niceverb.sty

—

Minimizing Markup for Documenting LaTeX packages*

Uwe Lück
http://contact-ednotes.sty.de.vu

February 23, 2009

Contents

*This manual describes package version version 0.1 as of February 23, 2009.

1 Presenting niceverb

1.1 Purpose

The niceverb package provides "minimal" markup for documenting LaTeX packages, reducing the number of keystrokes/visible characters needed (kind of poor man's WYSIWYG).[1] It conveniently handles command names in arguments of macros such as `\footnote` or even of sectioning commands. Commands for typesetting a package's code are inserted automatically (just using TeX). As opposed to tools that are rather common on UNIX/Linux, this operation should work at any TeX installation, irrespective of platform.

The package may at least be useful while working at a very new package and may suffice with small, simple packages. After having edited your package's code (typically in a `.sty` file), you just "`latex`" the manual file (maybe some `.tex` file) and get instantly the corresponding refreshed documentation.

niceverb may also help to generate without much effort documentations of nowadays commonly expected typographical quality for packages that so far only had ASCII documentations.

1.2 Acknowledgement/Basic Ideas

Three ideas of Stephan I. Böttcher's in documenting his lineno.sty inspired the present work:

1. The markup and its definitions are short and simple, markup commands are placed at the right "margin" of the ASCII file, so you hardly see them in reading the source file, you rather just read the text that will be printed.

2. An AWK script removes the %s starting *documentation* lines and inserts the commands for typesetting the package's *code* (you don't see them in the source).

3. An active character ('|') issues a `\string` *and* switches to typewriter typeface for typesetting a command verbatim—so this works without changing category codes (which is the usual idea of typesetting code), therefore it works even in macro arguments.

[1] "What you see is what you get." Novices are always warned that WYSIWYG is essentially impossible with LaTeX.

1.3 The Commands and Features of niceverb

Single quotes `‘`, `’`, “less than” `<` (accompanied with `>`), the hash mark `#`, ampersand `&`, and in an extended “auto mode” even backslash `\` become `\active` characters with “special effects.”

The package mainly aims at typesetting commands and descriptions of their syntax *if the latter is “standard LaTeX-like”*, using “meta-variables.” A string to be typeset “verbatim” thus is assumed to start with a single command like `\foo`, maybe followed by stars (‘`*`’) and pairs of square brackets (`[...]`) or curly braces (`{...}`), where those pairs contain strings indicating the typical kinds of contents for the respective arguments of that command. A typical example is this:

`\foo*[`⟨*opt-arg*⟩`]{`⟨*mand-arg*⟩`}`

This was achieved by typing `&\foo*[<opt-arg>]{<mand-arg>}`. In “auto mode” of the package, even typing

`\foo*[<opt-arg>]{<mand-arg>}`

would have sufficed—WYSIWYG! (I call such mixtures of *verbatim* and “meta-variables” *‘meta-code’*.)

Now for the details:

“Meta-variables:” The package supports the “angle brackets” style of “meta-variables” (as with ⟨*meta-variable*⟩). You just type ‘`<foo>`’ to get ‘⟨*foo*⟩’.

This works due to a sloppy variant `\SimpleVerb` of `\verb` which doesn’t care about possible ligatures and definitions of active characters. Instead, it assumes that the “verbatim” font doesn’t contain ligatures anyway. ‘`\verb|<foo>|`’, by contrast, just yields ‘`<foo>`’.

Almost the same feature is offered by ltxguide.cls which formats the basic guides from the LaTeX Project Team. The present feature, however, also works in plain text outside verbatim mode.

Single quotes (left/right) for “short verb:” The package “assumes” that *quoting* refers to *code*, therefore ‘‘`foo`’’ is typeset as ‘`foo`’. This somewhat resembles the `\MakeShortVerb` feature of doc.sty.

It will typically fail when you try to typeset commands (and their syntax) in *macro arguments*—e.g.,

```
\footnote{‘\bar’ is a celebrated fake example!}
```

will try to *execute* `\bar` instead of typesetting it, giving an "undefined" error so. `\verb` fails in the same situation, for the same reason. '`&`' (`\footnote{&\bar`⟨*remaining*⟩`}`) or "auto mode" (see below) may then work better.

