List of internal LaTeX2e Macros useful to Package Authors

Compiled by Martin Scharrer
martin@scharrer-online.de

Version 0.4a – 2017/12/09

Abstract

This document lists the internal macros defined by the LaTeX2e base files which can be also useful to package authors. The macros are hyper-linked to their description in *source2e*. For this to work both PDFs must be inside the same directory.

This document is not yet complete in content and format and may miss some macros.

Contents

1 Constants

1.1 Number Constants

Some of the following integer values are defined using `\countdef` (for −1), `\chardef` (for values between 0–255) and `\mathchardef` (> 255). They are robust and do not expand in an `\edef` context. When used on the right side of an assignment the act the same way as a count register. The `\m@ne` is a real count register and can still be modified, but doing so would certainly break various code. In the terms defined by *The TeXbook*, this integer constants yield *internal integers* rather than *integer denotations*.
The other numbers are defined as normal macros, which will simply expand to the containing number. They were defined to be used to set font sizes, which explains the non-integer numbers. Please note that if they are used for an assignment or other numeric context (e.g. `\ifnum`) TeX will keep expanding the following tokens until a non-numeric token is found.

Macro	Value	Defined using
`\@ne`	1	chardef
`\tw@`	2	chardef
`\thr@@`	3	chardef
`\sixt@@n`	16	chardef
`\@xxxii`	32	chardef
`\@cclv`	255	chardef
`\@cclvi`	256	mathchardef
`\@m`	1000	mathchardef
`\@M`	10000	mathchardef
`\@Mi`	10001	mathchardef
`\@Mii`	10002	mathchardef
`\@Miii`	10003	mathchardef
`\@Miv`	10004	mathchardef
`\@MM`	20000	mathchardef

Macro	Value	Defined using
`\@vpt`	5	def
`\@vipt`	6	def
`\@viipt`	7	def
`\@viiipt`	8	def
`\@ixpt`	9	def
`\@xpt`	10	def
`\@xipt`	10.95	def
`\@xiipt`	12	def
`\@xivpt`	14.4	def
`\@xviipt`	17.28	def
`\@xxpt`	20.74	def
`\@xxvpt`	24.88	def
`\m@ne`	−1	countdef

1.2 Dimension Constants

The following dimension and skip constants are defined using registers. They must not be changed.

Macro	Value	Defined using	Notes
`\p@`	1pt	newdimen	Can be used as a replacement of pt behind a number due to the resulting multiplication
`\z@`	0pt	newdimen	Can be used both for 0pt and 0
`\maxdimen`	16383.99999pt	newdimen	Largest valid dimension

Macro	Value	Defined using	Notes
`\z@skip`	0pt plust0pt minus0pt	newskip	
`\hideskip`	−1000pt plus 1fill	newskip	negative but can grow
`\@flushglue`	0pt plus 1fil	newskip	

1.3 String Constants

The following macros hold common strings and are fully expandable. Some special characters are defined here as their verbatim ASCII representation. This makes them useful if such characters are to be written into an auxiliary file.

Macro	Value	Note
\space	One space	An explicit space (catcode 10 "space").
\@spaces	Four spaces	Contains 4×\space.
\empty	Empty string	Commonly used to define empty macros using \let.
\@empty	Empty string	Same as above. Used by the LaTeX kernel commands.
\@backslashchar	"\"	Backslash with catcode 12 (other), i.e. simple ASCII backlash usable in e.g. \write.
\@percentchar	"%"	Percent character with catcode 12 (other), i.e. simple ASCII percent usable in e.g. \write.
\@charlb	"{"	Left brace with catcode 11 (letter).
\@charrb	"}"	Right brace with catcode 11 (letter).
\@clsextension	"cls"	Used for comparison with \@currext.
\@pkgextension	"sty"	Used for comparison with \@currext.
\@depth	"depth"	Used for box size declarations (\hb@xt@).
\@height	"height"	Used for box size declarations.
\@width	"width"	Used for box size declarations.
\@minus	"minus"	Used for skip declarations.
\@plus	"plus"	Used for skip declarations.

