

niceverb.sty

Minimizing Markup for Documenting L^AT_EX packages*

Uwe Lück[†]

September 9, 2011

Abstract

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

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

Contents

1	Presenting <code>niceverb</code>	2
1.1	Purpose	2
1.2	Acknowledgement/Basic Ideas	2
1.3	The Commands and Features of <code>niceverb</code>	3
1.4	Examples	6
1.5	What is Wrong with the Present Version	6
2	Implementation of the Markup Syntax	7
2.1	Switching Category Codes	7
2.2	Robustness by <code>\IfTypesetting</code>	9
2.3	<code>\NVerb</code>	9
2.4	Single Quotes Typeset Meta-Code	10
2.5	Ampersand (or <code>\cstx</code>) Typesets Meta-Code	11
2.6	Escape Character Typesets Meta-Code	12
2.7	Meta-Variables	14
2.8	Hash Mark is Code	15
2.9	Single Right Quotes for <code>\textsf</code>	15

*This document describes version [v0.44](#) of `niceverb.sty` as of 2011/09/09.

[†]<http://contact-ednotes.sty.de.vu>

2.10	Command-Highlighting Boxes	17
2.11	When <code>niceverb</code> Gets Nasty	19
2.11.1	Meta-Variables	19
2.11.2	Quotes	19
2.11.3	<code>hyperref</code>	20
2.11.4	<code>hyper-xr</code>	21
2.11.5	Turning off and on altogether	21
2.12	Activating the <code>niceverb</code> Syntax	22
2.13	Leave Package Mode	22
2.14	VERSION HISTORY	22

1 Presenting `niceverb`

1.1 Purpose

The `niceverb` package provides “minimal” markup for documenting L^AT_EX 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 T_EX). As opposed to tools that are rather common on UNIX/Linux, this operation should work at any T_EX 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 L^AT_EX.

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 L^AT_EX-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 L^AT_EX Project Team. The present feature, however, also works in plain text outside verbatim mode.

³On 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) `\code{<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 L^AT_EX. To get *usual* single quotes, you can use their standard substitutes `\lq` and `\rq`, or for pairs of them, `\lqtd{<text>}` in place of `\lq<text>\rq`—or even `\lq<text>\rq\lq`. 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 `\code{<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{<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!

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

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 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 a different package overtook the active `\<char>` (but I expect more failures

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

then), in this case `\MakeActiveLet\langle char \rangle\langle perm-alias \rangle` revives the `niceverb` meaning of `\langle char \rangle` where `\langle perm-alias \rangle` is the “permanent alias” for that active `\langle char \rangle` 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 *alltogether* 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

`\'\langle text \rangle\'`

replacing `\textit{\langle text \rangle}` or `\emph{\langle text \rangle}` 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`.

⁷<http://ctan.org/pkg/substr>

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}[2011/09/09 v0.44
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.

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

`\MakeLetter\langle char \rangle` is used for *private letters*, i.e., to allow `\langle char \rangle` in “internal”, non-user control sequences (*TEXbook* Chap. 3). `\MakeOther` is just a different implementation of L^AT_EX’s `\@makeother`.

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

... overriding `\finddo` if ...

`\MakeActive\langle char \rangle` just revives the meaning of `\langle char \rangle` 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 `\langle char \rangle` actively in the meantime ...

```
25 \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`.

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

`\MakeActiveLet\langle char \rangle\langle macro-name \rangle` activates `\langle char \rangle` and then gives it the meaning of `\langle macro name \rangle`.

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

We take a copy `\MakeActiveLetHere` as well.

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

We use the “underscore” as a private letter (the L^AT_EX2 Project Team likes it as well). Its usual meaning can be restored by `\MakeNormal_`. For restoring the usual category codes of T_EX’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.)

```
35 \def\do#1{\expandafter
36   \chardef \csname normal_catcode_\string#1\expandafter \endcsname
37   \CatCode#1\relax}
38 \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):

```
39 \MakeLetter\_
40
41 % \newcommand*{\make_iii_other}{\MakeOther\\\MakeOther\{\MakeOther\}}
42 %% <- replaced 2009/04/05
```

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

```
43 \newcommand*{\MakeNormal}[1]{%
44   \ifundefined{norm_catc_str#1}%
45     {\MakeOther#1}%
46     {\AssignCatCodeTo{csname\norm_catc_str#1\endcsname}#1}%
47 \newcommand*{\norm_catc_str}{normal_catcode_\string}
48 %% TODO add ^^I and ^^M
```

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

```
49 \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`.

