

# niceverb.sty

—

## Minimizing Markup for Documenting L<sup>A</sup>T<sub>E</sub>X packages\*

Uwe Lück<sup>†</sup>

November 21, 2015

### Abstract

`niceverb.sty` provides very decent syntax (through active characters) for describing L<sup>A</sup>T<sub>E</sub>X packages and the syntax of macros conforming to L<sup>A</sup>T<sub>E</sub>X syntax conventions.

**Keywords:** literate programming, syntactic sugar, `.txt` to `.tex` enhancement, macro programming

## Contents

<b>1 Presenting niceverb</b>	<b>2</b>
1.1 Purpose	2
1.2 Acknowledgement/Basic Ideas	2
1.3 The Commands and Features of niceverb	3
1.4 Examples	6
1.5 What is Wrong with the Present Version	7
<b>2 The Package File</b>	<b>8</b>
2.1 Preliminaries	8
2.1.1 File Header	8
2.1.2 <code>\newlet</code>	8
2.1.3 Switching Category Codes	8
2.1.4 Robustness by <code>\IfTypesetting</code> or So	10
2.1.5 Shared Shorthand Macros	11
2.2 Implementation of the “Nice” Syntax	12
2.2.1 <code>\NVerb</code>	12
2.2.2 Single Quotes Typeset Meta-Code	13

---

\*This document describes version [v0.62](#) of `niceverb.sty` as of 2015/11/21.

<sup>†</sup><http://contact-ednotes.sty.de.vu>

2.2.3	Ampersand (or <code>\cstx</code> ) Typesets Meta-Code . . . . .	14
2.2.4	Escape Character Typesets Meta-Code . . . . .	16
2.2.5	Meta-Variables . . . . .	18
2.2.6	Hash Mark is Code . . . . .	18
2.2.7	Single Right Quotes for <code>\textsf</code> . . . . .	19
2.2.8	Boxes Highlighting Commands and Syntax . . . . .	22
2.3	When <code>niceverb</code> Gets Nasty . . . . .	25
2.3.1	Meta-Variables . . . . .	25
2.3.2	Quotes . . . . .	25
2.3.3	<code>hyperref</code> . . . . .	26
2.3.4	<code>hyper-xr</code> . . . . .	28
2.3.5	Listings and Moving . . . . .	28
2.3.6	Turning off and on altogether . . . . .	29
2.4	Minor Final Things . . . . .	30
2.4.1	Activating the <code>niceverb</code> Syntax . . . . .	30
2.4.2	Leave Package Mode . . . . .	30
2.4.3	VERSION HISTORY . . . . .	30

## 1 Presenting `niceverb`

### 1.1 Purpose

The `niceverb` package provides “minimal” markup for documenting L<sup>A</sup>T<sub>E</sub>X packages, reducing the number of keystrokes/visible characters needed (kind of poor man’s WYSIWYG).<sup>1</sup> 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 T<sub>E</sub>X). As opposed to tools that are rather common on UNIX/Linux, this operation should work at any T<sub>E</sub>X 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:

---

<sup>1</sup>“What you see is what you get.” Novices are always warned that WYSIWYG is essentially impossible with L<sup>A</sup>T<sub>E</sub>X.

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).<sup>2</sup>
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>'`. (`'\lt'` 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 L<sup>A</sup>T<sub>E</sub>X-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:

---

<sup>2</sup>The corresponding part of the “present work” is `makedoc.sty`.

**“Meta-variables:”** The package supports the “angle brackets” style of “meta-variables” (as with  $\langle meta-variable \rangle$ ). You just type  $\langle bar \rangle$  to get  $\langle bar \rangle$ .

This works due to a sloppy variant  $\backslash NVerb$  of  $\backslash verb$  which doesn’t care about possible ligatures and definitions of active characters. Instead, it assumes that the “verbatim” font doesn’t contain ligatures anyway.<sup>3</sup>  $\backslash verb+\langle foo \rangle+$ , by contrast, just yields  $\langle foo \rangle$ .

Almost the same feature is offered by `ltxguide.cls` which formats the basic guides from the L<sup>A</sup>T<sub>E</sub>X 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  $\langle foo \rangle$  is typeset as  $\langle foo \rangle$ , or (generally)  $\langle content \rangle$  turns  $\langle content \rangle$  into meta-code with the meta-variable feature as above. This somewhat resembles the  $\backslash MakeShortVerb$  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*  $\backslash bar$  instead of typesetting it, giving an “undefined” error or so.  $\backslash verb$  fails in the same situation, for the same reason.  $\langle \& \backslash bar \langle remaining \rangle \rangle$  or “auto mode” (see below) may then work better.<sup>4</sup> 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  $\backslash string$ ). 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  $\backslash verb$  command from L<sup>A</sup>T<sub>E</sub>X. To get *usual* single quotes, you can use their standard substitutes  $\backslash lq$  and  $\backslash rq$ , or for pairs of them,  $\backslash qtd\{ \langle text \rangle \}$  in place of  $\backslash lq\langle text \rangle\backslash rq$ —or even  $\backslash lq\langle text \rangle\backslash rq$ . To get single quotes around some verbatim (*verb*), often  $\backslash qtd\{ \& \langle verb \rangle \}$  works. It is for this reason that I have refrained from different solutions as in `newverbs` (so far).

v0.44 provides  $\backslash AddQuotes$  after which single quotes *both* turn their content into metacode *and print* single quotes around them *automatically*. This can be turned off again by  $\backslash DontAddQuotes$ .

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

$$\backslash textsf\{ \langle text \rangle \}$$

<sup>3</sup>On the other hand,  $\backslash NVerb$  is more *careful* with `niceverb`’s special characters.

<sup>4</sup> $\backslash bar$  indeed!

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>}`.<sup>5</sup>

This choice of `&` rests on the assumption that there won’t be many tables in the documentation. 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 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

<sup>5</sup>Moreover, `&` currently has a limited `xspace` functionality only.

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.<sup>6</sup> 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 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.—The feature fails in certain occasions because a single right quote must not always be interpreted as `\textsf`, and deciding this by macros became quite laborious for me and is most likely still not perfect. There is a command `\nvAllRightQuotesSansSerif` to be used with care that interpretes *all* single right quotes as `\textsf`, which, e.g., means that you must use `\rq` for apostrophes.

**“Moving” arguments:** `\NiceVerbMove{⟨text⟩}` with v0.6 is for “moving” arguments so that niceverb syntax operates *locally* at the destination (table of contents or page headings). It is automatically used by niceverb’s variant of L<sup>A</sup>T<sub>E</sub>X’s sectioning commands; while with `\markboth`, `\markright`, `\addcontentsline` etc. you must include yourself (currently, `TODO?`).

## 1.4 Examples

The file `mdoccorr.cfg` providing some `.txt`→L<sup>A</sup>T<sub>E</sub>X 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

<sup>6</sup>This idea appeared 2009 on the L<sup>A</sup>T<sub>E</sub>X-L mailing list. It may be wrong, as I have sometimes experienced . . .

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`<sup>7</sup> 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 ‘`\_`’.
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  $\varepsilon$ - $\TeX$  features, I need to find out ...