1.4 Token Constants

The following macros are defined using \let⟨*macro*⟩=⟨*token*⟩ and therefore equal to this token. They are useful for \ifx comparisons with tokens read by \let or \futurelet.

Name	Token	Catcode	Note
\@sptoken	Space	10	Should not be confused with \space
\bgroup	{	1	Begin of group
\egroup	}	2	End of group

1.5 Other

Macro	Value	Defined using	Notes
\voidb@x	(void)	newbox	permanently void box register
\@undefined	(undef)	(undefined)	This macro is not defined. It is used to test if other macros are undefined (using \ifx) or set them to an undefined state (using \let).

2 Variables

2.1 Temporary Variables

The following variables are defined and used by the LaTeX kernel commands as scratch registers and macros. They can be used with care, but should only be redefined inside a local group to avoid interference with other code. Care must also be taken if they are used together with external macros, which might use them as well – maybe not yet but in the next release.

Temp variable	Type	Note
\count@	counter	
\@tempcnta	counter	
\@tempcntb	counter	
\dimen@	dimension	
\dimen@i	dimension	Marked as "global only"
\dimen@ii	dimension	
\@tempdima	dimension	
\@tempdimb	dimension	
\@tempdimc	dimension	
\@tempa	macro	
\@tempb	macro	
\@tempc	macro	
\@gtempa	macro	For temporary definitions which must be made global
\skip@	skip	
\@tempskipa	skip	
\@tempskipb	skip	
\toks@	token register	
\@temptokena	token register	
\if@tempswa	if switch	Comes with the usual setters \@tempswatrue and \@tempswafalse
\@tempboxa	box register	
\@let@token	'let'	Used by \@ifnextchar to temporary store the next token using \futurelet. Can be used for similar purposes.

2.2 Dimension Variables

Macro	Description
\@wholewidth	This dimen register hold the full line width (thickness) in picture environments.[=*
\@halfwidth	This dimen register holds the half line width (thickness) in picture environments.[=*

2.3 String and Other Variables

Macro	Description
\@currext	Extension of the current package or class file, empty outside.
\@currname	File name base of the current package or class file, empty outside.
\@currenvir	Name of the current environment.
\@currenvline	Line number of the begin of the current environment
\@currentlabel	The value a \label will point to. Set by \stepcounter and \refstepcounter.

3 Macros

3.1 Macro Definition

\@namedef{⟨*name*⟩}⟨*parameter list*⟩{⟨*definition*⟩}

Defines macro \⟨*name*⟩ using \def. Can be prefixed with \long and \global.

\@nameuse{⟨*name*⟩}

Expands to macro \⟨*name*⟩.

\@ifnextchar⟨*token*⟩{⟨*yes*⟩}{⟨*no*⟩}

Tests if next non-space token is equal to ⟨*token*⟩.

\@ifstar{⟨*yes*⟩}{⟨*no*⟩}

Tests of next non-space token is '*'. Removes star for ⟨*yes*⟩ branch.

\@dblarg{⟨*cmd*⟩}{⟨*arg*⟩}

Expands to ⟨*cmd*⟩[⟨*arg*⟩]{⟨*arg*⟩}.

\@ifundefined{⟨*name*⟩}{⟨*yes*⟩}{⟨*no*⟩}

Tests if \⟨*name*⟩ is defined (and not equal to \relax).

\@ifdefinable\⟨*name*⟩{⟨*yes*⟩}

Tests if \⟨*name*⟩ is undefined, ⟨*name*⟩ not 'relax' and doesn't start with 'end', and if \end⟨*name*⟩ is not defined.

\@onlypreamble⟨*macro*⟩

The given ⟨*macro*⟩ is marked as only be valid in the preamble. It will be redefined as an error message AtBeginDocument.

