niceverb.sty

—

Minimizing Markup for Documenting LaTeX packages*

Uwe Lück†

January 26, 2011

Abstract

niceverb.sty provides very decent syntax (through active characters) for describing LaTeX packages and the syntax of macros conforming to LaTeX syntax conventions.

Contents

*This document describes version v0.42 of niceverb.sty as of 2011/01/26.

†http://contact-ednotes.sty.de.vu

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. If you use makedoc.sty additionally, 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.

Both packages 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—⟨*jobname*⟩`.sty`), you just "`latex`" the manual file (maybe some `.tex` file—⟨*jobname*⟩`.tex`) and get instantly the corresponding updated documentation.

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

1.2 Acknowledgement/Basic Ideas

Four ideas of Stephan I. Böttcher's in documenting his lineno 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 these commands in the source).[2]

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

[2] The corresponding part of the "present work" is makedoc.sty.

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.

4. ‘`<meta-variable>`’ produces ‘⟨*meta-variable*⟩’. (‘`\lessthan`’ stores the original ‘`<`’.)

1.3 The Commands and Features of niceverb

Actually, it is the main purpose of niceverb to save you from “commands” ...

Single quotes ‘`‘`’, ‘`’`’, “less than” `<` (accompanied with `>`), the “vertical” `|`, 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 (‘`[`⟨*opt-arg*⟩`]`’) or curly braces (‘`{`⟨*mand-arg*⟩`}`’), 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’*.

Outside macro arguments, you obtain the same by typing

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

Details:

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

This works due to a sloppy variant `\NVerb` 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.[3] ‘`\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.

[3] On the other hand, `\NVerb` is more *careful* with niceverb's special characters.

Single quotes (left/right) for "short verb:" The package "assumes" that *quoting* refers to *code*, therefore ‘‘`foo`’’ is typeset as ‘`foo`’, or (generally) ‘‘⟨*content*⟩’’ turns ⟨*content*⟩ into meta-code with the meta-variable feature as above. This somewhat resembles the `\MakeShortVerb` feature of doc.sty. You can "abuse" our feature just to get typewriter typeface.

Problems with this feature will typically arise 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 or so. `\verb` fails in the same situation, for the same reason. ‘`&`’ (`\footnote{&\bar`⟨*remaining*⟩`}`) or "auto mode" (see below) may then work better.[4] More generally, the quoting feature still works in macro arguments in the sense that you then have to mark difficult characters with `&` (simply as short for `\string`). However, it still won't work with curly braces that don't follow a command name (such *pairs* of braces will simply get lost, *single* braces will give errors or so).

Double quotes and apostrophes should still work the usual way. For difficult cases, you can still use the standard `\verb` command from LaTeX. To get *usual* single quotes, you can use their standard substitutes `\lq` and `\rq`, or for pairs of them, `\qtd{`⟨*text*⟩`}` in place of `\lq`␣⟨*text*⟩`\rq`—or even `\lq`␣⟨*text*⟩`\rq\`␣. To get single quotes around some verbatim ⟨*verb*⟩, often `\qtd{&`⟨*verb*⟩`}` works. It is for this reason that I have refrained from different solutions as in newverbs (so far).

Single right quotes for `\textsf`: Package names are (by some convention I often yet not always see working) typeset with `\textsf`; it was natural to use a remaining case of using single quotes for abbreviating

```
\textsf{⟨text⟩}
```

by ’⟨*text*⟩’. This idea of switching fonts continues font switching of wiki.sty which uses the syntax for editing *Wikipedia* pages (font switching by sequences of right single quotes).

Verticals for setting-off command descriptions: `|`⟨*code*⟩`|` works like ‘‘⟨*code*⟩’’ except putting the result into a *framed box* (just as all around here)—or something else that you can achieve using some *hooks* described with the implementation. There are variants like `\cmdboxitem|`⟨*code*⟩`|`.

Ampersand shows command syntax &c. 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, in macro arguments this does not work with *private letters* (`@` and `_` here), for this case, use `\cs{`⟨*characters*⟩`}` or `\cstx{`⟨*characters*⟩`}`⟨*parameters*⟩.[5]

[4] `\bar` indeed!

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

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 \MakeNormal\& and turn the present special meaning on again by

\MakeActive\& or \MakeActiveLet\&\CmdSyntaxVerb

You could also redefine (\renewcommand) \descriptionlabel using \CmdSyntaxVerb (the "normal command" that is equivalent to &, its "permanent alias") so \item[\foo] works as wanted.

Another feature of niceverb's & is getting (some of the) special characters (as listed in the standard macro \dospecials) verbatim in arguments (where \verb and the like fail). It just acts similarly as TEX's (as listed in the standard macro \dospecials) verbatim in arguments (where \verb and the like fail). It just acts similarly as TEX's primitive \string (which it actually invokes—cf. discussion on the left quote feature above).

"Auto mode" typesets commands verbatim unless ... In "auto mode," the backslash '\' is an active character that builds a command 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, or add ⟨*macros*⟩ to it by \AddToNoVerbList{⟨*macros*⟩} 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. \AutoCmdInput{⟨*file*⟩} is probably most important.

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! (So the general syntax is #⟨*digit*⟩.)

