

The L^AT_EX3 Sources

The L^AT_EX3 Project*

2008/08/05

Contents

1	Conventions	1
1.1	Functions	1
1.2	Parameters	2
2	Modules	3
2.1	Using the modules	4
3	Starters	4
4	Setting up primitive names	5
4.1	Assignments	6
5	Basics	20
5.1	Predicates and conditionals	20
5.1.1	Primitive conditionals	20
5.1.2	Non-primitive conditionals	21
5.2	Selecting and discarding tokens from the input stream	23
5.3	Internal functions	24
5.4	Defining functions	25
5.4.1	Defining new functions	25
5.4.2	Undefining functions	28
5.4.3	Defining internal functions (no checks)	28
5.5	Defining test functions	32

*Frank Mittelbach, Denys Duchier, Chris Rowley, Rainer Schöpf, Johannes Braams, Michael Downes, David Carlisle, Alan Jeffrey, Morten Høgholm, Thomas Lotze, Javier Bezos

5.6	The innards of a function	32
5.7	Grouping and scanning	32
5.8	Engine specific definitions	33
5.9	The Implementation	33
5.9.1	Renaming some \TeX primitives (again)	33
5.9.2	Defining functions	34
5.9.3	Predicate implementation	35
5.9.4	Defining and checking (new) functions	35
5.9.5	More new definitions	39
5.9.6	Further checking	45
5.9.7	Freeing memory	46
5.9.8	Engine specific definitions	46
5.9.9	Selecting tokens	46
5.9.10	Gobbling tokens from input	48
5.9.11	Scratch functions	48
5.9.12	Strings and input stream token lists	48
5.9.13	Predicates and conditionals	49
6	The <code>chk</code> module	49
6.1	Functions	50
6.2	Constants	50
6.3	Internal functions	50
6.4	The Implementation	51
6.4.1	Checking variable assignments	51
6.4.2	Doing some tracing	53
6.4.3	Tracing modules	54
7	Token list pointers	54
7.1	Functions	55
7.2	Predicates and conditionals	57
7.3	Token lists	58
7.3.1	Internal functions	59
7.4	Variables and constants	60
7.4.1	Internal functions	60
7.5	Search and replace	61

7.6	Heads or tails?	61
7.7	The Implementation	62
7.7.1	Checking for and replacing tokens	73
7.7.2	Heads or tails?	76
8	\LaTeX3 functions	78
8.1	Expanding arguments of functions	78
8.2	Defining new variants	80
8.3	Manipulating the first argument	81
8.4	Manipulating two arguments	82
8.5	Manipulating three arguments	83
8.6	Internal functions and variables	83
8.7	The Implementation	84
8.7.1	General expansion	84
8.7.2	Preventing expansion	88
8.7.3	Single token expansion	89
9	Macro Counters	90
9.1	Functions	91
9.2	Formatting a counter value	92
9.3	Variable and constants	93
9.4	Primitive functions	93
9.5	The Implementation	94
9.5.1	Defining constants	97
10	Sequences	98
10.1	Functions	98
10.2	Predicates and conditionals	100
10.3	Internal functions	100
11	Sequence Stacks	101
11.1	Functions	101
11.2	Predicates and conditionals	101
11.3	Implementation	102
11.3.1	Stack operations	105

12 Allocating registers and the like	106
12.1 Functions	107
12.2 The Implementation	107
13 Low-level file i/o	109
13.1 Functions for output streams	109
13.2 Functions for input streams	111
13.3 Constants	112
13.4 Internal functions	112
13.5 The Implementation	112
13.5.1 Output streams	113
13.5.2 Input streams	116
14 Comma lists	117
14.1 Functions	117
14.2 Mapping functions	118
14.3 Predicates and conditionals	120
14.4 Internal functions	120
14.5 Comma list Stacks	121
14.6 The Implementation	121
14.6.1 Stack operations	127
15 Property lists	128
15.1 Functions	128
15.2 Predicates and conditionals	130
15.3 Internal functions	130
15.4 The Implementation	131
16 Integers	136
16.1 Functions	137
16.2 Formatting a counter value	138
16.2.1 Internal functions	138
16.3 Variable and constants	139
16.4 Testing and evaluating integer expressions	139
16.5 Conversion	141
16.6 The Implementation	141
16.6.1 Scanning and conversion	149

17 Length registers	152
17.1 Skip registers	152
17.1.1 Functions	152
17.1.2 Formatting a skip register value	154
17.1.3 Variable and constants	154
17.2 Dim registers	154
17.2.1 Functions	154
17.2.2 Variable and constants	156
17.3 Muskips	156
17.4 The Implementation	157
17.4.1 Skip registers	157
17.4.2 Dimen registers	160
17.4.3 Muskips	162
18 Token Registers	163
18.1 Functions	163
18.2 Predicates and conditionals	165
18.3 Variable and constants	165
18.3.1 Internal functions	165
18.4 The Implementation	165
19 Communicating with the user	170
19.1 Displaying the information	170
19.2 Storing the information	171
19.2.1 Dealing with the error file	171
19.2.2 Declaring an error message in the error file	172
19.3 Internal functions	172
19.4 Kernel specific functions	173
19.5 Variables and constants	173
19.6 The implementation	174
19.6.1 Code to be moved to other modules	174
19.6.2 Variables and constants	175
19.6.3 Displaying the information	176
19.6.4 Dealing with the error file	177
19.6.5 Declaring an error message in the error file	178
19.6.6 Kernel specific functions	179

20 Boxes	183
20.1 Generic functions	183
20.2 Horizontal mode	185
20.3 Vertical mode	186
20.4 The Implementation	187
20.4.1 Generic boxes	187
20.4.2 Vertical boxes	189
20.4.3 Horizontal boxes	190
21 Control sequence functions extended ...	191
21.1 Internal variables	192
21.2 The Implementation	192
22 Quarks	196
22.1 Functions	196
22.2 Recursion	197
22.3 Constants	198
22.4 The Implementation	198
23 Control structures	202
23.1 Choosing modes	202
23.1.1 Alignment safe grouping and scanning	202
23.2 Producing n copies	203
23.3 Conditionals and logical operations	203
23.3.1 The boolean data type	204
23.3.2 Logical operations	205
23.3.3 Case switches	206
23.3.4 Generic loops	206
23.4 Sorting	207
23.5 The Implementation	207
23.5.1 Choosing modes	207
23.5.2 Making n copies	209
23.5.3 Booleans	212
23.5.4 Generic testing	214
23.5.5 Case switch	215
23.5.6 Sorting	216

24 A token of my appreciation...	219
24.1 Character tokens	219
24.2 Generic tokens	221
24.2.1 Useless code: because we can!	225
24.3 Peeking ahead at the next token	225
24.3.1 Internal functions	227
24.4 Implementation	227
24.4.1 Character tokens	227
24.4.2 Generic tokens	229
24.4.3 Peeking ahead at the next token	238
25 Cross references	242
25.1 Implementation	243
26 Infix notation arithmetic	246
26.1 The Implementation	248
26.2 Higher level commands	257
 Change History	 260
 Index	 261

Abstract

This package sets up an experimental naming scheme for L^AT_EX commands. It allows the L^AT_EX programmer to systematically name functions and variables, and specify the argument types of functions.

The T_EX primitives are all given a new name according to these conventions.

Warning: This package, and all packages using it should be regarded as *experimental*!

The names of these packages, and the names and syntax of any commands defined in them might change at any time.

These conventions are being distributed in this form to encourage discussion and experimentation. It is *not* intended that these packages be used in ‘real’ documents at this stage.

1 Conventions

This section gives an overview of the syntax for L^AT_EX commands that is set up for use in these ‘experimental’ packages.

Commands in L^AT_EX3 are either functions or parameters. All primitive commands of T_EX have private names.

