# The etextools package

**An e-TeX package providing useful tools for LaTeX Users and package Writers**

**Florent CHERVET** florent.chervet@free.fr

**Version 2 ζ**

**12 août 2009**

## Contents



**Abstract**

The etextools package is based on the etex and etoolbox packages and defines more tools for LaTeX Users or package Writers. Before using this package, it is highly recommended to read the documentation (of this package and...) of the etoolbox package.

This package requires the etex package from David Carlisle and the etoolbox package from Philipp Lehman. Both of them are available on CTAN under the /latex/contrib/ directory [1].

The main contributions of this package are :

- the \expandnext macro and the \ifempty macro
- the ability to define fully expandable macros with parameter and/or star version (with a small restriction) – see \FE@testopt, \FE@ifstar
- a Command-List Parser constructor that fully uses this new feature : command-list parser are fully expandable – see \csvloop and \listloop
- three macros that are lacking from the etoolbox package for removing elements from lists : \listdel and \listgdel, \listedel, \listxdel
- macros that return the index of an item in a list: \getlistindex and \xgetlistindex and conversely the item at a given position in a list: \getlistitem

[1] This documentation is produced with the ltxdockit classe and package by Philipp Lehman using the DocStrip utility.
⟶ To get the documentation, run (twice): pdflatex etextools.dtx
⟶ To get the package, run: etex etextools.dtx

# 1 Introduction

## 1.1 Motivation

The first motivation for this package was to define a list-parser macro:

- just like the \docsvlist and \dolistloop macros from etoolbox
- fully expandable (i. e., in an \edef)
- that take the auxiliary command(s) (\do in etextools) as an optional argument.

As a result, a method for defining fully expandable macros with optional parameter is built here.

## 1.2 Purely Expandable macros

The fully expandable (or purely expandable) commands defined in this package can be easily spotted with the special marker displayed here in the margin for information.

Go directly to the implementation of a command by clicking on the symbol displayed here in the margin for information.

Now from the implementation, you can return to the description section just by clicking on the symbol displayed here in the margin for information.

## 1.3 The example file

The example file provided with etextools illustrates the macros defined here.

## 1.4 Requirements

This package requires the packages etex package by David Carlisle and etoolbox by Philipp Lehman.

## 1.5 Acknowledgements – Thank You !

Thanks to Philipp Lehman for the etoolbox package (and also for this nice class of documentation). Much of my work on lists are based on his work and package.

# All User Commands

## 2 Expansion control

\expandnext{⟨*cstoken*⟩}{⟨*first parameter of cstoken*⟩}

We often want a control sequence to be expanded after its first argument. For example, if you want to test if a command \foo has a blank replacement text you will say:

\expandafter\ifblank\expandafter{\foo}{⟨*true part*⟩}{⟨*false part*⟩}

With \expandnext you'll just have to say:

\expandnext\ifblank{\foo}{⟨*true part*⟩}{⟨*false part*⟩}

Now observe the following game :

```
\def\foo{foo}      ⟶    \def\Foo{\foo}   ↩
\def\\FOo{\Foo}    ⟶    \def\FOO{\FOo}   ↩
\def\fool{\FOO}
```

Guess how many \expandafter are needed to test "\ifblank{foo}" directly from \fool ???

\expandnext solves this problem : \fool has 5 degrees of expansion until it expands to "foo", therefore exactly 5 \expandnext are required. The solution is:

\expandnext\expandnext\expandnext\expandnext\expandnext\ifblank{\fool}

Other example: suppose you want to define a macro \detokenizecs{⟨*csname*⟩} that expands to the detokenized content of \csname ⟨*csname*⟩\endcsname. Without \expandnext you will have to say:

\expandafter\expandafter\expandafter\detokenize
\expandafter\expandafter\expandafter{\csname ⟨*csname*⟩\endcsname}

**six** \expandafter(s). With \expandnext you just have to say:
\expandnext\expandnext\detokenize{\csname #1\endcsname}

\noexpandcs{⟨*csname*⟩}

In an expansion context (\edef) we often want a control sequence whose name results from the expansion of some macros and/or other tokens to be created, but not expanded at that point. Roughly:

\edef{\noexpandcs{<balanced text to be expanded as a cs-name>}}

will expand to: \"cs-name" but this (new) control sequence itself will not be expanded. A typical use is shown in the following code:

⟶ \edef\abc{\noexpandcs{abc@\@gobblescape\controlword}}
⟶ if equivalent to: \def\abc{\abc@controlword}.

\noexpandcs may be abbreviated f.ex. in \‘#1‘ in \edef that take place in a group.

\noexpandafter

\noexpandafter only means \noexpand\expandafter and is shorter to type.

This command is used in the definition of \DeclareCmdListParser.

## 3 Characters and Strings

\ifempty{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

\ifempty is similar to \ifblank but it test if a string is really empty (it shall not contain any character nor spaces). To test if the replacement text of a macro is empty, one may use \ifempty in conjunction with \expandnext:

\expandnext\ifempty{\macro} ⟨*true*⟩⟨*false*⟩

\deblank{⟨*string*⟩}

\deblank removes all leading and trailing blank spaces from its argument.