Escaping from niceverb (generally). To get rid of the functionality of some active character ⟨*char*⟩ ('&', single quote, ampersand, hash mark—not "auto mode," see above) here, use \MakeNormal\⟨*char*⟩—may be within a group. To revive it again, use \MakeActive\⟨*char*⟩. This may fail when

[6] This idea appeared 2009 on the LATEX-L mailing list. It may be wrong, as I have sometimes experienced ...

a different package overtook the active ⟨*char*⟩ (but I expect more failures then), in this case `\MakeActiveLet\⟨char⟩\⟨perm-alias⟩` revives the niceverb meaning of ⟨*char*⟩ where \⟨*perm-alias*⟩ is the "permanent alias" for that active ⟨*char*⟩ according to the documentation below. E.g., `\LQverb` is the "permanent alias" for active single left quote, niceverb activates it by `\MakeActiveLet\'\LQverb`.—You can turn off niceverb syntax *altogether* by `\noNiceVerb` and revive it by `\useNiceVerb` (without "auto mode").

Right Quotes: Disabling/reviving replacement of `\textsf` by single right quotes requires

`\nvRightQuoteNormal` or `\nvRightQuoteSansSerif`

respectively.

1.4 Examples

The file mdoccorr.cfg providing some `.txt`→LaTeX functionality—i.e., typographical corrections—documents itself using niceverb syntax. Its code and the documentation that is typeset from it are in the 'examples' section of makedoc.pdf.—Moreover, the documentation niceverb.pdf of niceverb.sty was typeset from niceverb.tex and niceverb.sty using niceverb syntax, likewise fifinddo.pdf and makedoc.pdf. The example of niceverb shows the most frequent use of the `&` feature.

nicetext bundle release v0.4 contains a file substr.tex that should typeset the documentation of the version of Harald Harders' substr.sty[7] that your TeX finds first, as well as arseneau.tex typesetting a few packages by Donald Arseneau. The outcomes (with me) are substr.pdf and arseneau.pdf. These are the first applications of niceverb's "auto mode" to (unmodified) third-party package files. (I also made a more ambitious documentation of Donald Arseneau's import.sty v3.0 before I found that CTAN already has a nicely typeset documentation of import.sty v5.2.)

1.5 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. Especially, the feature of

   `''⟨text⟩''`

   replacing `\textit{⟨text⟩}` or `\emph{⟨text⟩}` may be considered missing.

2. Font switching or horizontal spacing may fail in certain situations. You can correct spacing by '\␣'.

[7] `http://ctan.org/pkg/substr`

3. The "vertical" character '|' produces inline boxes only at present. It might as well provide a version of the `decl` tabular environment of ltxguide.cls. The inline boxes badly deal with long command names and many arguments. Doubled verticals could ensure the `decl` mode. Moreover, such a box might issue an *index* entry.

4. One may have *opposite* ideas about using quotes—maybe rather "⟨*code*⟩" should typeset ⟨*code*⟩ *verbatim*. There might be a package option for this. If ordinary ‘‘‘⟨*text*⟩"' still should work, awful tricks as now with the right quote feature would be needed.

5. "auto mode" seems not to work in section titles. (2011/01/26)

6. Certain difficulties with typesetting code in macro arguments may be overcome easily using ε-TEX features, I need to find out ...

2 Implementation of the Markup Syntax

```
1  \NeedsTeXFormat{LaTeX2e}[1994/12/01]
2  \ProvidesPackage{niceverb}[2011/01/26 v0.42
3                             minimize doc markup (UL)]
4
5  %% Copyright (C) 2009-2011 Uwe Lueck,
6  %% http://www.contact-ednotes.sty.de.vu
7  %% -- author-maintained in the sense of LPPL below --
8  %%
9  %% This file can be redistributed and/or modified under
10 %% the terms of the LaTeX Project Public License; either
11 %% version 1.3a of the License, or any later version.
12 %% The latest version of this license is in
13 %%     http://www.latex-project.org/lppl.txt
14 %% We did our best to help you, but there is NO WARRANTY.
15 %%
16 %% Please report bugs, problems, and suggestions via
17 %%
18 %%   http://www.contact-ednotes.sty.de.vu
19 %%
```

2.1 Switching Category Codes

v0.3 introduces `\AssignCatCodeTo` and `\MakeNormal`.

`\CatCode{\`⟨*character*⟩`}` (or simply `\CatCode\`⟨*character*⟩) saves one token per use and works when the category code of ‘‘’ ("single left quote") has changed.

```
20 \newcommand*{\CatCode}{\catcode`}
21 % \newcommand*{\CatCode}[1]{\catcode`#1 } %% no better 2010/02/27
```

With `\AssignCatCodeTo{⟨number⟩}{\⟨char⟩}`, `\CatCode` may still be useful for displaying (debugging or playing). Note that `\⟨char⟩` is the *second* argument here.

```
\newcommand*{\AssignCatCodeTo}[2]{\catcode`#2=#1\relax}
```

`\MakeLetter\⟨char⟩` is used for *private letters*, i.e., to allow ⟨*char*⟩ in "internal", non-user control sequences (*TEXbook* Chap. 3). `\MakeOther` is just a different implementation of LaTeX's `\@makeother`.

```
\newcommand*\MakeLetter{\AssignCatCodeTo{11}}
\def        \MakeOther {\AssignCatCodeTo{12}}
```

... overriding fifinddo if ...