```
50 \providecommand*{\IfTypesetting}{%
51 %   \relax
```

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

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

2.3 `\NVerb`

`\begin_min_verb` is a beginning shared by some macros here. It begins like L^AT_EX’s `\verb`, apart from the final `\tt`.

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

`\NVerb⟨char⟩⟨code⟩⟨char⟩`

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

⁸L^AT_EX’s `\nfss@catcodes` is similar, but it makes space-like characters ignored. Also cf. `ltfinal.dtx`. [TODO](#): `\RestoreNormalCatcodes`.

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

```
60 \newcommand*{\HardNVerb}{%
61     \_no_nice_meta_verb_true \nice_maybe_meta_verb}
62 \newif\if_no_nice_meta_verb_
63 \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.

```
64 \IfTypesetting{%
65     \begin_min_verb
66     \let\do\MakeOther \dospecials
```

Turn off niceverb specials:

```
67     \MakeOther\|\MakeOther\'\MakeOther\'%
68     \if_no_nice_meta_verb_ \MakeOther\<%
69     %% \else \MakeActiveLet\<\MetaVar %% 2010/12/31
70     \else \MakeActiveLetHere\<\MetaVar %% 2011/06/20
71     \fi
72     \MakeActiveLetHere #1\niceverb_egroup
73     \verb@eol@error %% TODO change message 2009/04/09
74 }{\string\NVerb \string#1}}
```

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

```
75 \newcommand*{\niceverb_normal_egroup}{%
76     \egroup
```

2011/09/09 adding `\niceverb_maybe_rq` for `\AddQuotes`:

```
77 \niceverb_maybe_rq
78 \ifmode\else\@fi}
79 \@ifdefinable\niceverb_egroup
80 {\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.)

```
81 \MakeActive\‘
82 \newcommand*{\LQverb}{%
83     \IfTypesetting{\lq_double_test}{\protect\‘}
84     \MakeOther\‘
85     \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):

```

86     \begingroup
87     \let\do\MakeOther \dospecials
88     \MakeOther\|%% 2010/03/09!
89     \futurelet\let_token \lq_double_decide}
90 \newcommand*{\lq_double_decide}{%
91   \ifx\let_token\LQverb
92   \endgroup
93   ‘‘\expandafter \@gobble

```

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

```

94   \else
95   \ifx\let_token\protect
96   \expandafter\expandafter\expandafter \lq_double_decide_ii
97   \else
98   \endgroup
99   \niceverb_maybe_qs                      %% 2011/09/09
100   \expandafter\expandafter\expandafter \NVerb
101   \expandafter\expandafter\expandafter \'%
102   \fi
103   \fi}

```

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

```

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

```

2.5 Ampersand (or `\cstx`) Typesets Meta-Code

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

```

106 \MakeActive\&
107 \newcommand*{\CmdSyntaxVerb}{%
108   \IfTypesetting{%
109     \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!

```

110     \MakeLetter\@\MakeLetter\__%
111     \cmd_syntax_verb
112   }\protect&\string}}
113 \MakeNormal\&
114 \newcommand*{\cmd_syntax_verb}[1]{%
115   \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 `\labels`, so we provide something like `\cs{characters}` from `tugboat.sty`.

```
116 \DeclareRobustCommand*\cs}[1]{%
117   \begin_min_verb \backslash_verb #1\egroup}
118 \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.