\xifblank{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

\xifblank is similar to \ifblank except that the ⟨*string*⟩ is first expanded with \protected@edef.

\xifstrequal{⟨*string1*⟩}{⟨*string2*⟩}{⟨*true*⟩}{⟨*false*⟩}

\xifstrequal is similar to \ifstrequal but each ⟨*string*⟩ argument is expanded with \protected@edef before comparison.

\iffirstchar{⟨*string1*⟩}{⟨*string2*⟩}{⟨*true*⟩}{⟨*false*⟩}

\iffirstchar{⟨⟩}{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

\iffirstchar compares the character codes of the **first** characters of each ⟨*string*⟩. The comparison is *catcode agnostic* and the macro is fully expandable. Neither ⟨*string1*⟩ nor ⟨*string2*⟩ is expanded before comparison. Example:

```
\iffirstchar *{*hello*}{begins with a star}{begins with something else}
```

There is a "special" use of this command when one want to test if a ⟨*string*⟩ begins with an *escape* character (\) one may say:

\iffirstchar \@backslashchar{⟨*string*⟩}  or even easier:
\iffirstchar {}{⟨*string*⟩}

Please note that the tests:

\iffirstchar {⟨*whatever string1 is*⟩}{}
and: \iffirstchar {}{}

are **always expanded into** ⟨*false*⟩ (for consistency with the shortcut-test for \@backslashchar).

\ifsinglechar{⟨*string1*⟩}{⟨*string2*⟩}{⟨*true*⟩}{⟨*false*⟩}

\ifsinglechar performs the test \iffirstchar but only if ⟨*string2*⟩ has one single character.

Now with the macro \ifsinglechar it becomes possible to write fully expandable macros with an option, **provided that this macro has at least one non-optional argument**, as far as we don't use \futurelet nor any assignation. The "trick" is the following:

```
\def\MacroWithOption#1{\ifsinglechar[{#1}
                                    {\MacroHasOption[}
                                    {\MacroNoOption{#1}}}}
\def\MacroHasOption[#1]#2...    definition
\def\MacroNoOption#1...         definition
```

Moreover (in the style of \@testopt):

```
\def\MacroWithOption#1{\ifsinglechar[{#1}
                                    {\MacroHasOption#1}
                                    {\MacroHasOption[default]{#1}}}
```

# 4 Fully expandable macros with options

\FE@testopt{⟨*argument to be tested*⟩}{⟨*commands*⟩}{⟨*default option*⟩}

\FE@testopt mimics the behaviour of \@testopt but is Fully Expandable (FE) and can be used as follow:

```
\def\MacroWithOption#1{\FE@testopt{#1}\MacroHasOption{default}}
```

Note that \MacroWithOption **must have at least one mandatory argument.**

\FE@testopt is used in the definition of \DeclareCmdListParser.

\FE@ifstar{⟨*argument to be tested*⟩}{⟨*star-commands*⟩}{⟨*non-star commands*⟩}

Similarly, it becomes possible to mimic the behaviour of \@ifstar but in a fully expandable(FE) way. \FE@ifstar can be used as follow :

```
\def\StarOrNotCommand#1{\FE@ifstar{#1}
                                  {\StarredCommand}
                                  {\NotStarredCommand}}
```

Remember that \StarredCommand and \NotStarredCommand **must have at least one mandatory argument.**

\FE@ifstar is used in the definitions of \DeclareCmdListParser, \csvloop, \listloop and \csvtolist.

# 5 Lists management

## 5.1 The Command-List Parser

The etoolbox package provides a way to define list parsers a fully expandable macros: the list parser is able to expand the auxiliary command \do on each item of a list.

Here we provide a \DeclareCmdListParser macro that is compatible and slightly different, because the auxiliary command is not necessarily \do. Such a command-list-parser is fully expandable.

The idea is that if \csvloop has been defined as a command-parser then, thank to the fully expandable macro \FE@testopt we can call for expansion:

\csvloop*{item,item,...} as a shortcut for \csvloop*[\do]{item,...}
or: \csvloop*[\listadd\mylist]{item,item,...}

for example to convert the csv-list into internal etoolbox list.

The star-form of \csvloop will be explained below.

\DeclareCmdListParser{⟨*command*⟩}{⟨*separator*⟩}

\DeclareCmdListParser acts in the same way as etoolbox-\DeclareListParser and the command-list-parser are sensitive to the category code of the ⟨*separator*⟩ *(which-is-not-necessarily-a-single-character-and-shall-not-be-a-&-with-a-catcode-of-3)*.

A Command-List-Parser is then defined as a purely expandable macro. It has a *star form* for the case of the list given is already expanded, and a *normal form* if the list is given as a macro name.

You may then use the following syntaxes:

```
\CmdListParser\mylist
\CmdListParser*{item⟨sep⟩item⟨sep⟩item}
\CmdListParser[\Commands]\mylist
```
or: `\CmdListParser*[\Commands]{item⟨sep⟩item⟨sep⟩item}`

## 5.2 Loops into lists

Now we have to declare two command-list-parsers : `\listloop` for `etoolbox` lists and `\csvloop` for comma-separated lists.

