The moredefs LaTeX package
more defining commands (Frankenstein's brain)

Matt Swift <swift@alum.mit.edu>

Version: 1.8 Date: 2001/08/31
Documentation revision: 2001/08/31

Abstract

A delightful collection of defining, expansion, and debugging commands that make elegant programming in LaTeX fun and easy.

Contents

I Discussion **2**

1 Naming conventions **2**

2 Conditionals **2**

3 Defining commands **2**
3.1 * and no-* forms 2
3.2 User commands 2

4 Controlling expansion **5**

5 Gobbling **6**

6 Option declaration **6**

7 Toggle a boolean **7**

8 Debugging **7**

II Implementation **8**

9 Version control **8**

10 Conditionals **8**

11 Defining commands **9**

12 Controlling expansion **16**

13 Gobbling **17**

14 Option declaration **18**

15 Toggle a boolean **18**

16 Debugging **18**

Part I
Discussion

These macros were written in response to practical programming needs. Most of the packages I have written, whether distributed or not, depend on this package. Using these constructs has saved me a lot of time and made my code much more readable—that is, maintainable and improvable. For examples of these macros in useful applications, see the packages in the `Frankenstein` bundle.

1 Naming conventions

The convention is that a capital *E* means the macro expands something just once. A lowercase *e*, as in `\edef`, means the macro expands something all the way to unexpandable tokens.

The specification ⟨*csname*⟩ means a control sequence with a preceding backslash; the specification ⟨*csname*⟩ means a control sequence without a preceding backslash. ⟨*csname*⟩ arguments are expanded. Commands which take ⟨*csname*⟩ arguments have `Name` in their names.

When I write *package* in this documentation, I mean LaTeX package or class.

2 Conditionals

`\@ifundefined@cs`

`\@ifundefined@cs{`⟨*csname*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}` executes the ⟨*true*⟩ clause if ⟨*csname*⟩ is not defined, and the ⟨*else*⟩ clause otherwise.

`\IfElement...\In`

To check whether a token ⟨*thingma*⟩ is `\ifx`-equal to any token in a list of tokens stored in a macro ⟨*list*⟩, use `\IfElement` ⟨*thingma*⟩`\In` ⟨*list*⟩`{`⟨*true*⟩`}{`⟨*false*⟩`}`. The top-level expansion of ⟨*list*⟩ must be a list of tokens to compare with ⟨*thingma*⟩ with `\ifx`. If the ⟨*thingma*⟩ is in the ⟨*list*⟩, the ⟨*true*⟩ clause is executed; otherwise, the ⟨*false*⟩ clause is executed.

3 Defining commands

3.1 * and no-* forms

The naming convention of most defining commands in the LaTeX kernel and in *moredefs* is that the no-`*` form of the command is `long` and the `*`-form is not `long`.

3.2 User commands

`\InitCS`
`\InitCS*`
`\InitName`
`\InitName*`
`\ShortEmpty`
`\LongEmpty`
`\ReserveCS`
`\ReserveCS*`

`\InitCS` and `\InitCS*` take one argument, ⟨*csname*⟩, and initialize it to `{}`.

`\InitName` and `\InitName*` are the same but take an argument ⟨*csname*⟩ without a backslash.

To make it easier to avoid the problem of comparing `long` and non-`long` macros with `\ifx`, compare macros with `\ShortEmpty` and `\LongEmpty`.

`\ReserveCS {`⟨*csname*⟩`}` reserves ⟨*csname*⟩ for the current package's use. The variable is also initialized with the `\InitCS` or `\InitCS` * as appropriate.

\ReserveName
\ReserveName*

\ReserveName and \ReserveName* are the same but take an argument ⟨*csname*⟩ without a backslash.

\SaveCS
\RestoreCS

\SaveCS {⟨*csname*⟩} saves the present value of ⟨*csname*⟩ in a macro (\MDSaved⟨*csname*⟩). The saved value is restored to ⟨*csname*⟩ by \RestoreCS {⟨*csname*⟩}.

\SaveName
\RestoreName

\SaveName and \RestoreName are the same but take an argument ⟨*csname*⟩ without a backslash.

\requirecommand
\requirecommand*

\requirecommand takes arguments like \newcommand and behaves like \providecommand (defined in the kernel) with the following difference: if the control sequence is already defined, \requirecommand calls \CheckCommand to make sure that the new and existing definitions are identical, whereas \providecommand assumes that if the control sequence is already defined, the existing definition is appropriate. \requirecommand, like \defcommand, *guarantees* that a control sequence will have the given definition, but \requirecommand also warns you if there was a previous and different existing definition.

\newtokens
\newlet
\newboolean

\newtokens {⟨*csname*⟩}, \newlet {⟨*csname*⟩}{⟨*csname*⟩}, and \newboolean {⟨*csname*⟩} give an error if their control sequence argument is already defined. \newtokens creates a token variable. \newlet does a \let assignment. \newboolean {⟨*csname*⟩} creates three new control sequences: two switches, \csnametrue and \csnamefalse, and a test, \ifcsname. \newtokens is *not* outer. Is there any reason this really matters?

Warning: *Limitation: You can't use* \newlet *to* \let *a command sequence to a character with a catcode not equal to 10 (space), 11 (letter), 12 (other), or 13 (active). For example, you can't say* \newlet\foo#*. Also, you cannot use* = *with* \newlet *like you can with* \let*.*

\providetokens
\provideboolean
\providesavebox
\providecounter
\providelength

Like the kernel's \providecommand, the commands \providetokens {⟨*csname*⟩}, \provideboolean {⟨*csname*⟩}, \providesavebox {⟨*csname*⟩}, \providecounter {⟨*csname*⟩}, and \providelength {⟨*csname*⟩} will create a new object (or objects) based on the name ⟨*csname*⟩ or ⟨*csname*⟩ only if they are not already defined. See the corresponding commands that begin with \new instead of \provide for a description of what kind of object is created. In contrast with \providecommand, however, these commands will write a record to the log file if their argument was already defined (\providecommand does nothing at all in this case).

\UndefineCS
\UndefineName

\UndefineCS {⟨*csname*⟩} causes ⟨*csname*⟩ to be undefined. \UndefineName does the same for a ⟨*csname*⟩. Use with caution. \global works before them.

\defcommand
\defcommand*

\defcommand {⟨*csname*⟩}[⟨*# of args*⟩][⟨*default for an optarg*⟩] defines ⟨*csname*⟩ in the same manner as \newcommand except no warning or error is issued if ⟨*csname*⟩ is already defined.