Double quotes and apostrophes should still work the usual way; otherwise you could control the parsing mechanisms using curly braces (outside and inside don't interact). To get usual single quotes, you can use their standard substitutes `\lq` and `\rq`. You can "abuse" this "single quotes" feature just to get typewriter typeface.[2] For difficult cases, you can still use the standard `\verb` command from LaTeX.

Single right quotes for `\textsf`: Package names are (by some unwritten convention!?) typeset with `\textsf`; it was natural to use a remaining case of using single quotes for replacing `\textsf{`⟨*text*⟩`}` by `'`⟨*text*⟩`'`.[3]

This idea of switching fonts continues font switching of wiki.sty which uses the syntax for editing Wikipedia pages.

Ampersand '`&`' typesets command syntax even in arguments: e.g., type '`&\foo{<arg>}`' to get '`\foo{`⟨*arg*⟩`}`'. This may be even more convenient for typing than the single quotes method, although looking somewhat strange. However, `&` may terminate *verbatim* unexpectedly, being designed for displaying "LaTeX-like command syntax" in the first instance.[4][5]

This choice of `&` rests on the assumption that there won't be many tables in the documenation. You can restore the usual meaning of `&` by `\MakeAlign\&` and turn the present special meaning on again by

`\MakeActiveLet\&\CmdSyntaxVerb`.

You could also redefine (`\renewcommand`) `\descriptionlabel` using `\CmdSyntaxVerb` (the "normal command" equivalent to `&`) so `\item[\foo]` works as wanted.

"Auto mode" typesets commands verbatim unless ... In "auto mode", the backslash '\' is an active character that builds a command

[2] In macro arguments this requires that the right single quote `'` is `\active`.

[3] Font switching by sequences of single quotes is a feature of the syntax for editing Wikipedia pages and of wiki.sty.

[4] Moreover, `&` currently has a limited xspace functionality only.

[5] You can even use `&` for referring to active characters like `&` in footnotes etc.!

name from the ensuing letters and typesets the command (and its syntax, allowing meta-variables) verbatim. However, there are some exceptions, which are collected in a macro `\niceverbNoVerbList`. `\begin`, `\end`, and `\item` belong to this list, you can redefine (`\renewcommand`) it. There is also a command `\NormalCommand{`⟨*letters*⟩`}` *issuing* the command `\`⟨*letters*⟩ instead of typesetting it. Since auto mode is somewhat dangerous, you have to start it explicitly by `\AutoCmdSyntaxVerb`. You can end it by `\EndAutoCmdSyntaxVerb`.

Auto mode is motivated by the observation that there are package files containing their documentation as pure (well-readable) ASCII text—containing the names of the new commands without any kind of quotation marks or verbatim commands. Auto mode should typeset such documentation just from the same ASCII text.

Hash mark # comes verbatim. No macro definitions are expected in the `document` environment.[6] Rather, '`#`' is an active character for taking the next character (assuming it is a digit) to form a reference to a *macro parameter*—'`#1`' becomes '`#1`' (WYSIWYG indeed!).

1.4 What is Wrong with the Present Version

1. niceverb.sty should be an extension of wiki.sty; yet their font selection mechanisms are currently not compatible.

2. Font switching or horizontal spacing may fail in certain situations with parentheses. (You can correct spacing by '`\ `'.)

3. The feature of mixing high-quality-typeset comments into the package code listing is implemented in a very rudimentary way only. The "comment detector" detects Wikipedia-style subsection titles instead of lines beginning with percent characters.[7]

4. The code listing currently uses the `listing` and `listingcont` environments of moreverb.sty; the code font and the line numbers may be too large.

5. The "vertical" character '`|`' should be active and provide a version of the `decl` environment of ltxguide.cls.

[6] This idea recently appeared on the latex-l mailing list. It may be wrong, not sure at the moment, think of latexa.ltx ...

[7] Percent characters will definitely not be "ignored" as with `\DocInput`, rather they will hide rests of *documentation* lines as usually, while they will be typeset verbatim in *package code* lines.