`\csvloop[`⟨*auxiliary commands*⟩`]{`⟨*csvlistmacro*⟩`}`

`\csvloop*[`⟨*auxiliary commands*⟩`]{`⟨*item,item,item,...*⟩`}`

`\listloop[`⟨*auxiliary commands*⟩`]{`⟨*listmacro*⟩`}`

`\listloop*[`⟨*auxiliary commands*⟩`]{`⟨*expanded list*⟩`}`

`\listloop` acts exactly as `etoolbox-\dolistloop` with an optional argument to change the default auxiliary command `\do` to apply to each item of the list :

`\listloop\mylist` *is* `\listloop[\do]\mylist` *and is also* `\dolistloop\mylist`
`\csvloop\csvlist` *is* `\csvloop[\do]\csvlist` *and is also*: ↩
`\expandafter\docsvlist\expandafter{\csvlist}`

## 5.3 Converting csv lists to `etoolbox`-lists

`\csvtolist{`⟨*listmacro*⟩`}{`⟨*csvlistmacro*⟩`}`

`\csvtolist*{`⟨*listmacro*⟩`}{`⟨*item,item,item...*⟩`}`

`\csvtolist` converts a comma separated list into an internal `etoolbox` list. It is useful to insert more than one item at a time in a list. Those macros are not fully expandable because of the redefinition of ⟨*listmacro*⟩ done by `\listadd`...

It's also the first application of the `\csvloop` macro just defined.

`\csvtolistadd{`⟨*listmacro*⟩`}{`⟨*csvlistmacro*⟩`}`

`\csvtolistadd*{`⟨*listmacro*⟩`}{`⟨*item,item,item...*⟩`}`

`\csvtolistadd` acts similarly but does not reset the ⟨*listmacro*⟩ to `\@empty`:

## 5.4 Removing elements from `etoolbox`-lists

The `etoolbox` package provides `\listadd`, `\listgadd` and `\listxadd` commands to add items to a list. However, no command is designed to remove an element from a list.

`\listdel{`⟨*listmacro*⟩`}{`⟨*item*⟩`}`

`\listgdel{`⟨*listmacro*⟩`}{`⟨*item*⟩`}`

`\listedel{`⟨*listmacro*⟩`}{`⟨*item*⟩`}`

`\listxdel{`⟨*listmacro*⟩`}{`⟨*item*⟩`}`

The `\listdel` command removes the element ⟨*item*⟩ from the list ⟨*listmacro*⟩. Note that the ⟨*listmacro*⟩ is redefined after deletion:

Commands \listgdel and \listxdel are similar, except that the assignment (i. e., the redefinition of the list) is global; for the latter, the ⟨*item*⟩ is first expanded using \protected@edef:

### 5.5 Index of an element in a list

\getlistindex{⟨*item*⟩}{⟨*listmacro*⟩}
\xgetlistindex{⟨*item*⟩}{⟨*listmacro*⟩}
\getlistindex*{⟨*item*⟩}{⟨*list*⟩}
\xgetlistindex*{⟨*item*⟩}{⟨*list*⟩}

Sometimes it is interesting to know at which offset in a list lies a given item. \getlistindex answers to this question. \xgetlistindex does the same thing but expand the ⟨*item*⟩ (using \protected@edef) prior looking for it in the list. The result (i. e., the position of ⟨*item*⟩ in ⟨*list*⟩) is stored in \indexinlist.

- If ⟨*item*⟩ is not found in the ⟨*list*⟩ then \indexinlist is 0
- If ⟨*item*⟩ is found in first position then \indexinlit is 1 and so on.

As for the command-list-parser, the star versions are designed in case the list (in the second argument) is already expanded.

### 5.6 Extract by index from a list

\getlistitem{⟨*listmacro*⟩}{⟨*index [numexpr expression]*⟩}
\getlistitem*{⟨*list*⟩}{⟨*index [numexpr expression]*⟩}

Now the reverse operation of "getting the index" is "getting the element" whose index is known. Note that this macro is fully expandable for we don't have to compare the items (just count the loop using \numexpr – no counter). Therefore we use the fully expandable version of \ifstar : \FE@ifstar to keep the "fully-expand-ability" of the star-form as well:

- If the ⟨*index*⟩ is in the range $[1 \dots n]$ then the macro expands to $\langle item \rangle_n$
- If the ⟨*index*⟩ is non positive or $> n$ then the macro expands to nothing ({}).

