The etextools package

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

Florent CHERVET florent.chervet@free.fr

Version 2 θ

15 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 and \xifempty macros
- 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 uses this new feature : command-list parsers are fully expandable: \csvloop, \listloop, \toksloop and more...
- four macros that are lacking from the etoolbox package for removing elements from lists : \listdel and \listgdel, \listedel, \listxdel
- get the index of an item in a list: \getlistindex and \xgetlistindex and conversely the item at a given position in a list: \getlistitem
- a vectorized form of \futurelet: \collecttoks is a standard token scanner.

[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
The .dtx file is embeded in this pdf thank to embedfile by H. Oberdiek.

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⟩

\ifempty is based on \detokenize and accept anything in its argument.

\xifempty{⟨string or cs-token⟩}{⟨true⟩}{⟨false⟩}

pdfTEX \xifempty is similar to \ifempty but the argument is expanded during comparison.

```
\def\x{\@empty}\def\y{}
\xifempty{\x\y} ⟨true⟩⟨false⟩
```

The macro is based on the \pdfstrcmp primitive.

\ifnotempty{⟨string⟩}{⟨true⟩}{⟨false⟩}

\ifnotempty reverses the test of \ifempty

\xifblank{⟨string⟩}{⟨true⟩}{⟨false⟩}

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

\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}
```

Alternatively, you may use the \ifstrmatch test.

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.

Note that there is a small difference if you run on pdfTEX or on another system:

pdfTEX (use \pdfstrcmp)	Other
\ifsinglechar *{␣*} is **false**	\ifsinglechar *{␣*} is **true**

But: \ifsinglechar *{*␣} is **always false**.

\iflastchar{⟨string1⟩}{⟨string2⟩}{⟨true⟩}{⟨false⟩}

\iflastchar compares the end of ⟨string2⟩ with the totality of ⟨string1⟩ (⟨string1⟩ is not necessarily a single character but the macro has been designed for this purpose first, hence the name).

\iflastchar is NOT purely expandable (use \ifstrmatch for purely expandable test).

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

pdfTEX \ifstrcmp is the LATEX form of \pdfstrcmp primitive.

Neither ⟨*string1*⟩ not ⟨*string2*⟩ is expanded during comparison. The comparison is *catcode agnostic* (use of \detokenize).

\ifstrmatch{⟨*pattern*⟩}{⟨*string*⟩}{⟨*true*⟩}{⟨*false*⟩}

pdfTEX \ifstrmatch is based on the \pdfmatch primitive that implements POSIX-regex.

You can test the last character of a string in a purely expandable way by:

```
\ifstrmatch{[*]$}{⟨string⟩}
```

for example to test *.

4 Fully expandable macros with options

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}}}
```

\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 \toksloop.

\FE@ifchar{⟨*Variant Mark Character*⟩}{⟨*argument to be tested*⟩}{⟨*special-commands*⟩}{⟨*normal-commands*⟩}

As a generalisation of \FE@ifstar etextools provides \FE@ifchar for use with other variants than the star-form.

For example, to define a ' + ' variant:

```
\def\SpecialMacro#1{\FE@ifchar+{#1}
                           {\SpecialFormMacro}
                           {\NormalFormMacro}}
```

\SpecialMacro must have at least one mandatory argument.