\defcommand is very similar to the primitive \def, so why would you want to use it? For one thing, the syntax is the same as all the other LaTeX defining commands, so it is easier to read, and easier to change the word defcommand to one of the other defining commands. Second, \defcommands that take arguments have simpler syntax when defining commands are nested. You still have to double the # characters in the definition body, but the argument specification (e.g., [n]) is the same as if not nested.

There is a performance-syntax tradeoff; I choose to use \defcommand whenever the command to be defined is taking an argument. When it does not take an argument, there is no difference between \def and \defcommand except that \def is faster.

I see very little reason to ever use `\renewcommand`. It causes an error when the control sequence is *not* already defined. Conceivably this is useful during development to catch programming mistakes, but much more often I find that I don't care whether the control sequence was defined or not, and therefore the error `\renewcommand` might raise is inappropriate and a problem.

`\NewName` `\NewName*` `\NewName {⟨csname⟩}{⟨template⟩}{⟨body⟩}` defines ⟨*csname*⟩ to expand to ⟨*body*⟩ using a TEX-style argument ⟨*template*⟩, e.g. `#1#2\@nil` or simply `#1#2`. If ⟨*csname*⟩ is already defined, an error will be signalled.

`\DefName` `\DefName*` `\DefName` is like `\NewName` but no error is signalled if ⟨*csname*⟩ is already defined.

`\Global` If the command `\Global` immediately precedes `\NewName`, `\DefName`, or `\ToggleBoolean`, then the definition will be global.

To do: *Get something like* `\Global` *going for all the new commands, not just* `\DefName` *and* `\NewName`*.*

`\CheckName` `\CheckName*` `\CheckName` is like `\NewName` but instead of defining the control sequence, it checks whether the control sequence has the given definition. If so, no action is taken; if not, a warning is given.

`\RequireName` `\RequireName*` `\RequireName` is to `\requirecommand` as `\NewName` is to `\newcommand`. The syntax is `\RequireName {⟨csname⟩}{⟨template⟩}{⟨body⟩}`.

`\NewTextFontCommand` `\NewRobustCommand` `\NewTextFontCommand` and `\NewRobustCommand` are just like the kernel's `\DeclareTextFontCommand` and `\DeclareRobustCommand`, but they signal an error instead of just a warning if their first argument is already defined.

`\Elet` `\EElet` `\Elet` expands the second token after it once and then `\let`s the first token to the second token. `\global` works before it. `\EElet` expands the two tokens that come after it once each, and then `\let`s the first to the second. `\global` works before it.

`\NewUserInfo` `\NewUserInfo*` `\NewUserInfo [⟨user-cmd⟩]{⟨variable⟩}`, where ⟨*variable*⟩ has some capital letters, will define the lowercase version of ⟨*variable*⟩ to be a user command that redefines ⟨*variable*⟩ to its argument. The argument ⟨*user-cmd*⟩, if supplied, is used for the user command, overriding the default of the lowercased ⟨*variable*⟩.

For example, `\NewUserInfo*\Subtitle` defines a user command `\subtitle {`⟨*text*⟩`}` which does the equivalent of `\defcommand\Subtitle{`⟨*text*⟩`}`.

`\NewUserInfo` uses `\ReserveCS` to initialize ⟨*variable*⟩; `\NewUserInfo` ** uses `\ReserveCS` *.

These have `@` in their names because they are modelled after kernel commands.

To do: *Sort out naming comventions and write them down.*

`\addto@macro` `\lg@addto@macro` `\addto@macro{`⟨`\`*csname*⟩`}{`⟨*tokens*⟩`}` adds ⟨*tokens*⟩ to the end of `{`⟨`\`*csname*⟩`}`. The redefinition of `{`⟨`\`*csname*⟩`}` is local. The kernel provides the global equivalent, `\g@addto@macro`. `\lg@addto@macro` is both `long` and global.

Warning: *These commands won't work with a* ⟨`\`*csname*⟩ *that takes arguments.*

To do: *P*

robably it would not be too hard to handle that case. Here is how you would do it by hand for one example:

```
% something like: \def\@chapter[#1]#2{...}

\typeout{\meaning\@chapter}

\renewcommand\addto@macro [2] {%
```

```
  \sc@toks@a=\expandafter{#1[##1]{##2}#2}%
  \edef#1[##1]##2{%
    \the\sc@toks@a
  }%
}

\def\doodie#1{bobo \textsc{#1}}
\tracingonline1

\Debug2
\addto@macro\@chapter {\doodie blorful}
\Debug0

\typeout{\meaning\@chapter}
```

\prependto@macro
\g@prependto@macro
\lg@prependto@macro

`\prependto@macro{`⟨*csname*⟩`}{`⟨*tokens*⟩`}` adds ⟨*tokens*⟩ to the beginning of `{`⟨*csname*⟩`}`. The redefinition of `{`⟨*csname*⟩`}` is local. The global equivalent is `\g@prependto@macro`. `\lg@prependto@macro` is both `long` and global.

4 Controlling expansion

\EExpand...\In
\EExpand*...\In
\eExpand...\In
\eExpand*...\In
\eExecute
\eExecute*

A common construction is to `\edef` a scratch variable to something and then execute the scratch variable. The `\eExecute` macro takes a single argument, expands it fully, then executes it.

`\eExpand {`⟨*first tokens*⟩`}\In {`⟨*second tokens*⟩`}` expands the ⟨*first tokens*⟩ inside ⟨*second tokens*⟩ wherever `#1` occurs. `\EExpand` expands the first token of ⟨*first tokens*⟩ only once. These commands can nest.

For example,

```
\def\a {\b}
\def\b {Hello }
\def\x {d}
\eExpand\a worl\x\In {%
  \def\c {This is a good way to avoid lots of noexpands and
          expandafters.  #1.  And I continue.}%
  \def\x {boogaloo}% \x is already expanded in the def of \c
  \c
}

\EExpand\a BLOOB\x\In {%
  \def\x{avoid }
  \edef#1{\b world}% BLOOB\x is syntactic sugar
  This is a good way to \x lots of noexpands and
  expandafters.  \a BLOOB\x.  And I continue.%
}
```

LOOKS LIKE:

This is a good way to avoid lots of noexpands and expandafters. Hello world. And I continue.
This is a good way to avoid lots of noexpands and expandafters. Hello world. And I continue.