1.1 Functions

Functions have the following general syntax:

$$\backslash\langle module \rangle_ \langle description \rangle : \langle arg-spec \rangle$$

where $\langle module \rangle$ is one of the (to be) chosen module names and $\langle description \rangle$ is a verbal description of the functionality. $\langle arg-spec \rangle$ finally describes the type of arguments that the function takes and is left empty if it is a function without arguments.

All three parts consists of letters only $\langle description \rangle$ is allowed to take further `_` characters to separate words is necessary.

Currently there exists some functions which don’t have a proper $\langle module \rangle$ name.

As a semi-formalized concept the letter `g` is sometimes used to prefix the $\langle module \rangle$ name and certain parts of the $\langle description \rangle$ to mark the function as “globally acting”.

The $\langle arg-spec \rangle$ currently supports the following types of arguments:

- n** Unexpanded token (or token-list if in braces) braces.
- o** One time expanded token or token-list. In the latter case, effectively only the first token in the list gets expanded. Since the expansion might result in more than one token, the result is surrounded for further processing with braces.
- x** Fully expanded token or token-list. Like **o** but the argument is expanded using `\def:Npx` before it is passed on.

- c** A character string or a token-list that expands to characters of catcode 11 or 12. This string (after expansion) is used to construct a command name that is eventually passed on.
- N,O,X** Like **n**, **o**, **x** but the argument must be a single token without any braces around it.
- w** One or more arguments with “weird” syntax that one has to know by heart or better leave it alone.
- p** Denotes parameter text specification part, e.g. `#1#2\q_stop#3`.
- T,F** denotes the “true” or the “false” case in a functional predicate.

Especially for the new names of \TeX primitives there are is one more character to denote arguments. It implies that these functions should not be used outside this bootstrapping file.

- D** Zero or more arguments with “weird” syntax. Uppercase “D” means (DON’T USE IT), i.e., that this is a primitive \TeX command that should not show up in code except in the very basic functions of \LaTeX 3 that provide a more sensible interface.

One could perhaps envisage an extended system which allocated letters to denote the various primitive argument types available in \TeX , however it seems that this just complicates the system without adding any real benefit, as these primitives would never be used in production code, as higher level packages should offer a better interface. Thus the following letters, although they were considered have not been used. “D” is used in most cases in preference.

- i** Denotes an integer in \TeX notation (which might be a register or ...).
- d** Denotes a dimension in \TeX notation.
- g** Denotes a glue in \TeX notation.
- m** Denotes an muglue or mukern in \TeX notation.
- b** Denotes a box specification in \TeX notation (again something pretty arbitrary).
- r** Denotes a rule specification in \TeX notation.

Some of the primitive functions below are flagged “D” even if they actually might be useful in average code. So certainly there are some adjustments necessary. It all depends whether or not we provide some safer interface or leave them alone.

1.2 Parameters

Parameter names have the following general syntax:

$$\backslash\langle access\rangle_ \langle module\rangle_ \langle description\rangle_ \langle type\rangle$$

$\langle module \rangle$ and $\langle description \rangle$ is as above. $\langle type \rangle$ should denote the type of parameter if this helps in using it. The currently used types are:

int Integer valued.

factor Another integer value type. Used for things where the parameter is used as a factor for something else.

status The sort of boolean stuff \TeX provides. Essentially an integer with the meaning 0 = ‘off’ and other values may or may not have sensible meanings.

pen Another integer describing penalties.

dem The demerits.

dim A dimension.

skip A glue value.

toks A toks register (sort of).

char An integer denoting a character.

muskip A math unit.

$\langle access \rangle$ describes how the parameter can be accessed. The following characters are possible:

c A constant. Should not be set in the code except with special functions to define the value for the whole processing.

C A constant according to \TeX ’s rules. Can not be changed at all.

l A local variable which therefore should not be changed globally.

L A local variable that is usually set (and/or reset) by \TeX itself.

g A global variable.

G A global variable that is usually set (and/or reset by \TeX .

R A variable that is set (and changed) by \TeX and can not be changed by in the code (read-only).

2 Modules

Nearly all operations of \LaTeX 3 are carried out by calling control sequences. For better programming concepts many types of functions are identified and gathered in modules. Functions in such modules starts with special prefixes, for example $\backslash\text{tlp}_$ is the prefix for functions dealing with token list pointers.

2.1 Using the modules

Most of the modules can be used on top of L^AT_EX and are loaded with the usual `\usepackage` or `\RequirePackage` instructions. As the packages use a coding syntax different from standard L^AT_EX it provides a few functions for setting it up.

<code>\ExplSyntaxOn</code>	
<code>\ExplSyntaxOff</code>	<code>\ExplSyntaxOn <code> \ExplSyntaxOff</code>

Issues a catcode regime where spaces are ignored and colon and underscore are letters.

<code>l3names</code>	<code>\RequirePackage{l3names}</code>
<code>\ProvidesExplPackage</code>	<code>\ProvidesExplPackage {<package>}</code>
<code>\ProvidesExplClass</code>	<code>{<date>} {<version>} {<description>}</code>

The package `l3names` (this module) provides `\ProvidesExplPackage` which is a wrapper for `\ProvidesPackage` and sets up the L^AT_EX3 catcode settings for programming automatically. Similar for the relationship between `\ProvidesExplClass` and `\ProvidesClass`. Spaces are not ignored in the arguments of these commands.

<code>\GetIdInfo</code>	
<code>\filename</code>	
<code>\filenameext</code>	
<code>\filedate</code>	
<code>\fileversion</code>	
<code>\filetimestamp</code>	<code>\RequirePackage{l3names}</code>
<code>\fileauthor</code>	<code>\GetIdInfo \$Id: < cvs or svn info field > \$ {<description>}</code>

Extracts all information from a cvs or svn field. Spaces are not ignored in these fields. The information pieces are stored in separate control sequences with `\filename` for the part of the file name leading up to the period, `\filenameext` for the extension, `\filedate` for date, `\fileversion` for version, `\filetimestamp` for the time and `\fileauthor` for the author.

To summarize: Every single package using this syntax should identify itself using one of the above methods. Special care is taken so that every package or class file loaded with `\RequirePackage` or alike are loaded with usual L^AT_EX catcodes and the L^AT_EX3 catcode scheme is reloaded when needed afterwards. See implementation for details. If you use the `\GetIdInfo` command you can use the information when loading a package with

```
\ProvidesExplPackage{\filename}{\filedate}{\fileversion}{\filedescription}
```

3 Starters

This is the base part of L^AT_EX3 defining things like `catcodes` and redefining the T_EX primitives.

We start by setting up `\catcodes` that we need to define new commands. These are the ones for begin-group and end-group characters.¹

```
1 <*initex>
2 \catcode'\{=1 % left brace is begin-group character
3 \catcode'\}=2 % right brace is end-group character
4 \catcode'\#=6 % hash mark is macro parameter character
5 \catcode'\^=7 %
6 \catcode'\^^I=10 % ascii tab is a blank space
7 </initex>
```

Reason for `\endlinechar=32` is that a line ending with a backslash will be interpreted as the token `_` which seems most natural and since spaces are ignored it works as we intend elsewhere.

```
8 <*initex | package>
9 \catcode126=10\relax % tilde is a space char.
10 \catcode32=9\relax % space is ignored
11 \catcode9=9\relax % tab also ignored
12 \endlinechar=32\relax % endline is space
13 \catcode95=11\relax % underscore letter
14 \catcode58=11\relax % colon letter
```

4 Setting up primitive names

Here is the function that renames `TEX`'s primitives.

Normally the old name is left untouched, but the possibility of undefining the original names is made available by `docstrip` and package options. If nothing else, this gives a way of checking what 'old code' a package depends on...