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*⟩}

\toksloop[⟨*auxiliary commands*⟩]{⟨*tokenslistmacro*⟩}

\toksloop*[⟨*auxiliary commands*⟩]{⟨*list of tokens*⟩}

\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}
```

The token list is a list without delimiter.

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 Converting lists of tokens to etoolbox-lists

\tokstolist{⟨*listmacro*⟩}{⟨*tokenslistmacro*⟩}

\tokstolist*{⟨*listmacro*⟩}{⟨*list of tokens*⟩}

\tokstolist converts a list of tokens (no separator) into an internal etoolbox list.

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

\tokstolistadd{⟨*listmacro*⟩}{⟨*tokslistmacro*⟩}

\tokstolistadd*{⟨*listmacro*⟩}{⟨*list of tokens*⟩}

\tokstolistadd acts similarly but does not reset the ⟨*listmacro*⟩ to \@empty.

5.5 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:

5.6 Index of an element in a list

5.6.1 etoolbox-lists

\getlistindex[⟨myindex⟩]{⟨item⟩}{⟨listmacro⟩}
\xgetlistindex[⟨myindex⟩]{⟨item⟩}{⟨listmacro⟩}
\getlistindex*[⟨myindex⟩]{⟨item⟩}{⟨list⟩}
\xgetlistindex*[⟨myindex⟩]{⟨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 expands the ⟨item⟩ while looking for it in the list.

The comparison is *catcode agnostic*.

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

pdfTEX When used on pdfTEX, \pdfstrcmp does the comparison and the macros are purely expandable:

- If ⟨item⟩ is not found in the list, \getlistindex expands to 0
- If ⟨item⟩ is found in first position then \getlistindex expands to 1 and so on.

Those macros are purely expandable only if \pdfstrcmp exists. Otherwise, the comparison is done with etoolbox-\ifstrequal and you shall use the optional argument to let the result in:

\getlistindex[\myindex]{⟨item⟩}{⟨listmacro⟩}

N.B. If \myindex is not a counter it is (possibly *re*-)defined as macro.

5.6.2 Comma-separated lists

\getcsvlistindex[⟨myindex⟩]{⟨item⟩}{⟨csvlistmacro⟩}
\xgetcsvlistindex[⟨myindex⟩]{⟨item⟩}{⟨csvlistmacro⟩}
\getcsvlistindex*[⟨myindex⟩]{⟨item⟩}{⟨item,item,item,...⟩}
\xgetcsvlistindex*[⟨myindex⟩]{⟨item⟩}{⟨item,item,item,...⟩}

This is the same as \getlistindex but for comma-separated lists.

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

pdfTEX Those macros are purely expandable only if \pdfstrcmp exists.

Otherwise, use the optional argument to let the result in:

\getcsvlistindex*[\myindex]{⟨item⟩}{⟨item,item,item...⟩}

If \myindex is not a counter it is (possibly *re*-)defined as macro.

5.6.3 Lists of tokens

\gettokslistindex[⟨myindex⟩]{⟨item⟩}{⟨tokenslistmacro⟩}
\gettokslistindex*[⟨myindex⟩]{⟨item⟩}{⟨list of tokens⟩}

There is no x- version for those macros. They work in a different way: the comparison if done by \ifx and not \pdfstrcmp.

Therefore, \gettokslistindex is **always purely expandable**, but for symmetry reasons, the optional form:

\gettokslistindex[\myindex]{⟨*item*⟩}{⟨*tokenslistmacro*⟩}
and:\gettokslistindex*[\myindex]{⟨*item*⟩}{⟨*list of tokens*⟩}

are defined.

An interesting application is the following:

```
\ifcase  \gettokslistindex*{R}{LORG}
      Not in the list: Error ?%
\or   Here it's L%
\or   Here it's O%
\or   Here it's R%
\or   Here it's G%
\fi
```

Please, refer to the examples...

5.7 Extract by index from a list

5.7.1 etoolbox-lists

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

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 ({}).

5.7.2 Comma-separated lists

\getcsvlistitem{⟨*index [numexpr expression]*⟩}{⟨*csvlistmacro*⟩}
\getcsvlistitem*{⟨*index [numexpr expression]*⟩}{⟨*item,item,item,...*⟩}

This is the same as \getlistitem but for comma-separated lists.

5.7.3 Lists of tokens

\gettokslistitem{⟨*index [numexpr expression]*⟩}{⟨*tokenslistmacro*⟩}
\gettokslistitem*{⟨*index [numexpr expression]*⟩}{⟨*list of tokens*⟩}

The same as \getlistitem for list of tokens.

6 Vectorized \futurelet

etextools defines a vectorized version of \futurelet. The idea is to say:

\collecttoks\macro{⟨*list of allowed tokens*⟩}{⟨*commands to execute next*⟩}

Then \collecttoks is a kind of simple scanner for tokens. It can be used to define an ***undelimited macro*** i. e., a macro that has no delimiter but whose content of arguments is restricted.

\collecttoks{⟨*cs-token*⟩}{⟨*list of tokens*⟩}{⟨*commands to expand after*⟩}

The macro \collecttoks will read the following tokens with \futurelet. If that token is in the allowed list, then it will append it to \macro (⟨*cs-token*⟩) and continue, scanning the tokens one after another.

Until it founds a token which is not in the list. Then it stops reading and execute the ⟨*commands to expand*⟩. Those commands may use the \macro defined for analyse or whatever the user want.

As an application, it can be used to define an easy interface for \hdashline (the dashed lines in tabulars and arrays provided by the arydshln package): modifying \hline in order to give sense to the following:

\hline.. \hline-- \hline== \hline.- \hline.-. etc.

After having collected the allowed tokens with:
\collecttoks\nexttokens{.-=}{⟨*commands next*⟩} it is possible to test the pattern given using \pdfstrcmp and, for example, the **\switch** construction of the boolexpr package:

