Typesetting

*re*StructuredText

with ConTEXt

A Manual for *rst*ConTEXt

Mail any bug reports, patches or suggestions to
`string.format("%s@%s.com", "phg42.2a", "gmail")`
or pay a visit to **my BitBucket home**.

Contents

1 FEATURES NOT IMPLEMENTED 3

Nesting 3 Hyperlinks 3

2 USAGE 5

Invocation from the Command Line 5 Module 6 RST projects 8

3 EXAMPLES 9

Block Quotes 9 Numbered List 10 Line Blocks 11

4 DIRECTIVES 14

Admonitions 14 Images 14

5 SUBSTITUTION DIRECTIVES 17

6 SPECIAL FEATURES 18

Text Roles 18 Bibliography and Citations 19 Tabs 19

7 ABOUT THIS SOFTWARE 21

8 LICENSE 23

1 Features Not Implemented

1.1 Nesting

Proper nesting. So far only lists support real nested structures. There's no way you could have real paragraphs or bulleted lists inside table cells. The problem is that with true nesting some jobs like the dissection of tables would have to be moved from the formatter to the parser. If you feel you need thoroughly nested structures – e.g. grid tables in footnotes or bullet lists inside simple tables inside enumerations inside quotations inside footnotes – you should consider including ConTEXt code as substitution directives. (OTOH docutils' new and old LaTeX formatter seems to have problems with tables in footnotes as well. Not to mention its preference for enclosing random nested structures in `quote`-environments.)

Should you find yourself in desparate need of tables or whatever structures inside footnotes then I might agree to find a solution if you ask.

1.2 Hyperlinks

The hyperlink implementation should be fine in general use if you avoid certain situations.

- Never ever call your hyperlink targets `anon_#`, where `#` stands for any integer. Just don't do it, OK? Great.

- Referencing a structure element like a section heading by means of an *empty link block* does work. However, if the element in question requests a page break (e.g. the vanilla `chapter{#1}` command),

the reference will link to the previous page instead and become useless. You can avoid this behaviour by referencing the section directly or by targetting the first paragraph in the section instead.

- Link chaining does not work with internal references. This is considered a low-priority bug and will be addressed during the next big hyperlink overhaul.

2 Usage

2.1 Invocation from the Command Line

*rst*ConTEXt is integrated into the `mtxrun` command as a script, which relies, naturally, on the Lua interpreter of LuaTEX. Therefore, *rst*ConTEXt might not run at all on other Lua installations, at least not without modification of the source. Fortunately, every ConTEXt user is equipped with LuaTEX nowadays so this dependency should be trivial.

To generate ConTEXt code from a reStructuredText document named `infile.rst`, call `mtxrun`:

```
$mtxrun --script rst --if=infile.rst --of=outfile.tex
```

You should now have a file `outfile.tex` that is ready to be run by ConTEXt. With some exceptions the generated code is downward compatible with MkII, thus it does not matter for a start whether you decide to test it with `texexec` or `context`.

The resulting TEX file has rather a basic layout, if at all. This is intentional as you are expected to include it in a document after specifying your own setups. An example for prepended setups can be found in the environment for this manual (`mod/doc/context/third/rst/manual.tex`).

The output of *rst*ConTEXt automatically inserts necessary setups for the components found in the input. Therefore, the `starttext` and `stoptext` commands are part of the output and may not be specified in your setups file. For now you have to use the ConTEXt command `appendtoks <token> to starttext` to add

content like title pages and indices to the result. This mechanism works reliable as long as you have an eye on the order in which the tokens are given. Again, have a look at `manual.tex` to get an impression how useful this can be. User hooks for these and other common constructs are thought of but have yet to be implemented.