If the package option 'removeoldnames' is used then some trick code is run after the end of this file, to skip past the code which has been inserted by `LATEX 2ε` to manage the file name stack, this code would break if run once the `TEX` primitives have been undefined. (What a surprise!) **The option has been temporarily disabled.**

To get things started, give a new name for `\let`.

```
15 \let\tex_let:D\let
16 </initex | package>
```

and now an internal function to possibly remove the old name.

```
17 <*initex>
18 \long\def\name_undefine:N#1{
19   \tex_let:D#1\c_undefined}
20 </initex>

21 <*package>
22 \DeclareOption{removeoldnames}{
23   \long\def\name_undefine:N#1{
24     \tex_let:D#1\c_undefined}}
```

¹Well not needed while this file is running as a package on top of `LATEX 2ε`, so omitted from the package code

```

25 \DeclareOption{keepoldnames}{
26   \long\def\name_undefine:N#1{}}

27 \ExecuteOptions{keepoldnames}

28 \ProcessOptions
29 \</package>

```

The internal function to give the new name and possibly undefine the old name.

```

30 \<initex | package>
31 \long\def\name_primitive:NN#1#2{
32   \tex_let:D #2 #1
33   \name_undefine:N #1
34 }

```

4.1 Assignments

In the current incarnation of this package, all T_EX primitives are given a new name of the form `\tex_oldname:D`. But first three special cases which have symbolic original names. These are given modified new names, so that they may be entered without catcode tricks.

```

35 \name_primitive:NN \tex_space:D
36 \name_primitive:NN \tex_italiccor:D
37 \name_primitive:NN \tex_hyphen:D

```

Now all the other primitives.

```

38 \name_primitive:NN \let \tex_let:D
39 \name_primitive:NN \def \tex_def:D
40 \name_primitive:NN \edef \tex_edef:D
41 \name_primitive:NN \gdef \tex_gdef:D
42 \name_primitive:NN \xdef \tex_xdef:D
43 \name_primitive:NN \chardef \tex_chardef:D
44 \name_primitive:NN \countdef \tex_countdef:D
45 \name_primitive:NN \dimendef \tex_dimendef:D
46 \name_primitive:NN \skipdef \tex_skipdef:D
47 \name_primitive:NN \muskipdef \tex_muskipdef:D
48 \name_primitive:NN \mathchardef \tex_mathchardef:D
49 \name_primitive:NN \toksdef \tex_toksdef:D
50 \name_primitive:NN \futurelet \tex_futurelet:D
51 \name_primitive:NN \advance \tex_advance:D
52 \name_primitive:NN \divide \tex_divide:D
53 \name_primitive:NN \multiply \tex_multiply:D
54 \name_primitive:NN \font \tex_font:D
55 \name_primitive:NN \fam \tex_fam:D
56 \name_primitive:NN \global \tex_global:D
57 \name_primitive:NN \long \tex_long:D
58 \name_primitive:NN \outer \tex_outer:D
59 \name_primitive:NN \setlanguage \tex_setlanguage:D
60 \name_primitive:NN \globaldefs \tex_globaldefs:D
61 \name_primitive:NN \afterassignment \tex_afterassignment:D
62 \name_primitive:NN \aftergroup \tex_aftergroup:D

```

63	\name_primitive:NN	\expandafter	\tex_expandafter:D
64	\name_primitive:NN	\noexpand	\tex_noexpand:D
65	\name_primitive:NN	\begingroup	\tex_begingroup:D
66	\name_primitive:NN	\endgroup	\tex_endgroup:D
67	\name_primitive:NN	\halign	\tex_halign:D
68	\name_primitive:NN	\valign	\tex_valign:D
69	\name_primitive:NN	\cr	\tex_cr:D
70	\name_primitive:NN	\crcr	\tex_crcr:D
71	\name_primitive:NN	\noalign	\tex_noalign:D
72	\name_primitive:NN	\omit	\tex_omit:D
73	\name_primitive:NN	\span	\tex_span:D
74	\name_primitive:NN	\tabskip	\tex_tabskip:D
75	\name_primitive:NN	\everycr	\tex_everycr:D
76	\name_primitive:NN	\if	\tex_if:D
77	\name_primitive:NN	\ifcase	\tex_ifcase:D
78	\name_primitive:NN	\ifcat	\tex_ifcat:D
79	\name_primitive:NN	\ifnum	\tex_ifnum:D
80	\name_primitive:NN	\ifodd	\tex_ifodd:D
81	\name_primitive:NN	\ifdim	\tex_ifdim:D
82	\name_primitive:NN	\ifeof	\tex_ifeof:D
83	\name_primitive:NN	\ifhbox	\tex_ifhbox:D
84	\name_primitive:NN	\ifvbox	\tex_ifvbox:D
85	\name_primitive:NN	\ifvoid	\tex_ifvoid:D
86	\name_primitive:NN	\ifx	\tex_ifx:D
87	\name_primitive:NN	\iffalse	\tex_iffalse:D
88	\name_primitive:NN	\iftrue	\tex_iftrue:D
89	\name_primitive:NN	\ifhmode	\tex_ifhmode:D
90	\name_primitive:NN	\ifmmode	\tex_ifmmode:D
91	\name_primitive:NN	\ifvmode	\tex_ifvmode:D
92	\name_primitive:NN	\ifinner	\tex_ifinner:D
93	\name_primitive:NN	\else	\tex_else:D
94	\name_primitive:NN	\fi	\tex_fi:D
95	\name_primitive:NN	\or	\tex_or:D
96	\name_primitive:NN	\immediate	\tex_immediate:D
97	\name_primitive:NN	\closeout	\tex_closeout:D
98	\name_primitive:NN	\openin	\tex_openin:D
99	\name_primitive:NN	\openout	\tex_openout:D
100	\name_primitive:NN	\read	\tex_read:D
101	\name_primitive:NN	\write	\tex_write:D
102	\name_primitive:NN	\closein	\tex_closein:D
103	\name_primitive:NN	\newlinechar	\tex_newlinechar:D
104	\name_primitive:NN	\input	\tex_input:D
105	\name_primitive:NN	\endinput	\tex_endinput:D
106	\name_primitive:NN	\inputlineno	\tex_inputlineno:D
107	\name_primitive:NN	\errmessage	\tex_errmessage:D
108	\name_primitive:NN	\message	\tex_message:D
109	\name_primitive:NN	\show	\tex_show:D
110	\name_primitive:NN	\showthe	\tex_showthe:D
111	\name_primitive:NN	\showbox	\tex_showbox:D
112	\name_primitive:NN	\showlists	\tex_showlists:D
113	\name_primitive:NN	\errhelp	\tex_errhelp:D
114	\name_primitive:NN	\errorcontextlines	\tex_errorcontextlines:D
115	\name_primitive:NN	\tracingcommands	\tex_tracingcommands:D
116	\name_primitive:NN	\tracinglostchars	\tex_tracinglostchars:D