# LATEX code

## 6 Implementation

### 6.1 Package identification

```
1 ⟨*package⟩
2 \NeedsTeXFormat{LaTeX2e}[1996/12/01]
3 \ProvidesPackage{etextools}
4    [2009/07/14 v2e e-TeX more useful tools for LaTeX package writers]
5 \csname ettl@onlyonce\endcsname\let\ettl@onlyonce\endinput
```

### 6.2 Requirements

This package requires the packages etex package by David Carlisle and etoolbox by Philipp Lehman.

```
\RequirePackage{etex,etoolbox}
```

Furthermore, the space token must have its natural catcode (10) all along this package.

```
\edef\ettl@restore@space@catcode{\catcode`\ =\the\catcode`\ }
\AtEndOfPackage{\ettl@restore@space@catcode
   \let\ettl@restore@space@catcode\ettl@undefined}
\catcode`\ =10
```

## 6.3 A few (7) "helper" macros

```
\long\def\ettl@afterelse#1\else#2\fi{\fi#1}
\long\def\ettl@afterfi#1\fi{\fi#1}
\long\def\ettl@afterfifi#1\fi\fi{\fi\fi#1}
\long\def\ettl@afterelsefi#1\else#2\fi\fi{\fi#1}
```

`\ettl@cdr` `\ettl@cdr` is only used in `\ifsinglechar`

```
\begingroup\catcode`\|=4
\gdef\ettl@cdr#1#2|{#2}
\endgroup
```

`\@gobblescape` This sequence of command is very often used, even in `latex.ltx`. So it appears to be better to put it in a macro. It's aim is to reverse the mechanism of `\csname...\endcsname`:

```
\newcommand*\@gobblescape{\expandafter\@gobble\string}
```

May be we could do better, testing first if the next token is a control sequence...

`\str@gobblescape` With ε-TEX `\detokenize` it is now possible to invoke a `\string` on an arbitrary sequence of tokens. Unfortunately, `\detokenize` adds a space tokens at the very end of the sequence. `\str@gobblescape` acts exactly the same way as `\@gobblescape` i. e., it removes the first character from a string, but removes the trailing spaces as well:

```
\newcommand\str@gobblescape[1]{\expandafter\deblank\expandafter{%
          \expandafter\@gobble\detokenize{#1}}}
```

## 6.4 Expansion control

`\noexpandcs` `\noexpandcs` may be abbreviated f.ex. in `\`#1` or `\"#1"` in `\edef` that take place in a group.

```
\newcommand*\noexpandcs[1]{\expandafter\noexpand\csname #1\endcsname}
```

`\noexpandafter` `\noexpandafter` only means `\noexpand\expandafter` and is shorter to type.

```
\newcommand\noexpandafter{\noexpand\expandafter}
```

`\expandnext` This code is not properly tricky but if you're eager to understand the job of each expandafter, it's best to go straight at the log.

Note that the first argument of `\expandnext` must be a single ⟨*cstoken*⟩ (for `\expandafter` acts only on the first following token).

```
\newcommand\expandnext[2]{%
   \ifx#1\expandnext
      \ettl@afterelse\expandafter\expandafter\expandafter
            \expandafter\@expandnext{#2}{\expandafter\expandafter\expandafter}%
   \else\ettl@afterfi\expandafter#1\expandafter{#2}%
   \fi}
\long\def\@expandnext#1#2#3{%
   \ifx#1\expandnext
```

```
      \expandafter\ettl@afterelse\expandafter\expandafter\expandafter
            \expandafter\@expandnext{#3}{\expandafter#2#2}%
   \else
      \expandafter\ettl@afterfi#2#1#2{#3}%
   \fi}
```

## 6.5 String manipulation

\deblank Note that the leading spaces are automatically removed by TEX's mastication mechanism.

```
\newcommand\deblank{}
\newcommand\ifempty{}
\begingroup\catcode`\|=4% a | as a tab array
\long\gdef\deblank#1{\@deblank#1 |}
\long\gdef\@deblank#1 #2|{\ifblank{#2}{#1}{#1\@deblank#2 |}}
```

\ifempty \ifempty is amazing because the code is so concise!

```
\long\gdef\ifempty#1{\csname @\ifx|#1|first\else second\fi oftwo\endcsname}
\endgroup% catcode group
```

## 6.6 Expanded string comparison

\xifblank Just expands the parameter using \protected@edef before testing for \ifblank:

```
\newrobustcmd\xifblank[1]{\begingroup
   \protected@edef\@xifblank{\endgroup
         \noexpand\ifblank{#1}%
   }\@xifblank}
```

\xifstrequal Same idea: protected expansion before testing:

```
\newrobustcmd\xifstrequal[2]{%
   \begingroup\protected@edef\@tempa{#1}\protected@edef\@tempb{#2}%
      \ifx\@tempa\@tempb   \aftergroup\@firstoftwo
      \else                \aftergroup\@secondoftwo
      \fi\endgroup}
```

\iffirstchar \iffirstchar test if #1 and #2 begins with the same character.

```
\newcommand\iffirstchar[2]{%
   \if \expandafter\@car\string#2\relax\@nil\expandafter\@car#1\string\\\@nil
         \ettl@afterelse\ifblank{#2}\@secondoftwo\@firstoftwo
   \else \expandafter\@secondoftwo
   \fi}
```

\ifsinglechar Test if #2 is a single character equal to #1:

```
\newcommand\ifsinglechar{}
\begingroup\catcode`\|=4
\long\gdef\ifsinglechar#1#2{%
```

Remember that neither ⟨*string1*⟩ nor ⟨*string2*⟩ is expanded.
First: test if #2 is a single character:

```
   \csname @\expandafter\expandafter\expandafter\ifx
            \expandafter\expandafter\expandafter|%
               \expandafter\ettl@cdr\string#2||first\else second\fi
   oftwo\endcsname
```