`\@star@or@long`

Tests for a following '*', if found `\l@ngrel@x` will be let to `\relax`, but `\long` otherwise. It can the be used before `\def` as an potential prefix.

`\@testopt{`⟨*1*⟩`}{`⟨*2*⟩`}`

Short for `\@ifnextchar[{`⟨*1*⟩`}{`⟨*1*⟩`[{`⟨*2*⟩`}]}`, i.e. ⟨*2*⟩ will be used as default argument if non was given.

`\@protected@testopt`

Robust version of `\@testopt`?

`\addto@hook`⟨*token register*⟩`{`⟨*code*⟩`}`

Appends code to the token register.

`\g@addto@macro`⟨*macro*⟩`{`⟨*code*⟩`}`

Appends ⟨*code*⟩ to the definition of ⟨*macro*⟩.

3.2 Expanding/Gobbling Arguments

Macro	Description
`\@gobble{`⟨*arg*⟩`}`	Removes (gobbles) its argument. (long)
`\@gobbletwo{`⟨*arg 1*⟩`}{`⟨*arg 2*⟩`}`	Removes (gobbles) two arguments. (long)
`\@gobblefour{`⟨*1*⟩`}{`⟨*2*⟩`}{`⟨*3*⟩`}{`⟨*4*⟩`}`	Removes (gobbles) four arguments. (long)
`\@gobblecr`	Gobbles a following carriage return, ignores spaces otherwise
`\@firstofone{`⟨*arg*⟩`}`	Expands to ⟨*arg*⟩, i.e. is used to remove braces. (long)
`\@iden{`⟨*arg*⟩`}`	Identity. Same as `\@firstofone` for compatibility reasons. (long)
`\@firstoftwo{`⟨*1*⟩`}{`⟨*2*⟩`}`	Expands to ⟨*1*⟩, discards ⟨*2*⟩. (long)
`\@secondoftwo{`⟨*1*⟩`}{`⟨*2*⟩`}`	Expands to ⟨*2*⟩, discards ⟨*1*⟩. (long)
`\@thirdofthree{`⟨*1*⟩`}{`⟨*2*⟩`}{`⟨*3*⟩`}`	Expands to ⟨*3*⟩, discards ⟨*1*⟩ and ⟨*2*⟩. (long)
`\@expandtwoargs\macro{`⟨*1*⟩`}{`⟨*2*⟩`}`	Expands the two arguments using `\edef` and feeds it to `\macro`.

3.3 Lists

`\in@{`⟨*1*⟩`}{`⟨*2*⟩`}`

Checks if first argument occurs in the second and sets the switch `\ifin@` accordantly. The arguments are not expanded. This must be done beforehand.

`\@removeelement{`⟨*element*⟩`}{`⟨*list*⟩`}{`⟨*cs*⟩`}`

Removes an element from a comma-separated list and puts it into a control sequence.

3.4 Loops

Macro	Description
`\loop ... \iterate ... \repeat`	
`\@whilenum`⟨*test*⟩`\do{`⟨*body*⟩`}`	While loop with `\ifnum` test.
`\@whiledim`⟨*test*⟩`\do{`⟨*body*⟩`}`	While loop with `\ifdim` test.
`\@whilesw`⟨*switch*⟩`\fi{`⟨*body*⟩`}`	While loop with ⟨*switch*⟩ test.
`\@for`⟨*macro*⟩`:=`⟨*list*⟩`\do{`⟨*body*⟩`}`	For loop. The ⟨*list*⟩ is supposed to expand to a comma separated list. Defines ⟨*macro*⟩ to each element of the list and executes ⟨*body*⟩ each time. Supports an empty lists without errors.
`\@tfor`⟨*macro*⟩`:=`⟨*list*⟩`\do{`⟨*body*⟩`}`	For loop. The ⟨*list*⟩ is not expanded and taken as a list of tokens or `{...}`. Defines ⟨*macro*⟩ to each element of the list and executes ⟨*body*⟩ each time. Supports an empty lists without errors.
`\@break@tfor`	Break out of a `\@tfor` loop. This should be called *inside* the scope of an `\fi`.