---

<sup>7</sup><http://ctan.org/pkg/substr>

## 2 The Package File

### 2.1 Preliminaries

#### 2.1.1 File Header

```

1 \NeedsTeXFormat{LaTeX2e}[1994/12/01]
2 \ProvidesPackage{niceverb}[2015/11/21 v0.62
3             minimize doc markup (UL)]
4
5 %% Copyright (C) 2009-2012, 2014 2015 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 %%
```

#### 2.1.2 `\newlet`

`\newlet⟨cmd⟩⟨cmd⟩` counters the risk of mistyping `⟨cmd⟩` with `\@ifdefinable`, and even saves some code lines:

```

20 \providecommand*\newlet}[2]{\@ifdefinable#1{\let#1#2}}
21 \@onlypreamble\newlet
```

#### 2.1.3 Switching Category Codes

Underscore as a “private letter,” using `stacklet` with v0.5:

```

22 \RequirePackage{stacklet} \PushCatMakeLetter\_           %% 2012/08/27
```

v0.3 introduced `\AssignCatCodeTo` and `\MakeNormal`. v0.5 abolishes the former again and uses `actcodes` for some part of `\catcode` switching:

```

23 \RequirePackage{actcodes}
```

`\CatCode{\⟨character⟩}` (or simply `\CatCode\⟨character⟩`) saves one token per use and works when the category code of “ (single left quote) has changed. As of v0.5, it may be defined by a different package:

```

24 \providecommand*\CatCode{\catcode'}           %% \provi... 2012/08/27
25 % \newcommand*\CatCode}[1]{\catcode'#1 } %% no better 2010/02/27
```



`\CatCode` is near to be moved into the `catcodes` bundle, and basic commands from `stacklet` and `actcodes` may be reimplemented using it (`manycats`; `allcats` for loading entire `catcodes` in good order).

`\AssignCatCodeTo{⟨number⟩}{⟨char⟩}` no longer is considered useful (counted tokens in `memory.tex`) and replaced by `\CatCode`.

```
26 % \newcommand*\AssignCatCodeTo[2]{\catcode'#2=#1\relax}
```

`\MakeLetter⟨char⟩` is replaced by the `stacklet` package—I thought, but *here* it is also needed to declare the “private letters” of the package that is documented. This should be “variable.” OK, the new (v0.5) `\private_letters` is a step towards this:

```
27 \newcommand*\private_letters{\CatCode\@11\CatCode\_11\relax}
```

`\MakeOther⟨char⟩` and `\MakeActive⟨char⟩` were implemented here before v0.5, now they are in `actcodes` ...

```
28 % \def \MakeOther {\AssignCatCodeTo{12}}
```

`\MakeActiveLet⟨char⟩⟨macro-name⟩` likewise is in `actcodes`. `niceverb` takes a copy `\MakeActiveLetHere` of it for dealing with `hyperref` (see Sec. 2.3.3). `hyperref`-compatibility of mere `\MakeActive` is not provided any longer:

```
29 \newlet\MakeActiveLetHere\MakeActiveLet
```

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.)

```
30 \def\do#1{\expandafter
```

```
31 % \chardef \csname normal_catcode_\string#1\expandafter \endcsname
```

← v0.6 2014/03/22: First I thought “too few `\expandafter`”; actually the original `\expandafter` has no effect →

```
32 \chardef \csname normal_catcode_\string#1\endcsname
```

```
33 \CatCode#1\relax}
```

```
34 \dospecials
```

Tests: “normal category code” of `\` is 0, “normal category code” of `$` is 3; “normal category code” of `&` is 4.<sup>8</sup>

```
35 % \newcommand*\make_iii_other{\MakeOther\\\MakeOther\{\MakeOther\}}
```

```
36 %% <- replaced 2009/04/05
```

<sup>8</sup> $\LaTeX$ 's `\nfss@catcodes` is similar, but it makes space-like characters ignored. Also cf. `lfinal.dtx`. **TODO:** `\RestoreNormalCatcodes`.

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

```

37 \newcommand*\MakeNormal}[1]{%
38   \ifundefined{\norm_catc_str#1}%
39     {\MakeOther#1}%
40     {\CatCode#1\csname\norm_catc_str#1\endcsname\relax}}
41 \newcommand*\norm_catc_str{normal_catcode_\string}
42 %% TODO add ^^I and ^^M
43 %% TODO save char tokens %% 2012/08/27

```

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

```

44 \newlet\MakeNormalHere\MakeNormal

```

### 2.1.4 Robustness by `\IfTypesetting` or So

It seems we need some own ways of robustifying (as opposed to L<sup>A</sup>T<sub>E</sub>X’s `\protect` and `\DeclareRobustCommand`—sometimes, especially for certain active characters) to achieve various compatibilities—using

```
\IfTypesetting{⟨if⟩}{⟨unless⟩}
```

It also saves some `\expandafters`.

```

45 \providecommand*\IfTypesetting}{%
46 %   \relax

```

← This `\relax` suppressed ligatures of single right quotes!

```

47   \ifx \protect@typeset@protect
48     \expandafter \@firstoftwo
49   \else \expandafter \@secondoftwo \fi}

```

```
\nvSelfProtect{⟨cmd-char⟩}{⟨typeset⟩}
```

is another idea. In “typesetting mode,” `⟨typeset⟩` is run. Otherwise a single unexpanded token `⟨cmd-char⟩` remains. `TODO` bad at `\shipout`. No `\protect` appears, and as opposed to L<sup>A</sup>T<sub>E</sub>X’s protection mechanism, running `⟨typeset⟩` does not require a second macro name. The idea is that `\nvSelfProtect{#1}{#2}` is the *definition* (substitution text—on token level) of `⟨cmd-char⟩`.<sup>9</sup>

```

50 \newcommand*\nvSelfProtect}[2]{%
51   \ifx \protect@typeset@protect
52     \nv_expand_else{#2}%                               %% braces 2014/03/26
53 %   \else \protect#1\fi}
54   \else \noexpand#1\fi}                               %% works 2014/03/28
55 \def\nv_expand_else#1\else#2\fi{\fi#1}

```

<sup>9</sup>This may go into a separate package under a different name later.

Sometimes “control sequences” get definitions with `\svSelfProtect` below whose first argument then is an active character—the “control sequence” then is the “permanent alias” of the active character. This is a somewhat “indirect self”-protection. At other places, the “self”-protection is more direct. Then `\NewSelfProtectedCommand{<cmd>}{<def>}` avoids mistakes from mistyping `<cmd>` and saves some code. It works like `\newcommand*`, provides the `\svSelfProtect`, and your definition `<def>` needs to contain the second argument of `\svSelfProtect` only. *Arguments* are not supported currently (TODO—well, 3 applications 2014/03/27):

```
56 \newcommand*\NewSelfProtectedCommand[2]{%
57   \newcommand*#1{\nvSelfProtect#1{#2}}
58 \@onlypreamble\NewSelfProtectedCommand
```

TODO 3 applications for the permanent alias case, saving catcode changes ...