The test returned ⟨*true*⟩: therefore test further the character codes of #2 and #1, and switch to ⟨*true*⟩ only in case of equality:

```
  {\iffirstchar{#1}{#2}}
```

Otherwise, switch to ⟨*false*⟩

```
    \@secondoftwo}
\endgroup% catcode group
```

`\FE@testopt` Fully expandable `\@testopt`-like test:

```
\newcommand\FE@testopt[3]{\ifsinglechar [{#1}{#2#1}{#2[{#3}]{#1}}}
```

`\FE@ifstar` Fully expandable `\@ifstar`-like test:

```
\newcommand\FE@ifstar[3]{\ifsinglechar *{#1}{#2}{#3{#1}}}
```

`\DeclareCmdListParser` `\DeclareCmdListParser` acts in the same way as `etoolbox-\DeclareListParser` and the command-list-parser are sensitive to the category code of the ⟨*separator*⟩

The command-list-parser will be defined only if it is definable:

```
\newrobustcmd*\DeclareCmdListParser[2]{%
  \@ifdefinable#1
    {\expandafter\etextools@defcmdparser\expandafter{#1}{#2}}}
```

Then the job is done by `\etextools@defcmdparser`: we need the '&' to have a catcode of 3 and globally define the macro inside a purposeful group:

```
\begingroup\catcode`\&=3\catcode`\`=3
\gdef\etextools@defcmdparser#1#2{%
   \begingroup
   \def\`##1`{\expandafter\noexpand\csname ##1\endcsname}%
```

Now the parser definition is made inside an protected-`\edef` in order to expand control sequences names:

```
   \protected@edef\defineparser{\endgroup
```

Here we define the entry-point: we first test if the command was starred. This is possible because the list-parser has always a mandatory argument (the ⟨*list*⟩ itself or the ⟨*listmacro*⟩) :

```
   \long\def#1####1{\noexpand\FE@ifstar{####1}
      {\`ettl@lst@star\string#1`}
      {\`ettl@lst@nost\string#1`}}%
```

Both starred and not-starred versions have an optional argument which is the auxiliary command, whose name is `\do` if not specified. It is possible to test the option for the same reason:

```
   \long\csdef{ettl@lst@star\string#1}####1{\noexpand\FE@testopt{####1}
      {\`ettl@lst@star@pt\string#1`}{\noexpand\do}}%
   \long\csdef{ettl@lst@nost\string#1}####1{\noexpand\FE@testopt{####1}
      {\`ettl@lst@nost@pt\string#1`}{\noexpand\do}}%
```

Definition of the parser with its arguments : `[optional command]{list or listmacro}`. If the starred version was used, then we do not have to expand the (mandatory) ⟨*list*⟩:

```
   \long\csdef{ettl@lst@star@pt\string#1}[####1]####2{%
      \`ettl@lst\string#1`{####2}{####1}}%
```

On the other hand, if the parser was not starred, the (mandatory) ⟨*listmacro*⟩ is expanded once:

```
   \long\csdef{ettl@lst@nost@pt\string#1}[####1]####2{%
      \noexpandafter\`ettl@lst\string#1`\noexpandafter{####2}{####1}}%
```

ListParser is defined and leads to `\`ettl@lst\string\ListParser`` in all cases; here the ⟨*list*⟩ is in first position, the auxiliary commands come after, so we reverse the order and add a ⟨*separator*⟩ in case the ⟨*list*⟩ is empty:

```
   \long\csdef{ettl@lst\string#1}####1####2{%
         \`ettl@lst@\string#1`{####2}####1\unexpanded{#2}&}%
```

In the following macro, we extract the first item from the list (before the ⟨*separator*⟩ #2), for treatment:

```
   \long\csdef{ettl@lst@\string#1}####1####2\unexpanded{#2}####3&{%
```

Proceed with the first item, if not empty:

```
      \noexpand\ifblank{####2}
            {}
            {\noexpand\ettl@lst@doitem{####1}{####2}}%
```

If the remainder of the list is empty then break loop, otherwise restart extracting the next, first item for treatment:

```
      \noexpand\ifblank{####3}
            {\noexpand\ettl@listbreak}
            {\‘ettl@lst@\string#1‘{####1}####3}&}%
```

Now the definitions are ready, execute them:

```
   }\defineparser}%
```

`\ettl@lst@doitem` apply the auxiliary command(s) (in `#1`) to the item (`#2`): `\ettl@listbreak` breaks the loop, removing the extra `&`:

```
   \long\gdef\ettl@lst@doitem#1#2{#1{#2}}%
   \long\gdef\ettl@listbreak#1&{}%
```

Leave “catcode-group”:

```
\endgroup
```

`\csvloop` Definition of `\csvloop`:

```
\DeclareCmdListParser\csvloop{,}
```

Definition of `\listloop` (with a ‘|’ of catcode 3 – cf.`etoolbox`):

```
\begingroup\catcode‘\|=3
   \def\do{\DeclareCmdListParser\listloop{|}}%
\expandafter\endgroup\do
```

