ifnextok

—

\IfNextToken instead of \@ifnextchar Does Not Skip Blank Spaces, [and '\\ [' may print bracket in new line]*

Uwe Lück†

June 27, 2011

Abstract

The ifnextok package deals with the behavior of LaTeX's internal `\@ifnextchar` to skip blank spaces. This sometimes has surprising or for some users really *unwanted* effects, especially with brackets following `\\` when the user does *not* intend to specify an optional argument, rather wants that brackets are *printed*. The package offers commands and options for modifying this behavior, maybe limited to certain parts of the document source.
[It works!] It may also be useful with active characters in lieu of `\\`, e.g., the double quote `"` with `german.sty` or babel.

v0.3 fixes behavior in non-typesetting mode with `\MakeNotSkipping`, using a somewhat different technique than LaTeX's robustifications.

Keywords: macro programming, optional command arguments, manual line breaks, humanities

Related packages: amsmath, mathtools

*This document describes version v0.3 of `ifnextok.sty` as of 2011/06/27.

†`http://contact-ednotes.sty.de.vu`

Contents

1 Installing and Calling

The package file `ifnextok.sty` is provided ready, installation only requires putting it somewhere where TEX finds it (which may need updating the file-name data base).[1]

Below the `\documentclass` line(s) and above `\begin{document}`, you load `ifnextok.sty` (as usually) by

`\usepackage{ifnextok}` or by `\usepackage[`⟨*options*⟩`]{ifnextok}`

—⟨*options*⟩ described in Section 7. E.g., the *main goal* of writing the package is achieved by

```
\usepackage[stdbreaks]{ifnextok}
```

2 Header (Legalize)

```
1  \NeedsTeXFormat{LaTeX2e}[1994/12/01]
2  \ProvidesPackage{ifnextok}[2011/06/27 v0.3 test next token (UL)]
3
4  %% Copyright (C) 2011 Uwe Lueck,
5  %% http://www.contact-ednotes.sty.de.vu
6  %% -- author-maintained in the sense of LPPL below --
7  %%
8  %% This file can be redistributed and/or modified under
9  %% the terms of the LaTeX Project Public License; either
10 %% version 1.3c of the License, or any later version.
11 %% The latest version of this license is in
12 %%     http://www.latex-project.org/lppl.txt
13 %% We did our best to help you, but there is NO WARRANTY.
14 %%
15 %% Please report bugs, problems, and suggestions via
16 %%
17 %%   http://www.contact-ednotes.sty.de.vu
18 %%
```

3 Outline

The ifnextok package deals with the behavior of LATEX's internal `\@ifnextchar` to skip blank spaces. This sometimes has surprising or for some users really *unwanted* effects, especially with brackets following `\\` when the user does *not* intend to specify an optional argument, rather wants that brackets are *printed.* The package offers commands and options for modifying this behavior, maybe limited to certain parts of the document source. They are described in the sections below together with the presentation of the implementation.

[1] `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=inst-wlcf`

As after multiletter commands blank spaces are skipped anyway (TEXbook p. 46f.), the package makes a *difference only for one-symbol commands* such as `\\`, or for *active characters* such as the double quote with `german.sty` and babel. (v0.21: Or also optional arguments *following mandatory ones*—"trailing" optional arguments mentioned by Lars Hellström and Bruno Le Floch on `LATEX-L`. v0.21a: Moreover, with "starred" command versions having a first optional argument!)

Similar things happen in amsmath and mathtools, and as of v0.21, we discuss relations to these packages.

A little **overview** of the package's commands and options:

3.1 For Macro Writers

1. `\IfNextToken` is an alternative to `\@ifnextchar`, not skipping spaces (Section 5.1). This macro is the **low-level** backbone of all other modifications of LATEX commands.

2. `\IfStarNextToken` is an alternative to `\@ifstar`, not skipping spaces, using `\IfNextToken` in lieu of `\@ifnextchar` (Section 5.4).

