Zerodoc Documentation

Release 0.2.0

Pablo Martin Medrano <pablo@odkq.com>

August 17, 2014

Contents

Contents:

CHAPTER 1

Zerodoc

Version 0.2.0 Last updated 2014-08-17 pablo@odkq.com

Zerodoc is a "plain text format" in the spirit of asciidoc, POD, reStructuredText or markdown, with an emphasis on simplicity and extensibility. Very few formatting options are avaliable, both to keep the parser simple and to make it easy to write new generators for the whole format.

Included are a Python library can be used to translate an input file or buffer into a tree that for wich generators can be easily written, and a command line tool to call existing generators for HTML, reStructuredText (that can then be converted or integrated with other tools like Sphinx) and a JSON intermediate representation.

1.1 1. The zerodoc format

1.1.1 1.1 Paragraphs and lines

Zerodoc files are simple text files organized in paragraphs. A paragraph is a group of text lines separated from other paragraphs by blank lines. Lists and source code copied *verbatim* can be defined. An unintrusive format for links is used, based on a references system.

Lines are limited to 72 characters for regular (not code or diagrams) text. If you need to put something more into a line (for example, a long URL), divide it in two and put a backslash () with no spaces at the end.

Example:

```
This is a very long url that needs to be splitted in three:
http://www.reallyreallyreallylonguniformresourcelocatorredir\
ection.com/redirectionator.php?theredirectioncode=d72a565ab8\
7dedf7b5fa84b3ec4b9f11
```

Renders into:

This is a very long url that needs to be splitted in three: http://www.reallyreallyreallylonguniformresourcelocatorredirection.com/redirecti

1.1.2 1.2 Lists

Lists are defined as paragrahps prefixed with a dash, and can be nested. Example

```
- The first element in a list
 - A nested element into the first consisting of two lines
   that are joined on output
 - Another nested element
- The third element in a list
```

Renders into:

- The first element in a list
  - A nested element into the first consisting of two lines that are joined on output
  - Another nested element
- The third element in a list

Backslash joining also occur inside list elements:

```
- The first element in a list. as it have two lines
  with no backslash, an space is inserted between 'lines' and 'with'
- To join the two lines without adding a space a back\
  slash is used. Note that the two spaces formatting the listline are
  removed
```

renders into:

- The first element in a list. as it have two lines with no backslash, an space is inserted between 'lines' and 'with'
- To join the two lines without adding a space a backslash is used. Note that the two spaces formatting the listline are removed after the backslash

NOTE: There are no numbered lists. In the "phylosophy" of zerodoc, numbers can not be omited from the original text nor 'computed' because that will make the text less readable than it's processed output.

1.1.3 1.3 Formatting attributes

Some attributes for the text inherited from other common formats and email conventions are supported:

```
- This is an *emphasis*
- This is an _underline_ (cursive on certain displays or formats,
  as in manual pages)
- This is a 'cursive'
```

Renders into:

- This is an *emphasis*
- This is an _underline_ (cursive on certain displays or formats, as in manual pages)
- This is a 'cursive'

1.1.4 1.4 Links

Links can be included directly in the text along with their destination, or referenced first in the text and then 'resolved' in another line.

Source of a link:

```
This `link`:http://www.google.com will redirect to google
```

Will render as:

This link will redirect to google

Referenced links are 'resolved' in lists of links. This lists of links will be removed from output directly. If the list is contained in a section alone, the section is also removed from output. See the 'References' section at the end of the source code of this document for an example. An 'autocontained' example could be:

```
This line contains two referenced links: `firstlink` and `secondlink`

- `firstlink`:http://www.google.com
- `secondlink`:http://www.google.com
```

Wich renders into:

This line contains two referenced links: firstlink and secondlink

1.1.5 1.5 Source code

Source code is text that will be included verbatim in the output. In source code, newlines are meaningful and no limits on line-length are imposed. An example:

```
#include <stdio.h>

int main() {
        // print hello world 100 times
        for (int i = 0; i < 100; i++) {
                printf("Hello, world!\n");
        }
}
```