The two commands expand to the same three sentences. Here is one more example, showing (again) how `\EExpand` expands only the first token of its argument only once::

```
\def\x{XXX}
\def\a{AAA\x}
\def\b{BBB}
\EExpand\a\b\In{%
  \def\x{xxx}
  \def\a{aaa}
  \def\b{YYY}
  #1
}
```

LOOKS LIKE:

AAAxxxYYY

`\E@car...\@nil` `\E@cdr...\@nil`

~~~~~~Let T be the sequence of tokens between `\E@car` and `\@nil`. The first token of T is expanded once, and `\E@car...\@nil` expands to the first token of the result.

`\E@cdr...\@nil` is similar, but expands to the entire result *except* its first token.

For example, after

```
\def\a {Hello}
\def\b { world}
```

`\E@car \a there\b.\@nil` would expand first of all to `H`. And `\E@cdr \a there\b.\@nil` would expand first of all to `ellothere\b.`, and then eventually expand fully to `ellothere world.`.

The example is more complicated than you would normally use. Usually you want to `car` and `cdr` a sequence of tokens contained in macro `\foo`, and this is easy enough with `\E@car\foo\@nil`. To chop off the first token of `\foo`, `\edef\foo {\E@cdr\foo\@nil}`. (If you're wondering, the space after `\foo` is irrelevant.)

5 Gobbling

`\Gobble` `\GobbleM` `\GobbleO` `\GobbleMM` `\GobbleMO` `\GobbleOM`

It occurs fairly often that you want to gobble things while `\makeatother` is in effect, so these command names have no `@`'s. The `M` stands for a mandatory argument, and the `O` stands for an optional argument. For example, suppose there is a command `\foo[`⟨*optarg*⟩`]{`⟨*marg*⟩`}`. If you `\let\foo\GobbleOM`, then the arguments to `\foo` will be gobbled appropriately.

`\Gobble` is the same as `\GobbleM`, in imitation of the internal `\@gobble`.

6 Option declaration

The follwing two commands may be used in packages before the `\ProcessOptions` command is issued.

`\DeclareBooleanOptions`

`\DeclareBooleanOptions {`⟨*on*⟩`}{`⟨*off*⟩`}` declares a new boolean variable `@`⟨*on*⟩`@` and makes it `true` if the option ⟨*on*⟩ is given to the package, and `false`
~~~~~~

if the option ⟨*off*⟩ is given, or if neither is given. I think it is good programming style not to rely on the default, always declaring either ⟨*on*⟩ or ⟨*off*⟩ with an `\ExecuteOptions` statement.

`\DeclareBooleanUserOptions` `\DeclareBooleanUserOptions{`⟨*on*⟩`}{`⟨*off*⟩`}` is like `\DeclareBooleanOptions`, but additionally declares two user commands ⟨*on*⟩ and ⟨*off*⟩ which are `\let` to `\@`⟨*on*⟩`@true` and `\@`⟨*off*⟩`@false`, respectively. Use this command when it is sensible to change the status of the option after the package has been loaded.

7 Toggle a boolean

`\ToggleBoolean` `\ToggleBoolean{`⟨*boolean*⟩`}` changes the state of ⟨*boolean*⟩ from `false` to `true` or vice versa. The argument ⟨*boolean*⟩ should not include an initial `if` or final `true` or `false`. The redefinition is local unless `\Global` precedes `\ToggleBoolean`.

8 Debugging

`\VerboseErrors` LaTeX by default gives very little context for errors. `\VerboseErrors [`⟨*number*⟩`]` causes LaTeX to give ⟨*number*⟩ lines of context, or the maximum by default.

`\GVerboseErrors` Like `\VerboseErrors` but effective globally.

`\Debug` `\Debug {`⟨*number*⟩`}` sets a debugging parameter to ⟨*number*⟩. I have plans to turn this into a bitwise parameter like many C programs, but right now the behavior is to issue a message with `\typeout`, call `\VerboseErrors`, and use the parameter to assign values to `\tracingoutput`, `\tracingpages`, `\tracingmacros`, and `\tracingcommands`.

`\GDebug` `\GDebug {`⟨*number*⟩`}` is as `\Debug` but its assignments are `\global`.

`\DTypeout` `\DTypeout` expands to `\typeout` when `\Debug` is 1 or greater, and `\GobbleM` otherwise. `\DDTypeout` is `\GobbleM` unless `\Debug` is 2 or greater; `\DDDTypeout` is `\GobbleM` unless `\Debug` is 3 or greater.

`\DDTypeout`

`\DDDTypeout`

`\DGobbleM` Like `\GobbleM` but when `\Debug` is 1 or greater, tells you what it's gobbling with a `\typeout`.

`\FrankenError` The commands `\FrankenError`, `\FrankenWarning`, and `\FrankenInfo` are defined here for use by other `Frankenstein` packages and classes. They are simply wrappers for the obvious kernel commands (i.e., substitute "Generic" for "Franken").

`\FrankenWarning`

`\FrankenInfo`

Part II
Implementation

9 Version control

\fileinfo
\DoXUsepackagE
\HaveECitationS
\fileversion
\filedate
\docdate
\PPOptArg

These definitions must be the first ones in the file.

```
\def\fileinfo{more defining commands (Frankenstein's brain)}
\def\DoXPackageS {}
\def\fileversion{v1.8}
\def\filedate{2001/08/31}
\def\docdate{2001/08/31}
\edef\PPOptArg {%
  \filedate\space \fileversion\space \fileinfo
}
```

If we're loading this file from a `\ProcessDTXFile` command (see the *compsci* package), then `\JusTLoaDInformatioN` will be defined; othewise we assume it is not (that's why the FunkY NamE).

If we're loading from `\ProcessDTXFile`, we want to load the packages listed in `\DoXPackageS` (needed to typeset the documentation for this file) and then bail out. Otherwise, we're using this file in a normal way as a package, so do nothing. `\DoXPackageS`, if there are any, are declared in the `dtx` file, and, if you're reading the typeset documentation of this package, would appear just above. (It's OK to call `\usepackage` with an empty argument or `\relax`, by the way.)

```
\makeatletter% A special comment to help create bst files.  Don't change!
\@ifundefined{JusTLoaDInformatioN} {%
  }{% ELSE (we know the compsci package is already loaded, too)
  \UndefineCS\JusTLoaDInformatioN
  \SaveDoXVarS
  \eExpand\csname DoXPackageS\endcsname\In {%use \csname in case it's undefined
    \usepackage{#1}%
  }%
  \RestoreDoXVarS
  \makeatother
  \endinput
}% A special comment to help create bst files.  Don't change!
```

Now we check for LaTeX2e and declare the LaTeX package.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{moredefs}[\PPOptArg]
```

10 Conditionals

We start with the conditionals section because we want to use `\@ifundefined@cs` in this package to make some of the subsequent definitions easier to read.

\@ifundefined@cs