```
\switch[\pdfstrcmp{\nexttokens}]
\case{{..}}\hdashline[parameters]%
\case{{--}}\hdashline[parameters]%
\case{{==}}\hdashline[parameters]%
\case{{.-.}}\hdashline[parameters]%
\otherwise \original@hline%
\endswitch
```

\switch is purely expandable.

If you replace \pdfstrcmp by \ifstrequal, \switch works but is not purely expandable.

See boolexpr (CTAN:macros/latex/contrib/boolexpr/) for more information on \switch.

LATEX code

7 Implementation

7.1 Package identification

```
1 ⟨*package⟩
2 \NeedsTeXFormat{LaTeX2e}[1996/12/01]
3 \ProvidesPackage{etextools}
4    [2009/08/15 v2t e-TeX more useful tools for LaTeX package writers]
5 \csname ettl@onlyonce\endcsname\let\ettl@onlyonce\endinput
```

7.2 Requirements

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

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

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

```
\let\ettl@AtEnd\@empty
\def\TMP@EnsureCode#1#2{%
  \edef\ettl@AtEnd{%
    \ettl@AtEnd
    \catcode#1 \the\catcode#1\relax
  }%
  \catcode#1 #2\relax
}
\TMP@EnsureCode{32}{10}% space
\ifundef\pdfstrcmp{%
  \TMP@EnsureCode{33}{9}% ! ignore
  \TMP@EnsureCode{34}{14}% " comment
}{\TMP@EnsureCode{33}{14}% ! comment
  \TMP@EnsureCode{34}{9}% " ignore
}
\AtEndOfPackage{\ettl@AtEnd\undef\ettl@AtEnd}
```

7.3 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}
\long\def\ettl@afterorfi#1\or#2\fi{\fi#1}
```

\ettl@ifstrcmp Use of the pdfTEX primitive \pdfstrcmp and design a LATEX form.

```
\def\ettl@ifstrcmp#1#2{\csname @%
   \ifnum\pdfstrcmp{#1}{#2}=0 first\else second\fi oftwo\endcsname}
```

\@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...

```
\def\ettl@onlypdfTeX#1#2{\ifundef{#1}
   {\def#2{\PackageError{etextools}{\string#1\space primitive not found\MessageBreak
   pdfTeX seems not to be running}
   {\string#2\space works only if used with pdfTeX (requires \string#1)}}}
   {}}
```

7.4 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}
```

7.5 Character and Strings

`\ifempty` `\ifempty` is based on `\detokenize` and can manage with any argument.

```
\newcommand\ifempty[1]{\csname @\expandafter\ifx
   \expandafter\relax\detokenize{#1}\relax
   first\else second\fi oftwo\endcsname}
```

`\ifnotempty` `\ifnotempty` is based on `\detokenize` and can manage with any argument.

```
\newcommand\ifnotempty[1]{\csname @\expandafter\ifx
   \expandafter\relax\detokenize{#1}\relax
   second\else first\fi oftwo\endcsname}
```

`\xifempty` `\xifempty` is based on pdf-TeX `\pdfstrcmp` and work with any argument.

```
\newcommand\xifempty[1]{\xifstrcmp{#1}{}}
\ettl@onlypdfTeX\pdfstrcmp\xifempty
```

`\ifnotblank` `\ifnotblank` reverses the test of `\ifblank`.