117	\name_primitive:NN	\tracingmacros	\tex_tracingmacros:D
118	\name_primitive:NN	\tracingonline	\tex_tracingonline:D
119	\name_primitive:NN	\tracingoutput	\tex_tracingoutput:D
120	\name_primitive:NN	\tracingpages	\tex_tracingpages:D
121	\name_primitive:NN	\tracingparagraphs	\tex_tracingparagraphs:D
122	\name_primitive:NN	\tracingrestores	\tex_tracingrestores:D
123	\name_primitive:NN	\tracingstats	\tex_tracingstats:D
124	\name_primitive:NN	\pausing	\tex_pausing:D
125	\name_primitive:NN	\showboxbreadth	\tex_showboxbreadth:D
126	\name_primitive:NN	\showboxdepth	\tex_showboxdepth:D
127	\name_primitive:NN	\batchmode	\tex_batchmode:D
128	\name_primitive:NN	\errorstopmode	\tex_errorstopmode:D
129	\name_primitive:NN	\nonstopmode	\tex_nonstopmode:D
130	\name_primitive:NN	\scrollmode	\tex_scrollmode:D
131	\name_primitive:NN	\end	\tex_end:D
132	\name_primitive:NN	\csname	\tex_csname:D
133	\name_primitive:NN	\endcsname	\tex_endcsname:D
134	\name_primitive:NN	\ignorespaces	\tex_ignorespaces:D
135	\name_primitive:NN	\relax	\tex_relax:D
136	\name_primitive:NN	\the	\tex_the:D
137	\name_primitive:NN	\mag	\tex_mag:D
138	\name_primitive:NN	\language	\tex_language:D
139	\name_primitive:NN	\mark	\tex_mark:D
140	\name_primitive:NN	\topmark	\tex_topmark:D
141	\name_primitive:NN	\firstmark	\tex_firstmark:D
142	\name_primitive:NN	\botmark	\tex_botmark:D
143	\name_primitive:NN	\splitfirstmark	\tex_splitfirstmark:D
144	\name_primitive:NN	\splitbotmark	\tex_splitbotmark:D
145	\name_primitive:NN	\fontname	\tex_fontname:D
146	\name_primitive:NN	\escapechar	\tex_escapechar:D
147	\name_primitive:NN	\endlinechar	\tex_endlinechar:D
148	\name_primitive:NN	\mathchoice	\tex_mathchoice:D
149	\name_primitive:NN	\delimiter	\tex_delimiter:D
150	\name_primitive:NN	\mathaccent	\tex_mathaccent:D
151	\name_primitive:NN	\mathchar	\tex_mathchar:D
152	\name_primitive:NN	\mskip	\tex_mskip:D
153	\name_primitive:NN	\radical	\tex_radical:D
154	\name_primitive:NN	\vcenter	\tex_vcenter:D
155	\name_primitive:NN	\mkern	\tex_mkern:D
156	\name_primitive:NN	\above	\tex_above:D
157	\name_primitive:NN	\abovewithdelims	\tex_abovewithdelims:D
158	\name_primitive:NN	\atop	\tex_atop:D
159	\name_primitive:NN	\atopwithdelims	\tex_atopwithdelims:D
160	\name_primitive:NN	\over	\tex_over:D
161	\name_primitive:NN	\overwithdelims	\tex_overwithdelims:D
162	\name_primitive:NN	\displaystyle	\tex_displaystyle:D
163	\name_primitive:NN	\textstyle	\tex_textstyle:D
164	\name_primitive:NN	\scriptstyle	\tex_scriptstyle:D
165	\name_primitive:NN	\scriptscriptstyle	\tex_scriptscriptstyle:D
166	\name_primitive:NN	\nonscript	\tex_nonscript:D
167	\name_primitive:NN	\eqno	\tex_eqno:D
168	\name_primitive:NN	\leqno	\tex_leqno:D
169	\name_primitive:NN	\abovedisplayshortskip	\tex_abovedisplayshortskip:D
170	\name_primitive:NN	\abovedisplayskip	\tex_abovedisplayskip:D

171	\name_primitive:NN	\belowdisplaysshortskip	\tex_belowdisplaysshortskip:D
172	\name_primitive:NN	\belowdisplayskip	\tex_belowdisplayskip:D
173	\name_primitive:NN	\displaywidowpenalty	\tex_displaywidowpenalty:D
174	\name_primitive:NN	\displayindent	\tex_displayindent:D
175	\name_primitive:NN	\displaywidth	\tex_displaywidth:D
176	\name_primitive:NN	\everydisplay	\tex_everydisplay:D
177	\name_primitive:NN	\predisplaysize	\tex_predisplaysize:D
178	\name_primitive:NN	\predisplaypenalty	\tex_predisplaypenalty:D
179	\name_primitive:NN	\postdisplaypenalty	\tex_postdisplaypenalty:D
180	\name_primitive:NN	\mathbin	\tex_mathbin:D
181	\name_primitive:NN	\mathclose	\tex_mathclose:D
182	\name_primitive:NN	\mathinner	\tex_mathinner:D
183	\name_primitive:NN	\mathop	\tex_mathop:D
184	\name_primitive:NN	\displaylimits	\tex_displaylimits:D
185	\name_primitive:NN	\limits	\tex_limits:D
186	\name_primitive:NN	\nolimits	\tex_nolimits:D
187	\name_primitive:NN	\mathopen	\tex_mathopen:D
188	\name_primitive:NN	\mathord	\tex_mathord:D
189	\name_primitive:NN	\mathpunct	\tex_mathpunct:D
190	\name_primitive:NN	\mathrel	\tex_mathrel:D
191	\name_primitive:NN	\overline	\tex_overline:D
192	\name_primitive:NN	\underline	\tex_underline:D
193	\name_primitive:NN	\left	\tex_left:D
194	\name_primitive:NN	\right	\tex_right:D
195	\name_primitive:NN	\binoppenalty	\tex_binoppenalty:D
196	\name_primitive:NN	\relpenalty	\tex_relpenalty:D
197	\name_primitive:NN	\delimitershortfall	\tex_delimitershortfall:D
198	\name_primitive:NN	\delimiterfactor	\tex_delimiterfactor:D
199	\name_primitive:NN	\nulldelimiterspace	\tex_nulldelimiterspace:D
200	\name_primitive:NN	\everymath	\tex_everymath:D
201	\name_primitive:NN	\mathsurround	\tex_mathsurround:D
202	\name_primitive:NN	\medmuskip	\tex_medmuskip:D
203	\name_primitive:NN	\thinmuskip	\tex_thinmuskip:D
204	\name_primitive:NN	\thickmuskip	\tex_thickmuskip:D
205	\name_primitive:NN	\scriptspace	\tex_scriptspace:D
206	\name_primitive:NN	\noboundary	\tex_noboundary:D
207	\name_primitive:NN	\accent	\tex_accent:D
208	\name_primitive:NN	\char	\tex_char:D
209	\name_primitive:NN	\discretionary	\tex_discretionary:D
210	\name_primitive:NN	\hfil	\tex_hfil:D
211	\name_primitive:NN	\hfilneg	\tex_hfilneg:D
212	\name_primitive:NN	\hfill	\tex_hfill:D
213	\name_primitive:NN	\hskip	\tex_hskip:D
214	\name_primitive:NN	\hss	\tex_hss:D
215	\name_primitive:NN	\vfil	\tex_vfil:D
216	\name_primitive:NN	\vfilneg	\tex_vfilneg:D
217	\name_primitive:NN	\vfill	\tex_vfill:D
218	\name_primitive:NN	\vskip	\tex_vskip:D
219	\name_primitive:NN	\vss	\tex_vss:D
220	\name_primitive:NN	\unskip	\tex_unskip:D
221	\name_primitive:NN	\kern	\tex_kern:D
222	\name_primitive:NN	\unkern	\tex_unkern:D
223	\name_primitive:NN	\hrule	\tex_hrule:D
224	\name_primitive:NN	\vrule	\tex_vrule:D