```
\newcommand*\@ifundefined@cs [1] {%
  \edef\reserved@a{%
    \expandafter\@gobble\string #1%
```

```
  }%
  \@ifundefined\reserved@a
      \@firstoftwo
    \@secondoftwo
}
```

\IfElement...\In

```
\newcommand\IfElement{}
\def\IfElement #1\In#2{%
  \@tempswafalse
  \expandafter   \@tfor
    \expandafter \sc@t@a
    \expandafter :%
    \expandafter =#2\do {%
      \ifx #1\sc@t@a
%        \DTypeout{[\meaning #1] matches element [\meaning\sc@t@a]
%                  in [\string#2].}%
        \@tempswatrue
        \@break@tfor
      \else
%        \DTypeout{[\meaning #1] matches NO elements in [\string #2].}%
      \fi
    }%
  \if@tempswa
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
```

11 Defining commands

\sc@star@or@long
\sc@star@nothing

The macros `\sc@star@or@long` and `\sc@star@nothing` are parallel to the kernel's `\@star@or@long` and `\l@ngrel@x`, which control whether definitions are `long` or not. `\sc@star@or@long` causes the value of `\sc@star@nothing` to be either `*` or empty, depending on whether it finds a `*` when it is called. It also sets the kernel's `\l@ngrel@x` to nothing or `\long`, respectively. (We need both flags at least once.)

```
\newcommand*\sc@star@nothing{}
\newcommand*\sc@star@or@long [1] {% args: defining-command
  \@ifstar {%
      \let\l@ngrel@x\relax
      \def\sc@star@nothing {*}%
      #1%
    }{% ELSE
    \let\l@ngrel@x\long
    \def\sc@star@nothing {}%
    #1%
  }%
}
```

\md@check@star

Looks for a star with `\@ifstar` and sets `\sc@star@nothing` to `*` if there is a star and `\ShortEmpty` if not.

```
\newcommand\md@check@star {%
  \@ifstar {%
      \def\sc@star@nothing {*}%
    }{% ELSE
    \let\sc@star@nothing \ShortEmpty
  }%
}
```

\requirecommand
\requirecommand*
\require@command

A typical application of the star mechanisms is \requirecommand.

```
\newcommand\requirecommand {%
  \sc@star@or@long\require@command
}
\newcommand\require@command [1] {% args: \csname
  \@ifundefined@cs{#1} {%
      \expandafter\newcommand\sc@star@nothing
    }{% ELSE
    \expandafter\CheckCommand\sc@star@nothing
  }%
  {#1}%
}
```

\InitCS
\InitCS*
\InitName
\InitName*
\ReserveCS
\ReserveCS*
\ReserveName
\ReserveName*
\ShortEmpty
\LongEmpty

```
\newcommand\InitCS {%
  \@star@or@long\Init@CS
}
\newcommand\Init@CS [1] {% args: \csname
  \l@ngrel@x\def#1{}%
}
\newcommand\InitName {%
  \sc@star@or@long\Init@Name
}
\newcommand\Init@Name [1] {% args: csname
  \expandafter\DefName\sc@star@nothing{#1}{}{}%
}
\newcommand\ReserveCS {%
  \sc@star@or@long\Reserve@CS
}
\newcommand\Reserve@CS [1] {% args: \csname
  \expandafter\newcommand\sc@star@nothing{#1} {}%
}
\newcommand\ReserveName {%
  \sc@star@or@long\Reserve@Name
}
\newcommand\Reserve@Name [1] {% args: csname
  \expandafter\NewName\sc@star@nothing{#1}{} {}%
}
\InitCS*\ShortEmpty
\InitCS\LongEmpty
```

\sc@t@a
\sc@t@b
\sc@t@c
\sc@t@d
\sc@t@e
\sc@t@f
\sc@t@g

Scratch variables.

```
\ReserveCS\sc@t@a
\ReserveCS\sc@t@b
\ReserveCS\sc@t@c
\ReserveCS\sc@t@d
```

```
\ReserveCS\sc@t@e
\ReserveCS\sc@t@f
\ReserveCS\sc@t@g
```

\newtokens
\newlet

Because \newtoks is \outer, we have to fool \def into allowing it to be in its argument by using \@nameuse.

```
\newcommand\newtokens [1] {% args: \csname
  \@ifdefinable #1 {%
    \@nameuse{newtoks}#1%
  }%
}
\newcommand*\newlet [2] {% args: \csname-a \csname-b
  \@ifdefinable #1 {%
    \let #1#2%
  }%
}
```

\providetokens
\providelength
\providesavebox
\providecounter
\newboolean
\provideboolean

The \newboolean command is the same as the one in the *ifthen* package; so that package won't clash with this one. Isn't \requirecommand nice?

```
\newcommand*\providetokens [1] {% args: \csname
  \@ifundefined@cs{#1} {%
      \@nameuse{newtokens}#1%
    }{% ELSE
    \FrankenInfo{moredefs}{\protect\providetokens\space is not reallocating
                           token variable \protect#1.\MessageBreak
                           The existing contents are [\the#1]}%
  }%
}
\newcommand*\providelength [1] {% args: \csname
  \@ifundefined@cs{#1} {%
      \newlength{#1}%
    }{% ELSE
    \FrankenInfo{moredefs}{\protect\providelength\space is not reallocating
                           \protect#1.\MessageBreak
                           The existing value is [\the#1]}%
  }%
}
\newcommand*\providesavebox [1] {% args: \csname
  \@ifundefined@cs{#1} {%
      \newsavebox{#1}%
    }{% ELSE
    \FrankenInfo{moredefs}{\protect\providesavebox\space is not reallocating
                           box \protect#1.}%
  }%
}
\newcommand*\providecounter [1] {% args: string
  \@ifundefined{c@#1} {%
      \newcounter{#1}%
    }{% ELSE
    \FrankenInfo{moredefs}{\protect\providecounter\space is not reallocating
                           counter #1.\MessageBreak
                           The existing value is [\expandafter\number\csname c@#1\endcsname]}%
  }%
}
```

The following definition follows the one in the *ifthen* package:

```
% \ProvidesPackage{ifthen}
%          [1999/01/07 v1.1a Standard LaTeX ifthen package (DPC)]

\requirecommand*\newboolean [1] {% args: string
  \expandafter
    \@ifdefinable\csname if#1\endcsname {%
      \expandafter\newif\csname if#1\endcsname
    }%
}