Source code is identified by one space before the content of the first line and one or more spaces in the rest. No tabs can be used, so either transform tabs-only source code before pasting or use a tool like expand(1) to do it for you. Blank lines are also included verbatim, up to the one delimiting the next 'regular' paragraph (one that contains text and starts on the first column)

To illustrate source code, i am going to paste the source code (yo dawg) of the example above, along with the regular paragraph-lines sorrounding it:

```
source code, newlines are meaningful and no limits on line-length are
imposed. An example:

 #include <stdio.h>

 int main() {
         // print hello world 100 times
         for (int i = 0; i < 100; i++) {
                 printf("Hello, world!\n");
         }
 }

Source code is identified by one space before the content of
the first line and one or more spaces in the rest. No tabs can

When pygmentize is used, the default language for syntax highlighting
can be specified in options.
```

1.1.6 1.6 Diagrams and images

Diagrams can be either included directly in the output, just as source code, or optionally converted to images (when this is possible, for example in a manual page it does not make sense to include images). Diagrams are converted using ditaa, aafigure, ascii2svg or tikz depending on the options parsed to the renderer. Refer to the aafigure manual page or to the ditaa website for help on this formats.

Diagrams are recognized by using TWO or more spaces before the first line of them. Anything up to the next 'regular' paragraph is considered part of the diagram.

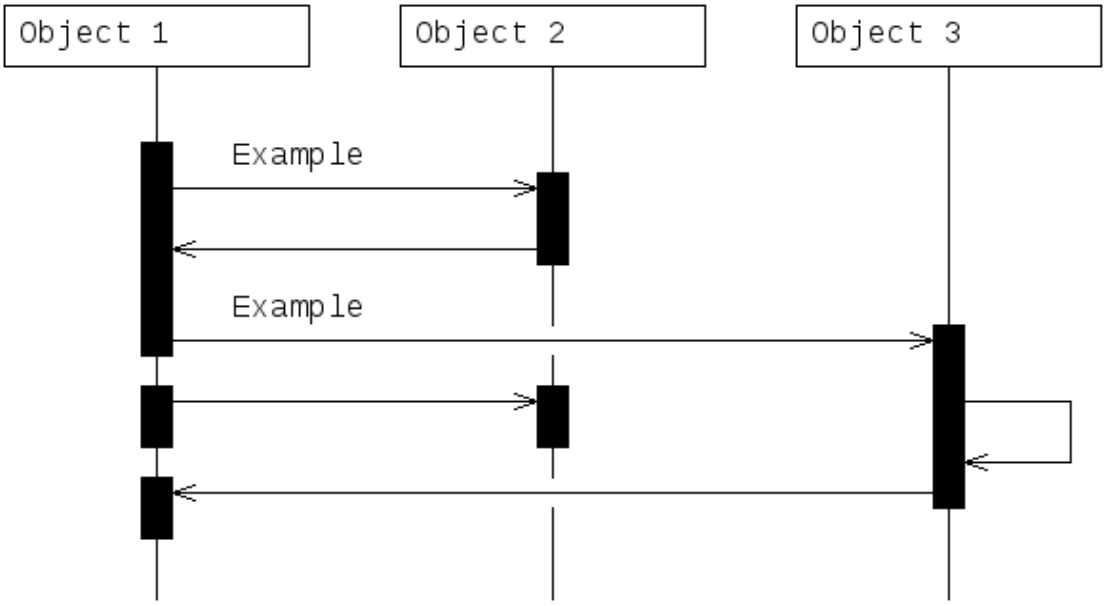
Source-code paragraphs and diagrams can not be adjacent; they need a 'regular' text paragraph (starting on the first column) between them. This makes sense since no diagram can follow source code or viceversa without at least an introduction of what the reader is seeing.

1.6.1 ASCIIToSVG ascii-art diagrams

The default for ascii art diagrams is asciitosvg. As it name implies, it converts text to SVG wich is quite convenient. It is written in php. Example diagram: (asciitosvg)

1.6.2 aafigure ascii-art diagrams

Another format to convert ascii art diagrams to graphics is aafigure. it is written in Python and have quite convenient idioms for things like sequence diagrams:



1.6.3 ditaa ascii-art diagrams

Another common format for ascii art diagrams is ditaa. It does not support svg output.