225	\name_primitive:NN	\leaders	\tex_leaders:D
226	\name_primitive:NN	\cleaders	\tex_cleaders:D
227	\name_primitive:NN	\xleaders	\tex_xleaders:D
228	\name_primitive:NN	\lastkern	\tex_lastkern:D
229	\name_primitive:NN	\lastskip	\tex_lastskip:D
230	\name_primitive:NN	\indent	\tex_indent:D
231	\name_primitive:NN	\par	\tex_par:D
232	\name_primitive:NN	\noindent	\tex_noindent:D
233	\name_primitive:NN	\vadjust	\tex_vadjust:D
234	\name_primitive:NN	\baselineskip	\tex_baselineskip:D
235	\name_primitive:NN	\lineskip	\tex_lineskip:D
236	\name_primitive:NN	\lineskiplimit	\tex_lineskiplimit:D
237	\name_primitive:NN	\clubpenalty	\tex_clubpenalty:D
238	\name_primitive:NN	\widowpenalty	\tex_widowpenalty:D
239	\name_primitive:NN	\exhyphenpenalty	\tex_exhyphenpenalty:D
240	\name_primitive:NN	\hyphenpenalty	\tex_hyphenpenalty:D
241	\name_primitive:NN	\linepenalty	\tex_linepenalty:D
242	\name_primitive:NN	\doublehyphendemerits	\tex_doublehyphendemerits:D
243	\name_primitive:NN	\finalhyphendemerits	\tex_finalhyphendemerits:D
244	\name_primitive:NN	\adjdemerits	\tex_adjdemerits:D
245	\name_primitive:NN	\hangafter	\tex_hangafter:D
246	\name_primitive:NN	\hangindent	\tex_hangindent:D
247	\name_primitive:NN	\parshape	\tex_parshape:D
248	\name_primitive:NN	\hsize	\tex_hsize:D
249	\name_primitive:NN	\lefthyphenmin	\tex_lefthyphenmin:D
250	\name_primitive:NN	\righthyphenmin	\tex_righthyphenmin:D
251	\name_primitive:NN	\leftskip	\tex_leftskip:D
252	\name_primitive:NN	\rightskip	\tex_rightskip:D
253	\name_primitive:NN	\looseness	\tex_looseness:D
254	\name_primitive:NN	\parskip	\tex_parskip:D
255	\name_primitive:NN	\parindent	\tex_parindent:D
256	\name_primitive:NN	\uchyph	\tex_uchyph:D
257	\name_primitive:NN	\emergencystretch	\tex_emergencystretch:D
258	\name_primitive:NN	\pretolerance	\tex_pretolerance:D
259	\name_primitive:NN	\tolerance	\tex_tolerance:D
260	\name_primitive:NN	\spaceskip	\tex_spaceskip:D
261	\name_primitive:NN	\xspaceskip	\tex_xspaceskip:D
262	\name_primitive:NN	\parfillskip	\tex_parfillskip:D
263	\name_primitive:NN	\everypar	\tex_everypar:D
264	\name_primitive:NN	\prevgraf	\tex_prevgraf:D
265	\name_primitive:NN	\spacefactor	\tex_spacefactor:D
266	\name_primitive:NN	\shipout	\tex_shipout:D
267	\name_primitive:NN	\vsize	\tex_vsize:D
268	\name_primitive:NN	\interlinepenalty	\tex_interlinepenalty:D
269	\name_primitive:NN	\brokenpenalty	\tex_brokenpenalty:D
270	\name_primitive:NN	\topskip	\tex_topskip:D
271	\name_primitive:NN	\maxdeadcycles	\tex_maxdeadcycles:D
272	\name_primitive:NN	\maxdepth	\tex_maxdepth:D
273	\name_primitive:NN	\output	\tex_output:D
274	\name_primitive:NN	\deadcycles	\tex_deadcycles:D
275	\name_primitive:NN	\pagedepth	\tex_pagedepth:D
276	\name_primitive:NN	\pagestretch	\tex_pagestretch:D
277	\name_primitive:NN	\pagefilstretch	\tex_pagefilstretch:D
278	\name_primitive:NN	\pagefillstretch	\tex_pagefillstretch:D

279	\name_primitive:NN	\pagefilllstretch	\tex_pagefilllstretch:D
280	\name_primitive:NN	\pageshrink	\tex_pageshrink:D
281	\name_primitive:NN	\pagegoal	\tex_pagegoal:D
282	\name_primitive:NN	\pagetotal	\tex_pagetotal:D
283	\name_primitive:NN	\outputpenalty	\tex_outputpenalty:D
284	\name_primitive:NN	\hoffset	\tex_hoffset:D
285	\name_primitive:NN	\voffset	\tex_voffset:D
286	\name_primitive:NN	\insert	\tex_insert:D
287	\name_primitive:NN	\holdinginserts	\tex_holdinginserts:D
288	\name_primitive:NN	\floatingpenalty	\tex_floatingpenalty:D
289	\name_primitive:NN	\insertpenalties	\tex_insertpenalties:D
290	\name_primitive:NN	\lower	\tex_lower:D
291	\name_primitive:NN	\moveleft	\tex_moveleft:D
292	\name_primitive:NN	\moveright	\tex_moveright:D
293	\name_primitive:NN	\raise	\tex_raise:D
294	\name_primitive:NN	\copy	\tex_copy:D
295	\name_primitive:NN	\lastbox	\tex_lastbox:D
296	\name_primitive:NN	\vsplit	\tex_vsplit:D
297	\name_primitive:NN	\unhbox	\tex_unhbox:D
298	\name_primitive:NN	\unhcopy	\tex_unhcopy:D
299	\name_primitive:NN	\unvbox	\tex_unvbox:D
300	\name_primitive:NN	\unvcopy	\tex_unvcopy:D
301	\name_primitive:NN	\setbox	\tex_setbox:D
302	\name_primitive:NN	\hbox	\tex_hbox:D
303	\name_primitive:NN	\vbox	\tex_vbox:D
304	\name_primitive:NN	\vtop	\tex_vtop:D
305	\name_primitive:NN	\prevdepth	\tex_prevdepth:D
306	\name_primitive:NN	\badness	\tex_badness:D
307	\name_primitive:NN	\hbadness	\tex_hbadness:D
308	\name_primitive:NN	\vbadness	\tex_vbadness:D
309	\name_primitive:NN	\hfuzz	\tex_hfuzz:D
310	\name_primitive:NN	\vfuzz	\tex_vfuzz:D
311	\name_primitive:NN	\overfullrule	\tex_overfullrule:D
312	\name_primitive:NN	\boxmaxdepth	\tex_boxmaxdepth:D
313	\name_primitive:NN	\splitmaxdepth	\tex_splitmaxdepth:D
314	\name_primitive:NN	\splittopskip	\tex_splittopskip:D
315	\name_primitive:NN	\everyhbox	\tex_everyhbox:D
316	\name_primitive:NN	\everyvbox	\tex_everyvbox:D
317	\name_primitive:NN	\nullfont	\tex_nullfont:D
318	\name_primitive:NN	\textfont	\tex_textfont:D
319	\name_primitive:NN	\scriptfont	\tex_scriptfont:D
320	\name_primitive:NN	\scriptscriptfont	\tex_scriptscriptfont:D
321	\name_primitive:NN	\fontdimen	\tex_fontdimen:D
322	\name_primitive:NN	\hyphenchar	\tex_hyphenchar:D
323	\name_primitive:NN	\skewchar	\tex_skewchar:D
324	\name_primitive:NN	\defaultshyphenchar	\tex_defaultshyphenchar:D
325	\name_primitive:NN	\defaultskewchar	\tex_defaultskewchar:D
326	\name_primitive:NN	\number	\tex_number:D
327	\name_primitive:NN	\romannumeral	\tex_romannumeral:D
328	\name_primitive:NN	\string	\tex_string:D
329	\name_primitive:NN	\lowercase	\tex_lowercase:D
330	\name_primitive:NN	\uppercase	\tex_uppercase:D
331	\name_primitive:NN	\meaning	\tex_meaning:D
332	\name_primitive:NN	\penalty	\tex_penalty:D

333	\name_primitive:NN	\unpenalty	\tex_unpenalty:D
334	\name_primitive:NN	\lastpenalty	\tex_lastpenalty:D
335	\name_primitive:NN	\special	\tex_special:D
336	\name_primitive:NN	\dump	\tex_dump:D
337	\name_primitive:NN	\patterns	\tex_patterns:D
338	\name_primitive:NN	\hyphenation	\tex_hyphenation:D
339	\name_primitive:NN	\time	\tex_time:D
340	\name_primitive:NN	\day	\tex_day:D
341	\name_primitive:NN	\month	\tex_month:D
342	\name_primitive:NN	\year	\tex_year:D
343	\name_primitive:NN	\jobname	\tex_jobname:D
344	\name_primitive:NN	\everyjob	\tex_everyjob:D
345	\name_primitive:NN	\count	\tex_count:D
346	\name_primitive:NN	\dimen	\tex_dimen:D
347	\name_primitive:NN	\skip	\tex_skip:D
348	\name_primitive:NN	\toks	\tex_toks:D
349	\name_primitive:NN	\muskip	\tex_muskip:D
350	\name_primitive:NN	\box	\tex_box:D
351	\name_primitive:NN	\wd	\tex_wd:D
352	\name_primitive:NN	\ht	\tex_ht:D
353	\name_primitive:NN	\dp	\tex_dp:D
354	\name_primitive:NN	\catcode	\tex_catcode:D
355	\name_primitive:NN	\delcode	\tex_delcode:D
356	\name_primitive:NN	\sfcode	\tex_sfcode:D
357	\name_primitive:NN	\lccode	\tex_lccode:D
358	\name_primitive:NN	\uccode	\tex_uccode:D
359	\name_primitive:NN	\mathcode	\tex_mathcode:D

Since L^AT_EX3 will require at least the ε -T_EX extensions, we also rename the additional primitives. These are all given the prefix \etex_.