% old def of \cs\newboolean I had before 15 Feb 00:
%  \csname newif\expandafter\endcsname\csname if#1\endcsname
```

Notice that `\defcommand` is not defined yet.

If the *ifthen* package is loaded *either* before or after this package, the `\provideboolean` command will be the one defined in *ifthen*. Otherwise, it will be the one defined here.

There are two minor differences between this definition and the one in the *ifthen* package: (1) my command will barf on undefined but "undefinable" commands, e.g., ones that begin with `\end`, which LaTeX reserves; (2) my command writes an informational message to the log file when the boolean variable is already defined. I'm not sure how useful the informational message is, but the first difference should I think also be in the *ifthen* package, so

To do: *I'm putting it on my list to write the LaTeX team requesting this change.*

```
\@ifpackageloaded{ifthen} {%
  }{% ELSE
  \requirecommand*\provideboolean [1] {% args: string
    \@ifundefined {if#1} {%
        \newboolean{#1}%
      }{% ELSE
      \FrankenInfo{moredefs}{\protect\provideboolean\space is not reallocating
                             \protect#1.\MessageBreak
                             The value is [\@nameuse{if#1}TRUE\else FALSE\fi]}%
    }%
  }%
}
```

The following definition is what's in the *ifthen* package, for reference.

```
% \requirecommand*\provideboolean [1] {% args: string
%   \@ifundefined{if#1}{%
%     \expandafter
%       \newif\csname if#1\endcsname}\relax
% }
```

\sc@toks@a
\sc@toks@b
\addto@macro
\lg@addto@macro
\prependto@macro
\g@prependto@macro
\lg@prependto@macro

There are still missing a couple of the permutations, but I won't add them until I need them. You can add them yourself in the configuration file `moredefs.cfg`.

```
\newtokens\sc@toks@a
\newtokens\sc@toks@b

\newcommand\addto@macro [2] {%
```

```
  \sc@toks@a=\expandafter{#1#2}%
  \edef#1{%
    \the\sc@toks@a
  }%
}
\newcommand\lg@addto@macro [2] {%
  \sc@toks@a=\expandafter{#1#2}%
  \long\xdef#1{%
    \the\sc@toks@a
  }%
}
\newcommand\prependto@macro [2] {%
  \sc@toks@a={#2}%
  \sc@toks@b=\expandafter{#1}%
  \edef#1{%
    \the\sc@toks@a\the\sc@toks@b
  }%
}
\newcommand\g@prependto@macro [2] {%
  \sc@toks@a={#2}%
  \sc@toks@b=\expandafter{#1}%
  \xdef#1{%
    \the\sc@toks@a\the\sc@toks@b
  }%
}
\newcommand\lg@prependto@macro [2] {%
  \sc@toks@a={#2}%
  \sc@toks@b=\expandafter{#1}%
  \long\xdef#1{%
    \the\sc@toks@a\the\sc@toks@b
  }%
}
```

\UndefineCS
\UndefineName

\global works before them.

```
\newcommand\UndefineCS [1] {% args: \csname
  \let#1\@undefined
}
\newcommand\UndefineName [1] {% args: csname
  \expandafter\let\csname#1\endcsname\@undefined
}
```

\defcommand
\defcommand*
\def@command

See the user documentation for a discussion of when to use this insteae of \def.

```
\newcommand\defcommand {%
  \@star@or@long\def@command
}
\newcommand\def@command {%
  \let\@ifdefinable\@rc@ifdefinable
  \new@command
}
```

\DefName
\DefName*
\def@name
\NewName
\NewName*
\new@name
\Global
\sc@global

\Global works before \DefName, \NewName, and \ToggleBoolean only!

```
\newcommand\DefName {%
  \@star@or@long\def@name
}
```

```
\newcommand\def@name [3] {% args: arglist csname body
  \sc@global\l@ngrel@x\@namedef{#1}#2{#3}%
}
\newcommand\NewName {%
  \@star@or@long\new@name
}
\newcommand\new@name [3] {% args: arglist csname body
  \@ifundefined{#1} {%
      \sc@global\l@ngrel@x\@namedef{#1}#2{#3}%
    }{% ELSE
    \defcommand\reserved@a {%
      #1%
    }%
    \@notdefinable
  }%
}
\newcommand\sc@global {%
  \relax
}
\newcommand\Global {%
  \def\sc@global {%
    \global\let\sc@global\relax\global
  }%
}
```

\CheckName
\CheckName*
\check@name
\RequireName
\RequireName*
\require@name

```
\newcommand\CheckName {%
  \@star@or@long\check@name
}
\newcommand\check@name [3] {% args: arglist csname body
  \expandafter\DefName\sc@star@nothing{reserved@a}{#2}{#3}%
  \expandafter\@check@eq\csname #1\endcsname\reserved@a
}
\newcommand\RequireName {%
  \sc@star@or@long\require@name
}
\newcommand\require@name [3] {% args: arglist csname body
  \@ifundefined{#1} {%
      \expandafter\DefName\sc@star@nothing{#1}{#2}{#3}%
    }{% ELSE
    \expandafter   \expandafter
      \expandafter \CheckName
      \expandafter \sc@star@nothing
                   \csname #1\endcsname
      {#2}{#3}%
  }%
}
```

\NewTextFontCommand
\NewRobustCommand
\new@robustcommand
\new@@robustcommand

```
\newcommand\NewTextFontCommand [2] {% args: \csname font-command
  \NewRobustCommand#1[1]{%
    \ifmmode
      \nfss@text{#2##1}%
    \else
```

```
      \leavevmode
      {\text@command{##1}%
       #2\check@icl ##1\check@icr
       \expandafter}%
    \fi
  }%
}
\newcommand\NewRobustCommand {%
  \@star@or@long\new@robustcommand
}
```

We need a second level here because otherwise the `\fi` that closes `\@ifdefinable` will become the definition of the closing `\new@command`. We could use a chain of `\expandafter`s but that would be confusing.

```
\newcommand\new@robustcommand [1] {%
  \let\sc@t@a\relax
  \@ifdefinable #1 {%
    \def\sc@t@a {%
      \new@@robustcommand #1%
    }%
  }%
  \sc@t@a
}
\newcommand\new@@robustcommand [1] {%
  \edef\reserved@a {\string#1}%
  \def\reserved@b {#1}%
  \edef\reserved@b {%
    \expandafter\strip@prefix\meaning\reserved@b
  }%
  \edef#1{%
    \ifx\reserved@a\reserved@b
      \noexpand\x@protect
      \noexpand#1%
    \fi
    \noexpand\protect
    \expandafter\noexpand\csname
      \expandafter\@gobble\string#1 \endcsname
  }%
 \let\@ifdefinable\@rc@ifdefinable
 \expandafter\new@command\csname
      \expandafter\@gobble\string#1 \endcsname
}
```

`\Elet`

```
\newcommand\Elet {%
  \expandafter\let\expandafter
}
```

`\EElet`

```
\newcommand*\EElet {%
  \expandafter\expandafter\expandafter\let\expandafter\expandafter
}
```

`\NewUserInfo`
`\NewUserInfo*`
`\new@userinfo`

Using `\lowercase` in this macro is tricky, since it gets expanded only in TEX's stomach.