To build the documentation, first create a temporary directory somewhere safe. Then copy or symlink the Lua files from `mod/tex/context/third/rst/` and the manual source there as well:

```
$mkdir tmp; cd tmp
$ln -s ../mod/doc/context/third/rst/documentation.rst .
```

Now run *rst*ConTEXt on the main documentation file as follows:

```
$mtxrun --script rst --if=documentation.rst --of=doc.tex
```

Now run ConTEXt on the layout file:

```
$context ../mod/doc/context/third/rst/manual.tex
```

This will include the generated code after a couple of setups – voilà, you have successfully built `manual.pdf`. (Note that the commands you have to issue in each of the steps vary across different OS. In the literal form the example might only work on Linux or POSIX compliant systems.)

2.2 Module

A provisional module for MkIV is included (`t-rst.mkiv`). Actually, the converter was thought of as a module for direct rendering of reStructuredText input initially, but certain objections diverted me from this path.

- *Typography.* It's all about the details. No matter how good your converter is, it still won't reach TEX's omnipotence and flexibility.

*rst*ConTEXt is a tool to generate raw material for your typesetting job, not a typesetting system in itself.

- *Testing.* Never underestimate the insights gained from reading the resulting ConTEXt file. Quite some effort has been undertaken to make it human-readable, especially the setups.

- *MkII.* I'm not an MkII user at all save for rapid testing and the occasional check for the sanity of ConTEXt's behaviour. Slow hardware forces me to run pdfTEX instead of LuaTEX whenever I need some result as quick as possible, so I wanted to keep the code MkII clean. Do not expect Unicode (as in this document) to work without precautions.

During the development readability of the generated code was alway one of the main goals of *rst*ConTEXt. Quite some computing effort is made to reflow even simple things as paragraphs into a shape understandable by more than only the TEX machine. If you should at one point decide that your project is ripe for the typographical finish and you want to add local changes in form of TEX code only, you should be able to use the output of *rst*ConTEXt as starting point.

However, using the module may have advantages when testing. There is a usage example in `moduletest.tex`, introducing the macro `\typesetRSTfile`. Another example in `hybridtest.tex` demonstrates the ConTEXt command `\RST` as well as the corresponding environment.

To install the module simply copy the files into your local TEX tree, i.e. if the minimals reside in `~/context/`, you would issue the following line:

```
$cp -r ./mod/* ~/context/tex/texmf-local/
```

Then rebuild the filename database running `context --generate`. The module should be ready for use now.

2.3 RST projects

In addition to the simple command `\typesetRSTfile` the module also provides means for handling multiple reStructuredText input files. This is achieved by so-called *inclusions*. An inclusion has to be defined first, using the macro `\defineRSTinclusion`, which receives up to three arguments in brackets. The first one specifies the *identifier* by which the inclusion will be referred to afterwards (cf. ConTEXt's `\useURL` command). The second argument, which is mandatory as well, takes the file to be associated with an inclusion. Finally, optional setups can be passed to the parser via the third argument (cf. the section on **Tabs**). E.g.:

```
\usemodule[rst]
\defineRSTinclusion [first][inc-first.rst]
\defineRSTinclusion[second][inc-second.rst][expandtab=true,shiftwidth=8]
\defineRSTinclusion [third][inc-third.rst]
```

Those inclusions are afterwards accessible *within* the `\[start|stop]project` environment, and can be dereferenced by `\RSTinclusion`, which takes the identifier as a single argument in brackets:

```
\startRSTproject
\RSTinclusion [first]
\RSTinclusion[second]
\RSTinclusion [third]
\stopRSTproject
```

Within the project environment, *rst*ConTEXt allows for arbitrary ConTEXt markup.

3 Examples

*rst*ConTEXt was developed for the largest part by going through the reStructuredText **specification** step by step and tested against the examples given both in the spec and in the **quick reference**. Therefore you should refer to those examples first (and drop me a note immediately if any of them stopped working). All kinds of text blocks and inline markup have been implemented with the exception of anything mentioned in the section on **Features Not Implemented**. Some of them that I have not found a real-world usage for (such as *definition lists*) do not yet have a presentable output – there is room for improvements that should be supplied by somebody who actually uses those features.