```
119 \DeclareRobustCommand*\cstx}[1]{%           %% corr. 2010/03/17
120   \begin_min_verb \backslash_verb #1\futurelet\let_token \after_cs}
121 \newcommand*\after_cs{%
122   \ifcat\noexpand\let_token a\egroup \space
123   \else \expandafter \decide_verb \fi}
124 \newcommand*\test_more_verb{\futurelet\let_token \decide_verb}
125 \newcommand*\decide_verb{%
126   \jumpteg_on_with\bgroup\braces_verb
127   \jumpteg_on_with[\brackets_verb
128   \jumpteg_on_with*\star_verb
129   \egroup}
130   %% CAUTION/TODO wrong before (... if cmd without arg
131   %%           use \ then or choose usual verb...
132   %%           or \MakeLetter\ ( etc. ... or \xspace
133 \newcommand*\jumpteg_on_with}[2]{%
134   \ifx\let_token#1\do_jumpteg_with#2\fi}
```

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

```
135 \def\do_jumpteg_with#1#2\egroup{\fi#1}
136 \def\braces_verb#1{\string{#1\string}\test_more_verb}
137 \def\brackets_verb[#1]{[#1]\test_more_verb}
138 \def\star_verb*{*\test_more_verb}
139   %% 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 `&`:

```
140 \let\endcell&
```

2.6 Escape Character Typesets Meta-Code

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

```

141 \DeclareRobustCommand*\BuildCsSyntax}{%
142   \futurelet\let_token \build_cs_syntax_sp}
143 \newcommand*\build_cs_syntax_sp}{%
144   \ifx\let_token\@sptoken
145     \@%                               %% 2010/12/30
146   \else %% TODO ^~M!?
147     \expandafter \start_build_cs_syntax
148   \fi}
149 \newcommand*\start_build_cs_syntax}[1]{%
150   \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:

```

151   \edef\before_build_cs_sub{\the\CatCode\_}%
152   \MakeLetter\_ \MakeLetter\@%% CAUTION, cf. ...
153   \test_more_cs}
154 \newcommand*\test_more_cs}{%
155   \futurelet\let_token \decide_more_cs}
156 \newcommand*\decide_more_cs}{%
157   \ifcat\noexpand\let_token a\expandafter \add_to_cs
158   \else
159   %   \MakeNormalHere\_

```

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

```

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

```

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

```

179 \newcommand*\AutoCmdSyntaxVerb}{%
180   \MakeActiveLetHere\\\BuildCsSyntax}
181 \newcommand*\EndAutoCmdSyntaxVerb}{\CatCode\\\z@}

```

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

```
182 \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:

```
183 % \futurelet\LetToken\relax \relax
184 % \show\LetToken \typeout{\ifcat\noexpand\LetToken aa\else x\fi}
```

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

```
185 \newcommand*{\niceverbNoVerbList}{%
186   \begin\end\item\verb\EndAutoCmdSyntaxVerb\NormalCommand
187   \section\subsection\subsubsection} %% TODO!?
```

`\AddToMacro{\niceverbNoVerbList}{<macros>}` can be used to add `<macros>` to that list.

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

Hey, or just `\AddToNoVerbList{<macros>}`:

```
191 \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:”

```
192 \newcommand*{\AutoCmdInput}[1]{%
193   \begingroup
194     \AddToMacro\niceverbNoVerbList{\ProvidesFile}%
195     %% <- removed ‘\endinput’, will be code! 2010/04/05
196     \AutoCmdSyntaxVerb
197     \input{#1}%
198     \EndAutoCmdSyntaxVerb
199   \endgroup
200 }
```

2.7 Meta-Variables

`\MetaVar<var-id>` will be a permanent alias for the active ‘<’.

```
201 \def\MetaVar#1>{%
202   \mbox{\normalfont\itshape $\langle$#1\rangle$}}
203   %% 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.

```
204 \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.

```
205 \MakeActive\'
206 \newcommand*\RQsansserif{%
207   \IfTypesetting{\niceverb_rq_sf_test}{\protect'}
208 \MakeOther\'
```

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

```
209 \newcommand*\niceverb_rq_sf_test{%
210   \ifhmode
211     \ifdim\lastskip>z@
212       \expandafter\expandafter\expandafter \DoRQsansserif
213     \else
214       \ifnum\niceverb_spacefactor
215         \expandafter\expandafter\expandafter\expandafter
216         \expandafter\expandafter\expandafter
217         \DoRQsansserif
218       \else '\fi
219     \fi
220   \else \ifvmode
221     \expandafter\expandafter\expandafter \DoRQsansserif
222   \else '\fi
223 \fi}
```

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