360	\name_primitive:NN	\ifdefined	\etex_ifdefined:D
361	\name_primitive:NN	\ifcsname	\etex_ifcsname:D
362	\name_primitive:NN	\unless	\etex_unless:D
363	\name_primitive:NN	\eTeXversion	\etex_eTeXversion:D
364	\name_primitive:NN	\eTeXrevision	\etex_eTeXrevision:D
365	\name_primitive:NN	\marks	\etex_marks:D
366	\name_primitive:NN	\topmarks	\etex_topmarks:D
367	\name_primitive:NN	\firstmarks	\etex_firstmarks:D
368	\name_primitive:NN	\botmarks	\etex_botmarks:D
369	\name_primitive:NN	\splitfirstmarks	\etex_splitfirstmarks:D
370	\name_primitive:NN	\splitbotmarks	\etex_splitbotmarks:D
371	\name_primitive:NN	\unexpanded	\etex_unexpanded:D
372	\name_primitive:NN	\detokenize	\etex_detokenize:D
373	\name_primitive:NN	\scantokens	\etex_scantokens:D
374	\name_primitive:NN	\showtokens	\etex_showtokens:D
375	\name_primitive:NN	\readline	\etex_readline:D
376	\name_primitive:NN	\tracingassigns	\etex_tracingassigns:D
377	\name_primitive:NN	\tracingscantokens	\etex_tracingscantokens:D
378	\name_primitive:NN	\tracingnesting	\etex_tracingnesting:D
379	\name_primitive:NN	\tracingifs	\etex_tracingifs:D
380	\name_primitive:NN	\currentiflevel	\etex_currentiflevel:D
381	\name_primitive:NN	\currentifbranch	\etex_currentifbranch:D
382	\name_primitive:NN	\currentiftype	\etex_currentiftype:D
383	\name_primitive:NN	\tracinggroups	\etex_tracinggroups:D

384	\name_primitive:NN	\currentgrouplevel	\etex_currentgrouplevel:D
385	\name_primitive:NN	\currentgrouptype	\etex_currentgrouptype:D
386	\name_primitive:NN	\showgroups	\etex_showgroups:D
387	\name_primitive:NN	\showifs	\etex_showifs:D
388	\name_primitive:NN	\interactionmode	\etex_interactionmode:D
389	\name_primitive:NN	\lastnodetype	\etex_lastnodetype:D
390	\name_primitive:NN	\iffontchar	\etex_iffontchar:D
391	\name_primitive:NN	\fontcharht	\etex_fontcharht:D
392	\name_primitive:NN	\fontchardp	\etex_fontchardp:D
393	\name_primitive:NN	\fontcharwd	\etex_fontcharwd:D
394	\name_primitive:NN	\fontcharic	\etex_fontcharic:D
395	\name_primitive:NN	\parshapeindent	\etex_parshapeindent:D
396	\name_primitive:NN	\parshapelength	\etex_parshapelength:D
397	\name_primitive:NN	\parshapedimen	\etex_parshapedimen:D
398	\name_primitive:NN	\numexpr	\etex_numexpr:D
399	\name_primitive:NN	\dimexpr	\etex_dimexpr:D
400	\name_primitive:NN	\glueexpr	\etex_glueexpr:D
401	\name_primitive:NN	\muexpr	\etex_muexpr:D
402	\name_primitive:NN	\gluestretch	\etex_gluestretch:D
403	\name_primitive:NN	\glueshrink	\etex_glueshrink:D
404	\name_primitive:NN	\gluestretchorder	\etex_gluestretchorder:D
405	\name_primitive:NN	\glueshrinkorder	\etex_glueshrinkorder:D
406	\name_primitive:NN	\gluetomu	\etex_gluetomu:D
407	\name_primitive:NN	\mutoglue	\etex_mutoglue:D
408	\name_primitive:NN	\lastlinefit	\etex_lastlinefit:D
409	\name_primitive:NN	\interlinepenalties	\etex_interlinepenalties:D
410	\name_primitive:NN	\clubpenalties	\etex_clubpenalties:D
411	\name_primitive:NN	\widowpenalties	\etex_widowpenalties:D
412	\name_primitive:NN	\displaywidowpenalties	\etex_displaywidowpenalties:D
413	\name_primitive:NN	\middle	\etex_middle:D
414	\name_primitive:NN	\savinghyphcodes	\etex_savinghyphcodes:D
415	\name_primitive:NN	\savingdiscards	\etex_savingdiscards:D
416	\name_primitive:NN	\pagediscards	\etex_pagediscards:D
417	\name_primitive:NN	\splitdiscards	\etex_splitdiscards:D
418	\name_primitive:NN	\TeXETstate	\etex_TeXETstate:D
419	\name_primitive:NN	\beginL	\etex_beginL:D
420	\name_primitive:NN	\endL	\etex_endL:D
421	\name_primitive:NN	\beginR	\etex_beginR:D
422	\name_primitive:NN	\endR	\etex_endR:D
423	\name_primitive:NN	\predisplaydirection	\etex_predisplaydirection:D
424	\name_primitive:NN	\everyeof	\etex_everyeof:D
425	\name_primitive:NN	\protected	\etex_protected:D

All major distributions² use pdf ϵ -T ϵ X as engine so we add these names as well. Since the pdfT ϵ X team has been very good at prefixing most primitives with pdf (so far only five do not start with pdf) we do not give then a double pdf prefix. The list below covers pdfT ϵ Xv 1.30.4.

426	%% integer registers:		
427	\name_primitive:NN	\pdfoutput	\pdf_output:D
428	\name_primitive:NN	\pdfminorversion	\pdf_minorversion:D
429	\name_primitive:NN	\pdfcompresslevel	\pdf_compresslevel:D
430	\name_primitive:NN	\pdfdecimaldigits	\pdf_decimaldigits:D

²At the time of writing MiK ϵ T ϵ X does not but I have a gut feeling that will change.