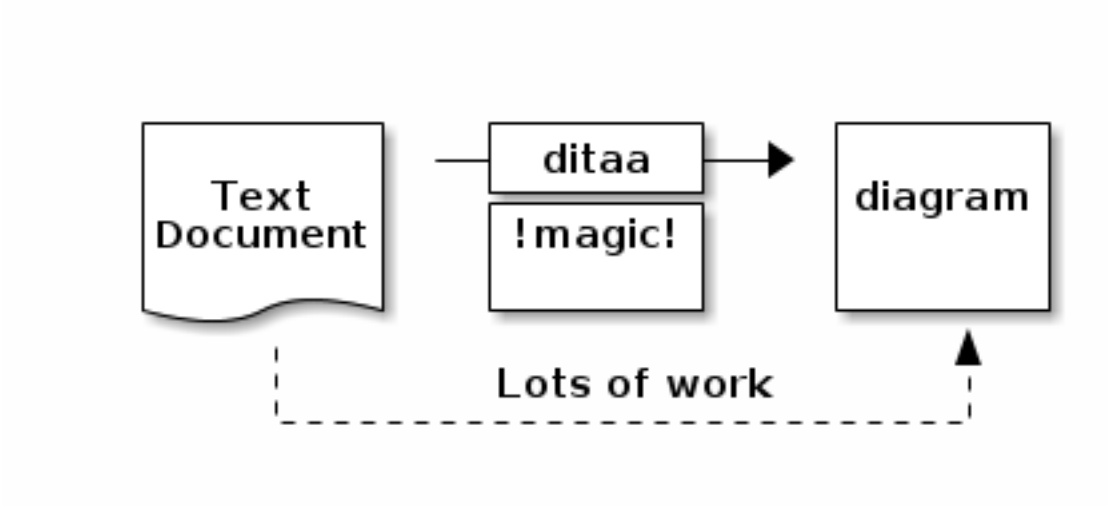
```
This is the source code of the following paragraph
(diagram taken from the 'ditaa website':

Example diagram: (ditaa)

+--------+   +-------+    +-------+
|        | --+ ditaa +--> |       |
|  Text  |   +-------+    |diagram|
|Document|   |!magic!|    |       |
|     {d}|   |       |    |       |
+---+----+   +-------+    +-------+
```

```
:                          ^
|       Lots of work       |
+--------------------------+
```

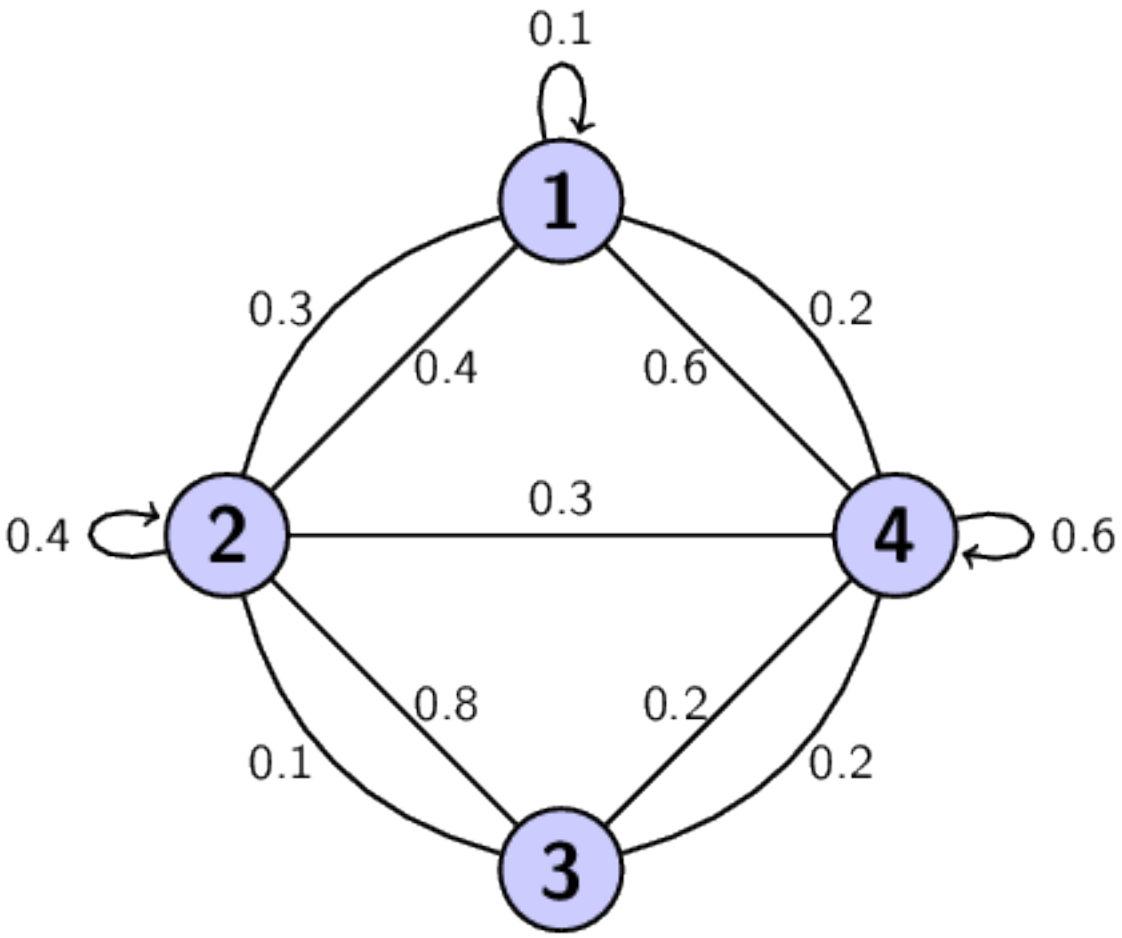
This is the source code of the following paragraph (diagram taken from the ditaa website