```
224 \MakeActive\'
225 \@ifdefinable\DoRQsansserif
226   {\def\DoRQsansserif#1'\textsf{#1}}
227 \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 L^AT_EX’s internal `\@sect` and of T_EX’s primitives

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

```
228 \newcommand*{\nvAllowRQSS}{%
229     \MakeActiveLetHere'\RQsansserif
230     \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:

```
231 \AtBeginDocument{%
232     \edef\before_niceverb_parenthesis{\the\scode'\{}%
233     \let \before_niceverb_ignore \ignorespaces    %% 2010/03/16
234     \let \before_niceverb_sect \@sect
235     \let \before_niceverb_noindent \noindent}    %% 2010/03/08
```

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

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

2010/03/20:

```
240 \newcommand*{\niceverb_spacefactor}{\spacefactor=1001\relax}
241 \newcommand*{\niceverb_noindent}{%
242     \before_niceverb_noindent \niceverb_spacefactor}
243 \newcommand*{\niceverb_ignore}{%
244     \ifhmode \niceverb_spacefactor \fi \before_niceverb_ignore}
```

Here are the main switches:

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

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

```
248 % \let\@footnotetext\niceverb_footnotetext
249 \let\ignorespaces\niceverb_ignore    %% 2010/03/16
250 \let\@sect\niceverb_sect
251 \let\noindent\niceverb_noindent}    %% 2010/03/08
252 \newcommand*{\nvRightQuoteNormal}{%
253     \MakeNormal'\%                %% 2010/03/21
254     \scode'\(=\before_niceverb_parenthesis\relax
255     \let\ignorespaces\before_niceverb_ignore    %% 2010/03/16
256     \let\@sect\before_niceverb_sect
257     \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.

```
258 \newcommand*{\nvAllRightQuotesSansSerif}{%
259     \nvRightQuoteNormal
260     \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>}`:

```
261 % \DeclareRobustCommand*{\ctanpkgref}[1]{%
262 %     \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>|`.

```
263 \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.

```
264 \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).

```
265 \newcommand*{\HardVerbBox}{\_no_nice_meta_verb_true \gen_cmd_box}
266 \newcommand*{\gen_cmd_box}{%
267     \bgroup
268     \let\niceverb_egroup\nice_collect_verb_egroup
269     \global %% TODO!? for \cmdboxitem 2010/03/15
270     %% <- TODO replace \niceverb_egroup by parameter,
271     %%         save one nesting level 2010/03/15
272     \setbox\niceverb_savebox \hbox\bgroup
273     \if_no_nice_meta_verb_
274         \expandafter \HardNVerb
275     \else \expandafter \NVerb \fi
276 \newcommand*{\nice_collect_verb_egroup}{%
277     \egroup \egroup
278     \ifvmode \expandafter \VerticalCmdBox
279     \else \ifmmode \hbox \fi
280         \expandafter \InlineCmdBox \fi
281     {\box\niceverb_savebox}%
282 \niceverb_normal_egroup}
```

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

```
283 \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`):

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

The command declaration boxes in the documentation of Nicola Talbot’s `data-tool` 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.

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

```
286 \bgroup
```

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

```
287 \fboxsep 1pt
288 \kern\SetOffInlineCmdBoxOuter
289 \smash{\SetOffInlineCmdBox{\kern\SetOffInlineCmdBoxInner
290 \InlineCmdBoxArea{#1}%
291 \kern\SetOffInlineCmdBoxInner}}%
292 \mathstrut
293 \kern\SetOffInlineCmdBoxOuter
294 \egroup
295 }
```

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

```
296 \@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[cmk]{.1,0,.2,.05}}
```

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

```
297 \newcommand*{\SetOffInlineCmdBoxInner}{-\fbboxsep\thinspace}
```

This choice is inspired by `\cstok` for “boxed” things in Knuth’s `manmac.tex` which formats *The T_EXbook*.

`\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).

```
298 \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.

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

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

```
301 \newcommand*{\cmdboxitem}{%
302   \bgroup
303   \let\niceverb_egroup\cmd_item_egroup
304   \global %% TODO!? 2010/03/15
305   \setbox\niceverb_savebox \hbox\bgroup
306   \NVerb}
307 \newcommand*{\cmd_item_egroup}{%
308   \egroup \egroup \egroup
309   \item[\InlineCmdBox{\box\niceverb_savebox}]}
```

2.11 When `niceverb` Gets Nasty

These things are new with v0.3.