3. Some "**patching**" commands aim at modifying existing (LATEX) macros without specifying the resulting new definition explicitly (Sections 5.2 and 5.4). As a package writer, you just must know which macros need to be modified and specify their names as arguments for the patching macros.

4. There are low-level commands `\INTstore` and `\INTrestore` for undoing modifications of existing macros (Section 5.3).

3.2 For End-Users

There are **high-level** commands for modifying `\\` and selecting LATEX **environments** to be affected (Section 6). Package **options** execute some of them (Section 7), e.g., `[stdbreaks]`.

3.3 Intermediate

`\MakeNotSkipping{`⟨*target*⟩`}{`⟨*on-space*⟩`}` described in Section 5.5 is somewhat "intermediate." It acts on a document-level command ⟨*target*⟩ without any assumptions about its internals. On the other hand, choosing ⟨*on-space*⟩ for the new behavior of ⟨*target*⟩ in front of a space token may need some knowledge ...

(TODO: how command names are composed)

4 Caveats

1. Testing has not been very comprehensive so far. Usage together with amsmath may require special care or fail altogether.

2. Switching into "don't-skip-spaces" mode *two times* without switching back into normal mode in between won't work with this version (v0.1–v0.3 TODO) of the package (TODO: permanent aliases). You will get the

   `Argument of` ⟨*patching*⟩ `has an extra }.`

   error. This also applies to commands that have been issued by package options.

3. Implementation may change much. (TODO 0.3)

5 For Making Macros

5.1 The Main Command `\IfNextToken`

`\IfNextToken`⟨*match*⟩`{`⟨*if*⟩`}{`⟨*else*⟩`}` is the obvious variant of LaTeX's internal `\@ifnextchar` executing ⟨*if*⟩ if actually the *"very next"* token is ⟨*match*⟩ and executing ⟨*else*⟩ otherwise. If ⟨*match*⟩ is *not* a *space token* (LaTeX's `\@sptoken`) but the next token *is*, ⟨*else*⟩ is executed; while `\@ifnextchar` tries matching the next token after ensuing space tokens.

```
\newcommand{\IfNextToken}[3]{%
    \let\nextok@match= #1%
```

... v0.21 adds '`= `' after Heiko Oberdiek's explanation on texhax, this allows `\@sptoken` as a possible `#1`.

```
    \def\nextok@if{#2}\def\nextok@else{#3}%
    \futurelet\@let@token\nextok@decide}
```

... apart from using different names, this is the same as `\new@ifnextchar` in `amsgen.sty` of the amsmath bundle:

```
\long\def\new@ifnextchar#1#2#3{%
  \let\reserved@d= #1%
  \def\reserved@a{#2}\def\reserved@b{#3}%
  \futurelet\@let@token\new@ifnch
}
```

... and the behavior is essentially the same ...

```
\def\nextok@decide{%
    \ifx\@let@token\nextok@match \expandafter\nextok@if
    \else                        \expandafter\nextok@else
    \fi}
```

The analogue to our `\nextok@decide` in amsmath/`amsgen.sty` is `\new@ifnch`:

```
\def\new@ifnch{%
  \ifx\@let@token\reserved@d \let\reserved@b\reserved@a \fi
  \reserved@b
}
```

. . . and these two macros (ifnextok's and amsmath's) actually make the difference to Standard LaTeX. The latter's `\@ifnch` tests for `\@sptoken` before looking for the actually wanted `char`, ifnextok and amsmath don't. As to `\new@ifnch` vs. `\nextok@decide`, the first has one token less than the latter, but one assignment more. What does this mean? TODO

When I decided to create the ifnextok package, I was not aware of the similarity to amsmath, and I am not sure what I would have done had I . . .

`\NoNextSkipping` now switches into "don't-skip-spaces" mode "altogether" (however . . .):

```
\newcommand*{\NoNextSkipping}{\let\@ifnextchar\IfNextToken}
```

This appears so dangerous to me that I don't want to support it much right now. `\RestoreNextSkipping` just switches back to LaTeX's original version, so some support for amsmath may be missing here.

```
\newcommand*{\RestoreNextSkipping}{%
    \let\@ifnextchar\kernel@ifnextchar}
```

Actually, because `\NoNextSkipping` does not affect `\kernel@ifnextchar`, those of LaTeX's commands using the latter still will skip spaces (with package version v0.1).