#### Testing:

```
59 \newcommand*\nvShowProtectedEdef[1]{%
60   \protected@edef\@tempa{#1}\show\@tempa}
```

### 2.1.5 Shared Shorthand Macros

`\begin_min_verb` is a beginning shared by some macros here. It begins like L<sup>A</sup>T<sub>E</sub>X’s `\verb`, apart from the final `\tt`. `\bgroup` is needed for `\hbox` and must be balanced by an `\egroup` counterpart later.

```
61 \newcommand*\begin_min_verb{%
62   \relax \ifmode \hbox \else \leavevmode\null \fi
63   \bgroup \tt}
```

For typographical additions (“decorations”) to the verbatim material, we collect it in a box register addressed by `\niceverb_savebox`:

```
64 \newsavebox\niceverb_savebox
```

`\SetNiceVerbSaveBox` starts reading the (“meta-”)verbatim material:

```
65 \newcommand*\SetNiceVerbSaveBox{%
66   \setbox\niceverb_savebox\hbox\bgroup}
```

`\NVerb`, `\HardNVerb`, or `\NiceMaybeMetaVerb` with an optional argument about as `[<id>_egroup]` should follow, cf. Sec. 2.2.1.—There have been two applications only up to now (2014/03/19), but this may change soon.

`\TheNiceVerbSaveBox` allows referring to the verbatim material collected, in order to place it a *single time*—and no surrounding braces are needed:

```
67 \newcommand*\TheNiceVerbSaveBox{\box\niceverb_savebox}
```

TODO:

```
\NewNiceVerbDecoration{<deco>}{<end-name>}{<start>}{<end-code>}
```

might save from typing `\<end-name>` twice and from typing the two `\egroups`.

TODO left quote verb moving: braces get lost.

## 2.2 Implementation of the “Nice” Syntax

### 2.2.1 `\NVerb`

Discovered mistakes in this section 2014/03/19, with respect to robustness. (i) `\NVerb` not really was meant to be a user command, to appear in documentation code (rather to be internal). (ii) The attempt to make it robust was incomplete. (iii) The code for “not typesetting” was strange. (iv) It is difficult to imagine that somebody attempts to use “verbatim” code, e.g., in a section title (while with our `&/\string` it’s ok). *At least handy replacement for `\textsf`*. Well, let’s see. May be I once find a useful application. So I repair the code—`\protect` before `\_...\_false`. `\NVerb<char><code><char>`:

```
68 \newcommand*{\NVerb}{%
69     \protect\_no_nice_meta_verb_false \NiceMaybeMetaVerb}
```

`\HardNVerb<char><code><char>` does not recognize meta-variables:

```
70 \newcommand*{\HardNVerb}{%
71     \protect\_no_nice_meta_verb_true \NiceMaybeMetaVerb}
72 \newif\if_no_nice_meta_verb_
```

v0.6 equips both `\NVerb[<end-cmd>]` and `\HardNVerb[<end-cmd>]` with an optional argument for a single parameter-less macro for what to do after reading verbatim text—for boxing or quoting etc.<sup>10</sup> `\niceverb_egroup` then is useless and removed. Macros that were assigned to it before v0.6 move into the new optional arguments.—Actually, the next macro `\NiceMaybeMetaVerb` shared by `\NVerb` and `\HardNVerb` gets the optional argument: **TODO!**

```
73 % \newcommand*{\nice_maybe_meta_verb}[1]{%
74 \newcommand*{\NiceMaybeMetaVerb}[2][\niceverb_normal_egroup]{%
```

`\newcommand` with v0.6 must suffice for robustness, so removing 2014/03/20:

```
75 % \IfTypesetting{%
```

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

```
76 \begin_min_verb
77 \let\do\MakeOther \dospecials
```

Turn off `niceverb` specials:

<sup>10</sup>The goal resembles that of `\collectverb` in Martin Scharrer’s `newverbs`. A difference in implementation is that the character delimiting the verbatim text is used as a parameter delimiter for a new/temporary macro. So the verbatim characters are fixed. Our approach will be collecting the verbatim material in a box, if we need something more complex than `\niceverb_normal_egroup`. This allows changing category codes with `\MetaVar` again, although there hasn’t been a need for this so far. It might be useful for allowing shorthand macros in `\MetaVar`’s argument.

```

78     \MakeOther\|\MakeOther\‘\MakeOther\’%
79     \if_no_nice_meta_verb_ \MakeOther\<%
80     %% \else \MakeActiveLet\<\MetaVar %% 2010/12/31
81     \else \MakeActiveLetHere\<\MetaVar %% 2011/06/20
82     \fi
83 % \MakeActiveLetHere #2\niceverb_egroup
84 \MakeActiveLetHere #2#1% %% 2014/03/18

```

After the previous line has worked, we use `\def` instead of `\let`, so there is no longer a need to choose a command name for the verbatim delimiter – well, **no**, don’t define the same macro several times. Also, the same “end” macro might be used for different purposes, e.g., when a macro in an eventual expansion of the “end” macro is modified.

```

85 % \MakeActiveDef #2{#1}% %% 2014/03/18
86 \verb@eol@error %% TODO change message 2009/04/09
87 % }{\string\NVerb \string#1}

```

← both `\string` very strange (second one finds `<char>`—maybe it’s active—but then its next occurrence delimiting the verbatim code will harm too!), also redirecting to `\NVerb`. (May have been ok for entries to auxiliary files.) New difficulties come from the optional argument, which needs protection as well.—Ok, the optional argument is not protected, and active characters `<char>` must “protect themselves,” so use of `\IfTypesetting` changes, cf. Sec. 2.1.4.

```
88 }
```

[2014/03/19 removing/hiding remarks from 2009f. that I don’t understand anymore ...]

```

89 \newcommand*{\niceverb_normal_egroup}{%
90     \egroup
91     \niceverb_maybe_rq %% 2011/09/09 for \AddQuotes
92     \ifmode\else@\fi}
93 % \@ifdefinable\niceverb_egroup %% rm 2014/03/18
94 % {\let\niceverb_egroup\niceverb_normal_egroup}

```

## 2.2.2 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, and we typeset it (thus the left quote feature does not allow to typeset something verbatim that starts with a single left quote). In page headers, a `\protect` could be in the way before v0.6. (A hook for `\relaxing` certain things in `\markboth` and `\markright` would have been an alternative. [TODO](#))

```

95 \MakeActive\‘
96 \newcommand*{\LQverb}{%
97 % \IfTypesetting{\lq_double_test}{\protect‘}}

```

New approach v0.6:

```

98     \nvSelfProtect'\lq_double_test}
99 %   \IfTypesetting{\lq_double_test}{\noexpand'}
100 \MakeOther'
101 \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): [TODO](#) switch what?

```

102 \begingroup
103 \let\do\MakeOther \dospecials
104 \MakeOther\|%% 2010/03/09!
105 \futurelet\let_token \lq_double_decide}
106 \newcommand*\lq_double_decide}{%
107 \ifx\let_token\LQverb
108 \endgroup
109 ‘\expandafter \@gobble
```

... alternative ...

```

110 %   \expandafter ‘%
```

does not recognize next left quote—why? [TODO](#)—Corresponding right quotes will become “other” due to having no space at the left. [TODO](#) to be changed with `wiki.sty`.