`\MakeActive\⟨char⟩` just revives the meaning of ⟨*char*⟩ it had most recently (as an `\active` character ... maybe "Undefined control sequence" unless ...) This is fine for reviving niceverb functionality after having disabled it by `\MakeNormal`—provided no other package used ⟨*char*⟩ actively in the meantime ...

```
\providecommand*{\MakeActive}{\AssignCatCodeTo\active} %% used v0.3
```

We take a copy `\MakeActiveHere` of `\MakeActive` as the latter may become a dangerous thing for compatibility with hyperref.

```
\@ifdefinable\MakeActiveHere{%
    \let\MakeActiveHere\MakeActive}
  %% <- TODO aliascid + elsewhere 2010/03/12
```

`\MakeActiveLet\⟨char⟩⟨macro name⟩` activates ⟨*char*⟩ and then gives it the meaning of ⟨*macro name*⟩.

```
\newcommand*{\MakeActiveLet}[2]{%% cf. \@sverb/\do@noligs (doc.sty)
  \MakeActiveHere#1%        %% 2010/03/12
  \begingroup
    \lccode`\~`#1\relax \lowercase{\endgroup \let~#2}}
```

We take a copy `\MakeActiveLetHere` as well.

```
\@ifdefinable\MakeActiveLetHere{%
    \let\MakeActiveLetHere\MakeActiveLet}
```

We use the "underscore" as a private letter (the LaTeX2 Project Team likes it as well). Its usual meaning can be restored by `\MakeNormal\_` For restoring the usual category codes of TeX's special characters later, we store them now. (I.e., these characters are listed in the macro `\dospecials` that expands to

```
\do\ \do\\\do\{\do\}\do\$\do\&\do\#\do\^\do\_\do\%\do\~
```

their category codes are 10, 0, 1, 2, 3, 4, 6, 7, 8, 14, 13 respectively; "end of line", "ignored", "letter", "other", and "invalid" are missing—cf. *TEXbook* Chap. 7.)

```
\def\do#1{\expandafter
  \chardef \csname normal_catcode_\string#1\expandafter \endcsname
    \CatCode#1\relax}
\dospecials
```

Tests: "normal category code" of \ is 0, "normal category code" of $ is 3; "normal category code" of & is 4.[8]

Here we switch to the "underscore" as a "letter" indeed (for the rest of the package):

```
\MakeLetter\_

% \newcommand*{\make_iii_other}{\MakeOther\\\MakeOther\{\MakeOther\}}
  %% <- replaced 2009/04/05
```

\MakeNormal\⟨*char*⟩ saves you from remembering ...

```
\newcommand*{\MakeNormal}[1]{%
  \@ifundefined{\norm_catc_str#1}%
             {\MakeOther#1}%
             {\AssignCatCodeTo{\csname\norm_catc_str#1\endcsname}#1}}
\newcommand*{\norm_catc_str}{normal_catcode_\string}
  %% TODO add ^^I and ^^M
```

We take a copy \MakeNormalHere of \MakeNormal as with \MakeActive.

```
\@ifdefinable\MakeNormalHere{\let\MakeNormalHere\MakeNormal}
```

2.2 Robustness by \IfTypesetting

It seems we need some own ways to achieve various compatibilities—using \IfTypesetting{⟨*if*⟩}{⟨*unless*⟩}. It also saves some \expandafters.

```
\providecommand*{\IfTypesetting}{%
%     \relax
```

This \relax suppressed ligatures of single right quotes!

```
    \ifx \protect\@typeset@protect
         \expandafter \@firstoftwo
    \else \expandafter \@secondoftwo \fi}
```

2.3 \NVerb

\begin_min_verb is a beginning shared by some macros here. It begins like LaTeX's \verb, apart from the final \tt.

```
\newcommand*{\begin_min_verb}{%
  \relax \ifmmode \hbox \else \leavevmode\null \fi
  \bgroup \tt}
```

[8] LaTeX's \nfss@catcodes is similar, but it makes space-like characters ignored. Also cf. ltfinal.dtx. TODO: \RestoreNormalCatcodes.

\NVerb⟨*char*⟩⟨*code*⟩⟨*char*⟩

```
\newcommand*{\NVerb}{%
    \_no_nice_meta_verb_false \nice_maybe_meta_verb}
```

\HardNVerb⟨*char*⟩⟨*code*⟩⟨*char*⟩ does not recognize meta-variables:

```
\newcommand*{\HardNVerb}{%
    \_no_nice_meta_verb_true \nice_maybe_meta_verb}
\newif\if_no_nice_meta_verb_
\newcommand*{\nice_maybe_meta_verb}[1]{%
```

Mainly avoid \verb's noligs list which overrides definitions of some active characters, while cmtt doesn't have any ligatures anyway.

```
  \IfTypesetting{%
    \begin_min_verb
      \let\do\MakeOther \dospecials
```

Turn off niceverb specials:

```
      \MakeOther\|\MakeOther\`\MakeOther\'%
      \if_no_nice_meta_verb_ \MakeOther\<%
      \else                  \MakeActiveLet\<\MetaVar   %% 2010/12/31
      \fi
      \MakeActiveLetHere #1\niceverb_egroup
      \verb@eol@error %% TODO change message 2009/04/09
  }{\string\NVerb \string#1}}
```

2009/04/11: about `etc.` [preceding a box!? 2010/03/14]

```
\newcommand*{\niceverb_normal_egroup}{\egroup \ifmmode\else\@\fi}
\@ifdefinable\niceverb_egroup
    {\let\niceverb_egroup\niceverb_normal_egroup}
```

2.4 Single Quotes Typeset Meta-Code

\LQverb will be a "permanent alias" for the active left single quote.