2 Implementation of the Markup Syntax

```
\NeedsTeXFormat{LaTeX2e}[1994/12/01]
\ProvidesPackage{niceverb}[2009/02/22 v0.1
                           minimize \string\verb\space code (UL)]
%% Copyright (C) 2009 Uwe Lueck,
%% http://www.contact-ednotes.sty.de.vu
%% -- author-maintained in the sense of LPPL below --
%%
%% This file can be redistributed and/or modified under
%% the terms of the LaTeX Project Public License; either
%% version 1.3a of the License, or any later version.
%% The latest version of this license is in
%%     http://www.latex-project.org/lppl.txt
%% We did our best to help you, but there is NO WARRANTY.
%%
%% Please report bugs, problems, and suggestions via
%%
%%   http://www.contact-ednotes.sty.de.vu
%%
```

2.1 Switching category codes

```
\providecommand{\CatCode}{\catcode`}
% \providecommand*{\MakeActive}[1]{\CatCode#1\active}
\providecommand*{\MakeAlign} [1]{\CatCode#14\relax}
\providecommand*{\MakeLetter}[1]{\CatCode#111\relax}
\providecommand*{\MakeOther} [1]{\CatCode#112\relax}
\providecommand*{\MakeSub}   [1]{\CatCode#18\relax}
\newcommand*{\MakeActiveLet}[2]{%% cf. \@sverb & \do@noligs
  \CatCode#1\active
  \begingroup
    \lccode`\~`#1\relax \lowercase{\endgroup \let~#2}}
\MakeLetter\_
\newcommand*{\make_iii_other}{\MakeOther\\\MakeOther\{\MakeOther\}}
```

2.2 Sloppy variant of \verb

```
\newcommand*{\begin_min_verb}{%
  \relax \ifmmode \hbox \else \leavevmode\null \fi
    %% standard, for $$...$$
```

```
  \bgroup \tt %%%\let\do\MakeOther \dospecials
    \MakeLetter\_\MakeLetter\@}
\newcommand*{\SimpleVerb}[1]{%
  %% mainly avoid \verb's noligs list which overrides definitions
  %% of some active characters, while cmtt doesn't have any
  %% ligatures anyway.
  \ifx\protect\@typeset@protect
    \begin_min_verb \make_iii_other \MakeActiveLet#1\egroup
  \else \string\SimpleVerb \string#1\fi}
```

2.3 Single quotes typeset meta-code

```
\newcommand*{\lq_verb}{%
  \ifx\protect\@typeset@protect
    \expandafter \lq_double_test
  \else \lq \fi}
%     \ifcat\noexpand'\noexpand~%
%       \expandafter\expandafter\expandafter
%         \protect_corresp_quotes
%     \else
%       \rq_verb
%     \fi
%   \fi}
\newcommand*{\lq_double_test}{%
  %% test settles next catcode, better switch to ``other''
  %% in advance:
  \begingroup \let\do\MakeOther \dospecials
    \futurelet\let_token \lq_double_decide}
\newcommand*{\lq_double_decide}{%
  \ifx\let_token\lq_verb
      \endgroup
    \lq\lq \expandafter \@gobble
    %% corresponding right quotes will become ``other'' due to
    %% having no space at the left. TODO to be changed with
    %% wiki.sty
  \else
      \endgroup
%     \expandafter \rq_verb
    \expandafter \SimpleVerb \expandafter \'%
  \fi}
% \newcommand*{\rq_verb}{\SimpleVerb\'}
```

```
% \AtBeginDocument{\MakeActiveLet\‘\rq_verb}
\AtBeginDocument{\MakeActiveLet\‘\lq_verb}
  %% Strings referred to will be code
  %% TODO to be changed with wiki.sty
```