Note that there are two spaces before the first +—

1.6.4 TikZ diagrams

A Tikz diagram (from the Tikz examples)



LaTeX source code for that Tikz chunk:

```
\begin{tikzpicture}[auto,node distance=3cm,
  thick,main node/.style={circle,fill=blue!20,draw,
  font=\sffamily\Large\bfseries}]

  \node[main node] (1) {1};
  \node[main node] (2) [below left of=1] {2};
  \node[main node] (3) [below right of=2] {3};
  \node[main node] (4) [below right of=1] {4};

  \path[every node/.style={font=\sffamily\small}]
    (1) edge node [left] {0.6} (4)
        edge [bend right] node[left] {0.3} (2)
        edge [loop above] node {0.1} (1)
    (2) edge node [right] {0.4} (1)
        edge node {0.3} (4)
        edge [loop left] node {0.4} (2)
        edge [bend right] node[left] {0.1} (3)
    (3) edge node [right] {0.8} (2)
        edge [bend right] node[right] {0.2} (4)
    (4) edge node [left] {0.2} (3)
        edge [loop right] node {0.6} (4)
        edge [bend right] node[right] {0.2} (1);
\end{tikzpicture}
```

1.6.6 Diagram tagging and autodetection

As with source code, the type of diagram is autodetected for Tikz and gnuplot diagrams. This detection can be overriden by specifing it in the first line of the diagram, between parenthesis.

1.1.7 1.7 Definition lists

A definition list is a list of terms and corresponding definitions. It usually renders (in HTML, man pages, ReST) in the text of the definition indented with respect to the title. It is useful for documenting functions and command line parameters.

Following is an example:

```
man ls
    Display the manual page for the item (program) ls.
man -a intro
    Display,  in  succession, all of the available intro manual
    pages contained within the manual.  It is possible
    to quit between successive displays or skip any of them.
```

that renders into:

man ls Display the manual page for the item (program) ls.

man -a intro Display, in succession, all of the available intro manual pages contained within the manual. It is possible to quit between successive displays or skip any of them.

1.1.8 1.8 The default zerodoc structure

1.8.1 Header

The header in a zerodoc document contains the title, an optional abstract and a table of contents. The table of contents needs to be updated by hand (this is different from other well known text formats but allow zerodoc to have free-form titles (no — nor ~~~ nor any other form of markup is needed):

```
This is the title, that can spawn several
lines

This are one or several paragraphs of abstract

1. Title 1
2. Title 2
```

1.8.1.1 Title

The title can spawn several lines (a whole paragraph) that will be joined together on output

The table of contents can be prefixed by a 'Table of conents' line that will be recognized automatically as the TOC title. If that line is not present, it will also be ommited on the transformed output.

