The **ltxnew*** package

provides the \new \renew and \provide prefixes for checking definitions.

Florent Chervet <florent.chervet@free.fr>

October 11, 2009

Abstract

ltxnew provides **\new \renew** and **\provide**: three expandable prefixes for use with \def, \gdef, \edef, \xdef, \countdef, \dimendef, \skipdef, \muskipdef, \box, \toksdef, \marks, \count, \dimen, \skip, \muskip, \savebox, \toks and the \glob*** and \loc*** variants of the etex package.

For example:
\new\def\macro will do *something like:* \newcommand\macro{} \def\macro
\new\let\macro will do *something like:* \newcommand\macro{} \let\macro

...But in fact \new does a little more than that... (see Using \new).

You may use \new or \renew for declaring macros, counters, dimensions, skips, muskips, boxes, tokens and ε-TEX's marks. Even with \let, \new can be used. Moreover, \renew can be used to redefine macros that were previously defined as \outer.

ltxnew is designed to work with an ε-TEX distribution of LATEX. It relies on the LATEX macro \@ifdefinable, on the etex[1] package and some macros of etoolbox[2].

Contents

---

* ltxnew: CTAN:macros/latex/contrib/ltxnew
1. etex: CTAN:macros/latex/contrib/etex-pkg
2. etoolbox: CTAN:macros/latex/contrib/etoolbox
This documentation is produced with the DocStrip utility.
⟶ To get the documentation, run (thrice): `pdflatex ltxnew.dtx`
for the index: `makeindex -s gind.ist ltxnew.idx`
⟶ To get the package, run: `etex ltxnew.dtx`
The .dtx file is embedded into this pdf file thank to embedfile by H. Oberdiek.

1 Introduction

1.1 Motivation

LaTeX provides \newcommand for defining new commands. However, comparing to \def the syntax is limited because we cannot use delimited arguments in such a command. The advantage of \newcommand (apart the optional argument[3]) is that the control sequence is first checked for availability (its meaning ought to be undefined or \relax before the definition).

etoolbox enhance this matter allowing to define \newrobustcmd and \renewrobustcmd.

Moreover, LaTeX does not provide an automatic check of control sequences when defining tokens (\newtoks), dimensions (\newdimen), skips (\newskip), etc. etc.

The only exceptions are:

- \newlength
  but there is no \renewlength command... because the name \renewlength sounds bad: it would have meant *"I know the control sequence I wish to define as a length has been defined before, as a macro may be, or a box or a token or whatever, and I wish to redefine this control sequence to be a length (*ie *a skip).* So it doesn't really make sense...
- \newcounter
  but \newcounter{name} does not define name but \c@name instead, as a counter.
- \newsavebox
- \newfont

All those \new* stuff define control sequences globally, *excepting* \newfont. The reason could to be found in the background[4].**

But it's a matter of fact : *fonts* are local to LaTeX while *length* (*ie. skips*) are global...

Thank to the etex package that provides a method for the local allocation of new quantity ltxnew puts the state of the affairs in a better order. ltxnew provides a way to define new control sequences, or redefine them, just by beginning the definition with a (expandable) prefix : \new or \renew.

1.2 What \new means...

Such a short and easy word as new ought to be defined !

\new means:

- Check if the control sequence to define is available (*ie* means undefined or \relax)
- If that's OK: go on (with a side effect if the package tracing is loaded)
- If not : throw an error, and if in scrollmode or nonstopmode or batchmode do not overwrite the last meaning.

That is really what means \new. No more, no less.

3. optional arguments are implemented in a much flexible way by xargs by Manuel Pégourié-Gonnard.
4. in fact, a new font is defined as a control sequence, just like a macro, whereas skips, dimens, tokens etc. are numbered and then, defining a new one require an allocation.

1.3 What \renew means

\renew means:

- Check if the control sequence to redefine already has a meaning (different from undefined and also from \relax)
- If that's OK : go on (with a side effect if the package tracing is loaded)
- If not : throw an error. But if in srollmode, nonstopmode or batchmode **do define** the control sequence.

1.4 What \provide means

\provide means:

- Check if the control sequence to define already has a meaning (different from undefined and also from \relax)
- If that's OK : go on (with a side effect if the package tracing is loaded)
- If not : silently do nothing.

1.5 Using \new

\new acts as a (expandable) prefix with the following syntax:

	\new ␣	
possibly	(\long␣|\global␣|\protected␣|\outer␣)	optional (zero or more)
in a macro	<DEFINITION WORD>	required: see below
	control sequence	required

␣ denote optional spaces, ignored by the \new-prefixes-scanner.