As opposed to amsmath, ifnextok aims at more choices as to what document-level commands are affected by the modified `next` checking. Of course, amsmath deals with breaks between math display lines, while the present package rather was motivated by experiences in the humanities.

`\@sptoken` was discussed under 'Some puzzling TeX' on texhax in 2011 (February/May/June), and the matter is discussed in *The TeXbook* in Exercise 24.6 and on pp. 376f.

5.2 "Bold" Patching Commands

`\INTpatch`⟨*replacer*⟩⟨*macro*⟩ replaces something in the definition of ⟨*macro*⟩ according to the replacement macro ⟨*replacer*⟩. This seems to work with the macros I thought of. It does *not* work when (for replacing `\@ifnextchar`) (a) there are *more* `\@ifnextchar`s in the macro to patch (outside braces), or when (b) the only `\@ifnextchar` is inside a pair of braces.

```
\newcommand*{\INTpatch}[2]{%
  \expandafter\expandafter\expandafter \def
    \expandafter\expandafter\expandafter #2%
      \expandafter\expandafter\expandafter {%
        \expandafter #1#2}}                    %% red. 2011/06/24
```

`\NextTestPatch`⟨*macro*⟩ replaces `\@ifnextchar` in the definition of ⟨*macro*⟩ by `\IfNextToken`.

```
\newcommand*{\NextTestPatch}{\INTpatch\nextok@patch}
\def\nextok@patch#1\@ifnextchar{#1\IfNextToken} %% red. 2011/06/24
```

Another application of `\INTpatch` is `\StarTestPatch` in Section 5.4.

5.3 Storing and Restoring

`\INTstore`⟨*macro*⟩ stores the meaning of the macro ⟨*macro*⟩ in a special name space.

```
\newcommand*{\INTstore}[1]{%
    \expandafter\let\csname\INT@save#1\endcsname#1}
```

In order to apply `\MakeNotSkipping` even to active characters below (v0.2), nothing must be gobbled from `\string`⟨*token*⟩:

```
% \newcommand*{\INT@save}{INT.save\expandafter\@gobble\string}
\newcommand*{\INT@save}{INT.save\string}
```

`\INTrestore`⟨*macro*⟩ *r*estores the meaning of ⟨*macro*⟩ that is expected to have been stored with `\INTstore`:

```
\newcommand*{\INTrestore}[1]{%
    \expandafter\let\expandafter#1\csname\INT@save#1\endcsname}
```

5.4 The Star Test

Before a LaTeX line-break command tests for an optional argument, it tests for a star using `\@ifstar`, which in turn invokes `\@ifnextchar`. So already `\@ifstar` needs to be modified. We do not so much want to change `\@ifstar` altogether, rather we will replace it at some places by a non-skipping variant `\IfStarNextToken`,[2] using the patching command `\StarTestPatch`⟨*macro*⟩. (`\@ifstar` has an argument and therefore cannot be patched as nicely as the line-break commands.)

```
\newcommand*{\IfStarNextToken}[1]{\IfNextToken*{\@firstoftwo{#1}}}
\newcommand*{\StarTestPatch}{\INTpatch\nextok@starpatch}
```

The macro to be patched may contain a `\par` (`\@centercr` is an example), so we need `\long`:

```
\long\def\nextok@starpatch#1\@ifstar{#1\IfStarNextToken}
```

`\StoreStarSkipping` stores the current meaning of `\@ifstar` ...

```
\newcommand*{\StoreStarSkipping}{\INTstore\@ifstar}
```

[2] TODO or `\IfNextStar`, cf. `\IfNextSpace`.