431	\name_primitive:NN	\pdfimageresolution	\pdf_imageresolution:D
432	\name_primitive:NN	\pdfpkresolution	\pdf_pkresolution:D
433	\name_primitive:NN	\pdftracingfonts	\pdf_tracingfonts:D
434	\name_primitive:NN	\pdfuniqueresname	\pdf_uniqueresname:D
435	\name_primitive:NN	\pdfadjustspacing	\pdf_adjustspacing:D
436	\name_primitive:NN	\pdfprotrudechars	\pdf_protrudechars:D
437	\name_primitive:NN	\efcode	\pdf_efcode:D
438	\name_primitive:NN	\lpcode	\pdf_lpcode:D
439	\name_primitive:NN	\rpcode	\pdf_rpcode:D
440	\name_primitive:NN	\pdfforcepagebox	\pdf_forcepagebox:D
441	\name_primitive:NN	\pdfoptionalwaysusepdfpagebox	\pdf_optionalwaysusepdfpagebox:D
442	\name_primitive:NN	\pdfinclusionerrorlevel	\pdf_inclusionerrorlevel:D
443	\name_primitive:NN	\pdfoptionpdfinclusionerrorlevel	\pdf_optionpdfinclusionerrorlevel:D
444	\name_primitive:NN	\pdfimagehicolor	\pdf_imagehicolor:D
445	\name_primitive:NN	\pdfimageapplygamma	\pdf_imageapplygamma:D
446	\name_primitive:NN	\pdfgamma	\pdf_gamma:D
447	\name_primitive:NN	\pdfimagegamma	\pdf_imagegamma:D
448	%% dimen registers:		
449	\name_primitive:NN	\pdfhorigin	\pdf_horigin:D
450	\name_primitive:NN	\pdfvorigin	\pdf_vorigin:D
451	\name_primitive:NN	\pdfpagewidth	\pdf_pagewidth:D
452	\name_primitive:NN	\pdfpageheight	\pdf_pageheight:D
453	\name_primitive:NN	\pdflinkmargin	\pdf_linkmargin:D
454	\name_primitive:NN	\pdfdestmargin	\pdf_destmargin:D
455	\name_primitive:NN	\pdfthreadmargin	\pdf_threadmargin:D
456	%% token registers:		
457	\name_primitive:NN	\pdfpagesattr	\pdf_pagesattr:D
458	\name_primitive:NN	\pdfpageattr	\pdf_pageattr:D
459	\name_primitive:NN	\pdfpageresources	\pdf_pageresources:D
460	\name_primitive:NN	\pdfpkmode	\pdf_pkmode:D
461	%% expandable commands:		
462	\name_primitive:NN	\pdftexrevision	\pdf_texrevision:D
463	\name_primitive:NN	\pdftexbanner	\pdf_texbanner:D
464	\name_primitive:NN	\pdfcreationdate	\pdf_creationdate:D
465	\name_primitive:NN	\pdfpageref	\pdf_pageref:D
466	\name_primitive:NN	\pdfxformname	\pdf_xformname:D
467	\name_primitive:NN	\pdffontname	\pdf_fontname:D
468	\name_primitive:NN	\pdffontobjnum	\pdf_fontobjnum:D
469	\name_primitive:NN	\pdffontsize	\pdf_fontsize:D
470	\name_primitive:NN	\pdfincludechars	\pdf_includechars:D
471	\name_primitive:NN	\leftmarginkern	\pdf_leftmarginkern:D
472	\name_primitive:NN	\rightmarginkern	\pdf_rightmarginkern:D
473	\name_primitive:NN	\pdfescapestring	\pdf_escapestring:D
474	\name_primitive:NN	\pdfescapename	\pdf_escapename:D
475	\name_primitive:NN	\pdfescapehex	\pdf_escapehex:D
476	\name_primitive:NN	\pdfunescapehex	\pdf_unescapehex:D
477	\name_primitive:NN	\pdfstrcmp	\pdf_strcmp:D
478	\name_primitive:NN	\pdfuniformdeviate	\pdf_uniformdeviate:D
479	\name_primitive:NN	\pdfnormaldeviate	\pdf_normaldeviate:D
480	\name_primitive:NN	\pdfmdfivesum	\pdf_mdfivesum:D
481	\name_primitive:NN	\pdffilemoddate	\pdf_filemoddate:D
482	\name_primitive:NN	\pdffilesize	\pdf_filesize:D
483	\name_primitive:NN	\pdffiledump	\pdf_filedump:D
484	%% read-only integers:		

```

485 \name_primitive:NN \pdftexversion      \pdf_texversion:D
486 \name_primitive:NN \pdflastobj         \pdf_lastobj:D
487 \name_primitive:NN \pdflastxform       \pdf_lastxform:D
488 \name_primitive:NN \pdflastximage      \pdf_lastximage:D
489 \name_primitive:NN \pdflastximagepages \pdf_lastximagepages:D
490 \name_primitive:NN \pdflastannot       \pdf_lastannot:D
491 \name_primitive:NN \pdflastxpos        \pdf_lastxpos:D
492 \name_primitive:NN \pdflastypos        \pdf_lastypos:D
493 \name_primitive:NN \pdflastdemerits    \pdf_lastdemerits:D
494 \name_primitive:NN \pdfelapsedtime     \pdf_elapsedtime:D
495 \name_primitive:NN \pdfrandomseed      \pdf_randomseed:D
496 \name_primitive:NN \pdfshellescape     \pdf_shellescape:D
497 %% general commands:
498 \name_primitive:NN \pdfobj              \pdf_obj:D
499 \name_primitive:NN \pdfrefobj           \pdf_refobj:D
500 \name_primitive:NN \pdfxform            \pdf_xform:D
501 \name_primitive:NN \pdfrefxform         \pdf_refxform:D
502 \name_primitive:NN \pdfximage           \pdf_ximage:D
503 \name_primitive:NN \pdfrefximage        \pdf_refximage:D
504 \name_primitive:NN \pdfannot            \pdf_annot:D
505 \name_primitive:NN \pdfstartlink        \pdf_startlink:D
506 \name_primitive:NN \pdfendlink          \pdf_endlink:D
507 \name_primitive:NN \pdfoutline          \pdf_outline:D
508 \name_primitive:NN \pdfdest             \pdf_dest:D
509 \name_primitive:NN \pdfthread           \pdf_thread:D
510 \name_primitive:NN \pdfstartthread      \pdf_startthread:D
511 \name_primitive:NN \pdfendthread        \pdf_endthread:D
512 \name_primitive:NN \pdfsavepos          \pdf_savepos:D
513 \name_primitive:NN \pdfinfo             \pdf_info:D
514 \name_primitive:NN \pdfcatalog          \pdf_catalog:D
515 \name_primitive:NN \pdfnames            \pdf_names:D
516 \name_primitive:NN \pdfmapfile          \pdf_mapfile:D
517 \name_primitive:NN \pdfmapline          \pdf_mapline:D
518 \name_primitive:NN \pdffontattr         \pdf_fontattr:D
519 \name_primitive:NN \pdftrailer          \pdf_trailer:D
520 \name_primitive:NN \pdffontexpand        \pdf_fontexpand:D
521 %%\name_primitive:NN \vadjust [<pre spec>] <filler> { <vertical mode material> } (h, m)
522 \name_primitive:NN \pdfliteral          \pdf_literal:D
523 %%\name_primitive:NN \special <pdfspecial spec>
524 \name_primitive:NN \pdfresettimer       \pdf_resettimer:D
525 \name_primitive:NN \pdfsetrandomseed    \pdf_setrandomseed:D
526 \name_primitive:NN \pdfnoligatures      \pdf_noligatures:D

```

What about Omega and Aleph? The status of both are unclear but let's start by adding a single primitive which we can use for testing if we have one of these engines.

```

527 \name_primitive:NN \texdir              \aleph_texdir:D

```

\CodeStart Here we define functions that are used to turn on and off the special conventions used in the kernel of L^AT_EX3.