The <DEFINITION WORD> may be one of the following:

General:	\let			
Macros:	\def	\gdef	\edef	\xdef
Type	*def-word*	*always global*	*local (unless* \global*)*	*global*
Counters:	\countdef	\count	\loccount	\globcount
Dimensions:	\dimendef	\dimen	\locdimen	\globdimen
Skip:	\skipdef	\skip or \length	\locskip	\globskip
Muskip:	\muskipdef	\muskip	\locmuskip	\globmuskip
Box:	\box	\savebox	\locbox	\globbox
Tokens:	\toksdef	\toks	\loctoks	\globtoks
Fonts:	\font			
Marks:	\marks		\locmarks[5]	\globmarks
Write:		\write		
Read:		\read		

Table 1: List of definition-words that may be used with \new \renew and \provide

5. The use of \locmarks is left to the appreciation of the user...

Examples:

\new\countdef\mycount	is the same as	\new\loccount\mycount
\new\global\countdef\mycount	is the same as	\new\globcount\mycount
\new\count\mycount	is the same as	\newcount\mycount (with control sequence checking)
\new\write\fileout	is the same as	\newwrite\fileout (with control sequence checking)

Therefore: (all of the following are global excepting \newfont):

\new\count	is an improved version of	\newcount	close. to	\new\global\countdef
\new\dimen	is an improved version of	\newdimen	close. to	\new\global\dimendef
\new\skip	is an improved version of	\newskip	close. to	\new\global\skipdef
\new\skip	is also the same as	\newlength		
\new\muskip	is an improved version of	\newmuskip	close. to	\new\global\muskipdef
\new\savebox	is the same as	\newsavebox	close. to	\new\global\box
\new\toks	is an improved version of	\newtoks	close. to	\new\global\toksdef
\new\font	is the same as	\newfont		
\new\marks	is an improved version of	\newmarks	equiv. to	\new\global\locmarks

The \loc*** and \glob*** words and also \newmarks are defined by etex[6].
Please note that there is no **\new\command** and there will most probably never be. Nor is there any \new\keycmd if you use the keycommand[7] package.

1.6 Using \renew and \provide

\renew and \provide shares the same syntax as \new.

★ ★ ★

2 Implementation

2.1 Identification

This package is intended to use with LaTeX so we don't check if it is loaded twice.

```
1 ⟨*package⟩
2 \NeedsTeXFormat{LaTeX2e}% LaTeX 2.09 can't be used (nor non-LaTeX)
3    [2005/12/01]% LaTeX must be 2005/12/01 or younger (see kvsetkeys.dtx).
4 \ProvidesPackage{ltxnew}
5    [2009/10/11 v1.1 provides the new and renew prefixes for checking definitions]
```

2.2 Requirements

ltxnew requires etex[8] for local allocation of counters, tokens, skips etc.

```
6 \RequirePackage{etex}
```

2.3 Helper macro

\ltxn@expandonce

\ltxn@expandonce is the copy of the \expandonce macro from etoolbox[9]. As long as this is the only macro from etoolbox we use here, we avoid loading this package.

```
7 \def\ltxn@expandonce#1{\unexpanded\expandafter{#1}}
```

6. etex: CTAN:macros/latex/contrib/etex-pkg
7. keycommand: CTAN:macros/latex/contrib/keycommand
8. etex: CTAN:macros/latex/contrib/etex
9. etoolbox: CTAN:macros/latex/contrib/etoolbox

2.4 The prefixes scanner

The prefixes scanner is very simple in fact! All the job is based of \futurelet: \futurelet reads the next token but does not remove it from the input string. We then just have to test it with \ifx to conditionally append it into the prefix buffer: \ltxn@prfx. Otherwise, we expand the prefix once and try again. Namely:

\futurelet\x\testmacro ⟶ if \testmacro "returned false" then:
\expandafter\futurelet\expandafter\x\expandafter\testmacro

easy easy easy...

If it happens that the expanded prefix is the same before and after expansion, then it means that was a primitive. The only primitives allowed between \new and \def are:

\long	\global	\protected	\outer
\expandafter	\noexpand	*and*	\relax

\ltxn@prefix This is the prefix scanner. We open a group at the very beginning for all definitions will be local until the final definition:

```
\def\ltxn@prefix{\begingroup
   \newif\ifglobal
   \let\ltxn@prfx\@empty
   \let\ltxn@rubbish\relax
   \futurelet\x\ltxn@@prefix}
```