... so that it can be restored by `\RestoreStarSkipping`:

```
47 \newcommand*{\RestoreStarSkipping}{\INTrestore\@ifstar}
```

`\NoStarSkipping` renders `\@ifstar` non-skipping altogether:

```
48 \newcommand*{\NoStarSkipping}{\let\@ifstar\IfStarNextToken}
```

This again seems to be so dangerous that it will not be supported much with package version v0.1 (by a package option).

On the other hand, amsmath (`amsgen.sty`) is not as scrupulous as we are and indeed redefines `\@ifstar` *altogether*, equivalent to our `\NoStarSkipping`, except that the latter provides a method to restore. I.e., as soon as you have loaded `amsgen.sty` (invoked by any amsmath package), you have decided that a star appearing after whitespace is printed as a star, rather than choosing the "starred" version of the respective command. What we actually find in `amsgen.sty` is

```
\def\@ifstar#1#2{\new@ifnextchar *{\def\reserved@a*{#1}\reserved@a}{#2}}
```

The `\reserved@a` trick seems to be due to amsmath's idea of implementing the conditional (see the code we are quoting in Section 5.1).

5.5 More General Patching with `\@sptoken`

This section deals with modifying macros by a more general technique than employed in Section 5.2. We do not use any knowledge of internals of the target user command (a "control symbol" like `\\` or an "active character"), and the command may take arguments, as the active double quote does with german.sty or babel.

5.5.1 `\IfNextSpace`

`\IfNextSpace{`⟨*if*⟩`}{`⟨*else*⟩`}` is an auxiliary macro that executes ⟨*if*⟩ if the next token is a space token (LaTeX's `\@sptoken`), otherwise it executes ⟨*else*⟩:

```
49 \newcommand*{\IfNextSpace}{\IfNextToken\@sptoken}
```

This did not work with the v0.2 version of `\IfNextToken` that didn't have '`= `', due to `\@sptoken` being an "implicit space token," as Heiko Oberdiek pointed out on texhax. He also provided the remedy that actually was amsmath's way ...

v0.2 was:

```
\newcommand*{\IfNextSpace}[2]{%
    \def\nextok@if{#1}\def\nextok@else{#2}%
    \futurelet\@let@token\nextok@ifspace}
\newcommand*{\nextok@ifspace}{%
    \ifx\@let@token\@sptoken \expandafter \nextok@if
    \else                    \expandafter \nextok@else
    \fi}
```

... not so bad from an efficiency point of view, but ...—

5.5.2 \MakeNotSkipping

`\MakeNotSkipping{`⟨*target*⟩`}{`⟨*on-space*⟩`}` should modify ⟨*target*⟩ so that it acts in its usual way when no space token is ahead while executing ⟨*on-space*⟩ otherwise. E.g., ⟨*target*⟩ may be the active double quote `"` from babel, and on the left of a space token you want that the double quote just prints an ordinary double quote from the ligature ‘’’ (the first pair of argument braces may be omitted):

```
\MakeNotSkipping{"}{''}
```

. . . while I don't really recommend this right now (v0.2f.).

⟨*target*⟩, being on document level and probably appearing in moving arguments, must be robust, while `\IfNextSpace` is not. When ⟨*target*⟩ has been defined using `\DeclareRobustCommand` (`\\` from the `document` environment is an example), we would loose the original behavior of ⟨*target*⟩ if we used `\DeclareRobustCommand` ourselves.

v0.21 was horribly flawed at this point; I had not tested all cases, I had not studied how LaTeX's `\DeclareRobustCommand` handles control symbols (such as `\_`, see `source.pdf`), and my implementation did not obey the warning in my ancompanying documentation . . .