```
\newcommand\NewUserInfo {%
  \sc@star@or@long\new@userinfo
}
\newcommand*\new@userinfo [2][] {% args: [\csname] \csname
  \expandafter\ReserveCS\sc@star@nothing{#2}%
  \def\sc@t@b {#1}%
```

If we were not given the optional user-cmd, define scratch `b` to be a lowercase version of the variable, without the backslash. Otherwise use the user-cmd given, without the backslash.

```
  \ifx\sc@t@b\ShortEmpty
    \edef\sc@t@a {%
      \edef\noexpand\sc@t@b{%
        \E@cdr\string#2\@nil
      }%
    }%
    \lowercase\expandafter{\sc@t@a}%
  \else
    \edef\sc@t@b {\E@cdr\string#1\@nil}%
  \fi
```

Now define the user-cmd to be a redefinition of the variable.

```
  \edef\sc@t@a {%
    \noexpand\NewName\sc@star@nothing{\sc@t@b}{####1}
      {\noexpand\renewcommand\sc@star@nothing\noexpand#2{####1}}
  }%
  \sc@t@a
}
```

\SaveCS
\RestoreCS
\SaveName
\RestoreName

```
\newcommand\SaveCS [1] {% args: \csname
  \expandafter\newlet\csname MDSaved\E@cdr\string#1\@nil\endcsname#1%
}
\newcommand\RestoreCS [1] {% args: \csname
  \Elet#1\csname MDSaved\E@cdr\string#1\@nil\endcsname
  \UndefineName{MDSaved\E@cdr\string#1\@nil}%
}
\newcommand\SaveName [1] {% args: csname
  \ReserveName{MDSaved#1}%
  \EElet\csname MDSaved#1\endcsname
        \csname #1\endcsname
}
\newcommand\RestoreName [1] {% args: csname
  \EElet\csname #1\endcsname
        \csname MDSaved#1\endcsname
  \UndefineName{MDSaved#1}%
}
```

12 Controlling expansion

\EExpand...\In
\EExpand*...\In
\sc@EExpand
\eExpand...\In
\eExpand*...\In
\sc@eExpand
\eExecute
\eExecute*
\sc@eExecute

Uses \sc@t@a, \sc@t@b, \sc@t@c.

```
\newcommand\eExpand {%
  \sc@star@or@long\sc@eExpand
```

```
}
\NewName{sc@eExpand} {#1\In#2} {% args: object body
  \l@ngrel@x\edef\sc@t@a{#1}%
  \expandafter\defcommand\sc@star@nothing\sc@t@b [1] {#2}%
  \expandafter   \sc@t@b
    \expandafter {\sc@t@a}%
}
\newcommand\EExpand {%
  \sc@star@or@long\sc@EExpand
}
```

When this is `short`, both the two args are `short`. `\sc@star@nothing` gets reset by the first `\defcommand`, so we save it in `\sc@t@c`.

```
\NewName{sc@EExpand}{#1\In#2} {% args: object body
  \let\sc@t@c\sc@star@nothing
  \expandafter   \expandafter
    \expandafter \defcommand
    \expandafter \sc@t@c
    \expandafter \sc@t@a
    \expandafter {#1}%
  \expandafter\defcommand\sc@t@c\sc@t@b [1] {#2}%
  \expandafter\sc@t@b
    \expandafter{\sc@t@a}%
}
\newcommand\eExecute {%
  \sc@star@or@long\sc@eExecute
}
\newcommand\sc@eExecute [1] {% args: body
  \l@ngrel@x\edef\sc@t@a {#1}%
  \sc@t@a
}
```

`\E@car`
`\E@cdr`

```
\NewName{E@cdr} {#1\@nil} {%
  \expandafter\@cdr #1\@nil
}
\NewName{E@car} {#1\@nil} {%
  \expandafter\@car #1\@nil
}
```

13 Gobbling

`\Gobble` `\GobbleM` `\GobbleO` `\GobbleMM` `\GobbleMO` `\GobbleOM` `\sc@gobbleO` `\sc@gobbleOM`

M for mandatory arg, i.e., one token. O for optional arg, i.e., a square-brace pair.

```
\newlet\Gobble\@gobble
\newlet\GobbleM\@gobble
\newcommand\GobbleO {%
  \@ifnextchar [
      \sc@gobbleO
    \relax
}
\newlet\GobbleMM\@gobbletwo
\newcommand\GobbleOM {%
  \@ifnextchar [
```

```
      \sc@gobbleOM
    \Gobble
}
\newcommand\GobbleMO [1] {%
  \@ifnextchar [
     \sc@gobbleO
    \relax
}
\NewName{sc@gobbleOM} {[#1]#2}
  {}
\NewName{sc@gobbleO} {[#1]}
  {}
```

14 Option declaration

\DeclareBooleanOptions
\DeclareBooleanUserOptions

```
\newcommand\DeclareBooleanOptions [2] {% args: on off
  \newboolean{@#1@}%
  \DeclareOption{#1} {%
    \@nameuse{@#1@true}
  }%
  \DeclareOption{#2} {%
    \@nameuse{@#1@false}
  }%
}
\newcommand\DeclareBooleanUserOptions [2] {% args: on off
  \DeclareBooleanOptions{#1}{#2}%
  \ReserveName{#1}%
  \ReserveName{#2}%
  \EElet \csname#1\endcsname\csname @#1@true\endcsname
  \EElet \csname#2\endcsname\csname @#1@false\endcsname
}
```

15 Toggle a boolean

\ToggleBoolean

```
\newcommand\ToggleBoolean [1] {% arg: boolean
  \csname if#1\endcsname
    \sc@global\csname #1false\endcsname
  \else
    \sc@global\csname #1true\endcsname
  \fi
}
```

16 Debugging

\VerboseErrors
\GVerboseErrors

We do not use \setcounter but rather set these counters locally.