\ltxn@@prefix This is the test macro: it is very long because there are many many \ifx... and as many fees!

```
\def\ltxn@@prefix{%
   \let\ltxn@next@addto\ltxn@next@prefix
   \ifx\x\@sptoken          \let\next\ltxn@space@prefix%%1
   \else                    \let\next\ltxn@addto@prfx
      \ifx\x\long           \def\z{\long}%%2
      \else\ifx\x\protected\def\z{\protected}%%3
      \else\ifx\x\global    \let\z\@empty\globaltrue%%4
      \else\ifx\x\outer     \def\z{\outer}%%5
      \else
         \ifx\x\expandafter \def\z{\expandafter}%%6
         \else\ifx\x\noexpand    \def\z{\noexpand}%%7
         \else\ifx\x\relax       \def\z{\relax}%%8
         \else
                  \def\ltxn@next@addto{\expandafter\ltxn@def\noexpand}%
            \ifx\x\let              \def\z{\let}%%9
                                    \let\ltxn@cancel\ltxn@cancel@let
            \else                   \let\ltxn@cancel\ltxn@cancel@def
               \ifx\x\def           \edef\z{\ifglobal\global\fi\def}%%10
               \else\ifx\x\edef     \edef\z{\ifglobal\global\fi\edef}%%11
               \else\ifx\x\gdef     \def\z{\gdef}%%12
               \else\ifx\x\xdef     \def\z{\xdef}%%13
               \else                \let\ltxn@cancel\ltxn@cancel@new
                  \ifx\x\count       \def\z{\newcount}%%14
                  \else\ifx\x\countdef%%15
                     \ifglobal\def\z{\globcount}\else\def\z{\loccount}\fi
                  \else\ifx\x\loccount%%16
                     \ifglobal\def\z{\globcount}\else\def\z{\loccount}\fi
                  \else\ifx\x\globcount   \def\z{\globcount}%%17
                  \else\ifx\x\dimen       \def\z{\newdimen}%%18
                  \else\ifx\x\dimendef%%19
                     \ifglobal\def\z{\globdimen}\else\def\z{\locdimen}\fi
                  \else\ifx\x\locdimen%%20
                     \ifglobal\def\z{\globdimen}\else\def\z{\locdimen}\fi
                  \else\ifx\x\globdimen   \def\z{\globdimen}%%21
                  \else\ifx\x\skip        \def\z{\newskip}%%22
                  \else\ifx\x\skipdef%%23
```

```
                  \ifglobal\def\z{\globskip}\else\def\z{\locskip}\fi
               \else\ifx\x\locskip%%24
                  \ifglobal\def\z{\globskip}\else\def\z{\locskip}\fi
               \else\ifx\x\globskip    \def\z{\globskip}%%25
               \else\ifx\x\muskip      \def\z{\newmuskip}%%26
               \else\ifx\x\muskipdef%%27
                  \ifglobal\def\z{\globmuskip}\else\def\z{\locmuskip}\fi
               \else\ifx\x\locmuskip%%28
                  \ifglobal\def\z{\globmuskip}\else\def\z{\locmuskip}\fi
               \else\ifx\x\globmuskip  \def\z{\globmuskip}%%29
               \else\ifx\x\savebox     \def\z{\newsavebox}%%30
               \else\ifx\x\box%%31
                  \ifglobal\def\z{\globbox}\else\def\z{\locbox}\fi
               \else\ifx\x\locbox%%
32
                  \ifglobal\def\z{\globbox}\else\def\z{\locbox}\fi
               \else\ifx\x\globbox     \def\z{\globbox}%%33
               \else\ifx\x\toksdef%%34
                  \ifglobal\def\z{\globtoks}\else\def\z{\loctoks}\fi
               \else\ifx\x\toks        \def\z{\newtoks}%%35
               \else\ifx\x\loctoks%%36
                  \ifglobal\def\z{\globtoks}\else\def\z{\loctoks}\fi
               \else\ifx\x\globtoks    \def\z{\globtoks}%%37
               \else\ifx\x\locmarks%%38
                  \ifglobal\def\z{\globmarks}\else\def\z{\locmarks}\fi
               \else\ifx\x\marks       \def\z{\newmarks}%%39   %\newmarks=\globmarks
               \else\ifx\x\globmarks   \def\z{\globmarks}%%40
               \else\ifx\x\font        \def\z{\font}%%41
               \else\ifx\x\write       \def\z{\newwrite}%%42
               \else\ifx\x\read        \def\z{\newread}%%43
               \else\ifx\x\protect     \ltxn@error@prefix%%44
               \else
                  \let\ltxn@next@addto\ltxn@next@prefix
                  \ifx\y\x\ltxn@error@prefix
                  \else\let\y\x
                  \fi
                  \let\next\ltxn@expand@prefix
               \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
               \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
               \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
            \fi\fi\fi\fi
         \fi
      \fi\fi\fi
   \fi\fi\fi\fi% so many fees...
\fi\next}

\def\ltxn@next@prefix{\futurelet\x\ltxn@@prefix}
\def\ltxn@expand@prefix{%
   \expandafter\futurelet\expandafter\x\expandafter\ltxn@@prefix}
\def\ltxn@addto@prfx#1{\let\y\@undefined
   \edef\ltxn@prfx{\ltxn@expandonce{\ltxn@prfx}\ltxn@expandonce{\z}}%
   \ltxn@next@addto}
\expandafter\def\expandafter\ltxn@space@prefix\space{\ltxn@next@prefix}
\def\ltxn@error@prefix{\@latex@error{A \string\def\space
   (or \string\countdef\space or\string\toksdef\space etc.)\MessageBreak
   was expected after \string\new\MessageBreak
   I found a \meaning\x!\MessageBreak
   see ltxnew documentation for more information}\@ehd}
```