```

111 \else
112 %   \ifx\let_token\protect                %% rm. 2014/03/28
113 % %   \show\let_token                    %% indeed before v0.6, 2014/03/24
114 %   \expandafter\expandafter\expandafter \lq_double_decide_ii
115 %   \else
116 \endgroup
117 \niceverb_maybe_qs                        %% 2011/09/09
118 %   \expandafter\expandafter\expandafter \NVerb
119 %   \expandafter\expandafter\expandafter \’%
```

← 2015/11/20 with one conditional less, less →

```

120 \expandafter \NVerb \expandafter \’%
121 %   \fi
122 \fi}
```

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

```

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

### 2.2.3 Ampersand (or `\cstx`) Typesets Meta-Code

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

```

125 \MakeActive\&
126 \newcommand*\CmdSyntaxVerb}{%
127 \IfTypesetting{%
128 \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!

```

129 \private_letters % v0.5
130 \cmd_syntax_verb
131 % }{\protect&\string}}
132 % }{\noexpand&\string}}

```

... with `\string`, in an `\edef`, the following command cannot be properly typeset, so

```

133 }{\noexpand&\noexpand}} % 2014/03/26

```

**TODO** actually test non-typesetting, maybe introduce macros that perform tests anywhere ...

```

134 \MakeNormal\&
135 \newcommand*\cmd_syntax_verb}[1]{%
136 \string#1\futurelet\let_token \after_cs}

```

However, `&` (or `\CmdSyntaxVerb`) may fail with private letters, especially in *macro arguments*<sup>11</sup> and with `hyperref` in titles of *sections bearing \labels*, so we provide something like `\cs{characters}` from `tugboat.sty`.

```

137 \DeclareRobustCommand*\cs}[1]{%
138 % \begin_min_verb \backslash_verb #1\egroup}

```

... fails with `_` in footnote today (2014/03/19) so:

```

139 \begin_min_verb \withcsname\string#1\endcsname\egroup} % v0.6
140 \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{characters}*[<opt>]{<mand>}` as follows.

```

141 \DeclareRobustCommand*\cstx}[1]{% % corr. 2010/03/17
142 % \begin_min_verb \backslash_verb #1%

```

v0.6 like above:

---

<sup>11</sup>**TODO**: `vfoot2e.sty` – see notes.

```

143     \begin_min_verb \withcsname\string#1\endcsname
144     \futurelet\let_token \after_cs}
145 \newcommand*{\after_cs}{%
146   \ifcat\noexpand\let_token a\egroup \space
147   \else \expandafter \decide_verb \fi}
148 \newcommand*{\test_more_verb}{\futurelet\let_token \decide_verb}
149 \newcommand*{\decide_verb}{%
150   \jumpteg_on_with\bgroup\braces_verb
151   \jumpteg_on_with[\brackets_verb
152   \jumpteg_on_with*\star_verb
153   \egroup}
154   %% CAUTION/TODO wrong before (... if cmd without arg
155   %%           use \ then or choose usual verb...
156   %%           or \MakeLetter\ ( etc. ... or \xspace
157 \newcommand*{\jumpteg_on_with}[2]{%
158   \ifx\let_token#1\do_jumpteg_with#2\fi}

```

TODO cf. xfor, xspace (`\break@loop`); `\DoOrBranch#1...#1` or so.

```

159 \def\do_jumpteg_with#1#2\egroup{\fi#1}
160 \def\braces_verb#1{\string{#1}\string}\test_more_verb}
161 \def\brackets_verb[#1]{[#1]\test_more_verb}
162 \def\star_verb*{*}\test_more_verb}
163   %% 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 `&`:

```
164 \let\endcell&
```

## 2.2.4 Escape Character Typesets Meta-Code

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

```

165 \DeclareRobustCommand*{\BuildCsSyntax}{%
166   \futurelet\let_token \build_cs_syntax_sp}
167 \newcommand*{\build_cs_syntax_sp}{%
168   \ifx\let_token@sptoken
169     \@%                               %% 2010/12/30
170   \else %% TODO ^^M!?
171     \expandafter \start_build_cs_syntax
172   \fi}
173 \newcommand*{\start_build_cs_syntax}[1]{%
174   \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:

```

175   \edef\before_build_cs_sub{\the\CatCode\_}%
176   \private_letters                               %% v0.5

```



```

177 \test_more_cs}
178 \newcommand*\test_more_cs){%
179 \futurelet\let_token \decide_more_cs}
180 \newcommand*\decide_more_cs){%
181 \ifcat\noexpand\let_token a\expandafter \add_to_cs
182 \else
183 % \MakeNormalHere\

```

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

```

184 \CatCode\_ \before_build_cs_sub
185 \MakeOther\_%
186 % \expandafter \in@ \expandafter
187 % {\csname \string_built \expandafter \endcsname
188 % \expandafter}\expandafter{\niceverbNoVerbList}%
189 %% <- useless braces 2014/07/17 ->
190 \expandafter \in@ \csname \string_built \expandafter
191 \endcsname \expandafter {\niceverbNoVerbList}%
192 \ifin@
193 \csname \string_built
194 \expandafter\expandafter\expandafter \endcsname
195 \else
196 \begin_min_verb \backslash_verb\string_built
197 \expandafter\expandafter\expandafter \test_more_verb
198 \fi
199 \fi}
200 %% TODO such \if nestings with ifthen!?
201 %% cf.:
202 % \let\let_token,\typeout{\meaning\let_token}
203 %% TEST TODO fuer xspace!? (\ifin@)
204 \newcommand*\add_to_cs}[1]{%
205 \edef\string_built{\string_built#1}\test_more_cs}

```

`\AutoCmdSyntaxVerb` starts, `\EndAutoCmdSyntaxVerb` ends “auto mode.”

```

206 \newcommand*\AutoCmdSyntaxVerb){%
207 \MakeActiveLetHere\\BuildCsSyntax}
208 \newcommand*\EndAutoCmdSyntaxVerb){\CatCode\\z@}

```

`\NormalCommand{<characters>}` executes `\<characters>` in “auto mode.”

```

209 \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:

```

210 % \futurelet\LetToken\relax \relax
211 % \show\LetToken \typeout{\ifcat\noexpand\LetToken aa\else x\fi}

```



### 2.2.7 Single Right Quotes for `\textsf`

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

One essential 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.

Another problem is that with and L<sup>A</sup>T<sub>E</sub>X (as with Plain T<sub>E</sub>X—*The T<sub>E</sub>Xbook* p. 357), the right single quote is needed for primes in math mode, and L<sup>A</sup>T<sub>E</sub>X enforces this in `\@outputpage` preparing `\writes` (why? `TODO`) as well as page headers.

```
235 \MakeActive\
236 \newcommand*\{niceverb_rq_choice}[1]{%           %% 2014/03/27
```

We make a deal with `\active@math@prime`: in math mode, the prime functionality acts; outside, the “right quote sansserif mode” acts. Test: *a'* now works with `niceverb`.—For page headers, in expanding without typesetting, the expansion of `\RQsansserif` must contain another active single right quote.

```
237 \nvSelfProtect'\ifmmode
238 \expandafter\active@math@prime
239 \else
240 \expandafter#1%
241 \fi}}
```

The following `\do_rq_sansserif` is what `\DoRQsansserif` below was before v0.6. This, too, must be changed for `\active@math@char`, and earlier use of `\DoRQsansserif` in `\niceverb_rq_sf_test` must be replaced.