`\csvtolist` This is the first application of `\csvloop`:

```
\newcommand\csvtolist{\@ifstar\star@csvtolist\nost@csvtolist}%
\def\star@csvtolist#1{\let#1\@empty\csvloop*[{\unexpanded{\listadd#1}}]}%
\def\nost@csvtolist#1{\let#1\@empty\csvloop[{\unexpanded{\listadd#1}}]}%
```

`\csvtolistadd` It’s quite the same as the latter except it does not reset the list to empty at the very beginning:

```
\newcommand\csvtolistadd{\@ifstar\star@csvtolistadd\nost@csvtolistadd}%
\def\star@csvtolistadd#1#2{\ifblank{#2}
         {\let#1\@empty}{\csvloop*[{\unexpanded{\listadd#1}}]{#2}}}%
\def\nost@csvtolistadd#1#2{\ifblank{#2}
         {\let#1\@empty}{\csvloop[{\unexpanded{\listadd#1}}]{#2}}}%
```

`\listdel` `\listdel` removes an item from a list:

```
\newrobustcmd\listdel[2]{\@listdel\def{#1}{#2}}%

\newrobustcmd\listgdel[2]{\@listdel\gdef{#1}{#2}}%
\newrobustcmd\listxdel[2]{\begingroup
   \protected@edef\@listxdel{\endgroup
      \unexpanded{\@listdel\gdef{#1}}{#2}%
   }\@listxdel}%
\newrobustcmd\listedel[2]{\begingroup
   \protected@edef\@listedel{\endgroup
      \unexpanded{\@listdel\def{#1}}{#2}%
   }\@listedel}
```

Now you noticed the job is delayed to a general macro `\@listdel` which is relatively tricky ! We need first to be placed in an environment where the ‘|’ delimiter has a category code of 3 (cf `etoolbox`-lists definitions):

```
\begingroup\catcode‘\|=3\catcode‘\&=3
```

Inside this “catcode-group” the definition of \@listdel ought to be global:

```
\long\gdef\@listdel#1#2#3{%
```

#1=\def or \gdef, #2=⟨*listmacro*⟩, #3=⟨*item*⟩ to remove.

In order to preserve the hash-table from temporary definitions, a group is opened:

```
\begingroup
      \def\@tempa##1|#3|##2&{##1|##2\@tempb}%
```

\@tempa is a delimited macro whose aim is to remove the first ⟨*item*⟩ found in the list and it adds \@tempb after the result. If the ⟨*item*⟩ was not in the list, then ##2 will be empty. Note that \@tempa is the only macro whose definition depends on the ⟨*item*⟩ (and then leads \@listdel not to be fully-expandable).

The result of \@tempa expansion is then given to \@tempb whose purpose is to cancel out whatever is found between the two delimiters: |\@tempb...|\@tempb:

```
   \def\@tempb|##1|\@tempb##2|\@tempb{%
      \ifblank{##2}{\unexpanded{##1}}
```

If the ⟨*item*⟩ **was not** is the list, then ##2 will be empty, and

\@tempb|...|\@tempb...|\@tempb

is replaced by the *original*-list (i. e., ##1 – that we shall not expand), then the loop is broken; otherwise the ⟨*item*⟩ was in the list and ##1 is the *shortened*-list without the ⟨*item*⟩. We have to loop to remove all the ⟨*item*s⟩ of the list, except in the case where the *shortened*-list is empty after having removing ⟨*item*⟩ (##1 empty):

```
      {\ifblank{##1}{}{\@tempx##1&}}}%
```

Now we just have to (define and) expand \@tempx in an \edef which is going to redefine the ⟨*listmacro*⟩. \@tempx expands first \@tempa and then \@tempb on the result of the expansion of \@tempa. The macro \@tempx itself has an argument: it is (at first stage) the replacement text of ⟨*listmacro*⟩:

```
   \def\@tempx##1&{\expandafter\@tempb\@tempa|##1|\@tempb|#3|&}%
   \edef\@redef{\endgroup
      \unexpanded{#1#2}{% ie: \def or \gdef   \listmacro
```

\@redef redefines the list using (\def or \gdef), the replacement text is the result of the expansion of \@tempx on the ⟨*listmacro*⟩ which is expanded once (to see its items...):

```
      \expandafter\@tempx\unexpanded\expandafter{#2}&}%
```

Then just expand \@redef to proceed the redefinition of ⟨*listmacro*⟩:

```
   }\@redef}% end of \@listdel
```

And ends the “catcode-group”:

```
\endgroup
```

\getlistindex \getlistindex may be defined, with its star form (no expansion of the list) and normal form (⟨*listmacro*⟩ expanded once); The search-index is initialised at 1:

We first need to get into a group where delimiter ‘|’ and ‘&’ have catcode 3:

```
\begingroup\catcode`\|=3\catcode`\&=3
\gdef\getlistindex#1{\FE@ifstar{#1}{\ettl@getlistindex{}}
                                   {\ettl@getlistindex\expandnext}}
%   \@ifstar{\ettl@getlistindex{}}{\ettl@getlistindex{\expandnext}}}
```