When ⟨*target*⟩ is a single (active) character, it may have been robustified by making it expand to a robust control sequence token, such as `~` via `\nobreakspace{}`—we don't know, or don't try to find out now. We make it robust, accepting that this may just introduce an unnecessary extra macro.

```
\newcommand*{\INT@modified}[1]{%
    \ifx\protect\@typeset@protect
        \expandafter\expandafter\expandafter \IfNextSpace
            \csname\INT@mod#1\expandafter\endcsname
    \else
        \protect#1%
    \fi}
```

`\INT@mod`⟨*cs*⟩ is another *“name modifier”:*

```
\newcommand*{\INT@mod}{INT@mod.\string}
```

Here is the main command of the section. `\@tempa` will store the meaning of the command that ⟨*target*⟩ would call in typesetting mode after `\DeclareRobustCommand`⟨*target*⟩. `\@tempb` will store what ⟨*target*⟩ does in in typesetting mode, maybe that is just the meaning of ⟨*target*⟩ (kind of flag, like `\if@tempswa`):

```
\newcommand*{\MakeNotSkipping}[2]{%
    \expandafter \let \expandafter \@tempa
        \csname\expandafter\@gobble\string#1 \endcsname
    \let\@tempb#1%
```

When `#1` is a control word, and its name, extended by a space, is the name of a defined token, we *lazily* (like makerobust, TODO) assume that its current

meaning was assigned by `\DeclareRobustCommand`. One exception: if that token is the control space `\␣`, `#1` is a single character (hopefully active, TODO check!?).

```
62      \ifx\@tempa\relax \else \ifx\@tempa\ \else
63          \let\@tempb\@tempa \fi \fi
64      \expandafter\let\csname\INT@save#1\endcsname\@tempb
```

We have analyzed `#1` and now may modify it:

```
65      \def#1{\INT@modified#1}%
```

We do not know beforehand what ⟨*on-space*⟩ will contain, in any case it should *not* be expanded right here, that's why we use the token register `\@toks`:

```
66      \toks@{#2}%
67      \expandafter\edef\csname\INT@mod#1\endcsname{%
68          {\the\toks@}%
69          \expandafter\noexpand\csname\INT@save#1\endcsname}%
70  }
```

This still is experimental, and you must care not to apply the patch two times when it has not been undone in between. The main application may be a macro like `\\` that some (non-standard) environment defines; then you could redefine the environmont so that its start finally modifies that macro according to your wishes. In the latter situation, the end of the environment will undo your `\MakeNotSkipping`;

TODO: In the case of `"`, this might be a starting point for handling conventions about moving an ensuing punctuation mark to the left of the quotation mark. Moreover, getting something really useful would require dealing with `"` at the left of a bracket too.

6 “Manual” Line Breaks

6.1 Outline of Implementation

In the first instance, the present package aims at rendering `\\` a command that interpretes a left-hand square bracket as a start of an optional argument only if the bracket is not preceded by any other token (apart from the star in `\\*`), especially not by a space token.

Indeed, an author may expect that when a bracket opens in a *different* line than the `\\`, then it will be *printed* rather than interpreted as an *optional-argument delimiter* (the package author has been such an author some times). Now, when the bracket only is in a line *following* the line carrying the `\\`, the end-line character normally produces a space token (TEXbook p. 47), so the present idea of implementation will cover the case of a bracket in the next line.

In `latex.ltx`, the names of the commands implementing the line break have some “pivot” part ⟨*pivot*⟩ that we can use to patch them in a uniform way. They

are two in each case: The first starts with `\@`⟨*pivot*⟩ and invokes `\@ifstar`, the second starts with `\@x`⟨*pivot*⟩ and invokes the left-hand-bracket test. Both of them need to be patched.

6.2 "Normal" Manual Line Breaks

If I had been aware of the difficulties of this part, I probably would not have started writing this package, hoping it would be the work of about an hour.

`\@xnewline` must be patched in order to get a non-skipping version of the bracket test, and this patch suffices for the optional-argument goal.

The `\@ifstar` call is in `\@normalcr` and its alias `\\␣`; the latter is invoked by `\\` according to `\DeclareRobustCommand\\`.