```
\begingroup\catcode`\&=3\catcode`\:=12
\ifcsundef{ifnotblank}{%
\long\gdef\ifnotblank#1{\etb@ifblank@i#1&&\@firstoftwo\@secondoftwo:}}
\relax\endgroup
```

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

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

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

```
\newcommand\iffirstchar[2]{\csname @%
   \if \expandafter\@car\detokenize{#2}\relax\@nil\expandafter\@car\detokenize{#1}\stri
      \ifblank{#2}{second}{first}%
   \else second%
   \fi oftwo\endcsname}
```

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

```
\newcommand\ifsinglechar[2]{%
```

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

```
   \csname @\ifblank{#2}{second}{%
"       \ifnum\pdfstrcmp{\detokenize{#1}}{\detokenize{#2}}=0
"          \ifnum\pdfstrcmp{\detokenize{#2}}{\expandafter\@car\detokenize{#2}\relax\@nil
```

```
76 "            first%
77 "         \else second%
78 "         \fi
79 "      \else second%
80 "      \fi oftwo\endcsname}}
81 !     \expandafter\expandafter\expandafter\ifx
82 !         \expandafter\expandafter\expandafter\relax\expandafter\@cdr\detokenize{#2}\@ni
83 !         \iffirstchar{#1}{#2}{first}{second}\else second\fi oftwo\endcsname}}
```

`\iflastchar` This macro is not purely expandable.

```
84 \begingroup\catcode`\&=3
85 \long\gdef\iflastchar#1#2{%
86    \long\def\ettl@iflastchar##1#1##2&{%
87       \ifstrcmp{##2}{#1}\@firstoftwo{%
88          \ifempty{##2}\@secondoftwo{\ettl@iflastchar##2&}}
89    }\ettl@iflastchar#2#1&}
90 \endgroup
```

`\ifstrcmp` The macro is base on the `\pdfstrcmp` primitive.

```
91 \newcommand\ifstrcmp[2]{\csname @%
92   \ifnum\pdfstrcmp{\detokenize{#1}}{\detokenize{#2}}=0 first\else second\fi
93   oftwo\endcsname}
94 \ettl@onlypdfTeX\pdfstrcmp\ifstrcmp
```

`\xifstrcmp` The macro is based on the `\pdfstrcmp` primitive.

```
95 \newcommand\xifstrcmp[2]{\csname @%
96   \ifnum\pdfstrcmp{#1}{#2}=0 first\else second\fi
97   oftwo\endcsname}
98 \ettl@onlypdfTeX\pdfstrcmp\ifxstrcmp
```

`\xifstrequal` The macro is based on `etoolbox-\ifstrequal`.

```
99 \newrobustcmd*\xifstrequal[2]{\begingroup
100    \protected@edef\@xifstrequal{\endgroup\noexpand\ifstrequal{#1}{#2}%
101    }\@xifstrequal}
```

`\ifstrmatch` The macro is base on the `\pdfmatch` primitive.

```
102 \newcommand\ifstrmatch[2]{\csname @%
103  \ifnum\pdfmatch{\detokenize{#1}}{\detokenize{#2}}=1 first\else second\fi
104   oftwo\endcsname}
105 \ettl@onlypdfTeX\pdfmatch\ifstrmatch
```

7.6 Purely expandable macros with options

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

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

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

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

`\FE@ifchar` Plus généralement, on peut tester si une macro est suivie d'un caractère donné:

```
108 \newcommand\FE@ifchar[4]{\ifsinglechar{#1}{#2}{#3}{#4{#2}}}
```

\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@opt\string#1`{}}
      {\`ettl@lst@opt\string#1`\noexpandafter}}%
```

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@opt\string#1}####1####2{\noexpand\FE@testopt{####2}
      {\`ettl@lst@@pt\string#1`{####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@@pt\string#1}####1[####2]####3{%
      ####1\`ettl@lst\string#1`####1{####3}{####2}}%
```

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

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
```

Definition of \toksloop (with no delimiter):

```
\DeclareCmdListParser\toksloop{}%
```

\csvtolist This is the first application of \csvloop:

```
\newrobustcmd*\csvtolist{\@ifstar
      {\ettl@tolist\z@{}\csvloop}
      {\ettl@tolist\z@\expandnext\csvloop}}%
\long\def\ettl@tolist#1#2#3#4#5{\ifx#1\z@\let#4\@empty\fi
   \def\ettl@to@list{#3*[{\unexpanded{\listadd#4}}]%
   }#2\ettl@to@list{#5}}
```

\tokstolist This is the first application of \toksloop:

```
\newrobustcmd*\tokstolist{\@ifstar{\ettl@tokstolist\z@{}}{\ettl@tokstolist\z@\expandnext
\def\ettl@tokstolist#1#2#3#4{\ifx#1\z@\let#3\@empty\fi
      \def\ettl@toks@tolist{\toksloop*[{\unexpanded{\listadd#3}}]}%
         #2\ettl@toks@tolist{#4{}}}
```