\xgetlistindex is similar with prior expansion of ⟨*item*⟩:

```
\gdef\xgetlistindex#1{\FE@ifstar{#1}{\ettl@xgetlistindex{}}
                                    {\ettl@xgetlistindex\expandnext}}
\gdef\ettl@xgetlistindex#1#2#3{\begingroup
   \protected@edef\next{\endgroup\unexpanded{\ettl@getlistindex{#1}{#2}}{#3}%
   }\next}
```

Then the following macro does the job in a loop, which is not fully expandable because the use of \ifstrequal:

```
\gdef\ettl@getlistindex#1#2#3{\begingroup#1\ettl@get@list@idx{#3}{#2}{\numexpr1}}
\gdef\ettl@get@list@idx#1#2#3{% #1=expanded list, #2=item, #3=index
```

```
147    \ifblank{#1}0% the (remainder of) the list is empty
148       {\expandnext\ifstrequal{\ettl@list@first@item#1&}{#2}
149          {\endgroup\edef\indexinlist{\number#3\relax}}
150          {\expandnext\ettl@get@list@idx{\ettl@list@other@item#1&}{#2}{#3+1}}}}
```

The next two macros expand to the first item in a list or the remainder of the list respectively :

```
151 \gdef\ettl@list@first@item#1|#2&{#1}
152 \gdef\ettl@list@other@item#1|#2&{#2}
153 \endgroup% catcode group
```

\getlistitem 

```
154 \newcommand\getlistitem[1]{\FE@ifstar{#1}\ettl@getlistitem
155                                       {\expandnext\ettl@getlistitem}}
156 \begingroup\catcode`\&=3
157 \long\gdef\ettl@getlistitem#1#2{%#1=listmacro, #2=index
158    \ettl@get@list@item{#1}{\number\numexpr#2}}
159 \long\gdef\ettl@get@list@item#1#2{%
160    \ifblank{#1}{}
161       {\ifnum#2=1 \ettl@afterelsefi
162             \expandonce{\ettl@list@first@item#1&}
163       \else\ifnum#2>0 \ettl@afterfifi
164             \expandnext\ettl@getlistitem{\ettl@list@other@item#1&}{#2-1}%
165       \fi\fi}}
166 \endgroup
```