1.8.1.2 Abstract

The abstract is a group of paragraphs that appear before the Table of content.

1.8.1.3 Table of contents

The table of contents is a list of the titles of the different sections, for example

```
- 1. Section one
- 2. Section two
- 3. Third section
```

Will define the table of contents of a document, if found in the header (after the abstract). If a title listed here is not found in the document, an error is yielded.

1.8.2 Body

The body is formed by several paragraphs. Paragraphs are divided into sections by lines with titles. The lines with titles should appear in the TOC and should have the same content as the TOC. Optionally they can be in uppercase for clarity. As the transformed document usually will have better ways to emphasize the title, the lowercase format used in the TOC will be used regardless of uppercase being used. For example, the next section of this document starts with

```
2. INSTALLING ZERODOC

2.1 Prerrequisites
```

And in the TOC the pertinent lines appear as:

```
-- toc fragment --
  - 1.7.1.3 Table of contents
 - 1.7.2 Body
- 2. Installing zerodoc
 - 2.1 Prerrequisites
```

As you can see on the start of the next section, the title appears in lowercase (as in the TOC above)

1.2 2. Installing zerodoc

1.2.1 2.1 Prerrequisites

Zerodoc needs Python (2.6 or newer) the Python PLY 'lex and yacc' utilities (2.5 or newer) and distutils for installation. Aditionally when generating diagrams, the programs to parse them need to be installed as well.

As an example, in a GNU/Linux Debian 6.0 'Squeeze' system, the requirements can be installed using:

```
# apt-get install python-ply python-aafigure ditaa
```

To generate diagrams with gnuplot or tikz, install the pertinent packages

```
# apt-get install gnuplot
```

```
# apt-get install texlive-picture
```

1.2.2 2.2 Installing the library and interpreter

2.2.1 Using a git snapshot

Clone the github repository using

```
$ git clone git://github.com/odkq/zerodoc.git
```

Change to the zerodoc dir and call setup.py as root

```
$ cd zerodoc/
$ sudo ./setup.py install
```

2.2.2 Using pypi

1.3 3. Using the command line converter

zerodoc - converts a zerodoc text file to HTML and many other formats

1.3.1 3.1 SYNOPSIS

Usage: zerodoc [options]

Options:

-h, –help show this help message and exit

-f FORMAT, –format=FORMAT Output format. If ommited, 'html'

-o OPTIONS, –options=OPTIONS Options for format renderer

-i FILE, –input=FILE Use <filename> as input file. If ommited, use stdin.

-O FILE, –output=FILE Use <filename> as output file. If ommited,use stdout.

1.3.2 3.2 HTML output options

ditaa Use ditaa to format diagrams. When this option is used, you can specify the path of the ditaa .jar file with jarpath:<path>. If jarpath is ommited, 'ditta' will be called (you can install a command-line ditta wraper in Debian and others with apt-get install ditaa)

jarpath:<path> Location of the .jar path (there is no default, 'java' must be in the $PATH)

aafigure Use aafigure to format diagrams

svg Prefer svg in output when applicable (when the converter outputs it and when the rendered format allows for scalable graphics)

datauri Do not generate image files, embbed the images directly in the HTML using *DataURIscheme*

1.3.3 3.3 reStructuredText output options

notoc Usually *reStructuredText* processors attach their own index in the side (*sphinx-doc*, for example). In that case, you better do not output the toc (it is still used to get section titles)

1.3.4 3.4 JSON output options

Json output has no options. It's output is the json render of the parsed tree with no interpretation whatsoever

1.3.5 3.5 Confluence output options

ditaa, jarpath, aafigure, datauri With the same meaning as in the HTML output options

You can specify an output file, and paste it by hand into the confluence 'edition' formulary, or you can make zerodoc client upload it directly with this options:

folder:<folder> Folder (path) for the uploaded document

user:<user> User to use

passwd:<passd> Password

host:<host> Host

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*