2.4 Ampersand typesets meta-code

```
\newcommand*{\CmdSyntaxVerb}{%
  \ifx\protect\@typeset@protect
    \expandafter \cmd_syntax_verb
  \else %% thinking of .aux only
    \string\CmdSyntaxVerb \expandafter \string
  \fi}
\newcommand*{\cmd_syntax_verb}[1]{%
  \begin_min_verb \string#1\futurelet\let_token \after_cs}
\AtBeginDocument{\MakeActiveLet\&\CmdSyntaxVerb}
  %% not needed with \Auto... OTHERWISE useful in args!
  %% TODO \MakeAmpCmdSyntax
  %% TODO \let\endcell& (wie \endline, \endgraf) \MakeEndCell
\newcommand*{\after_cs}{%
  \ifcat\noexpand\let_token a\egroup \space
  \else \expandafter \decide_verb \fi}
\newcommand*{\test_more_verb}{\futurelet\let_token \decide_verb}
\newcommand*{\decide_verb}{%
%  \show\let_token
    \jumpteg_on_with\bgroup\braces_verb
    \jumpteg_on_with[\brackets_verb
    \jumpteg_on_with*\star_verb
  \egroup}
    %% CAUTION/TODO wrong before (... if cmd without arg
    %%         use \ then or choose usual verb...
    %%         or \MakeLetter\( etc. ... or \xspace
\newcommand*{\jumpteg_on_with}[2]{%
  \ifx\let_token#1\do_jumpteg_with#2\fi}
 %% TODO cf. xfor, xspace (break@loop);
 %% \DoOrBranch#1...#1 or so
\def\do_jumpteg_with#1#2\egroup{\fi#1}
\def\braces_verb#1{\string{#1\string}\test_more_verb}
\def\brackets_verb[#1]{[#1]\test_more_verb}
\def\star_verb*{*\test_more_verb}
\makeatletter
```

2.5 Escape character typesets meta-code

```
\DeclareRobustCommand*{\BuildCsSyntax}{%
  \futurelet\let_token \build_cs_syntax_sp}
\newcommand*{\build_cs_syntax_sp}{%
  \ifx\let_token\@sptoken \else %% TODO ^^M!?
    \expandafter \start_build_cs_syntax
  \fi}
\newcommand*{\start_build_cs_syntax}[1]{%
  \edef\string_built{\string#1}%% #1 may be active
  \MakeLetter\_\MakeLetter\@%% CAUTION, cf. ...
  \test_more_cs}
\newcommand*{\test_more_cs}{%
  \futurelet\let_token \decide_more_cs}
\newcommand*{\decide_more_cs}{%
  \ifcat\noexpand\let_token a\expandafter \add_to_cs
  \else
    \MakeSub\_\MakeOther\@%
    \expandafter \in@ \expandafter
      {\csname \string_built \expandafter \endcsname
        \expandafter}\expandafter{\niceverbNoVerbList}%
    \ifin@
      \csname \string_built
        \expandafter\expandafter\expandafter \endcsname
    \else
      \begin_min_verb \@backslashchar\string_built
        \expandafter\expandafter\expandafter \test_more_verb
    \fi
  \fi}
  %% TODO such \if nestings with ifthen!?
 %% cf.:
%  \let\let_token,\typeout{\meaning\let_token}
  %% TEST TODO fuer xspace!? (\ifin@)
\newcommand*{\add_to_cs}[1]{%
  \edef\string_built{\string_built#1}\test_more_cs}
\newcommand*{\AutoCmdSyntaxVerb}{\MakeActiveLet\\\BuildCsSyntax}
  %% TODO or \niceverbswitch...
\newcommand*{\EndAutoCmdSyntaxVerb}{\CatCode\\\z@}
\newcommand*{\NormalCommand}{} \let\NormalCommand\@nameuse
%% Were tests:
%  \futurelet\LetToken\relax \relax
```

```
  %  \show\LetToken \typeout{\ifcat\noexpand\LetToken aa\else x\fi}
\newcommand*{\niceverbNoVerbList}{%
  \begin\end\item\verb\EndAutoCmdSyntaxVerb\NormalCommand
  \section\subsection}%% TODO!?
```

2.6 Meta-variables

```
\def\niceverb_meta#1>{%
  \mbox{\normalfont\itshape $\langle$#1\/$\rangle$}}
  %% TODO offer without angles as well
\AtBeginDocument{\MakeActiveLet\<\niceverb_meta}
  %% difference to ltxguide.cls: also outside verbatim
```