The verbatim feature must not act when another single left quote is ahead—we assume a double quote is intended then (thus the left quote feature does not allow to typeset something verbatim that starts with a single left quote). Rather, double quotes should be typeset then. In page headers, a \protect may be in the way. (A hook for \relaxing certain things in \markboth and \markright would have been an alternative.)

```
\MakeActive\`
  \newcommand*{\LQverb}{%
    \IfTypesetting{\lq_double_test}{\protect`}}
\MakeOther\`
\newcommand*{\lq_double_test}{%
```

This test settles the next catcode, so better switch to "other" in advance (won't harm if left quote isn't next):

```
    \begingroup
      \let\do\MakeOther \dospecials
      \MakeOther\|%% 2010/03/09!
      \futurelet\let_token \lq_double_decide}
  \newcommand*{\lq_double_decide}{%
    \ifx\let_token\LQverb
      \endgroup
      ``\expandafter \@gobble
```

Corresponding right quotes will become "other" due to having no space at the left. TODO to be changed with wiki.sty.

```
    \else
      \ifx\let_token\protect
        \expandafter\expandafter\expandafter \lq_double_decide_ii
      \else
        \endgroup
        \expandafter\expandafter\expandafter \NVerb
          \expandafter\expandafter\expandafter \'%
      \fi
    \fi}
```

`\lq_double_decide_ii` continues test behind `\protect`.

```
  \newcommand*{\lq_double_decide_ii}[1]{%
      \futurelet\let_token \lq_double_decide}
```

2.5 Ampersand (or \cstx) Typesets Meta-Code

`\CmdSyntaxVerb` will be a permanent alias for the active `&`.

```
  \MakeActive\&
    \newcommand*{\CmdSyntaxVerb}{%
      \IfTypesetting{%
        \begin_min_verb
```

v0.3 moves the previous line from `\cmd_syntax_verb` where it is too late to establish private letters according to next line which was in `\begin_min_verb` earlier—an important bug fix!

```
        \MakeLetter\@\MakeLetter\_%
        \cmd_syntax_verb
      }{\protect&\string}}
  \MakeNormal\&
  \newcommand*{\cmd_syntax_verb}[1]{%
    \string#1\futurelet\let_token \after_cs}
```

However, `&` (or `\CmdSyntaxVerb`) may fail with private letters (there should be a hook for them), especially in *macro arguments* and with hyperref in titles of *sections bearing* `\label`*s*, so we provide something like `\cs{`⟨*characters*⟩`}` from doc.sty.

```
\DeclareRobustCommand*{\cs}[1]{%
  \begin_min_verb \backslash_verb #1\egroup}
\newcommand*{\backslash_verb}{\char`\\}
```

Moreover, typing `&\par` in "short" *macro arguments* fails, you better type `\cs{par}` then. Likewise, `\cs{if`⟨*letters*⟩`}` and `\cs{fi}` is safer in case you want to skip some part of the documentation (e.g., a package option skips commented code) by `\if`⟨*letters*⟩`\fi`. Finally, there will be PDF bookmarks support for `\cs` rather than for a real `&` or `\CmdSyntaxVerb` analogue like `\cstx{`⟨*charcters*⟩`}*[`⟨*opt*⟩`]{`⟨*mand*⟩`}` as follows.

```
\DeclareRobustCommand*{\cstx}[1]{%              %% corr. 2010/03/17
  \begin_min_verb \backslash_verb #1\futurelet\let_token \after_cs}
\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}{%
    \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}
  %% not needed with \Auto... OTHERWISE useful in args!
```

As latex.ltx has `\endgraf` as a permanent alias for the primitive version of `\par` and `\endline` for `\cr`, we offer `\endcell` as a replacement for the original `&`:

```
\let\endcell&
```

2.6 Escape Character Typesets Meta-Code

`\BuildCsSyntax` will be a permanent alias for the active escape character.

```
\DeclareRobustCommand*{\BuildCsSyntax}{%
  \futurelet\let_token \build_cs_syntax_sp}
\newcommand*{\build_cs_syntax_sp}{%
  \ifx\let_token\@sptoken
    \@%                                 %% 2010/12/30
  \else %% TODO ^^M!?
```

```
142       \expandafter \start_build_cs_syntax
143     \fi}
144   \newcommand*{\start_build_cs_syntax}[1]{%
145     \edef\string_built{\string#1}%
```

`#1` may be active.—With Donald Arseneau's import.sty (e.g.), '_' may be needed to be `\active` with the meaning of `\textunderscore`, therefore restoring its category code needs some more care than with v0.32 and earlier:

```
146     \edef\before_build_cs_sub{\the\CatCode\_}%
147     \MakeLetter\_\MakeLetter\@%% CAUTION, cf. ...
148     \test_more_cs}
149   \newcommand*{\test_more_cs}{%
150     \futurelet\let_token \decide_more_cs}
151   \newcommand*{\decide_more_cs}{%
152     \ifcat\noexpand\let_token a\expandafter \add_to_cs
153     \else
154   %     \MakeNormalHere\_
```

Restoring '_' more carefully with v0.4 (`\begingroup` ... `\endgroup`!?):

```
155       \CatCode\_\before_build_cs_sub
156       \MakeOther\@%
157       \expandafter \in@ \expandafter
158         {\csname \string_built \expandafter \endcsname
159           \expandafter}\expandafter{\niceverbNoVerbList}%
160       \ifin@
161         \csname \string_built
162           \expandafter\expandafter\expandafter \endcsname
163       \else
164         \begin_min_verb \backslash_verb\string_built
165           \expandafter\expandafter\expandafter \test_more_verb
166       \fi
167     \fi}
168     %% TODO such \if nestings with ifthen!?
169    %% cf.:
170   %  \let\let_token,\typeout{\meaning\let_token}
171     %% TEST TODO fuer xspace!? (\ifin@)
172   \newcommand*{\add_to_cs}[1]{%
173     \edef\string_built{\string_built#1}\test_more_cs}
```

`\AutoCmdSyntaxVerb` starts, `\EndAutoCmdSyntaxVerb` *ends* "auto mode."

```
174   \newcommand*{\AutoCmdSyntaxVerb}{%
175       \MakeActiveLetHere\\\BuildCsSyntax}
176   \newcommand*{\EndAutoCmdSyntaxVerb}{\CatCode\\\z@}
```

`\NormalCommand{`⟨*characters*⟩`}` executes \⟨*characters*⟩ in "auto mode."

```
177   \newcommand*{\NormalCommand}{} \let\NormalCommand\@nameuse
```

Once I may want to use this feature in *Wikipedia*-like section titles as supported by makedoc, yet I cannot really apply the present feature soon, so this must wait ... (There is a special problem with `\newlabel` and hyperref ...)