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

```
\newrobustcmd*\csvtolistadd{\@ifstar
      {\ettl@tolist\@ne{}\csvloop}
      {\ettl@tolist\@ne\expandnext\csvloop}}
```

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

```
\newrobustcmd*\tokstolistadd{\@ifstar{\ettl@tokstolist\@ne{}}{\ettl@tokstolist\@ne\expa
```

\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
      \long\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:

```
\long\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*⟩:

```
\long\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:

```
\newcommand\getlistindex[1]{\FE@ifstar{#1}
            {\ettl@getlistindex{}\unexpanded}
            {\ettl@getlistindex\expandnext\unexpanded}}
```

\xgetlistindex

\xgetlistindex is similar but the ⟨*item*⟩ is expanded during string comparison (inside \pdfstrcmp is pdfTEX, or before the test if this primitive is not found).

```
\newcommand\xgetlistindex[1]{\FE@ifstar{#1}
           {\ettl@getlistindex{}\@firstofone}
           {\ettl@getlistindex\expandnext\@firstofone}}
```

Then the following macro does the job in a loop, which is fully expandable when we can use \pdfstrcmp:

```
\begingroup\catcode`\|=3\catcode`\&=3
\long\gdef\ettl@getlistindex#1#2#3{\FE@testopt{#3}{\ettl@get@listindex{#1}{#2}}{}}
\long\gdef\ettl@get@listindex#1#2[#3]#4#5{#1\ettl@get@list@idx{#5}{#4}{\numexpr1}{#2}[{#
\long\gdef\ettl@get@list@idx#1#2#3#4[#5]{% #1=expanded list, #2=item, #3=index, #4=\une
   \ifblank{#1}0% the (remainder of) the list is empty
"      {\xifstrcmp{\expandonce{\ettl@list@first@item#1&}}{#4{#2}}}
!      {\ettl@listitem@cmp{\ettl@list@first@item#1&}{#2}{#4}}
         {\ifblank{#5}{\number#3\relax}{\ettl@ifiscount{#5}{#5=\number#3\relax}{\edef#5
         {\expandnext\ettl@get@list@idx{\ettl@list@other@item#1&}{#2}{#3+1}{#4}[{#5}]}}
```

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

```
\long\gdef\ettl@list@first@item#1|#2&{#1}
```

```
\long\gdef\ettl@list@other@item#1|#2&{#2}
!\long\gdef\ettl@listitem@cmp#1#2#3{%
!    \ifx#3\unexpanded\def\ettl@listitem@cmp@{#2}%
!    \else\protected@edef\ettl@listitem@cmp@{#2}%
!    \fi
!    \expandafter\ifstrequal\expandafter{#1}{#2}}
\edef\ettl@ifiscount{%
   \gdef\noexpand\ettl@ifiscount##1{%
      \noexpand\expandafter\noexpand\ettl@ifiscount@i
         \noexpand\meaning##1\string\count\noexpand\@nil}%
}\ettl@ifiscount
\edef\ettl@ifiscount@i{%
   \gdef\noexpand\ettl@ifiscount@i##1\string\count##2\noexpand\@nil{%
      \noexpand\ifempty{##1}}%
}\ettl@ifiscount@i
```

\getcsvlistindex

```
\long\gdef\getcsvlistindex#1{\FE@ifstar{#1}
                  {\ettl@getcsvlistindex{}\unexpanded}
                  {\ettl@getcsvlistindex\expandnext\unexpanded}}
```

\xgetcsvlistindex is similar but the ⟨*item*⟩ is expanded during string comparison (inside \pdfstrcmp is pdfTEX, or before the test if this primitive is not found).

```
\long\gdef\xgetcsvlistindex#1{\FE@ifstar{#1}
                  {\ettl@getcsvlistindex{}\@firstofone}
                  {\ettl@getcsvlistindex\expandnext\@firstofone}}
```