3.1 Block Quotes

The *block quote* syntax is fully supported, including attributions. For instance, the next snippet:

```
Some people have made the mistake of seeing Shunt's work as a
load of rubbish about railway timetables, but clever people
like me, who talk loudly in restaurants, see this as a
deliberate ambiguity, a plea for understanding in a
mechanized world.

--- Gavin Millarrrrrrrrrr on Neville Shunt
```

gets you a neatly indented quotation, typeset in a slightly smaller font magnitude.

> Some people have made the mistake of seeing Shunt's work as a load of rubbish about railway timetables, but clever people like me, who talk loudly in restaurants, see this as a deliberate ambiguity, a plea for understanding in a mechanized world.
>
> *Gavin Millarrrrrrrrrr on Neville Shunt*

Don't forget proper indentation.

3.2 Numbered List

Save for nesting lists are fully implemented in *rst*ConTEXt. The following code typesets a triple-nested list with different kinds of bulleting / numbering:

```
i   First order list, first entry.

ii  First order list, second entry.

iii First order list, third entry.

    -   Second order list, first entry.

        #   Third order list, first entry.

        #   Third order list, second entry.

        #   Third order list, third entry.
            Real nesting rules!

    -   Second order list, second entry.

iv  First order list, fourth entry.

v   First order list, fifth entry.
```

The result looks like this:

i. First order list, first entry.

ii. First order list, second entry.

iii. First order list, third entry.

   – Second order list, first entry.

      1. Third order list, first entry.

      2. Third order list, second entry.

      3. Third order list, third entry. Real nesting rules!

   – Second order list, second entry.

iv. First order list, fourth entry.

v. First order list, fifth entry.

Don't forget the blank lines between list items.

3.3 Line Blocks

Line blocks are a convenient environment for parts of the text that need to preserve line breaks and indentation. This makes it the first choice for most kinds of poems:

```
| When does a dream begin?
|   Does it start with a goodnight kiss?
|       Is it conceived or simply achieved?
| When does a dream begin?
```

```
|
| When does a dream begin?
|   Is it born in a moment of bliss?
|       Or is it begun when two hearts are one?
| When does a dream exist?
|
| The vision of you appears somehow
    Impossible to resist
| But I'm not imagining seeing you
    For who could have dreamed of this?
|
| When does a dream begin?
|   When reality is dismissed?
|       Or does it commence when we lose all pretence?
| When does a dream begin?
```

Indentation, continued lines, etc. should work out without problems:

When does a dream begin?
  Does it start with a goodnight kiss?
    Is it conceived or simply achieved?
When does a dream begin?

When does a dream begin?
  Is it born in a moment of bliss?
    Or is it begun when two hearts are one?
When does a dream exist?

The vision of you appears somehow Impossible to resist
But I'm not imagining seeing you For who could have dreamed of this?

When does a dream begin?
  When reality is dismissed?
    Or does it commence when we lose all pretence?
When does a dream begin?

4 Directives

4.1 Admonitions

The following admonition directives have been implemented:

4.1.1 Caution

The *caution* directive results in the text being prefixed by one "dangerous bend" symbol in order to resemble the "wizards only" passages of the TeXbook. For example, the directive:

```
.. caution:: White mice do worse in experiments than grey mice.
```

will result in the following:

White mice do worse in experiments than grey mice.

4.1.2 Danger

Similar to the *caution* directive, the *danger* directive prefixes the given text with two "dangerous bends" giving it the look of Knuths's "esoteric" annotations.

Be nice to the parser: Don't forget to align paragraphs that end a literal block!

4.2 Images

Including pictures is easy using the *image* directive: simply supply it the name of the image file as in `.. image:: cow`. If the format is

supported by ConTEXt the suffix can be neglected.

The placement of images can be controlled via a set of optional arguments, each of which has to be specified on single line in `key: value` style:

```
.. image:: cow
    width: hsize
    caption: A generic Dutch cow.
```

This will place your image somewhere close to the spot where you defined it. (The placement parameter to `placefigure` will be set to `here` by default.)