NOTES:

1. These macros use no `\@temp` sequences.
2. These macros do not work if the body contains anything that looks syntactically to TeX like an improperly balanced `\if \else \fi`.

3.5 Colors

The following macros are defined equal to `\relax` by LaTeX2e as placeholder for the real code added by the `color` or `xcolor` package. They are needed to handle colors inside saved boxes correctly. See the definitions of `\sbox` for an example.

Macro	Definition given by `color`
`\color@begingroup`	`\begingroup`
`\color@endgroup`	`\endgroup`
`\color@setgroup`	`\begingroup\set@color`
`\color@hbox`	`\hbox\bgroup\color@begingroup`
`\color@vbox`	`\vbox\bgroup\color@begingroup`
`\color@endbox`	`\color@endgroup\egroup`

3.6 Auxiliary Macros

This auxiliary macros were originally defined to handle font changes but can be used for other code as well.

Macro	Description
`\ifnot@nil{`⟨*1*⟩`}{`⟨*2*⟩`}`	Checks if ⟨*1*⟩ is the token `\@nil`. If so gobbles ⟨*2*⟩ otherwise uses `\@firstofone` to remove the braces around ⟨*2*⟩.
`\@nil`	This macro is undefined on purpose and used as endmarker in loops and other macros which process token lists.
`\@nnil`	Definition contains only `\@nil` and is used beside others to test for the presents of `\@nil` in `\ifnot@nil`. Is also used as endmarker.
`\remove@to@nnil`	Removes everything behind it until and including `\@nnil`.
`\remove@star`	Removes everything behind it until and including *.
`\zap@space`⟨*text*⟩␣`\@empty`	Removes all spaces from ⟨*text*⟩. Expandable.[=*
`\strip@prefix`	Removes everything up to and including to the next >.

3.7 Catcodes

Name	Description
`\@makeother{`⟨*letter*⟩`}`	Changes the catcode of the letter to 'other' (12). Special letters must be escaped with a backslash.
`\@sanitize`	Changes catcodes of everything except braces to 'other' (12).
`\@onelevel@sanitize`⟨*macro*⟩	Sanitizes ⟨*macro*⟩, turns it definition into verbatim code. Resulting characters except spaces are in catcode 'other' (12)! Uses `\meaning` and `\strip@prefix`. (With ϵ-TEX many applications can be replaced by `\detokenize{`⟨*content*⟩`}`.)

3.8 Messages

`\MessageBreak`

Inside a message this macro create a new line followed by a continuation text. Outside it is equal to `\relax`.

`\GenericInfo{`⟨*continuation*⟩`}{`⟨*message*⟩`}`

Prints ⟨*message*⟩ to a log file. An included `\MessageBreak` will cause a new line which start with ⟨*continuation*⟩.

`\GenericError{`⟨*continuation*⟩`}{`⟨*error message*⟩`}`
`{`⟨*where to go for further information*⟩`}{`⟨*help text*⟩`}`

Print error message to log file followed by the 'further information' line. The help text is displayed if the user presses '`h`'.

`\wlog{`⟨*log message*⟩`}`

Write on log file only.

3.9 Dimensions, Length and Skips

Name	Description
`\@settopoint`⟨*register*⟩	Rounds register to whole number of points.
`\rem@pt`⟨*dimension value*⟩	Awaits a value dimension value (⟨*int*⟩`.`⟨*frac*⟩`pt`) as string where the "`pt`" is removed. If ⟨*frac*⟩ is numerical equal to 0, then it and the decimal dot are removed as well.
`\strip@pt`⟨*dimension*⟩	Expands dimension using `\the` and strips the "`pt`" using `\rem@pt`.
`\@killglue`	Removes (`\unskip`s) the last and then all further skips (`\lastskip`) till one with a size of zero is reached.