Then the following macro does the job in a loop, which is fully expandable when we can use \pdfstrcmp:

```
\long\gdef\ettl@getcsvlistindex#1#2#3{\FE@testopt{#3}{\ettl@get@csvlistindex{#1}{#2}}{}
\long\gdef\ettl@get@csvlistindex#1#2[#3]#4#5{#1\ettl@get@csvlist@idx{#5}{#4}{\numexpr1}
\long\gdef\ettl@get@csvlist@idx#1#2#3#4[#5]{% #1=expanded list, #2=item, #3=index, #4=\
   \ifblank{#1}0% the (remainder of) the list is empty
"      {\xifstrcmp{\expandonce{\ettl@csvlist@first@item#1&}}{#4{#2}}}
!      {\ettl@listitem@cmp{\ettl@csvlist@first@item#1&}{#2}{#4}}
         {\ifblank{#5}{\number#3\relax}{\ettl@ifiscount{#5}{#5=\number#3\relax}{\edef#5
         {\expandnext\ettl@get@csvlist@idx{\ettl@csvlist@other@item#1&}{#2}{#3+1}{#4}[{#
```

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

```
\long\gdef\ettl@csvlist@first@item#1,#2&{#1}
\long\gdef\ettl@csvlist@other@item#1,#2&{#2}
```

\gettokslistindex

```
\long\gdef\gettokslistindex#1{\FE@ifstar{#1}
                  {\ettl@gettokslistindex{}}
                  {\ettl@gettokslistindex\expandnext}}

\long\gdef\ettl@gettokslistindex#1#2{\FE@testopt{#2}{\ettl@get@tokslistindex{#1}}{}}
\long\gdef\ettl@get@tokslistindex#1[#2]#3#4{#1\ettl@get@tokslist@idx{#4}{#3}{\numexpr1}
\long\gdef\ettl@get@tokslist@idx#1#2#3[#4]{% #1=expanded list, #2=item, #3=index
   \ifblank{#1}0% the (remainder of) the list is empty
      {\expandafter\ifx\expandafter#2\ettl@tokslist@first@item#1&%
         \ettl@afterelse\ifblank{#4}{\number#3\relax}
                  {\ettl@ifiscount{#4}{#4=\number#3\relax}{\edef#4{\number#3}}}%
       \else \ettl@afterfi
         \expandnext\ettl@get@tokslist@idx{\ettl@tokslist@other@item#1&}{#2}{#3+1}[{#4}
       \fi}}
```

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

```
\long\gdef\ettl@tokslist@first@item#1#2&{#1}
```

```
\long\gdef\ettl@tokslist@other@item#1#2&{#2}
\endgroup% catcode group
```

\getlistitem

```
\newcommand\getlistitem[1]{\FE@ifstar{#1}{\ettl@getlistitem{}}
                                        {\ettl@getlistitem\expandnext}}
\begingroup\catcode`\&=3
\long\gdef\ettl@getlistitem#1#2#3{%#2=index, #3=list or list-macro
   #1\ettl@get@list@item{#3}{\numexpr#2}}
\long\gdef\ettl@get@list@item#1#2{%
   \ifblank{#1}{}
      {\ifnum#2=1 \ettl@afterelsefi
            \expandonce{\ettl@list@first@item#1&}%
      \else\ifnum#2>0 \ettl@afterfifi
            \expandnext\ettl@get@list@item{\ettl@list@other@item#1&}{#2-1}%
      \fi\fi}}
\endgroup
```

\getcsvlistitem

```
\newcommand\getcsvlistitem[1]{\FE@ifstar{#1}{\ettl@getcsvlistitem{}}
                                            {\ettl@getcsvlistitem\expandnext}}
\begingroup\catcode`\&=3
\long\gdef\ettl@getcsvlistitem#1#2#3{%#2=index, #3=list or list-macro
   #1\ettl@get@csvlist@item{#3,}{\numexpr#2}}
\long\gdef\ettl@get@csvlist@item#1#2{%
   \ifblank{#1}{}
      {\ifnum#2=1 \ettl@afterelsefi
            \expandonce{\ettl@csvlist@first@item#1&}%
      \else\ifnum#2>0 \ettl@afterfifi
            \expandnext\ettl@get@csvlist@item{\ettl@csvlist@other@item#1&}{#2-1}%
      \fi\fi}}
\endgroup
```