```
\newcommand*\VerboseErrors [1][\@M] {% args: [number]
  \c@errorcontextlines #1%
  \showboxbreadth #1%
```

```
  \showboxdepth #1%
}
\newcommand*\GVerboseErrors [1][\@M] {% args: [number]
  \global\c@errorcontextlines #1%
  \global\showboxbreadth #1%
  \global\showboxdepth #1%
}
```

\Debug
\GDebug
\md@maybe@global

Set \debug to 0, 1, or 2.

```
\ReserveCS\md@maybe@global
\newcommand*\Debug {%
  \let\md@maybe@global\relax
  \md@debug
}
\newcommand*\GDebug {%
  \let\md@maybe@global\global
  \md@debug
}
\newcommand*\md@debug [1] {% args: debug-level
  \ifnum #1 > 0%
    \let\sc@t@a\@M
    \md@maybe@global\def\DTypeout ##1{%
      \typeout{##1}%
    }%
    \md@maybe@global\def\DGobbleM ##1{%
      \typeout{DGobbleM: [##1]}%
    }%
    \ifnum #1 > 1%
      \md@maybe@global\def\DDTypeout ##1{%
        \typeout{##1}%
      }%
      \ifnum #1 > 2%
        \md@maybe@global\def\DDDTypeout ##1{%
          \typeout{##1}%
        }%
      \fi
    \fi
  \else
    \let\sc@t@a\m@ne
    \md@maybe@global\let\DTypeout\GobbleM
    \md@maybe@global\let\DDTypeout\GobbleM
    \md@maybe@global\let\DDDTypeout\GobbleM
    \md@maybe@global\let\DGobbleM\GobbleM
  \fi
  \md@maybe@global\tracingoutput#1 %
  \md@maybe@global\tracingpages#1 %
  \md@maybe@global\tracingmacros#1 %
  \md@maybe@global\tracingcommands#1 %
  \ifx\md@maybe@global\relax
    \VerboseErrors[\sc@t@a]%
    \typeout{++++ Debugging [#1]\on@line}%
  \else
    \GVerboseErrors[\sc@t@a]%
    \typeout{++++ Global debugging [#1]\on@line}%
```

```
  \fi
}
```

\DTypeout
\DTypeout
\DTypeout
\DGobbleM

When the debugging parameter is not set, these commands gobble their argument.

To do: *Streamline dox about "debugging parameter"; should be something checkable, no?*

```
\newlet\DTypeout\GobbleM
\newlet\DDTypeout\GobbleM
\newlet\DDDTypeout\GobbleM
\newlet\DGobbleM\GobbleM
```

\FrankenError
\FrankenWarning
\FrankenInfo