`\@defaultunits`

Used to provide a default unit for `dimen` or `skip` assignment.
Usage: `\@defaultunits\dimen=#1pt\relax\@nnil`. Other units can be used instead of '`pt`'.

3.10 Class and Package Options

Macro	Description
`\@classoptionslist`	List of options of the main class.
`\@unusedoptionlist`	List of options to the main class that haven't been declared.
`\@declaredoptions`	Comma separated list of the options declared by the current package or class. The list is in the order in which the options were declared.

3.11 Files

Macro	Description
\if@filesw	If false the package should not be produce or write to output files. Set to false by \nofiles.
\if@partsw	
\@currdir	Holds the current directory, e.g. "./" in an Unix OS.
\input@path	List of input paths. Each path should be enclosed in braces with no delimiters between paths.
\@filelist	The comma separated list of all files read so far. Only active if \listfiles is used in the preamble.
\@inputcheck	Input file handle to check for the existence of the file.
\@unused	Output file handle used to reserve the standard output. Used in \typeout to write to the terminal.
\@mainaux	Output file handle for the main aux file.
\@partaux	Output file handle for include file aux files.
\@auxout	Let to \@partaux inside include files, but to \@mainaux otherwise.
\@partlist	Holds the comma-separated list defined by \includeonly.
\@pushfilename	pushes file name, extension and current catcode of "@" onto the file stack.
\@popfilename	pushes file name, extension and current catcode of "@" onto the file stack.
\@currnamestack	file name stack.

\filename@parse{⟨*filename*⟩}

Parses ⟨*filename*⟩ and provides its directory, name base and extension in \filename@area, \filename@base and \filename@ext. The latter is let to \relax if it does not exists.

\@filef@und

The macro \IfFileExists(which is used by \InputIfFileExists{)} stores the found file followed by a space in this macro.

\@starttoc{⟨*ext*⟩}

Reads the file with the given extension (\jobname.⟨*ext*⟩) and opens it for writing afterwards. The file is initially empty. Creates the output file handle \tf@⟨*ext*⟩.

\@writefile{⟨*ext*⟩}{⟨*code*⟩}

Writes code using the output handle \tf@⟨*ext*⟩ if it exists.

\@iffileonpath{⟨*filename*⟩}

Check if given file is found by TEX directly or in any of the directories given by \input@path.

`\@obsoletefile{`⟨*new*⟩`}{`⟨*obsolete*⟩`}`

Prints warning message (only) that now a different file is used a input.

`\@addtofilelist{`⟨*filename*⟩`}`

Adds the given filename to the list of files. Only active if `\listfiles` is used in the preamble.

`\@ptionlist{`⟨*filename*⟩`}`

Expands the option list of package, class or file given by full filename.

`\@ifpackageloaded{`⟨*package*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\@ifclassloaded{`⟨*class*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\@ifl@aded{`⟨*extension*⟩`}{`⟨*file base*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Tests if given package/class/file has been loaded.

`\@ifpackagewith{`⟨*name*⟩`}{`⟨*option-list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\@ifclasswith{`⟨*name*⟩`}{`⟨*option-list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\@if@ptions{`⟨*extension*⟩`}{`⟨*name*⟩`}{`⟨*option-list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Tests if given package/class/file has been loaded with the given options.

`\@ifpackagelater{`⟨*name*⟩`}{`⟨*date YYYY/MM/DD*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\@ifclasslater{`⟨*name*⟩`}{`⟨*date YYYY/MM/DD*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\@ifl@ter{`⟨*extension*⟩`}{`⟨*file base*⟩`}{`⟨*date YYYY/MM/DD*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Tests if given package/class/file has been loaded with a version more recent than ⟨*version*⟩.