167 ⟨/package⟩

## 6.7 Example

```
168 ⟨*example⟩
169 \documentclass[11pt,french,a4paper,oneside]{scrartcl}
170 \usepackage[latin1]{inputenc}
171 \usepackage[T1]{fontenc}
172 \usepackage[american]{babel}
173 \usepackage{geometry,doc,ltxdockit,txfonts,fancyhdr}
174 \usepackage{etextools}
175 \hypersetup{colorlinks,pdfstartview={FitH}}
176 \geometry{top=2cm,bottom=2cm,left=2.5cm,right=1cm}
177 \fancyhf{}
178 \fancyhead[L]{Examples for the \sty{etextools} package}
179 \pagestyle{fancy}
180 \MakeShortVerb{\|}
181
182 \makeatletter
183 \def\smex{\leavevmode\hb@xt@2em{\hfil$\longrightarrow$\hfil}}
184 \def\strip@meaning{\expandafter\strip@prefix\meaning}
185 \def\strip@macro{\expandafter\strip@macroprefix\meaning}
186 \def\get@params#1{\expandafter\@get@params\meaning#1\@nil}
187 \edef\@get@params{%
188    \def\noexpand\@get@params\detokenize{macro:}##1\detokenize{->}##2\noexpand\@nil{##1}
189 }\@get@params
190 \def\make@macro#1{\string\def\string#1\get@params#1\string{\strip@meaning#1\string}}
191 \newcommand\preline{\@ifstar{\@preline}{\hrulefill\par\@preline}}
192 \def\@preline#1{\noindent\hskip6pt\texttt{\make@macro#1}\par\vskip1.5ex}
193
194 \def\meaningcs#1{\expandafter\meaning\csname#1\endcsname}
195 \def\Meaningcs#1{\expandafter\strip@meaning\csname #1\endcsname}
196
197 \newcommand\test[1]{%
198    \csname test#1\endcsname
```

```
    \edef\usercmd{\Meaningcs{test#1}}\edef\result{\meaningcs{#1Test}}\noindent
    \begin{tabular}{lp{15cm}}
    \multicolumn{2}{l}{\textcolor{blue}{\llap\smex\tt \usercmd}} \\[1.5ex]
    \cmd{#1Test}= & \tt\bfseries\result
    \end{tabular}\par\nobreak\hrulefill\null\goodbreak}


\begin{document}
\title{etextools examples}
\subtitle{Examples for some macros provided by the \sty{etextools} package}
\author{Florent Chervet}
\date{July 22, 2009}
\maketitle

\tableofcontents


\section{\cmd{expandnext} examples}

\subsection{Test if the replacement text of  macro is really empty}

\def\xx{   }
\def\testexpandnext{%
    \edef\expandnextTest{\string\xx\ is \expandnext\ifempty{\xx}{}{not} empty}
}
\preline\xx
\test{expandnext}

\def\xx{}
\preline\xx
\test{expandnext}


\clearpage
\subsection{Test if the replacement text of a macro is blank (empty or spaces)}

\def\xx{something}
\def\testexpandnext{%
    \edef\expandnextTest{\string\xx\ is \expandnext\ifblank{\xx}{}{not} blank}
}
\preline\xx
\test{expandnext}

\def\xx{    }
\preline\xx
\test{expandnext}


\subsection{Detokenize the replacement text of a named-sequence}
\def\detokenizecs#1{\expandnext\expandnext\detokenize{\csname #1\endcsname}}
\def\testexpandnext{%
    \edef\expandnextTest{\detokenizecs{document}}}
\preline\detokenizecs
\test{expandnext}


\section{Testing characters}
\subsection{\cmd{ifsinglechar} versus \cmd{iffirstchar}}
\def\testifsinglechar{%
    \edef\ifsinglecharTest{\ifsinglechar *{*hello*}{ single star }{ something else }}
```

```
}\hrulefill\par
\test{ifsinglechar}

\def\testiffirstchar{%
   \edef\iffirstcharTest{\iffirstchar *{*hello*}{ first char is star }{ something else
}\hrulefill\par
\test{iffirstchar}

\subsection{Fully Expandable starred macros}
\def\starmacro#1{\FE@ifstar{#1}\starred\notstarred}
\def\starred#1{your "#1" will be processed by the STAR form}
\def\notstarred#1{your "#1" will be processed by the NORMAL form}
\def\testFE@ifstar{%
   \edef\FE@ifstarTest{\starmacro{sample text}}}
\preline\starmacro
\preline*\starred
\preline*\notstarred
\test{FE@ifstar}

\def\testFE@ifstar{%
   \edef\FE@ifstarTest{\starmacro*{sample text}}}
\hrulefill\par
\test{FE@ifstar}

\subsection{Fully Expandable macros with options}
\def\optmacro#1{\FE@testopt{#1}\OPTmacro{Mr.}}
\def\OPTmacro[#1]#2{#1 #2}
\def\testFE@testopt{%
   \edef\FE@testoptTest{\optmacro{Woody Allen}}}
\preline\optmacro
\preline*\OPTmacro
\test{FE@testopt}

\def\testFE@testopt{%
   \edef\FE@testoptTest{\optmacro[Ms.]{Vanessa Paradis}}}
\hrulefill\par
\test{FE@testopt}

\section{Lists management}
\subsection{\cmd{csvloop} and \cmd{csvloop*} examples}
\subsubsection{\cmd{makequotes}}
\def\makequotes#1{"#1"\space}
\def\testcsvloop{%
   \edef\csvloopTest{\csvloop*[\makequotes]{hello,world}}
}
\preline\makequotes
\test{csvloop}
\subsubsection{\cmd{detokenize}}
\def\testcsvloop{%
   \edef\csvloopTest{\csvloop*[\detokenize]{\un,\deux}}
}\hrulefill\par
\test{csvloop}
\subsubsection{\cmd{numexpr}}
\def\mylist{1,2,3,4,5}\def\BySeven#1{$#1\times 7 = \number\numexpr#1*7\relax$\par}
\def\testcsvloop{%
   \edef\csvloopTest{\csvloop[\BySeven]\mylist}}
\preline\mylist
\preline*\BySeven
\test{csvloop}
\subsubsection{protected \cmd{textbf}}
\def\testcsvloop{%
```

```
    \protected@edef\csvloopTest{\csvloop*[\textbf]{hello ,my ,friends}}
}\hrulefill\par
\test{csvloop}

\subsection{\cmd{getlistitem}}
\csvtolist*\mylist{one,two,three,four,five,alpha,beta,gamma}
\def\testgetlistitem{%
    \edef\getlistitemTest{\getlistitem\mylist{4}}
}\hrulefill\par
\noindent\hskip6pt|\csvtolist*\mylist{one,two,three,four,five,alpha,beta,gamma}|\par\vsk
\test{getlistitem}

\subsection{\cmd{getlistindex} {\mdseries(not expandable)}}
\getlistindex{four}\mylist
\hrulefill\par
\noindent\hskip6pt|\csvtolist*\mylist{one,two,three,four,five,alpha,beta,gamma}|\par\vsk
\noindent\hskip6pt\textcolor{blue}{\llap\smex\cmd{getlistindex}\string{four\string}\cmd
\noindent\hskip6pt\cmd{indexinlist}=\quad{\bfseries\meaning\indexinlist}\par\hrulefill\p

\end{document}
```

⟨/example⟩

# 7 Revision history

### 2z 2009-08-12

Addition of `\ifempty`

Modification of `\ifsinglechar`

`\ifsinglechar` now works with `\ifempty` so that:

| | |
|---|---|
| `\macro{ * }` | is no more considered as a starred form because of the spaces following the `*` however, the spaces **before** are skipped, as does `\@ifnextchar` from the LATEX kernel. |

Index added to this documentation paper.

### 2e 2009-07-14

First version (include an example file)

# References


[1] David Carlisle and Peter Breitenlohner *The etex package*; 1998/03/26 v2.0; CTAN:macros/latex/contrib/etex-pkg/.

[2] Philipp Lehman *The etoolbox package*; 2008/06/28 v1.7; CTAN:macros/latex/contrib/etoolbox/.

# Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.