Former tests:

```
178  %  \futurelet\LetToken\relax \relax
179  %  \show\LetToken \typeout{\ifcat\noexpand\LetToken aa\else x\fi}
```

`\niceverbNoVerbList` is the list of macros that will be *executed* instead of being typeset.

```
180  \newcommand*{\niceverbNoVerbList}{%
181    \begin\end\item\verb\EndAutoCmdSyntaxVerb\NormalCommand
182    \section\subsection\subsubsection} %% TODO!?
```

`\AddToMacro{\niceverbNoVerbList}{`⟨*macros*⟩`}` can be used to add ⟨*macros*⟩ to that list.

```
183  \providecommand*{\AddToMacro}[2]{%   %% TODO move to ... 2010/03/05
184    \expandafter \def \expandafter #1\expandafter {#1#2}}
185    %% <- was very wrong 2010/03/18
```

Hey, or just `\AddToNoVerbList{`⟨*macros*⟩`}`:

```
186  \newcommand*{\AddToNoVerbList}{\AddToMacro\niceverbNoVerbList}
```

"Auto mode" probably ain't mean a thing if it ain't invoked using

`\AutoCmdInput{`⟨*file*⟩`}`

for typesetting ⟨*file*⟩ in "auto mode:"

```
187  \newcommand*{\AutoCmdInput}[1]{%
188      \begingroup
189        \AddToMacro\niceverbNoVerbList{\ProvidesFile}%
190        %% <- removed '\endinput', will be code! 2010/04/05
191        \AutoCmdSyntaxVerb
192        \input{#1}%
193        \EndAutoCmdSyntaxVerb
194      \endgroup
195  }
```

2.7 Meta-Variables

`\MetaVar<`⟨*var-id*⟩`>` will be a permanent alias for the active '<'.

```
196  \def\MetaVar#1>{%
197    \mbox{\normalfont\itshape $\langle$#1\/$\rangle$}}
198    %% TODO offer without angles as well
```

As opposed to ltxguide.cls, this works outside verbatim as well.

2.8 Hash Mark is Code

`\HashVerb⟨digit⟩` will be a permanent alias for the active hash mark.

```
\newcommand*{\HashVerb}[1]{{\tt\##1}}
```

2.9 Single Right Quotes for \textsf

`\RQsansserif` will be a permanent alias for the active single right quote.

The basic problem with the "single right quote feature" is that a single right quote may be meant to be an apostrophe. This is certainly the case at the right of a letter. On the other hand, we assume that it is *not* an apostrophe (i) in vertical mode (opening a new paragraph), (ii) after a horizontal skip.

For page headers, in expanding without typesetting, the expansion of `\RQsansserif` must contain another active single right quote.

```
\MakeActive\'
  \newcommand*{\RQsansserif}{%
    \IfTypesetting{\niceverb_rq_sf_test}{\protect'}}
\MakeOther\'
```

Another macro just to avoid more sequences of `\expandafter`:

```
\newcommand*{\niceverb_rq_sf_test}{%
  \ifhmode
    \ifdim\lastskip>\z@
      \expandafter\expandafter\expandafter \DoRQsansserif
    \else
      \ifnum\niceverb_spacefactor
        \expandafter\expandafter\expandafter\expandafter
          \expandafter\expandafter\expandafter
            \DoRQsansserif
      \else '\fi
    \fi
  \else \ifvmode
    \expandafter\expandafter\expandafter \DoRQsansserif
    \else '\fi
  \fi}
```

`\DoRQsansserif` is *another* (possible) alias for the active single right quote, see below.

```
\MakeActive\'
  \@ifdefinable\DoRQsansserif
    {\def\DoRQsansserif#1'{\textsf{#1}}}
\MakeOther\'
```

The following cases are typical and cannot be decided by the previous criteria: (i) parenthesis, (ii) footnotes and after "horizontal" environments like `\[⟨math⟩\]`, (iii) section titles, (iv) `\noindent`. We introduce some dangerous tricks—redefinitions of LaTeX's internal `\@sect` and of TeX's primitives

\noindent and \ignorespaces as well as by a signal \spacefactor value of 1001. In page headers, LaTeX equips the single right quote with the meaning of \active@math@prime which must be overridden.

```
\newcommand*{\nvAllowRQSS}{%
    \MakeActiveLetHere\'\RQsansserif
    \niceverb_ignore}                    %% 2010/03/16
```

These and the entire right quote functionality are activated by

\nvRightQuoteSansSerif and disabled by \nvRightQuoteNormal

—at \begin{document}—where we collect previous settings—or later:

```
\AtBeginDocument{%
    \edef\before_niceverb_parenthesis{\the\sfcode`\(}%
    \let \before_niceverb_ignore   \ignorespaces    %% 2010/03/16
    \let \before_niceverb_sect     \@sect
    \let \before_niceverb_noindent \noindent}       %% 2010/03/08
```

We assume that \@sect has the same parameters there as in LaTeX (even if redefined by another package, like hyperref).

```
\def\niceverb_sect#1#2#3#4#5#6[#7]#8{%
    \before_niceverb_sect{#1}{#2}{#3}{#4}{#5}{#6}%
                         [{\protect\nvAllowRQSS #7}]%
                          {\protect\nvAllowRQSS #8}}
```