Things seem to be easier when `\\␣` *expands* to `\@normalcr` instead of being an *alias* of it (**CAUTION!**). Then we just need to control `\@normalcr`:

```
\@namedef{\@backslashchar\space}{\@normalcr}
```

`\StoreNewlineSkipping` *stores* the skipping behavior of `\\` outside special environments:

```
\newcommand*{\StoreNewlineSkipping}{%
    \INTstore\@normalcr \INTstore\@xnewline}
```

`\RestoreNewlineSkipping` *re*stores the skipping behavior of `\\` outside special environments:

```
\newcommand*{\RestoreNewlineSkipping}{%
    \INTrestore\@normalcr \INTrestore\@xnewline}
```

`\NoNewlineSkipping` *suppresses* skipping blank spaces with `\\` outside special environments:

```
\newcommand*{\NoNewlineSkipping}{%
    \StarTestPatch\@normalcr \NextTestPatch\@xnewline}
```

6.3 Manual Line Breaks in LaTeX Environments

The macros in the present section should modify LaTeX's `\\` in environments (⟨*env*⟩ being one of:) `center`, `tab`, `array`, and `tabular`. These *environment names* are the expected *arguments* of those macros. However, argument `center` also affects the `flushleft`, `flushright`, and `verse` environments,[3] and `array` and `tabular` should also affect their enhanced variants from other LaTeX packages. When this internal structure of LaTeX changes, the present section may become obsolete ...

`\INTactOnEnv{`⟨*action1*⟩`}{`⟨*action2*⟩`}{`⟨*env*⟩`}` is the backbone of these macros. ⟨*action1*⟩ and ⟨*action2*⟩ are one of

`\INTstore`, `\INTrestore`, `\StarTestPatch`, `\NextTestPatch`.

⟨*action1*⟩ deals with `\@ifstar`, ⟨*action2*⟩ deals with `\@ifnextchar`:

[3] `verse` is provided by LaTeX's standard classes only, while `flushleft` and `flushright` belong to the LaTeX kernel.

```
78  \newcommand*{\INTactOnEnv}[3]{%
79      \expandafter#1\csname  @#3cr\endcsname
80      \expandafter#2\csname @x#3cr\endcsname}
```

`\StoreSkippingCRs{`⟨*env*⟩`}` *stores* the skipping behavior of `\\` in environments ⟨*env*⟩:

```
81  \newcommand*{\StoreSkippingCRs}{%
82      \INTactOnEnv\INTstore\INTstore}
```

`\RestoreSkippingCRs{`⟨*env*⟩`}` *re*stores the skipping behavior of `\\` in environments ⟨*env*⟩:

```
83  \newcommand*{\RestoreSkippingCRs}{%
84      \INTactOnEnv\INTrestore\INTrestore}
```

`\NotSkippingCRs{`⟨*env*⟩`}` *suppresses* space skipping of `\\` in environments ⟨*env*⟩:

```
85  \newcommand*{\NotSkippingCRs}{%
86      \INTactOnEnv\StarTestPatch\NextTestPatch}
```

Do these commands work?
[Or do they not?]

By contrast, the environments `quotation` and `quote` from LaTeX's standard classes use the "normal" newline command essentially provided by `\@normalcr`.

6.4 amsmath and mathtools

Just discussing related functionality in the amsmath and mathtools packages, without any own code:

amsmath modifies the star (`*`) test all over the document (see our Section 5.4), while providing own (not skipping) versions of `\\` rather in math displays and math environments only. This applies quite obviously to the `{cases}` and `{matrix}` environments. I am not sure about amsmath's use of `\displaybreak`, (cf. `amstex.sty` and the `\intertext` command) and `\math@cr`.

mathtools modifies amsmath's line breaking behavior in turn on its options `[allowspaces]` and `[disallowspaces]`, referring to some strange behavior of amsmath. Still I don't understand what is going on entirely, and my impression is that nobody else has understood these things entirely so far. mathtools is not the first package suppressing space skipping with `\\`, amsmath has done this already; the question is where, where not, and why ... mathtools's `[disallowspaces]` just seems to provide a more straightforward policy ... TODO

7 Package Options

7.1 Behavior *without* Options

If the package is called without any option, it only defines `\IfNextToken`, `\IfStarNextToken` and the other package-writer or user commands, without actually changing behavior of any LaTeX command.