```
242 \@ifdefinable\do_rq_sansserif
243 {\def\do_rq_sansserif#1'\{textsf{#1}}}
244 \newcommand*\RQsansserif}{%
245 % \IfTypesetting{\niceverb_rq_sf_test}{\protect'}}
246 \niceverb_rq_choice\niceverb_rq_sf_test}
247 \MakeOther\'
```

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

```
248 \newcommand*\{niceverb_rq_sf_test}{%
249 \ifhmode
250 \ifdim\lastskip>\z@
251 % \expandafter\expandafter\expandafter \DoRQsansserif
252 \expandafter\expandafter\expandafter \do_rq_sansserif
253 \else
254 \ifnum\niceverb_spacefactor
255 \expandafter\expandafter\expandafter\expandafter
256 \expandafter\expandafter\expandafter
257 \do_rq_sansserif %% \DoRQsansserif
258 \else '\fi
259 \fi
```

```

260 \else \ifvmode
261 % \expandafter\expandafter\expandafter \DoRQsansserif
262 \expandafter\expandafter\expandafter \do_rq_sansserif
263 \else '\fi
264 \fi}
265 % \nvShowProtectedEdef{'niceverb'}
266 \MakeOther\'

```

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

```

267 \newcommand*{\DoRQsansserif}{%
268 \niceverb_rq_choice\do_rq_sansserif} %% 2014/03/27

```

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 L<sup>A</sup>T<sub>E</sub>X’s internal `\@sect` and of T<sub>E</sub>X’s primitives `\noindent` and `\ignorespaces` as well as by a signal `\spacefactor` value of 1001.

`\nvAllowRQSS` becomes more powerful with v0.6, for Sec. 2.3.5:

```

269 \NewSelfProtectedCommand{\nvAllowRQSS}{%
270 \MakeActiveLetHere'\RQsansserif
271 \niceverb_rqsf %% 2014/03/27
272 \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:

```

273 \AtBeginDocument{%
274 \edef\before_niceverb_parenthesis{\the\sfcode'\{ }%
275 \newlet \before_niceverb_ignore \ignorespaces %% 2010/03/16
276 \newlet \before_niceverb_sect \@sect %% \newlet 2014/03/25
277 \newlet \before_niceverb_noindent \noindent} %% 2010/03/08

```

We assume that `\@sect` has the same parameters there as in L<sup>A</sup>T<sub>E</sub>X (even if redefined by another package, like `hyperref`).

```

278 \def\niceverb_sect#1#2#3#4#5#6[#7]#8{%
279 \before_niceverb_sect{#1}{#2}{#3}{#4}{#5}{#6}%
280 % [\protect\nvAllowRQSS #7]}%
281 % {\protect\nvAllowRQSS #8]}

```

With v0.6, a more general `\NiceVerbMove{text}` is introduced, defined in Sec. 2.3.5:

```

282 [\NiceVerbMove{#7}]%
283 {\NiceVerbMove{#8}}

```

2010/03/20:

```

284 \newcommand*{\niceverb_spacefactor}{\spacefactor=1001\relax}
285 \newcommand*{\niceverb_noindent}{%
286     \before_niceverb_noindent \niceverb_spacefactor}
287 \newcommand*{\niceverb_ignore}{%
288     \ifhmode \niceverb_spacefactor \fi \before_niceverb_ignore}

```

Here are the main switches. With v0.6, `\nvRightQuoteSansSerif` is divided into two parts, for Sec. 2.3.5:

```

289 \newcommand*{\niceverb_rqsf}{%                               %% 2014/03/27
290 % \MakeActiveLet'\RQsansserif
291     \sfcode'\'(=1001 %% enable in parentheses 2009/04/10

```

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

```

292 % \let\@footnotetext\niceverb_footnotetext
293 \let\ignorespaces\niceverb_ignore                               %% 2010/03/16
294 % \let\@sect\niceverb_sect
295 \let\noindent\niceverb_noindent                               %% 2010/03/08
296 \newcommand*{\nvRightQuoteSansSerif}{%
297     \niceverb_rqsf
298     \MakeActiveLet'\RQsansserif
299     \let\@sect\niceverb_sect
300     \def\niceverb_rqsf_kind{\nvAllowRQSS}}

```

← It really must be `\def` in order to transmit the choice to the table of contents.

With v0.6, in dealing with moving things in Sec. 2.3.5, section titles are handled in a more complex way. We divide the former `\nvRightQuoteNormal` into two parts:

```

301 \newcommand*{\niceverb_rq_normal}{%
302 % \MakeNormal\'%                                             %% 2010/03/21
303     \sfcode'\'(=\before_niceverb_parenthesis\relax
304     \let\ignorespaces\before_niceverb_ignore                   %% 2010/03/16
305     \let\noindent\before_niceverb_noindent                     %% 2010/03/08
306     \MakeActive\'
307 \newcommand*{\nvRightQuoteNormal}{%
308     \MakeNormal\'%                                             %% 2010/03/21
309     \niceverb_rq_normal
310     \let\nv_rqsf_kind\@empty
311     \ifnum\CatCode\'=\active                                   %% '=' missing 2015/11/09
312         \ifx'\LQverb \else
313             \let\@sect\before_niceverb_sect
314             \fi
315         \else
316             \let\@sect\before_niceverb_sect
317             \fi}
318 \MakeOther\'

```

`\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.

```
319 \newcommand*{\nvAllRightQuotesSansSerif}{%
320     \niceverb_rq_normal %%% \nvRightQuoteNormal      %% 2014/03/27
```

That's one use of `\DoRQsansserif` with v0.6:

```
321     \MakeActiveLet'\DoRQsansserif
322     \def\niceverb_rqsf_kind{\nvAllRQSS}}      %% 2014/03/27
```

← must be `\def` for transmissions.

```
323 \NewSelfProtectedCommand{\nvAllRQSS}{%
324     \niceverb_rq_normal
```

That's the other use of `\DoRQsansserif` with v0.6:

```
325     \MakeActiveLetHere'\DoRQsansserif}
```

[Hiding remarks from 2010f. (`\ctanpkgref`) 2014/03/23]

## 2.2.8 Boxes Highlighting Commands and Syntax

With v0.3, we include one kind of command syntax boxes whose *content* is (in `\niceverb` syntax) delimited as `[content]`. `\GenCmdBox<char><content><char>` works like `\NVerb<char><content><char>` except putting the latter's result into a framed (or coloured or ...) box.

```
326 \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).

```
327 \newcommand*{\HardVerbBox}{\_no_nice_meta_verb_true \gen_cmd_box}
328 \newcommand*{\gen_cmd_box}{%
329 % \let\niceverb_egroup\nice_collect_verb_egroup %% rm 2014/03/18
330 % \setbox\niceverb_savebox \hbox\bgroup
```

← 2014/03/19 →

```
331 \SetNiceVerbSaveBox
332 % \if_no_nice_meta_verb_
333 % \expandafter \HardNVerb
334 % \else \expandafter \NVerb \fi
```

← 2014/03/19 → [TODO use generalization]

```

335     \NiceMaybeMetaVerb[\nice_collect_verb_egroup]%
336   }
337   \newcommand*{\nice_collect_verb_egroup}{%
338     \egroup \egroup
339     \ifvmode \expandafter \VerticalCmdBox
340     \else    \ifmmode \hbox \fi
341             \expandafter \InlineCmdBox \fi
342   %        {\box\niceverb_savebox}%

```

← 2014/03/19 →

```

343     \TheNiceVerbSaveBox

```

(Removing a remark that I don't understand 2014/03/19.)