2.11.1 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:

```
310 \providecommand*{\gt}{>}
311 \providecommand*{\lt}{<}
```

2.11.2 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.

```
312 \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.

```
313 % TODO \qtdverb!? alternative meaning for \LQverb!? 2010/03/06
314 %      rather rare, & takes less space                2010/03/09
```

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

```
315 \newcommand*{\AddQuotes}{%
316     \let\niceverb_maybe_qs\niceverb_add_qs}
317 \newcommand*{\niceverb_add_qs}{%
```

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

```
318     \ifmmode\else
319     ‘\let\niceverb_maybe_rq\niceverb_rq
320     \fi}
321 \@ifdefinable\niceverb_maybe_rq{\let\niceverb_maybe_rq\relax}
322 \newcommand*{\niceverb_rq}{‘\let\niceverb_maybe_rq\relax}
```

You can undo this by `\DontAddQuotes`:

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

The default will be the behaviour that we had before:

```
324 \DontAddQuotes
```

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

```
325 \newcommand*{\dqtd}[1]{‘‘#1’’}
```

2.11.3 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.⁹

```
326 \AtBeginDocument{%
327     \@ifpackageloaded{hyperref}{%
328         \newcommand*{\PDFcstring}{%          %% moved here 2010/03/09
329             \134\expandafter@gobble\string}% %% ASCII octal encoding
330         \pdfstringdefDisableCommands{%
331             \let\nvAllowRQSS\empty           %% not \relax 2010/03/12
332             %% 2010/03/12
333             \MakeActiveLetHere\‘\lq \MakeActiveLetHere\’\rq
334             \MakeActiveLetHere\&\PDFcstring
335             \def\cs{134}%                     %% 2010/03/17, 2011/06/27
336         }%
```

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

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`.

```

337     \newcommand*{\niceverb_aux_cat}[2]{%           %% 2010/03/14
338         \protected@write\@auxout{}\string#1\string#2}%
339     \renewcommand*{\MakeActive}[1]{%
340         \MakeActiveHere#1%
341         \niceverb_aux_cat\MakeActiveHere#1}%
342     \renewcommand*{\MakeActiveLet}[2]{%
343         \MakeActiveLetHere#1#2%
344         \niceverb_aux_cat\MakeActiveHere#1}%
345     \renewcommand*{\MakeNormal}[1]{%
346         \MakeNormalHere#1%
347         \niceverb_aux_cat\MakeNormalHere#1}%
348     }{}%
349 }
```

[TODO](#) doesn't babel have the same problem? 2010/03/12

2.11.4 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.

2.11.5 Turning off and on altogether

These commands are new with v0.3.

`\noNiceVerb` *disables* all niceverb features.

```

350 \newcommand*{\noNiceVerb}{\MakeNormal\`%
351                               \MakeNormal\&%
352                               \MakeNormal\<%
353                               \MakeNormal\#%
354                               \nvRightQuoteNormal
355                               \MakeNormal\|}
```

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

```

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

[TODO](#) to be changed with `wiki.sty` v0.2

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

```

357 \MakeActiveLet\&\CmdSyntaxVerb
358 \MakeActiveLet\<\MetaVar
359 \MakeActiveLet\#\HashVerb
360 \nvRightQuoteSansSerif
361 \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.