```
\newcommand\FrankenWarning [2] {% args: package warning
  \GenericWarning  % continuation message
    {(#1)\@spaces\@spaces\@spaces\@spaces\@spaces}
    {Frankenstein (#1) WARNING: #2}%
}
\newcommand\FrankenError [3] {% args: package error-message help-text
  \GenericError % args: continuation message where-help what-help
    {(#1)\@spaces\@spaces\@spaces\@spaces\@spaces}
    {Frankenstein (#1) error: #2}
    {See the documenation for the #1 package for more information.}
    {#3}%
}
\newcommand\FrankenInfo [2] {% args: package info
  \GenericInfo    % continuation message
    {(#1)\@spaces\@spaces\@spaces\@spaces\@spaces}
    {Frankenstein (#1) says: #2}%
}
```

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.

Symbols

\@M 457, 462, 478
\@break@tfor 42
\@car 410
\@cdr 407
\@check@eq 272
\@firstoftwo 28, 48
\@gobble 25, 325, 329, 412, 413
\@gobbletwo 419
\@ifdefinable 117, 122, 167, 237, 305, 327
\@ifnextchar 415, 421, 426
\@ifpackageloaded . 174
\@ifstar 55, 66
\@ifundefined 10, 27, 153, 177, 187, 250, 278
\@ifundefined@cs *2*, 23, 76, 127, 136, 145
\@namedef 244, 251
\@nameuse 118, 128, 182, 437, 440
\@nil 346, 351, 360, 363, 364, 406, 407, 409, 410
\@notdefinable 256
\@rc@ifdefinable 237, 327
\@secondoftwo . . . 29, 50
\@spaces . . 520, 525, 532
\@star@or@long 84, 234, 241, 247, 268, 301
\@tempswafalse 33
\@tempswatrue 41
\@tfor 34
\@undefined . . . 228, 231

A

\addto@macro *4*, 191

C

\c@errorcontextlines 458, 463
\check@icl 295
\check@icr 295
\check@name 267
\CheckCommand 79
\CheckName *4*, 267
\CheckName* *4*, 267
\cs 172
\csname 14, 75, 86, 98, 116, 121, 126, 135, 144, 158, 167, 168, 173, 189, 227, 231, 272, 284, 288, 324, 328, 340, 359, 360, 362, 363, 368, 369, 372, 373, 447, 448, 451, 452, 454

D

\DDDTypeout *7*, 490, 499, 516
\DDTypeout *7*, 486, 498, 515
\Debug *7*, 467
\DeclareBooleanOptions *6*, 434
\DeclareBooleanUserOptions *7*, 434
\DeclareOption 436, 439
\def 1–5, 32, 57, 61, 67, 87, 263, 306, 314, 342, 479, 482, 486, 490
\def@command 233
\def@name 240
\defcommand . . *3*, 233, 253, 381, 391, 395
\defcommand* *3*, 233
\DefName *4*, 93, 240, 271, 279
\DefName* *4*, 240
\DGobbleM *7*, 482, 500, 514
\do 37
\docdate 1
\DoXPackageS 2
\DoXUsepackagE 1
\DTypeout *7*, 39, 44, 479, 497, 514

E

\E@car 406
\E@car...\@nil␣ *6*
\E@cdr 346, 351, 360, 363, 364, 406
\E@cdr...\@nil␣ *6*
\edef 6, 24, 196, 209, 313, 315, 318, 344, 345, 351, 353, 380, 403
\EElet *4*, 334, 368, 372, 447, 448
\eExecute *5*, 376
\eExecute* *5*, 376
\EExpand 385
\eExpand 14, 376
\EExpand*...\In . *5*, 376
\eExpand*...\In . *5*, 376
\EExpand...\In . . *5*, 376
\eExpand...\In . . *5*, 376
\Elet *4*, 331, 363
\else 43, 49, 182, 292, 350, 453, 495, 509
\endcsname 14, 158, 167, 168, 173, 189, 231, 272, 284, 325, 329, 360, 363, 368, 369, 372, 373, 447, 448, 451, 452, 454
\endinput 19
\expandafter . . . 25, 34–37, 48, 50, 77, 79, 93, 99, 105, 158, 166, 168, 173, 188, 195, 201, 208, 215, 222, 231, 271, 272, 279, 281–283, 296, 316, 324, 325, 328, 329, 332, 335, 341, 349, 360, 381–383, 390–397, 407, 410

F

\fi . 45, 51, 182, 297, 322, 352, 455, 493, 494, 501, 512
\filedate 1
\fileinfo 1
\fileversion 1
\FrankenError . . . *7*, 518
\FrankenInfo *7*, 130, 139, 148, 156, 180, 518
\FrankenWarning . *7*, 518

G

\g@prependto@macro *5*, 191
\GDebug *7*, 467
\GenericError 524
\GenericInfo 531
\GenericWarning . . . 519
\Global *4*, 240
\global 264, 463–465, 473
\Gobble *6*, 412
\GobbleM *6*, 412, 497–500, 514–517
\GobbleMM *6*, 412
\GobbleMO *6*, 412
\GobbleO *6*, 412
\GobbleOM *6*, 412
\GVerboseErrors *7*, 457, 510

H

\HaveECitationS 1

I

\if@tempswa 47
\IfElement 31, 32
\IfElement...\In . *2*, 31
\ifmmode 290
\ifnum 477, 485, 489
\ifx . . . 38, 319, 343, 506
\In 14
\Init@CS 84, 86
\Init@Name 90, 92
\InitCS *2*, 83
\InitCS* *2*, 83
\InitName *2*, 83
\InitName* *2*, 83

J

\JusTLoaDInformatioN 12

L

\l@ngrel@x 56, 60, 87, 244, 251, 380, 403
\leavevmode 293
\let 56, 60, 69, 123, 228, 231, 237, 264, 304, 327, 332, 335, 389, 469, 473, 478, 496–500
\lg@addto@macro . *4*, 191
\lg@prependto@macro *5*, 191
\long 60, 202, 223
\LongEmpty *2*, 83
\lowercase 349

M

\m@ne 496
\makeatletter 9
\makeatother 18
\md@check@star 65
\md@debug . 470, 474, 476
\md@maybe@global . . 467
\meaning . . . 39, 44, 316
\MessageBreak 131, 140, 157, 181

N

\NeedsTeXFormat . . . 21
\new@@robustcommand 288
\new@command . . 238, 328
\new@name 240
\new@robustcommand . 288
\new@userinfo 337
\newboolean . *3*, 126, 435
\newcommand 23, 31, 53, 54, 65, 72, 75, 77, 83, 86, 89, 92, 95, 98, 99, 101, 104, 116, 121, 126, 135, 144, 152, 194, 200, 206, 213, 220, 227, 230, 233, 236, 240, 243, 246, 249, 259, 262, 267, 270, 274, 277, 288, 300, 303, 312, 331, 334, 337, 340, 359, 362, 366, 371, 376, 385, 399, 402, 414, 420, 425, 434, 443, 450, 457, 462, 468, 472, 476, 518, 523, 530
\newcounter 154
\newif 168, 189
\newlength 137
\newlet *3*, 116, 360, 412, 413, 419, 514–517
\NewName *4*, 105, 240, 354, 379, 388, 406, 409, 430, 432
\NewName* *4*, 240
\NewRobustCommand *4*, 288
\newsavebox 146
\NewTextFontCommand *4*, 288
\newtokens *3*, 116, 191, 192
\NewUserInfo *4*, 337
\NewUserInfo* . . . *4*, 337
\nfss@text 291
\noexpand 320, 321, 323, 324, 345, 354, 355
\number 158

O

\on@line 508, 511

P

\PPOptArg 1, 22
\prependto@macro *5*, 191
\protect 130, 131, 139, 140, 148, 149, 156, 180, 181, 323
\provideboolean . *3*, 126
\providecounter . *3*, 126
\providelength . . *3*, 126
\providesavebox . *3*, 126
\ProvidesPackage 22, 162
\providetokens . . *3*, 126

R

\relax 56, 189, 260, 264, 304, 417, 428, 469, 506
\renewcommand 355
\require@command . . 72
\require@name 267
\requirecommand . . . *3*, 72, 165, 176, 186

\requirecommand* . *3*, 72
\RequireName *4*, 267
\RequireName* ... *4*, 267
\Reserve@CS 96, 98
\Reserve@Name . 102, 104
\ReserveCS .. *2*, 83, 109–115, 341, 467
\ReserveCS* *2*, 83
\reserved@a .. 24, 27, 253, 272, 313, 319
\reserved@b 314–316, 319
\ReserveName *3*, 83, 367, 445, 446
\ReserveName* *3*, 83
\RestoreCS *3*, 359
\RestoreDoXVarS ... 17
\RestoreName *3*, 359

S

\SaveCS *3*, 359
\SaveDoXVarS 13
\SaveName *3*, 359
\sc@eExecute 376
\sc@EExpand 376
\sc@eExpand 376
\sc@global 240, 452, 454
\sc@gobble0 412
\sc@gobbleOM 412
\sc@star@nothing 53, 67, 69, 77, 79, 93, 99, 105, 271, 279, 283, 341, 354, 355, 381, 389
\sc@star@or@long 53, 73, 90, 96, 102, 275, 338, 377, 386, 400
\sc@t@a 35, 38, 39, 109, 304, 306, 310, 344, 349, 353, 357, 380, 383, 393, 397, 403, 404, 478, 496, 507, 510
\sc@t@b 109, 342, 343, 345, 351, 354, 381, 382, 395, 396
\sc@t@c 109, 389, 392, 395
\sc@t@d 109
\sc@t@e 109
\sc@t@f 109
\sc@t@g 109
\sc@toks@a 191
\sc@toks@b 191
\ShortEmpty *2*, 69, 83, 343
\showboxbreadth 459, 464
\showboxdepth . 460, 465
\space 7, 130, 139, 148, 156, 180
\string 25, 40, 44, 313, 325, 329, 346, 351, 360, 363, 364
\strip@prefix 316

T

\text@command 294
\the .. 132, 141, 197, 203, 210, 217, 224
\ToggleBoolean .. *7*, 450
\tracingcommands .. 505
\tracingmacros 504
\tracingoutput 502
\tracingpages 503
\typeout .. 480, 483, 487, 491, 508, 511

U

\UndefineCS .. *3*, 12, 227
\UndefineName *3*, 227, 364, 374
\usepackage 15

V

\VerboseErrors *7*, 457, 507

X

\x@protect 320
\xdef 202, 216, 223