7.2 Option `newline`

Package option `newline` stores and disables space skipping for `\\` in "normal" mode according to Section 6.2:

```
\DeclareOption{newline}{\StoreNewlineSkipping\NoNewlineSkipping}
```

7.3 Environments

The next package options are just the environment names according to Section 6.3 (`center`, `tab`, `array`, `tabular`). Option ⟨*env*⟩ stores and disables the skipping behavior of `\\` in ⟨*env*⟩ environments. We abuse the our temporary macro `\nextok@match` from Section 5.1:

```
\def\nextok@match#1{%
    \DeclareOption{#1}{\StoreSkippingCRs{#1}\NotSkippingCRs{#1}}}
\nextok@match{center}
\nextok@match{tab}
\nextok@match{array}
\nextok@match{tabular}
```

7.4 "All Options" or "Standard Options"

Package Options `all` and (v0.11:) `stdbreaks` have the same effect as using the `newline` option and the environment package options `center`, `tab`, `array`, and `tabular` at once.

```
\def\nextok@match#1{\csname ds@#1\endcsname}
```

(... must not be changed before `\ProcessOptions` ...)

```
\DeclareOption{all}{%
    \nextok@match{newline} \nextok@match{center}
    \nextok@match{tab} \nextok@match{array} \nextok@match{tabular}}
```

Behavior of option `all` may *change* in the future of the package, while option `stdbreaks` should rather *keep* its present behavior.

```
\DeclareOption{stdbreaks}{\nextok@match{all}}   %% v0.11
```

8 Processing Options and Leaving the Package

```
\ProcessOptions
\endinput
```

9 Acknowledgments

While I experienced the problem myself some years ago with a critical edition, I finally decided to do this work after postings by Susan Dittmar (March 2011) and Philipp Stephani (December 2010) on the 'texhax' mailing list. The latter pointed to mathtools.

Moreover, the space skipping matter was discussed on the LATEX-L mailing list ('xparse and space skipping') in mid of May 2011, and the present package may be considered a contribution to that discussion (saying something like: keep the simple standard for beginners, offer something advanced for advanced users if you think some of them want it ... maybe just as contrib). Bruno Le Floch (May 11) and Frank Mittelbach (May 15) made me aware of the similar functionality in amsmath. Wordings in describing ifnextok may resemble wordings in that LATEX-L very much. Only for v0.21, I actually read these LATEX-L postings, rather than only their subject lines.

See sections 5.1 and 5.5 for Heiko Oberdiek's contribution.

10 VERSION HISTORY

```
v0.1   2011/05/23   very first
v0.11  2011/05/23   typo 'mathc' fixed, where/when
       2011/05/27   &, more structure, option [stdbreaks]
       2011/05/27   doc. mentions 'verse', 'quotation', 'quote';
                    ack.s
v0.2   2011/05/30   \IfNextSpace, \MakeNotSkipping
       2011/05/31   using \@normalcr differently;
                    corrected \IfNextSpace
v0.21  2011/06/02   reworked \IfNextToken and \IfNextSpace
                    after Heiko Oberdiek, regarding amsmath;
                    documentation discusses amsmath and mathtools
                    and refers to LATEX-L
       2011/06/03   ... \ , \rq\ ; corr. version string
v0.21a 2011/06/14   quoting \@ifstar from amsmath, extended comments
                    on amsmath and mathtools, starred versions,
                    Heiko Oberdiek for sec:main
       TO CTAN
v0.22  2011/06/24   ack's extended; reduced sec:patch; sec:genpatch:
                    horribly fragile robustification fixed
       2011/06/25   re-implementations in sec:patch using \WCS...
       2011/06/26   was named v0.3, renamed v0.22
       JUST STORED
v0.3   2011/06/26   sec:stored as v0.11;
```

```
124                     sec:genpatch code similar to v0.2;
125                     different sectioning; \pagebreak's
126          2011/06/27 \INT@modified streamlined, rm. useless \\ example
127                     and wrong description of \INT@modified
128
```