\gettokslistitem

```
\newcommand\gettokslistitem[1]{\FE@ifstar{#1}{\ettl@gettokslistitem{}}
                                            {\ettl@gettokslistitem\expandnext}}
\begingroup\catcode`\&=3
\long\gdef\ettl@gettokslistitem#1#2#3{%#2=index, #3=list or list-macro
   #1\ettl@get@tokslist@item{#3{}}{\numexpr#2}}
\long\gdef\ettl@get@tokslist@item#1#2{%
   \ifblank{#1}{}
      {\ifnum#2=1 \ettl@afterelsefi
            \expandonce{\ettl@tokslist@first@item#1&}%
      \else\ifnum#2>0 \ettl@afterfifi
            \expandnext\ettl@get@tokslist@item{\@gobble#1}{#2-1}%
      \fi\fi}}
\endgroup
```

\collecttoks A simple token scanner, vectorized version of \futurelet.

```
\newrobustcmd*\collecttoks{\@ifstar\ettl@scantoks\ettl@collecttoks}
\def\ettl@scantoks{\@latex@error{The starred version of \noexpand\collecttoks
   \MessageBreak (i.e. scantoks) is NOT implemented YET.}\@ehd}
```

\ettl@collecttoks

```
\def\ettl@collecttoks#1#2#3{\begingroup
   \def\ettl@collecttoks@list{#2}%
   \let#1=\@empty
   \def\Next{\ettl@collect@toks{#1}}%
```

```
    \def\finale{#3}%
    \futurelet\ettl@x\Next}
\def\ettl@collect@toks#1{%
    \edef\ettl@i{\gettokslistindex\ettl@x\ettl@collecttoks@list}%
    \ifcase\ettl@i \edef\next{\endgroup
       \def\noexpand#1{\expandonce{#1}}\expandonce{\finale}}%
    \else \def\next{\ettl@collect@toks@next{#1}}%
    \fi\next}
\def\ettl@collect@toks@next#1#2{%
    \edef#1{\expandonce{#1}\unexpanded{#2}}%
    \futurelet\ettl@x\Next}

⟨/package⟩
```