2010/03/20:

```
\newcommand*{\niceverb_spacefactor}{\spacefactor=1001\relax}
\newcommand*{\niceverb_noindent}{%
    \before_niceverb_noindent \niceverb_spacefactor}
\newcommand*{\niceverb_ignore}{%
    \ifhmode \niceverb_spacefactor \fi \before_niceverb_ignore}
```

Here are the main switches:

```
\newcommand*{\nvRightQuoteSansSerif}{%
    \MakeActiveLet\'\RQsansserif
    \sfcode`\(=1001   %% enable in parentheses 2009/04/10
```

I also added \sfcode`/=1001 in the preamble of makedoc.tex.

```
%     \let\@footnotetext\niceverb_footnotetext
    \let\ignorespaces\niceverb_ignore       %% 2010/03/16
    \let\@sect\niceverb_sect
    \let\noindent\niceverb_noindent}        %% 2010/03/08
\newcommand*{\nvRightQuoteNormal}{%
    \MakeNormal\'%                                  %% 2010/03/21
    \sfcode`\(=\before_niceverb_parenthesis\relax
    \let\ignorespaces\before_niceverb_ignore        %% 2010/03/16
    \let\@sect\before_niceverb_sect
    \let\noindent\before_niceverb_noindent}         %% 2010/03/08
```

`\nvAllRightQuotesSansSerif` (after `\begin{document}`!) forces the `\textsf` feature *without* testing for apostrophes. You then must be sure—DANGER! CARE!—to use '`\rq`' only for obtaining an apostrophe and the double quote character '`"`' for closing double quotes, or our `\dqtd{`⟨*text*⟩`}` for the entire quoting.

```
\newcommand*{\nvAllRightQuotesSansSerif}{%
    \nvRightQuoteNormal
    \MakeActiveLet\'\DoRQsansserif}
```

I started v0.31 (signal `\sfcode`=1000, lowercase letters get `\sfcode`=1001) because `\href{http://ctan.org/pkg/`⟨*pkg*⟩`}{`⟨*pkg*⟩`}` failed. However, what I actually needed was `\ctanpkgref{`⟨*pack-name*⟩`}`:

```
% \DeclareRobustCommand*{\ctanpkgref}[1]{%
%     \href{http://ctan.org/pkg/#1}{\textsf{#1}}}
```

... moves to texlinks.sty 2011/01/24.

2.10 Command-Highlighting Boxes

With v0.3, we include one kind of command syntax boxes whose ⟨*content*⟩ is (in niceverb syntax) delimited as `|`⟨*content*⟩`|`.

```
\newsavebox\niceverb_savebox
```

`\GenCmdBox{`⟨*char*⟩⟨*content*⟩⟨*char*⟩`}` works like `\NVerb`⟨*char*⟩⟨*content*⟩⟨*char*⟩ except putting the latter's result into a framed (or coloured or ...) box.

```
\newcommand*{\GenCmdBox}  {\_no_nice_meta_verb_false \gen_cmd_box}
```

`\HardVerbBox` is a variant of `\GenCmdBox` with the meta-variable feature disabled (for the documentation of the present package).

```
\newcommand*{\HardVerbBox}{\_no_nice_meta_verb_true  \gen_cmd_box}
\newcommand*{\gen_cmd_box}{%
  \bgroup
    \let\niceverb_egroup\nice_collect_verb_egroup
    \global %% TODO!? for \cmdboxitem 2010/03/15
    %% <- TODO replace \niceverb_egroup by parameter,
    %%         save one nesting level 2010/03/15
    \setbox\niceverb_savebox \hbox\bgroup
      \if_no_nice_meta_verb_
            \expandafter \HardNVerb
      \else \expandafter \NVerb     \fi}
\newcommand*{\nice_collect_verb_egroup}{%
      \egroup \egroup
    \ifvmode \expandafter \VerticalCmdBox
    \else    \ifmmode \hbox \fi
             \expandafter \InlineCmdBox \fi
    {\box\niceverb_savebox}%
  \niceverb_normal_egroup}
```

`\nvCmdBox` will be the permanent alias for '`|`'.

```
\newcommand*{\nvCmdBox}{\GenCmdBox\|}
```

`\VerticalCmdBox{`⟨*content*⟩`}` may eventually start a `decl` environment as in ltxguide.cls, looking ahead for another '`|`' in order to (perhaps) append another row. Another possibility is first to do some

```
\if@nobreak\else␣\pagebreak[2]\fi
```

etc. and then invoke `\InlineCmdBox`. The user can choose later by some `\renewcommand`. We do the perhaps most essential thing here (again cf. `\begin_min_verb`):

```
\newcommand*{\VerticalCmdBox}{\leavevmode\null\InlineCmdBox}
```

The command declaration boxes in the documentation of Nicola Talbot's datatool would be an especially nice realization of `\VerticalCmdBox`.