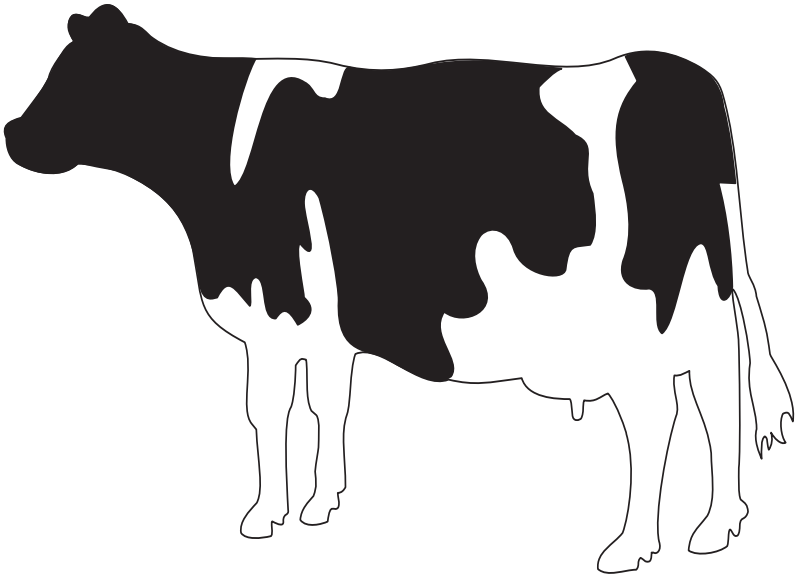

Figure 4.1 A generic Dutch cow (*bos primigenius taurus*).

The supported parameters are `width` (alias: `size`), `caption` and

`scale`. The *width* parameter accepts the values `hsize` (alias: `fit`, `broad`) or `normal`. Alternatively, the *scale* parameter allows for arbitrary manipulation of the desired magnification; it defaults to `1` (unscaled). The value passed as *caption* parameter will be used in as the caption text of the image.

5 Substitution Directives

There are substitution directives for simple *replacing* and for insertion of LuaTEX's three languages: METAPOST, Lua and, of course, TEX.

Ordinary text replacement is done via the `replace` substitution directive. E.g. in the main text you consistently use `|replaceme|` and have all its occurences substituted by `I wasn't in the mood to write out this long sentence.` like in the next snippet:

```
.. |replaceme| replace::
    I wasn't in the mood to write out this long sentence.
```

The code insertions work similarly. You have to specify some phrase that gets substituted by the code you supply. E.g. this document accesses the fancy logos predefined in the ConTEXt core via substitutions:

```
.. |CONTEXT| ctx:: \CONTEXT
.. |LUATEX| ctx:: \LUATEX
```

Etc. pp. The respective directive names are `ctx`, `mp` and `lua`. In order to get a ● drawn on spot, you would define a Metapost substitution:

```
.. |circle| mp::
    fill fullcircle scaled(8) withcolor blue;
```

6 Special Features

6.1 Text Roles

The default *role* for interpreted text is *emphasis*.

The role marker provides explicit access to formatting commands. The formatting routine for inline literals can be called with the role marker `literal`, strong emphasis likewise is achieved via the role marker `strong_emphasis`.

Other roles that lack an equivalent among inline markup are `bold`, `ss` (alias `sans_serif`), `uppercase`, `lowercase` and colors. Color roles begin with the string `color_` (the underscore is compulsive), followed by either the string `rgb_` or a **valid color name**. An rgb vector is specified in decimal. Its values can be separated by either dashes or underscores. Thus, `color_rgb_.3_.5_.8` is a valid rgb expression, as is `color_rgb_0-1-0`. Unforturnately, the colon character `:` has to be escaped in color expressions, e.g. `color_gray:5`.