```

344   \ifmmode\else@\fi
345   % \let\niceverb_egroup\niceverb_normal_egroup %%    rm 2014/03/19
346   }

```

`\nvCmdBox` will be the permanent alias for ‘|’.

```

347   \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`):

```

348   \newcommand*{\VerticalCmdBox}{%

```

v0.6 encourages a page break here according to the above idea, in order to avoid a page break after explaining subsequent code ([TODO](#): that's a major functionality change):

```

349             \if@nobreak\else \pagebreak[2]\fi
350             \leavevmode\InlineCmdBox}

```

(2011/11/05 removing `\null`.) The command declaration boxes in the documentation of Nicola Talbot's `datatool` would be an especially nice realization of `\VerticalCmdBox`.<sup>12</sup>

`\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.

<sup>12</sup>I find the documentation of Martin Scharrer's `newverbs` package similarly impressive.

```

351 \newcommand*\InlineCmdBox}[1]{%
352   \bgroup
... needed in math mode with \begin_min_verb.
353   \fboxsep 1pt
354   \kern\SetOffInlineCmdBoxOuter
355   \smash{\SetOffInlineCmdBox{\kern\SetOffInlineCmdBoxInner
356                                     \InlineCmdBoxArea{#1}%
357                                     \kern\SetOffInlineCmdBoxInner}}%
358   \mathstrut
359   \kern\SetOffInlineCmdBoxOuter
360 \egroup
361 }

```

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

```

362 \newlet\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[cmym]{.1,0,.2,.05}}

```

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

```

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

```

This choice is inspired by `\cstok` for “boxed” things in Knuth’s `manmac.tex` which formats *The T<sub>E</sub>Xbook*.

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

```

364 \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.

```

365 \newcommand*\InlineCmdBoxArea}[1]{%
366   \smash{#1}\vphantom{gjq\backslash_verb}}

```

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

```

367 \newcommand*\cmdboxitem}{%
368 % \bgroup
369 % \let\niceverb_egroup\cmd_item_egroup
370 % \global %% TODO!? 2010/03/15
371 % \setbox\niceverb_savebox \hbox\bgroup

```



← 2014/03/19 →

```
372     \SetNiceVerbSaveBox
373 %     \NVerb}
```

← 2014/03/19 →

```
374     \NVerb[\cmd_item_egroup]}
375 \newcommand*{\cmd_item_egroup}{%
376     \egroup \egroup %%% \egroup           %% 1 less 2014/03/19
377 \item[\InlineCmdBox\TheNiceVerbSaveBox]}
```

Does it work?

`\foo{<arg>}` could be defined for a test.

`\bar{<arg>}` could be defined for a test as well.

## 2.3 When niceverb Gets Nasty

These things are new with v0.3.

### 2.3.1 Meta-Variables

This is even newer than v0.3.

In case you actually need `<` and `>` in math mode, `\lt` and `\gt` are “provided” as aliases:

```
378 \providecommand*\gt{>}
379 \providecommand*\lt{<}
```

### 2.3.2 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.

```
380 \providecommand*\qtd}[1]{‘#1’}           %% provide 2012/11/27
```

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.

```
381 % todo \qtdverb!? alternative meaning for \LQverb!? 2010/03/06
382 %     rather rare, & takes less space                2010/03/09
```

... see approaches below ...

`\AddQuotes` automatically surrounds code with single quotes. I have so often felt that it was a design mistake to drop them (2011/09/09):

```
383 \newcommand*\AddQuotes}{%
384     \let\niceverb_maybe_qs\niceverb_add_qs}
385 \newcommand*\niceverb_add_qs}{%
```

In a math display, quotes are suppressed even with `\AddQuotes`:

```
386     \ifmode\else
387     ‘\let\niceverb_maybe_rq\niceverb_rq
388     \fi}
389 \newlet\niceverb_maybe_rq\relax
390 \newcommand*{\niceverb_rq}{‘\let\niceverb_maybe_rq\relax}
```

You can undo this by `\DontAddQuotes`:

```
391 \newcommand*{\DontAddQuotes}{\let\niceverb_maybe_qs\relax}
```

The default will be the behaviour that we had before:

```
392 \DontAddQuotes
```

With v0.6, `\qtdnverb⟨char⟩⟨m-verb⟩⟨char⟩` encloses the “meta-verbatim” material with single quotes:

```
393 \newcommand*{\qtdnverb}{%
```

Useless after `\AddQuotes`:

```
394 \ifx\niceverb_maybe_qs\niceverb_add_qs
395 \expandafter \NVerb
396 \else
397 \lq
398 \expandafter\NVerb\expandafter
399 [\expandafter\niceverb_egroup_rq\expandafter]%
```

← The comment mark proved essential 2015/11/20. The line was commented out previously, obviously I had forgotten that the feature didn’t work.

```
400 \fi}
```

Completing the work v0.62:

```
401 \newcommand*{\niceverb_egroup_rq}{\niceverb_normal_egroup\rq}
```

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

```
402 \providecommand*{\dqtd}[1]{‘‘#1’’}           %% 2012/11/27
```

### 2.3.3 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.<sup>13</sup>

<sup>13</sup>An alternative approach would be using `afterpackage` by Alex Rozhenko.

```

403 \AtBeginDocument{%
404   \ifpackageloaded{hyperref}{%
405     \newcommand*\PDFcstring{%           %% moved here 2010/03/09
406       \134\expandafter\@gobble\string}% %% ASCII octal encoding
407   \pdfstringdefDisableCommands{%
408     \let\nvAllowRQSS\empty             %% not \relax 2010/03/12
409     \let\niceVerbGeneral\empty         %% 2014/03/27
410     \let\nvAllRQSS\empty               %% 2014/03/27
411     \MakeActiveLetHere\<<%           %% 2014/03/28
412     %% 2010/03/12
413     \MakeActiveLetHere\‘\lq \MakeActiveLetHere\’\rq
414     \MakeActiveLetHere\&\PDFcstring
415     \def\cs{134}%                     %% 2010/03/17, 2011/06/27

```

The typesetting version of `\BuildCsSyntax` (Sec. 2.2.4): 2014/07/16

```

416   \withcsname\def BuildCsSyntax \endcsname{\cs}%

... disables \niceverbNoVerbList; better switch off auto mode with section
headings TODO (modify \@startsection)

417   \let\decide_more_cs\bookmark_more_cs
418   }%

```

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.<sup>14</sup> as `\active`. We use `\protected@write` as this cares for `\nofiles`. `\@auxout` may be `\@partaux` for `\include`.