`\InlineCmdBox{`⟨*content*⟩`}`, according to our idea, should not change baseline skip, even with some `\fboxsep` and `\fboxrule`. (However, it may be a good idea to increase the overall normal baseline skip.) We therefore replace actual height and depth of the content by the height and depth of math parentheses.

```
\newcommand*{\InlineCmdBox}[1]{%
  \bgroup
```

... needed in math mode with `\begin_min_verb`.

```
    \fboxsep 1pt
    \kern\SetOffInlineCmdBoxOuter
    \smash{\SetOffInlineCmdBox{\kern\SetOffInlineCmdBoxInner
                               \InlineCmdBoxArea{#1}%
                               \kern\SetOffInlineCmdBoxInner}}%
    \mathstrut
    \kern\SetOffInlineCmdBoxOuter
  \egroup
}
```

The default choice for `\SetOffInlineCmdBox` is `\fbox`:

```
\@ifdefinable\SetOffInlineCmdBox{\let\SetOffInlineCmdBox\fbox}
```

You can `\renewcommand` it to change `\fboxsep`, `\fboxrule` etc. or to use a `\colorbox` with the color package, e.g., I used the following setting so far:

```
\RequirePackage{color}
\renewcommand*{\SetOffInlineCmdBox}
              {\colorbox[cmyk]{.1,0,.2,.05}}
```

`\SetOffInlineCmdBoxInner` enables controlling the inner horizontal space to the box margin independently of `\fboxsep`.

```
\newcommand*{\SetOffInlineCmdBoxInner}{-\fboxsep\thinspace}
```

This choice is inspired by `\cstok` for "boxed" things in Knuth's manmac.tex which formats *The TEXbook*.

`\SetOffInlineCmdBoxOuter` allows that the box hangs out into the margin horizontally. We set it to 0 pt as default (it is a macro only, for a while).

```
\newcommand*{\SetOffInlineCmdBoxOuter}{\z@}
```

The height and depth of the frame should be the same for all inline boxes, we think. The present choice `\InnerCmdBoxArea` for the spacing respects code characters rather than the height and depth of the angle brackets that surround meta-variable names.

```
\newcommand*{\InlineCmdBoxArea}[1]{%
    \smash{#1}\vphantom{gjpq\backslash_verb}}
```

`\cmdboxitem|`⟨*content*⟩`|` is another variant of `\GenCmdBox`. It should replace `\item[`⟨*content*⟩`]` in the `description` environment.

```
\newcommand*{\cmdboxitem}{%
  \bgroup
    \let\niceverb_egroup\cmd_item_egroup
    \global %% TODO!? 2010/03/15
    \setbox\niceverb_savebox \hbox\bgroup
      \NVerb}
\newcommand*{\cmd_item_egroup}{%
      \egroup \egroup \egroup
  \item[\InlineCmdBox{\box\niceverb_savebox}]}
```

2.11 When niceverb Gets Nasty

These things are new with v0.3.

2.11.1 Quotes

In order to get *real* single quotes, you could use `\lq␣`⟨*text*⟩`\rq`, maybe appending a `\␣`, but the code `\qtd{`⟨*text*⟩`}` may look better and be easier to type.

```
\newcommand*{\qtd}[1]{‘#1’}
```

However, here we get the problem that the left quote in `\qtd{‘`⟨*code*⟩`’}` will be unable to switch into verbatim mode entirely—then use `&`, e.g., '`\qtd{&&}`' typesets "`&`", i.e., the ampersand in single (non-verbatim) quotes.

```
% TODO \qtdverb!? alternative meaning for \LQverb!? 2010/03/06
%      rather rare, & takes less space              2010/03/09
```

`\dqtd{`⟨*text*⟩`}` can be used for enclosing in *double* quotes with the dangerous `\nvAllRightQuotesSansSerif` (see above).

```
\newcommand*{\dqtd}[1]{‘‘#1"}
```

2.11.2 hyperref

This is for/about compatibility with the hyperref package. (One preliminary thing: in doubt, don't load niceverb earlier than hyperref.)

We need some substitutions for PDF bookmarks with hyperref. We issue them at `\begin{document}` when we know if hyperref is at work.[9]

```
\AtBeginDocument{%
  \@ifpackageloaded{hyperref}{%
    \newcommand*{\PDFcstring}{%         %% moved here 2010/03/09
      \134\expandafter\@gobble\string}% %% ASCII octal encoding
    \pdfstringdefDisableCommands{%
      \let\nvAllowRQSS\empty            %% not \relax 2010/03/12
      %% 2010/03/12
      \MakeActiveLetHere\‘\lq \MakeActiveLetHere\’\rq
      \MakeActiveLetHere\&\PDFcstring
      \def\cs{\134}%                    %% 2010/03/17
    }%
```

Moreover, in order to avoid spurious `Label(s) may have changed` with hyperref, a single right quote must be *read* as active by a `\newlabel` if and only if it has been active when `\@currentlabelname` was formed.[10] as `\active`. We use `\protected@write` as this cares for `\nofiles`. `\@auxout` may be `\@partaux` for `\include`.

```
    \newcommand*{\niceverb_aux_cat}[2]{%                %% 2010/03/14
        \protected@write\@auxout{}{\string#1\string#2}}%
    \renewcommand*{\MakeActive}[1]{%
        \MakeActiveHere#1%
        \niceverb_aux_cat\MakeActiveHere#1}%
    \renewcommand*{\MakeActiveLet}[2]{%
        \MakeActiveLetHere#1#2%
        \niceverb_aux_cat\MakeActiveHere#1}%
    \renewcommand*{\MakeNormal}[1]{%
        \MakeNormalHere#1%
        \niceverb_aux_cat\MakeNormalHere#1}%
  }{}%
}
```

TODO doesn't babel have the same problem? 2010/03/12

2.11.3 hyper-xr

With the hyper-xr package creating links into external documents, preceding `\externaldocument{`⟨*file*⟩`}` with `\MakeActiveLet\&\CmdSyntaxVerb` may be needed. I do not want to redefine something here right now as I have too little experience with this situation.

[9] An alternative approach would be using afterpackage by Alex Rozhenko.

[10] This uses `\@onelevelsanitize`, therefore `\protect` doesn't change the behaviour of "active" characters.