3.12 Saved plainTEX primitives

The following plainTEX macros are redefined by LATEX and therefore saved away first:

LATEX Macro	plainTEX original	Description/Note
`\@@par`	`\par`	Some LATEX environments redefine `\par` locally
`\@@input`	`\input`	Syntax: `\input` ⟨*filename*⟩
`\@@end`	`\end`	
`\@@underline`	`\underline`	
`\frozen@everymath`	`\everymath`	
`\frozen@everydisplay`	`\everydisplay`	

3.13 Fonts

Macro	Description
\f@encoding	Holds the current font encoding.
\f@family	Holds the current font family.
\f@series	Holds the current font series.
\f@shape	Holds the current font shape.
\f@size	Holds the current font size (in pt but without unit).
\f@baselineskip	Holds the current baseline skip (in pt but without unit).
\f@linespread	Holds the current internal value for \baselinestretch.
\@currsize	Let to the last font size command (e.g. \small).
\curr@math@size	Holds locally the current math size.
\curr@fontshape	\f@encoding /\f@family /\f@series /\f@shape

3.14 Paragraph

Macro	Description
\@@par	PlainTEX primitive \par.
\@setpar{⟨*val*⟩}	Used to make environment-wide changes to \par. Sets both \par and \@par to ⟨*val*⟩.
\@restorepar	Defines \par to \@par.

3.15 Space Hack

Macro	Description
\@bsphack	
\@esphack	Both of these macro ensure that the code between them does not insert any spaces into the document. The code itself should not produce any text and not change the mode (e.g. start or stop math mode).
\@Esphack	Variant of \@esphack which sets the @ignore switch to true which causes an \ignorespaces after the \end of the environment.
\@vbsphack	Variant of \@bsphack which ensure the invisible material is *not* set in vmode. Not used by LATEX itself at the moment.

3.16 Boxes

Macro	Description
\null	Empty \hbox. Good to fill places which must not be empty.
\strutbox	Box with dimension of \strut, i.e. maximum height and depth of the current font and zero width. Can be used to extract this dimension with \ht\strutbox and \dp\strutbox.
\@arstrutbox	Defined inside array and tabular. Like \strutbox but stretched by \arraystretch.
\@begin@tempboxa⟨*box*⟩{⟨*content*⟩}	Stores ⟨*content*⟩ into \@tempboxa as ⟨*box*⟩ (\hbox or \vbox) and stores its dimension into \width, \height, \depth and \totalheight.
\@end@tempboxa	Ends a \@begin@tempboxa environment.
\hb@xt@	\hbox to
\hmode@bgroup	\leavevmode\bgroup

\@settodim{⟨*dimension cs*⟩}{⟨*length register*⟩}{⟨*content*⟩}

Sets the ⟨*length register*⟩ to the dimension given by the ⟨*dimension*⟩ (\ht, \dp and \wd) of the ⟨*content*⟩. Example: \@settodim{\wd}{\@tempdima}{Hello World} will set \@tempdima to the width of "Hello World".

3.17 Base Conversion

Macro	Description
\hexnumber@{⟨*number*⟩}	Returns a single digit hexadecimal number (0–9, A–F) from given ⟨*number*⟩, which must either be a numeric register or a number ending with a space!
\@alph{⟨*number*⟩}	Expands to lower case letter corresponding to the given number (1=a, 2=b, . . .). Expands to \@ctrerr if number is larger then 26.
\@Alph{⟨*number*⟩}	Expands to upper case letter corresponding to the given number (1=A, 2=B, . . .). Expands to \@ctrerr if number is larger then 26.
\two@digits{⟨*number*⟩}	Returns ⟨*number*⟩ (e.g. a count register) as string and ensures that it has at least two digits by appending a '0' if required.[=*

3.18 Conditionals

Macro	Description
\if@compatibility	Switch to indicate if the LaTeX2.09 compatibility mode is active.
\if@ignore	Whether or not to ignore spaces after an environment. Set to true by \ignorespacesafterend.
\if@minipage	True for a minipage, false for a parbox. Responsible for adding space, skips and paragraph indents for a parbox.
\if@twoside	True for two-sided documents
\if@twocolumn	Indicates if two-column mode is active
\if@firstcolumn	Indicates if the first column is processed