7.7 Example

```
⟨*example⟩
\documentclass[11pt,french,a4paper,oneside]{scrartcl}
\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[american]{babel}
\usepackage{geometry,doc,ltxdockit,txfonts,fancyhdr,stmaryrd,graphicx,fancyvrb}
\usepackage{etextools}
\hypersetup{colorlinks,pdfstartview={FitH}}
\geometry{top=1.5cm,bottom=1.2cm,left=2.5cm,right=1cm}
\fancyhf{}
\fancyhead[L]{Examples for the \sty{etextools} package}
\pagestyle{fancy}
\MakeShortVerb{\|}

\makeatletter
\def\smex{\leavevmode\hb@xt@2em{\hfil$\longrightarrow$\hfil}}
\def\FE{\setbox8\hbox{$\m@th\bindnasrepma$}%
          \textcolor{fecc}{\scalebox{2}{$\copy8\mkern-13.5mu\copy8\mkern-13.5mu\copy8$}}
\definecolor{fecc}{rgb}{.2,.6,.2}
\def\strip@meaning{\expandafter\strip@prefix\meaning}
\def\strip@macro{\expandafter\strip@macroprefix\meaning}
\def\get@params#1{\expandafter\@get@params\meaning#1\@nil}
\edef\@get@params{%
    \def\noexpand\@get@params\detokenize{macro:}##1\detokenize{->}##2\noexpand\@nil{##1}
}\@get@params
\def\make@macro#1{\string\def\string#1\get@params#1\string{\strip@meaning#1\string}}
\newcommand\preline{\@ifstar{\@preline}{\hrulefill\par\@preline}}
\def\@preline#1{\noindent\hskip6pt\texttt{\make@macro#1}\par\vskip1.5ex}

\def\meaningcs#1{\expandafter\meaning\csname#1\endcsname}
\def\Meaningcs#1{\expandafter\strip@meaning\csname #1\endcsname}
\@ifundefined{pdfstrcmp}{\let\ifpdfTeX\iffalse}{\let\ifpdfTeX\iftrue}

\newcommand*\test{\@ifstar{\let\fe\relax\testi}{\let\fe\FE\testi}}
\newcommand\testi[1]{%
    \csname test#1\endcsname
    \edef\usercmd{\Meaningcs{test#1}}\edef\result{\meaningcs{#1Test}}\noindent
    \begin{tabular}{lp{15cm}}
    \multicolumn{2}{l}{\textcolor{blue}{\llap{\fe\,\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\testifsinglechar{%
   \edef\ifsinglecharTest{\ifsinglechar *{  *}{ single star }{ something else }}
}\hrulefill\par
\test{ifsinglechar}

\def\testifsinglechar{%
   \edef\ifsinglecharTest{\ifsinglechar *{ *  }{ single star }{ something else }}
```

```
}\hrulefill\par
\test{ifsinglechar}
{\small Note the space \textbf{after} the star $\uparrow$.}

\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{Index in lists and items by index}

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

\subsubsection{\cmd{getlistindex}}
\ifpdfTeX
\leavevmode\vadjust{\textsl{Require the }\string\pdfstrcmp\ \textsl{primitive (pdf\TeX{}
\def\testgetlistindex{%
   \edef\getlistindexTest{\getlistindex{alpha}\mylist}
}\hrulefill\par
\noindent\hskip6pt|\csvtolist*\mylist{one,two,three,four,five,alpha,beta,gamma}|\par\vsk
\test{getlistindex}

\leavevmode\vadjust{\textsl{Require the }\string\pdfstrcmp\ \textsl{primitive (pdf\TeX{}
\def\testgetlistindex{%
   \edef\getlistindexTest{\getcsvlistindex*{alpha}{one,two,three,four,five,alpha,beta}}
}\hrulefill\par
\test{getlistindex}
\fi

\hrulefill\par
\getlistindex[\myindex]{alpha}\mylist
{\color{blue}\noindent\hskip6pt\llap\smex|\getlistindex[\myindex]{alpha}\mylist|\par\vsk
{\tt\string\myindex=\quad\textbf{\meaning\myindex}}
\hrulefill\par\vskip.5ex

\hrulefill\par
\newcount\myindex
\getcsvlistindex*[\myindex]{alpha}{one,two,three,four,five,alpha,beta}
\noindent\hskip6pt|\newcount\myindex|\par
{\color{blue}\noindent\hskip6pt\llap\smex|\getcsvlistindex*[\myindex]{alpha}{one,two,th
{\tt\string\the\string\myindex=\qquad\textbf{\the\myindex}}
\hrulefill\par\vskip.5ex


\subsubsection{\cmd{xgetlistindex}}
\ifpdfTeX
\leavevmode\vadjust{\textsl{Require the }\string\pdfstrcmp\ \textsl{primitive (pdf\TeX{}
\def\x{beta}
\def\testxgetlistindex{%
   \edef\xgetlistindexTest{\xgetlistindex{\x}\mylist}
}\hrulefill\par
\noindent\hskip6pt|\csvtolist*\mylist{one,two,three,four,five,alpha,beta,gamma}|\par\vsk
\preline*\x
\test{xgetlistindex}
\fi
```

```
\subsubsection{\cmd{getlistindex} with \cmd{ifcase}}

\leavevmode\vadjust{Always purely expandable (no need of |\pdfstrcmp|, comparison done
\hrulefill\par\noindent
\llap{\FE\,\smex}\par\vskip-2.5\baselineskip\strut
\begin{Verbatim}
      \ifcase  \gettokslistindex*{D}{LRDF\relax O}
            Problem
      \or   What do to if L
      \or   What do to if R
      \or   What do to if D
      \or   What do to if F
      \or   What do to if \relax
      \or   What do to if O
      \fi
\end{Verbatim}

{\tt Result=\qquad\bfseries
\ifcase\gettokslistindex*{D}{LRDF}
      Problem
\or   What do to if L
\or   What do to if R
\or   What do to if D
\or   What do to if F
\fi}\par
\hrulefill\par\vskip.5ex



\end{document}
⟨/example⟩
```

8 Revision history

2t 2009-08-15

Addition of \ifnotempty, \iflastchar, \ifstrcmp, \ifstrmatch

2h 2009-08-14

\getlistindex is now fully expandable

Addition of

- \toksloop and \toksloop*
- \tokstolist and \tokstolist*
- \getcsvlistindex and \getcsvlistitem
- \gettokslistindex and \gettokslistitem

Addition of

- \collecttoks – a vectorized version of \futurelet.
- \FE@ifchar as a generalization of \FE@ifstar.

2z 2009-08-12

Addition of

\ifempty, \toksloop, \tokstolist and \tokstolistadd

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.