2.11.4 Turning off and on altogether

These commands are new with v0.3.

`\noNiceVerb` *disables* all niceverb features.

```
\newcommand*{\noNiceVerb} {\MakeNormal\‘%
                           \MakeNormal\&%
                           \MakeNormal\<%
                           \MakeNormal\#%
                           \nvRightQuoteNormal
                           \MakeNormal\|}
```

`\useNiceVerb` *activates* all the niceverb features (apart from “auto mode”).

```
\newcommand*{\useNiceVerb}{\MakeActiveLet\‘\LQverb
```

TODO to be changed with wiki.sty v0.2

```
                           \MakeActiveLet\&\CmdSyntaxVerb
                           \MakeActiveLet\<\MetaVar
                           \MakeActiveLet\#\HashVerb
                           \nvRightQuoteSansSerif
                           \MakeActiveLet\|\nvCmdBox}
```

2.12 Activating the niceverb Syntax

niceverb features are activated at `\begin{document}` so (some) other packages can be loaded *after* niceverb. For v0.3, we do this after possible settings for compatibility with hyperref.

```
\AtBeginDocument{\useNiceVerb}
```

2.13 Leave Package Mode

```
\MakeNormalHere\_                   %% 2010/03/12
\endinput
```

2.14 VERSION HISTORY

```
v0.1   2009/02/21   very first, sent to CTAN
v0.2   2009/04/04   ...NoVerbList: \subsubsection, \AddToMacro,
       2009/04/05   \SimpleVerb makes more other than iii
       2009/04/06   just uses \dospecials
       2009/04/08   debugging code for rq/sf, +\relax
       2009/04/09   +\verb@eol@error, prepared for new doc method,
                    removed spurious \makeat..., -\relax (ligature),
       2009/04/10   (’-trick
       2009/04/11   \@ after \SimpleVerb
       2009/04/14   noted TODO below
       2009/04/15   change v0.1 to 2009/02/21
v0.30   2010/02/27   short, more explained, \AssignCatCodeTo,
```

```
360                     use \MakeActive for re-activating, \MakeNormal
361         2010/02/28  fixed @ and _ with & by moving \begin_min_verb;
362                     replaced \lq by `; Capitals in Titles
363         2010/03/05  \SimpleVerb -> \NVerb;
364                     use \MakeActive + \MakeNormal; \rq -> ';
365                     renamed some sections; \lq_verb -> \LQverb,
366                     \niceverb_meta -> \MetaVar,
367                     \param_verb -> \HashVerb
368         2010/03/06  removed \MakeAlign; removed @ and _ todo below;
369                     \NVerb makes ` and ' other;
370                     \nvAllowRQSF allows ' in column titles,
371         2010/03/08  \LQverb and & work in column titles,
372                     \RQverb works with \noindent;
373                     bookmark substitutions
374         2010/03/09  extended notes on 'hyperref' (in)compatibility;
375                     \MakeLetter\@ in \CmdSyntaxVerb only;
376                     |...| implemented as \prepareCmdBox etc.!
377         2010/03/10  \colorbox example, \thinspace; ltxguide!;
378                     removed todo; ..._exec -> \DoRQsansserif;
379                     minor doc changes in ``Nasty"
380         2010/03/11  doc changes in ``Escape Character ..." and
381                     ``Ampersand"
382         2010/03/12  \niceverb_aux_cat, \MakeActiveHere etc.,
383                     \IfTypesetting, \noNiceVerb, \useNiceVerb,
384                     corr. bracing mistake in \MakeNormal!
385         2010/03/14  0.31 -> 0.3; \HardNVerb, \GenCmdBox,
386                     \prepareCmdBox -> \nvCmdBox
387         2010/03/15  \endcell; \cmdboxitem; remark on \sfcode`/
388         2010/03/16  corr. -> \endline;
389                     advice on \cs{par}, \cs{if...}, \cs{fi};
390                     redefined \ignorespaces for RQ feature
391         2010/03/17  corr. `\fututelet', corr. \cs PDF substitution
392         2010/03/18  |\niceverbNoVerbList|, |\AddToMacro| etc.;
393                     corr. \AddToMacro;
394                     \lastskip-fix of \niceverb_ignore,
395                     another fix of \niceverb_noindent
396         2010/03/19  another fix of \niceverb_ignore: \spacefactor
397         2010/03/20  ... again: \niceverb_spacefactor
398
399 NOT DISTRIBUTED, just stored saved as separate version
400
401 v0.31   2010/03/20  right quote feauture: letters get \sfcode=1001
402                     `column title' -> `page headers', \ctanpkgref
403
404 NOT DISTRIBUTED, just stored as separate version
405
406 v0.32   2010/03/21  taking best things from v0.30 and v0.31
407         2010/03/23  removed \relax from \IfTypesetting
408 SENT TO CTAN
409
```

```
410 v0.4   2010/03/27   restoring ‘_’ with "auto mode" safer
411        2010/03/28   \AddToNoVerbList
412        2010/03/29   note above, renamed v0.4
413 SENT TO CTAN
414
415 v0.41  2010/04/03   v0.33 -> v0.4
416        2010/04/05   corrected \AutoCmdInput list
417 SENT TO CTAN as part of NICETEXT release r0.41
418
419 v0.41a 2010/11/09   typo corrected
420 v0.42  2010/12/30   corr. ‘\ ’ emulation in auto mode
421        2010/12/31   \MetaVar in ...maybe_meta...
422        2011/01/19   ‘...’ fix
423        2011/01/24   \ctanpkgref moves to texlinks.sty
424        2011/01/26   update (C)
425
```