```

362 \AtBeginDocument{\useNiceVerb}

```

2.13 Leave Package Mode

```

363 \MakeNormalHere\_% 2010/03/12
364 \endinput

```

2.14 VERSION HISTORY

```

365 v0.1 2009/02/21 very first, sent to CTAN
366 v0.2 2009/04/04 ...NoVerbList: \subsubsection, \AddToMacro,
367 2009/04/05 \SimpleVerb makes more other than iii
368 2009/04/06 just uses \dospecials
369 2009/04/08 debugging code for rq/sf, +\relax
370 2009/04/09 +\verb@eol@error, prepared for new doc method,
371 removed spurious \makeat..., -\relax (ligature),
372 ('-trick
373 2009/04/11 \@ after \SimpleVerb
374 2009/04/14 noted TODO below
375 2009/04/15 change v0.1 to 2009/02/21
376 v0.30 2010/02/27 short, more explained, \AssignCatCodeTo,
377 use \MakeActive for re-activating, \MakeNormal
378 2010/02/28 fixed @ and _ with & by moving \begin_min_verb;
379 replaced \lq by ‘; Capitals in Titles
380 2010/03/05 \SimpleVerb -> \NVerb;
381 use \MakeActive + \MakeNormal; \rq -> ‘;
382 renamed some sections; \lq_verb -> \LQverb,
383 \niceverb_meta -> \MetaVar,
384 \param_verb -> \HashVerb
385 2010/03/06 removed \MakeAlign; removed @ and _ todo below;
386 \NVerb makes ‘ and ’ other;
387 \nvAllowRQSF allows ‘ in column titles,
388 2010/03/08 \LQverb and & work in column titles,
389 \RQverb works with \noindent;
390 bookmark substitutions
391 2010/03/09 extended notes on ‘hyperref’ (in)compatibility;
392 \MakeLetter\@ in \CmdSyntaxVerb only;

```

```

393      [...] implemented as \prepareCmdBox etc.!
394      2010/03/10 \colorbox example, \thinspace; ltxguide!;
395      removed todo; ..._exec -> \DoRQsansserif;
396      minor doc changes in "Nasty"
397      2010/03/11 doc changes in "Escape Character ..." and
398      "Ampersand"
399      2010/03/12 \niceverb_aux_cat, \MakeActiveHere etc.,
400      \IfTypesetting, \noNiceVerb, \useNiceVerb,
401      corr. bracing mistake in \MakeNormal!
402      2010/03/14 0.31 -> 0.3; \HardNVerb, \GenCmdBox,
403      \prepareCmdBox -> \nvCmdBox
404      2010/03/15 \endcell; \cmdboxitem; remark on \sfcode'/
405      2010/03/16 corr. -> \endline;
406      advice on \cs{par}, \cs{if...}, \cs{fi};
407      redefined \ignorespaces for RQ feature
408      2010/03/17 corr. 'futelet', corr. \cs PDF substitution
409      2010/03/18 \niceverbNoVerbList, \AddToMacro etc.;
410      corr. \AddToMacro;
411      \lastskip-fix of \niceverb_ignore,
412      another fix of \niceverb_noindent
413      2010/03/19 another fix of \niceverb_ignore: \spacefactor
414      2010/03/20 ... again: \niceverb_spacefactor
415
416      NOT DISTRIBUTED, just stored saved as separate version
417
418      v0.31 2010/03/20 right quote feature: letters get \sfcode=1001
419      'column title' -> 'page headers', \ctanpkgref
420
421      NOT DISTRIBUTED, just stored as separate version
422
423      v0.32 2010/03/21 taking best things from v0.30 and v0.31
424      2010/03/23 removed \relax from \IfTypesetting
425      SENT TO CTAN
426
427      v0.4 2010/03/27 restoring '_' with "auto mode" safer
428      2010/03/28 \AddToNoVerbList
429      2010/03/29 note above, renamed v0.4
430      SENT TO CTAN
431
432      v0.41 2010/04/03 v0.33 -> v0.4
433      2010/04/05 corrected \AutoCmdInput list
434      SENT TO CTAN as part of NICETEXT release r0.41
435
436      v0.41a 2010/11/09 typo corrected
437      v0.42 2010/12/30 corr. '\ ' emulation in auto mode
438      2010/12/31 \MetaVar in ...maybe_meta...
439      2011/01/19 '...' fix
440      2011/01/24 \ctanpkgref moves to texlinks.sty
441      2011/01/26 update (C)
442      with nicetext RELEASE r0.42

```

```
443 v0.43 2011/05/09 \gt, \lt
444         2011/05/27 \cs uses \@backslashchar
445         2011/06/20 \MakeActiveLetHere in \nice_maybe_meta_verb !!!
446         2011/06/27 2011/05/27 undone
447         2011/08/20 'r0.42', 'v0.43'
448 with nicetext RELEASE r0.43
449 v0.44 2011/09/09 \AddQuotes, \DontAddQuotes
450
451
```