For example, to give Mr. Neville Shunt's work an apt typographic representation you can use these roles instead of the standard inline markup:

```
:color_rgb_.9_.2_.7:`Chuff`, chuff, :literal:`chuffwooooooch`,
wooooooch! Sssssssss, sssssssss!  :uppercase:`Diddledum`,
`diddledum`, diddlealum.  :literal:`Toot`, toot. The train
:bold:`now` standing :color_gray\:5:`at` platform :ss:`eight,
tch`, tch, :color_rgb_0-1-0:`tch`,
:color_rgb_.5-.6-.2:`diddledum`, diddledum.
:lowercase:`Chuffff`, :strong_emphasis:`chuffffiTff`
eeeeeeeeeaaaaaaaaa :color_red:`Vooooommmmm`.
```

which yields when passed through *rst*ConTEXt:

Chuff, chuff, `chuffwooooooch`, wooooooch! Sssssssss, sssssssss! DIDDLEDUM, *diddledum*, diddlealum. `Toot`, toot. The train **now** standing at platform `eight`, `tch`, tch, tch, diddledum, diddledum. chuffff, CHUFFFFITFF eeeeeeeeeaaaaaaaaa Vooooommmmm.

6.2 Bibliography and Citations

Not much for now concerning the usage of Taco's bib system. It's just that I use my own bibliography system and never became sufficiently familiar with the standard ConTEXt approach. *If you feel that the current support should be improved then feel free to contact me!* I will need somebody for testing.

When *rst*ConTEXt first encounters a citation (`[texbook]_`) it automatically looks up a bibliography in the working directory by the name of `jobname`. E.g. with a main file `manual.tex` bibtex will use the database called `manual.bib`. Symlinking your bibliography file in the local tree should suffice and you can keep whatever directory structure you prefer. (Speaking for myself, bib data usually resides in its own subdirectory, so I'd use symlinks, too.)

6.3 Tabs

The reStructuredText specification requests that tabs (ASCII no 9) be treated as **spaces**. Converting your tabs to spaces might be a good preparation for an *rst*ConTEXt run. However, as of version 123 *rst*ConTEXt comes with built-in tab expansion. It can be enabled by supplying an optional argument to the `typesetRSTfile` command:

```
\usemodule[rst]
\typesetRSTfile[expandtab=true,shiftwidth=4]{myfile.rst}
```

The argument `expandtab` triggers a prepocessing step which expands all tabulation characters (`t` and `v`) into the correct amount of

spaces. Optionally, the tab stop distance can be configured using the `shiftwidth` parameter, which defaults to 4.

The **docutils** provide means to translate the extra-convenient markup language reStructuredText into various formats like PDF, HTML and LaTeX, unfortunately omitting the One True Macro System: ConTeXt.

As far as I am aware of it, there is some support for reStructuredText in **Pandoc** but as it relies on a rather large set of dependencies it proved very difficult (too difficult for me) to install on my favourite distribution. From the **interactive demo** I gather that support for reStructuredText's language features is not very extensive and the result did not even come with proper setups. Additionally, it's written in a language I am not familiar with and that does not make use of one the most awesome features of all the the extended capabilities LuaTeX provides: the Lua interpreter.

For quite some time I was thinking about how to implement an reStructuredText parser in LuaTeX, until some **discussion** emerged on the ConTeXt mailing list that indicates a broader interest in convenient markup languages across the community. As the alternatives mentioned above don't meet the expectations of a normal ConTeXt user, the initial step to write *rst*ConTeXt was done. Handling most of the corner cases and usability features of reStructuredText proved in the end not nearly as easy as I imagined.

*rst*ConTeXt is experimental software and neither feature complete nor thoroughly commented. Keep this in mind before you start using it. Anything might still be subject to change, so expect breakage *in case you start relying on exceptional behaviour* (read: bugs) that does not conform to the reStructuredText specification. Consider filing a bug report instead and wait for me (the

maintainer) to fix it, because regardless of how much testing I do myself I alway run into the weirdest issues only during the actual deployment of the software. Thus, if you notice that *rst*ConTEXt does not adhere to the **outline** of reStructuredText according to the Docutils documentation, very likely you have discovered a corner case I was not aware of.

8 License