://www.princexml.com)
  - <http://www.princexml.com>
- [See Salt](http://plantuml.sourceforge.net/salt.html)
  - <http://plantuml.sourceforge.net/salt.html>
- [YAML](https://en.wikipedia.org/wiki/Yaml)
  - <https://en.wikipedia.org/wiki/Yaml>

## 16 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
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.



- 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

Version	Date	Changes
0.1	2014-04-12	<ul style="list-style-type: none"> <li>• removed ConvertFile.pm</li> <li>• using Path::Tiny rather than other things</li> <li>• changed to use pandoc fences ~~{.tag} rather than xml format</li> </ul>
0.006	2014-04-10	<ul style="list-style-type: none"> <li>• first release to github</li> </ul>

## 17 Start a new page - page

Nice and simple, starts a new page

### Example

```
~~~~{.page}
~~~~
```

## 18 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.

## **19 HTML templates**

Within the samples directory of the release, there are some sample templates.

(c) Property of kmulholland 2013

2014-05-12