2.7 Hash mark is code

```
\newcommand*{\param_verb}[1]{{\tt\##1}}
\AtBeginDocument{\MakeActiveLet\#\param_verb}
```

2.8 Single right quote for \textsf

```
%% TODO fails in parentheses due to preserving apostrophes
\newcommand*{\niceverb_rq_sf}{%
  \ifx\protect\@typeset@protect
    \expandafter \niceverb_rq_sf_test
  \else \rq \fi}
  %% introduced another macro just to avoid more sequences
  %% of \expandafter ...
\newcommand*{\niceverb_rq_sf_test}{%
  \ifhmode \ifdim\lastskip>\z@
    \expandafter\expandafter\expandafter \niceverb_rq_sf_exec
    %% TODO but after `('!? make \( active, also for \after_cs!?
    \else \rq \fi
  \else \ifvmode
    \expandafter\expandafter\expandafter \niceverb_rq_sf_exec
    \else \rq \fi
  \fi}
{\CatCode\'\active \gdef\niceverb_rq_sf_exec#1'{\textsf{#1}}}
%% TODO to be changed with wiki.sty:
\AtBeginDocument{\MakeActiveLet\'\niceverb_rq_sf}
%% TODO \niceverbswitch{<mode>}{<on/off>} (bzw. Doku ohne {})
```

2.9 Leave package mode

```
\makeatother
\MakeSub\_
\endinput
```

3 Code Preparing Source for Typesetting

This is at present in a file `makedoc.tex`.

```
\ProvidesPackage{makedoc.tex}[2009/02/21 (UL)]
%% make packagecode environments and remove percents starting lines

\makeatletter \catcode`\_=11

\openin\@inputcheck=niceverb.sty %% JUST HERE
\newwrite\result_file
  \immediate\openout \result_file=niceverb.doc %% JUST HERE

\newif\if_package_code_     \_package_code_false
% \newif\if_empty_code_lines_ \_empty_code_lines_false
  %% <- FAILED SO FAR
% \let\maybe_result_empty_line\empty
% \def\result_empty_line{^^J}

\def\write_result{\immediate\write\result_file}

\def\process_file{%
  %% This macro here to avoid category changes
  %% affecting the present code
  \begingroup
    \let\do\@makeother \dospecials
    %% from docstrip.tex:
    \@makeother\^^A\@makeother\^^K\endlinechar\m@ne
    %% <- cf. TeXbook "extended keyboards" up-/downarrow
    %%    -> "math specials", cf. "space specials"
    \@makeother\^^I% ASCII horizontal tab -- guessed!? ^^L!?
%    \tracingmacros=1
    \loop \ifeof\@inputcheck \else
      \read\@inputcheck to \InputLine
      \expandafter \process_line \InputLine ======&%% primitive version
```

```
    \repeat
  \endgroup}

\def\process_line#1===#2===#3&{%% may be preferable to wiki.sty
  \ifx$#2$%
    \ifx$#1$%
%      \show\InputLine
      \if_package_code_
%         \if_empty_code_lines_
%           \write_result{}\_empty_code_lines_false
%         \fi
      \else
          \write_result{}%
%         \let\maybe_result_empty_line\result_empty_line
      \fi
    \else
%      \show\InputLine
      \if_package_code_
%         \_empty_code_lines_true
      \else
        \write_result{\string\begin{packagecode}}%
        \_package_code_true
      \fi
      \write_result{#1}%
%       \write_result{\maybe_result_empty_line #1}%
%       \let\maybe_result_empty_line\empty
    \fi
  \else
    \write_result{%
      \string\end{packagecode}^^J^^J%
      \string\subsection{\ignorespaces#2\unskip}^^J}%
    \_package_code_false
%     \_empty_code_lines_false
  \fi}

\process_file

\write_result{\string\end{packagecode}}
\closein\@inputcheck \immediate\closeout\result_file

\endinput
```

```
\stop
  %% TODO or \ThankYou; which NICEVERB.TEX may redefine into \endinput
  %%      or \ThankYou issues \endinput if jobname ...
  %%      the whole file may be enclosed in \begingroup ... \endgroup
  %%      ... this is a "driver file"!?

TODO make own listing environment like this -- own linewise processing,
     without verbatim.sty
```