```

419   \newcommand*\{ \niceverb_aux_cat}[2]{%           %% 2010/03/14
420     \protected@write\@auxout{\string#1\string#2}}%

```

v0.5 restricts “activating” to `\MakeActiveLet`:

```

421 %   \renewcommand*\{ \MakeActive}[1]{%
422 %     \MakeActiveHere#1%
423 %     \niceverb_aux_cat\MakeActiveHere#1}%
424 \renewcommand*\{ \MakeActiveLet}[2]{%
425   \MakeActiveLetHere#1#2%
426 %   \niceverb_aux_cat\MakeActiveHere#1}%
427   \protected@write\@auxout{\string#1\string#2}}%
428   \renewcommand*\{ \MakeNormal}[1]{%
429     \MakeNormalHere#1%
430     \niceverb_aux_cat\MakeNormalHere#1}%
431   }{}%
432 }
433 }

```

<sup>14</sup>This uses `\@onelevelsanitize`, therefore `\protect` doesn’t change the behaviour of “active” characters.

### 2.3.4 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.

### 2.3.5 Listings and Moving

Working on v0.6, in testing I discovered a problem with the *listing environments*. The present documentation uses code listings with `makedoc`, which build on the `moreverb` package and eventually call L<sup>A</sup>T<sub>E</sub>X's `\@noligs` macro.<sup>15</sup> The problem also appears with the `{verbatim}` environment from the L<sup>A</sup>T<sub>E</sub>X kernel (`latex.ltx`) as well as from the `verbatim` package—with anything that calls L<sup>A</sup>T<sub>E</sub>X's `\@noligs`. The latter assigns special meanings to the active characters listed in the `\verbatim@nolig@list`, three of them need a different meaning with `niceverb`. When a page break happens after such an environment has been entered (this may well be when the environment falls to the beginning of the next page), these settings are used in L<sup>A</sup>T<sub>E</sub>X's `\@outputpage` for running the `\writes` of the page as well as for page headers. And this happens quite often in a package documentation!

The problem was reported by Walter Schmidt with respect to math primes as `latex/3104` in 1999. I cannot reproduce it, and I see two reasons in recent `latex.ltx` code why it cannot happen anymore. However, one remedy in `latex.ltx` is activating `\active@math@prime` in `\@resetactivechars` of `\@outputpage`. But this is bad for `niceverb`'s single right quote. We override the `\active@math@prime` functionality and `verbatim \@noligs` by appending some protection of the characters collected in `\verbatim@nolig@list` to `\@resetactivechars`. This solves the problem for `\writes` at `\shipout`. `\do_protect_noligs` is used for this purpose; actually it is applied in `\useNiceVerb` (Sec. 2.3.6):

```
434 \newcommand*\do_protect_noligs}[1]{%           %% 2014/03/28
435     \MakeActiveLetHere#1\relax}             %% 'Here' missing 2015/11/09
```

`\nvResetPages` can be used to restore L<sup>A</sup>T<sub>E</sub>X's `\@resetactivechars`. I don't add it to `\noNiceVerb` because it could corrupt `\writes`, so should be used with care.

```
436 \AtBeginDocument{%
437     \newlet\latex_reset_actives\@resetactivechars}
438 \newcommand*\nvResetPages}{%
439     \let\@resetactivechars\latex_reset_actives}
```

---

<sup>15</sup>`moreverb` is used, and its listing environments use `\@verbatim` from the `verbatim` package, then `\verbatim@font` calls `\@noligs`...

`\NiceVerbMove{text}` with v0.6 is for “moving” arguments so that niceverb syntax operates *locally* at the destination, I think of table of contents and page headers. It is automatically used by niceverb’s variant of L<sup>A</sup>T<sub>E</sub>X’s sectioning commands (Sec. 2.2.7); while with `\markboth`, `\markright`, `\addcontentsline` etc. you must include yourself (currently, `TODO`?). This is meant as a remedy against L<sup>A</sup>T<sub>E</sub>X’s and `verbatim`’s `\@noligs` with respect to page headers. However, another purpose is that you could switch off the niceverb syntax at the beginning of your document (by `\noNiceVerb`), though certain entries to the table of contents can use niceverb syntax without affecting other entries (where some active characters may have different meanings, perhaps from a different package).

```
440 \newcommand*{\NiceVerbMove}[1]{%
```

What goes to .aux files must not have underscores:

```
441     {\NiceVerbGeneral\niceverb_rqsf_kind#1}}
442 \NewSelfProtectedCommand{\NiceVerbGeneral}{%
443 % \newcommand*{\useNiceVerbHere}{%           %% 2014/03/28
444     \let\MakeActiveLet\MakeActiveLetHere \useNiceVerbI}
445 % \newcommand*{\NiceVerbGeneral}{%
446 %     \nvSelfProtect\niceverb_rqsf_kind#1\useNiceVerbHere}
```

### 2.3.6 Turning off and on altogether

These commands are new with v0.3.

`\noNiceVerb` *disables* all niceverb features.

```
447 \newcommand*{\noNiceVerb} {\MakeNormal\‘%
448     \MakeNormal\&%
449     \MakeNormal\<%
450     \MakeNormal\#%
451     \nvRightQuoteNormal
452     \MakeNormal\|}%
453 \let\@sect\niceverb_before_sect}           %% 2014/03/27
```

`\useNiceVerb` *activates* all the niceverb features (apart from “auto mode”). With v0.6, it is divided into two parts for `\NiceVerbMove` in Sec. 2.3.5:

```
454 \newcommand*{\useNiceVerbI}{\MakeActiveLet\‘\LQverb
```

`TODO` to be changed with `wiki.sty` v0.2

```
455     \MakeActiveLet\&\CmdSyntaxVerb
456     \MakeActiveLet\<\MetaVar
457     \MakeActiveLet\#\HashVerb
458     \nvRightQuoteSansSerif
459     \MakeActiveLet\|\nvCmdBox}
460 \newcommand*{\useNiceVerb} {\useNiceVerbI           %% 2014/03/27
461 \let\@sect\niceverb_sect}
```

```

462     \g@addto@macro\@resetactivechars{%
463     %                               \useNiceVerbHere           %% 2014/03/28
464     \let\do\do_protect_noligs \verbatim@nolig@list           %% 2014/03/28
465     }}

```

## 2.4 Minor Final Things

### 2.4.1 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.