2.5 The cancel macros

\ltx@cancel These are the macros used in case we have to cancel definition (nonstopmode)

```
\def\ltxn@cancel@let{\afterassignment\endgroup\let\ltxn@rubbish}
\def\ltxn@cancel@def{\afterassignment\endgroup\def\ltxn@rubbish}
\def\ltxn@cancel@new{\endgroup}
```

2.6 The defining macros

\ltxn@new \ltxn@new defines the new control sequence, or cancels definition depending on the result of \@ifdefinable. \ltxn@new does all the jobs: it is called by both \ltxn@renew and \ltxn@provide:

```
\def\ltxn@new#1{%
   \let\next\ltxn@cancel
   \ifdefined#1\unless\ifx#1\relax\def#1{ltxn@throw@error}\fi\fi
   \expandafter\@ifdefinable\noexpand#1{%
      \expandafter\let\noexpand#1=\relax
      \edef\next{\endgroup\ltxn@expandonce{\ltxn@prfx}\noexpand#1}}%
   \next}
```

\ltxn@renew \ltxn@renew throws an error if the control sequence is undefined or if its meaning is \relax. In case of nonstopmode the control sequence is redefined, however.
To handle the case where the control sequence was an outer macro, we "stringify" its name first, in order not to give the control sequence itself as an argument for the error message.

```
\def\ltxn@renew#1{%
   \edef\ltxn@name{\string#1 }%
   \ifdefined#1   \ifx#1\relax    \ltxn@error{renew: \ltxn@name undefined}\fi
   \else          \ltxn@error{renew: \ltxn@name undefined}%
   \fi
   \let#1=\relax
   \def\next{\ltxn@new#1}%
   \next}
```

\ltxn@provide \ltxn@provide never throws an error, but define the control sequence only if it is undefined or \relax (*ie* if it is definable):
To handle the case where the control sequence was an outer macro, we "stringify" its name first, in order not to put the control sequence itself in the definition of \next. It's admittedly tricky here, because if the control sequence is already defined, \provide will cancel out the new definition, however, as a borderline effect, \ltxn@new should have been called with this very control sequence as an argument, if it had been definable. Even if this \iffalse-call (not expanded) is prepared into \ifx...\fi conditional, the \outer control sequence is there, and TEX (not LATEX) will throw an error: `Forbidden control sequence found...`.

```
\def\ltxn@provide#1{%
   \let\next\ltxn@cancel
   \edef\ltxn@name{\string#1}%
   \ifdefined#1 \ifx#1\relax \ltxn@provide@new\fi
   \else \ltxn@provide@new
   \fi
   \next}
\def\ltxn@provide@new{%
   \edef\next{\noexpand\ltxn@new\csname\expandafter\@gobble\ltxn@name\endcsname}}
```

2.7 The prefixes: \new, \renew and \provide

\new \new: the entry point: just let the definition macro to be \ltxn@new and start scanning prefixes.

```
\def\new{\let\ltxn@def\ltxn@new\ltxn@prefix}
```

\renew `\renew`: the entry point: just let the definition macro to be `\ltxn@renew` and start scanning prefixes.

```
\def\renew{\let\ltxn@def\ltxn@renew\ltxn@prefix}
```

\provide `\provide`: the entry point: just let the definition macro to be `\ltxn@provide` and start scanning prefixes.

```
\def\provide{\let\ltxn@def\ltxn@provide\ltxn@prefix}
```

\ltxn@error In case of redefinition, throws an `\ehc`-type error:

```
\def\ltxn@error#1{\@latex@error{#1}\@ehc}
```

```
⟨/package⟩
```

3 History

[2009/10/11 v1.1]

- Correction of `.sty` header.

[2009/07/22 v1.0]

- First version.

4 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/.

5 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.