```

466     \AtBeginDocument{\useNiceVerb}

```

### 2.4.2 Leave Package Mode

```

467     \PopLetterCat\_                                           %% 2012/08/27
468     \endinput

```

### 2.4.3 VERSION HISTORY

```

469     v0.1    2009/02/21    very first, sent to CTAN
470     v0.2    2009/04/04    ...NoVerbList: \subsubsection, \AddToMacro,
471             2009/04/05    \SimpleVerb makes more other than iii
472             2009/04/06    just uses \dospecials
473             2009/04/08    debugging code for rq/sf, +\relax
474             2009/04/09    +\verb@eol@error, prepared for new doc method,
475             removed spurious \makeat..., -\relax (ligature),
476             2009/04/10    ('-trick
477             2009/04/11    \@ after \SimpleVerb
478             2009/04/14    noted TODO below
479             2009/04/15    change v0.1 to 2009/02/21
480     v0.30   2010/02/27    short, more explained, \AssignCatCodeTo,
481             use \MakeActive for re-activating, \MakeNormal
482             2010/02/28    fixed @ and _ with & by moving \begin_min_verb;
483             replaced \lq by ‘; Capitals in Titles
484             2010/03/05    \SimpleVerb -> \NVerb;
485             use \MakeActive + \MakeNormal; \rq -> ‘;
486             renamed some sections; \lq_verb -> \LQverb,
487             \niceverb_meta -> \MetaVar,
488             \param_verb -> \HashVerb
489             2010/03/06    removed \MakeAlign; removed @ and _ todo below;
490             \NVerb makes ‘ and ’ other;
491             \nvAllowRQSF allows ‘ in column titles,
492             2010/03/08    \LQverb and & work in column titles,
493             \RQverb works with \noindent;
494             bookmark substitutions
495             2010/03/09    extended notes on ‘hyperref’ (in)compatibility;
496             \MakeLetter\@ in \CmdSyntaxVerb only;

```

497 |...| implemented as \prepareCmdBox etc.!  
 498 2010/03/10 \colorbox example, \thinspace; ltxguide!  
 499 removed todo; ...\_exec -> \DoRQsansserif;  
 500 minor doc changes in "Nasty"  
 501 2010/03/11 doc changes in "Escape Character ..." and  
 502 "Ampersand"  
 503 2010/03/12 \niceverb\_aux\_cat, \MakeActiveHere etc.,  
 504 \IfTypesetting, \noNiceVerb, \useNiceVerb,  
 505 corr. bracing mistake in \MakeNormal!  
 506 2010/03/14 0.31 -> 0.3; \HardNVerb, \GenCmdBox,  
 507 \prepareCmdBox -> \nvCmdBox  
 508 2010/03/15 \endcell; \cmdboxitem; remark on \sfcode/  
 509 2010/03/16 corr. -> \endline;  
 510 advice on \cs{par}, \cs{if...}, \cs{fi};  
 511 redefined \ignorespaces for RQ feature  
 512 2010/03/17 corr. '\fututelet', corr. \cs PDF substitution  
 513 2010/03/18 |\niceverbNoVerbList|, |\AddToMacro| etc. ;  
 514 corr. \AddToMacro;  
 515 \lastskip-fix of \niceverb\_ignore,  
 516 another fix of \niceverb\_noindent  
 517 2010/03/19 another fix of \niceverb\_ignore: \spacefactor  
 518 2010/03/20 ... again: \niceverb\_spacefactor  
 519  
 520 NOT DISTRIBUTED, just stored saved as separate version  
 521  
 522 v0.31 2010/03/20 right quote feature: letters get \sfcode=1001  
 523 'column title' -> 'page headers', \ctanpkgref  
 524  
 525 NOT DISTRIBUTED, just stored as separate version  
 526  
 527 v0.32 2010/03/21 taking best things from v0.30 and v0.31  
 528 2010/03/23 removed \relax from \IfTypesetting  
 529 SENT TO CTAN  
 530  
 531 v0.4 2010/03/27 restoring '\_' with "auto mode" safer  
 532 2010/03/28 \AddToNoVerbList  
 533 2010/03/29 note above, renamed v0.4  
 534 SENT TO CTAN  
 535  
 536 v0.41 2010/04/03 v0.33 -> v0.4  
 537 2010/04/05 corrected \AutoCmdInput list  
 538 SENT TO CTAN as part of NICETEXT release r0.41  
 539  
 540 v0.41a 2010/11/09 typo corrected  
 541 v0.42 2010/12/30 corr. '\ ' emulation in auto mode  
 542 2010/12/31 \MetaVar in ...maybe\_meta...  
 543 2011/01/19 '...' fix  
 544 2011/01/24 \ctanpkgref moves to texlinks.sty  
 545 2011/01/26 update (C)  
 546 with nicetext RELEASE r0.42

```

547 v0.43 2011/05/09 \gt, \lt
548         2011/05/27 \cs uses \@backslashchar
549         2011/06/20 \MakeActiveLetHere in \nice_maybe_meta_verb !!!
550         2011/06/27 2011/05/27 undone
551         2011/08/20 'r0.42', 'v0.43'
552 with nicetext RELEASE r0.43
553 v0.44 2011/09/09 \AddQuotes, \DontAddQuotes
554 with nicetext RELEASE r0.44
555 v0.45 2011/11/05 mod. \niceverb_collect_egroup/\VerticalCmdBox,
556         tried \output problem without avail
557         2011/12/05 clarified "r0.44"
558 with nicetext RELEASE r0.5
559
560 v0.5 2012/08/27 using 'catcodes', \providecommand\CatCode,
561         rm. \AssignCatCodeTo, \private_letters
562         2012/08/28 fixed \private_letters;
563         rewording for filling lines
564         2012/09/27 corrections about \MakeActive...
565 with nicetext RELEASE r0.6
566
567 v0.6 2012/11/27 \[d]qtd only \provide'd
568 v0.61 2014/03/18 doc.: rm. TODO on private letters hook,
569         folding history tighter,
570         RM CODE COMMENTED OUT IN 2011;
571         \VerticalCmdBox gets \pagebreak[2]
572         2014/03/19 doc.: strange replaced, restructured,
573         Command-Highlighting Boxes -> Boxes
574         Highlighting ...;
575         opt. arg. for \NVerb etc. replaces
576         \niceverb_egroup, \cs/\cstx enhanced,
577         reimpl.s with \SetNiceVerbSaveBox,
578         \nice_maybe_meta_verb -> \NiceMaybeMetaVerb
579         2014/03/20 reworking robustness -- doc., ...; doc. on
580         'Shared ...', \qtdnverb
581         2014/03/21 "debugging": \noexpand vs. \protect
582         2014/03/22 ... continued; mod. \MakeNormal
583         2014/03/23 ..., hiding ...
584         2014/03/24 TODO on left quotes, doc. test there,
585         rm. babel-TODO
586         2014/03/25 doc. about left quotes shorter, rm. earlier page
587         breaks, doc. problems with right quotes; \newlet;
588         dealing with \active@math@prime
589         2014/03/26 corr. test for right single quote, more about
590         \active@math@prime, corr. \CmdSyntaxVerb
591         2014/03/27 different treatment of \active@math@prime ...
592         main work for sec:listmv and independent
593         switching for rqsf, \NewSelfProtectedCommand
594         (3 applications); doc. corr., TODO;
595         \typeout test
596         2014/03/28 test section; [TODO?]; \nvShowProtectedEdef,

```



```
597             \MetaVar: protection and hyperref version;
598             remark \NVerb; [\NiceVerbHere]; \protect test
599             with RQ removed; \do_protect_noligs fixed
600             and used
601             2014/07/16 \BuildCsSyntax with hyperref
602             2014/07/17 remarks on \BuildCsSyntax and removing useless
603             braces there (hours of trying better)
604             2015/02/23 doc.: \PDFstring -> \pdfstring
605             2015/04/07 doc.: page headings -> page headers
606             2015/11/09 bugfixes \nvRightQuoteNormal and
607             \do_protect_noligs; doc. typo fix
608 with nicetext RELEASE r0.66
609 v0.62 2015/11/20f. \qtdnverb was disabled!
610             doc.: fn. on \qtdnverb obsolete
611
612
```