\CodeStop

First of all, the space, tab and the return characters will all be ignored inside L^AT_EX3 code, the latter because endlines are set to a space instead. When space characters are needed in L^AT_EX3 code the ~ character will be used for that purpose.

```

528 \tex_def:D\ExplSyntaxOn{
529   \tex_def:D\ExplSyntaxStatus{00}
530   \tex_catcode:D 126=10 \tex_relax:D % tilde is a space char.
531   \tex_catcode:D 32=9 \tex_relax:D % space is ignored
532   \tex_catcode:D 9=9 \tex_relax:D % tab also ignored
533   \tex_endlinechar:D =32 \tex_relax:D % newline is space
534   \tex_catcode:D 95=11 \tex_relax:D % underscore letter
535   \tex_catcode:D 58=11 \tex_relax:D % colon letter
536 }

```

```

537 \tex_def:D\ExplSyntaxOff{
538   \tex_def:D\ExplSyntaxStatus{01}
539   \tex_catcode:D 126=13 \tex_relax:D
540   \tex_catcode:D 32=10 \tex_relax:D
541   \tex_catcode:D 9=10 \tex_relax:D
542   \tex_endlinechar:D =13 \tex_relax:D
543   \tex_catcode:D 95=8 \tex_relax:D
544   \tex_catcode:D 58=12 \tex_relax:D
545 }

```

Temporary while names change.

```

546 \tex_let:D \CodeStart \ExplSyntaxOn
547 \tex_let:D \CodeStop \ExplSyntaxOff

```

\NamesStart Sometimes we need to be able to use names from the kernel of L^AT_EX3 without adhering it's
\NamesStop conventions according to space characters. These macros provide the necessary settings.

```

548 \tex_def:D \NamesStart{
549   \tex_catcode:D '\_ =11\tex_relax:D
550   \tex_catcode:D '\: =11\tex_relax:D
551 }
552 \tex_def:D \NamesStop{
553   \tex_catcode:D '\_ =8\tex_relax:D
554   \tex_catcode:D '\: =12\tex_relax:D
555 }

```

\GetIdInfo Extract all information from a cvs or svn field. The formats are slightly different but
\GetIdInfoAux:w at least the information is in the same positions so we check in the date format so see
\GetIdInfoAuxii:w if it contains a / after the four-digit year. If it does it is cvs else svn and we extract
\GetIdInfoAuxCVS:w information. To be on the safe side we ensure that spaces in the argument are seen.
\GetIdInfoAuxSVN:w

```

556 \tex_def:D\GetIdInfo{
557   \tex_begingroup:D
558   \tex_catcode:D 32=10 \tex_relax:D % needed? Probably for now.
559   \GetIdInfoAux:w
560 }
561 \tex_def:D\GetIdInfoAux:w$#1~#2.#3~#4~#5~#6~#7~#8$#9{
562   \tex_endgroup:D
563   \tex_def:D\filename{#2}
564   \tex_def:D\fileversion{#4}
565   \tex_def:D\filedescription{#9}
566   \tex_def:D\fileauthor{#7}
567   \GetIdInfoAuxii:w #5\tex_relax:D

```

```

568 #3\tex_relax:D#5\tex_relax:D#6\tex_relax:D
569 }
570 \tex_def:D\GetIdInfoAuxii:w #1#2#3#4#5#6\tex_relax:D{
571   \tex_ifx:D#5/
572   \tex_expandafter:D\GetIdInfoAuxCVS:w
573   \tex_else:D
574   \tex_expandafter:D\GetIdInfoAuxSVN:w
575   \tex_fi:D
576 }
577 \tex_def:D\GetIdInfoAuxCVS:w #1,v\tex_relax:D
578                               #2\tex_relax:D#3\tex_relax:D{
579   \tex_def:D\filedate{#2}
580   \tex_def:D\filenameext{#1}
581   \tex_def:D\filetimestamp{#3}

```

When creating the format we want the information in the log straight away.

```

582 <initex>\tex_immediate:D\tex_write:D-1
583 <initex> {\filename;~ v\fileversion,~\filedate;~\filedescription}
584 }
585 \tex_def:D\GetIdInfoAuxSVN:w #1\tex_relax:D#2-#3-#4
586                               \tex_relax:D#5Z\tex_relax:D{
587   \tex_def:D\filenameext{#1}
588   \tex_def:D\filedate{#2/#3/#4}
589   \tex_def:D\filetimestamp{#5}
590 <-package>\tex_immediate:D\tex_write:D-1
591 <-package> {\filename;~ v\fileversion,~\filedate;~\filedescription}
592 }
593 </initex | package>

```

Finally some corrections in the case we are running over L^AT_EX 2_ε.

We want to set things up so that experimental packages and regular packages can coexist with the former using the L^AT_EX3 programming catcode settings. Since it cannot be the task of the end user to know how a package is constructed under the hood we make it so that the experimental packages have to identify themselves. As an example it can be done as

```

\RequirePackage{l3names}
\ProvidesExplPackage{agent}{2007/08/28}{007}{bonding module}

```

or by using the `\file{field}` informations from `\GetIdInfo` as the packages in this distribution do like this:

```

\RequirePackage{l3names}
\GetIdInfo$Id: l3names.dtx 621 2007-09-01 20:14:19Z morten $
  {L3 Experimental Box module}
\ProvidesExplPackage
  {\filename}{\filedate}{\fileversion}{\filedescription}

```

```

\ProvidesExplPackage First up is the identification. Rather trivial
\ProvidesExplClass

```



```

594 (*package)
595 \tex_def:D \ProvidesExplPackage#1#2#3#4{
596   \ProvidesPackage{#1}[#2~v#3~#4]
597   \ExplSyntaxOn
598 }
599 \tex_def:D \ProvidesExplClass#1#2#3#4{
600   \ProvidesClass{#1}[#2~v#3~#4]
601   \ExplSyntaxOn
602 }

```

`\org@onefilewithoptions` The idea behind the code is to record whether or not the L^AT_EX3 syntax is on or off
`\@onefilewithoptions` when about to load a file with class or package extension. This status stored in the
`\@popfilename` parameter `\ExplSyntaxStatus` and set by `\ExplSyntaxOn` and `\ExplSyntaxOff` to 00
and 01 respectively is pushed onto the stack `\ExplSyntaxStack`. Then the catcodes are
set back to normal, the file loaded with its options and finally the stack is popped again.

`\@popfilename` is appended with a preamble check. If the catcode of `@` is being reset
it is a fair assumption that we are back in the usual preamble and so we switch off our
syntax as well.

```

603 \tex_let:D \org@onefilewithoptions\@onefilewithoptions
604 \tex_def:D \@onefilewithoptions#1[#2][#3]#4{
605   \tex_edef:D \ExplSyntaxStack{ \ExplSyntaxStatus\ExplSyntaxStack }
606   \ExplSyntaxOff
607   \org@onefilewithoptions{#1}[{#2}][{#3}]{#4}
608   \tex_expandafter:D\ExplSyntaxPopStack\ExplSyntaxStack\tex_relax:D
609 }
610 \g@addto@macro\@popfilename{%
611   \tex_ifnum:D\tex_the:D\tex_catcode:D'\@=12\tex_relax:D
612   \ExplSyntaxOff
613   \tex_fi:D
614 }

```

`\ExplSyntaxPopStack` Popping the stack is simple: Take the first two tokens which are either the sequence 00
`\ExplSyntaxStack` or 01 and use them in an if test. The stack is initially empty.

```

615 \tex_def:D\ExplSyntaxPopStack#1#2#3\tex_relax:D{
616   \tex_def:D\ExplSyntaxStack{#3}
617   \tex_if:D#1#2
618     \ExplSyntaxOn
619   \tex_else:D
620     \ExplSyntaxOff
621   \tex_fi:D
622 }
623 \tex_def:D\ExplSyntaxStack{}

```

A few of the ‘primitives’ assigned above have already been stolen by L^AT_EX, so assign
them by hand to the saved real primitive.

```

624 \tex_let:D\tex_input:D      \@@input
625 \tex_let:D\tex_underline:D  \@@underline
626 \tex_let:D\tex_end:D        \@@end
627 \tex_let:D\tex_everymath:D  \frozen@everymath

```

```

628 \tex_let:D\tex_everydisplay:D \frozen@everydisplay
629 \tex_let:D\tex_italiccorr:D \@@italiccorr
630 \tex_let:D\tex_hyphen:D \@@hyph

```

T_EX has a nasty habit of inserting a command with the name `\par` so we had better make sure that that command at least has a definition.

```

631 \tex_let:D\par \tex_par:D

632 \tex_ifx:D\name_undefine:N\@gobble

633 \AtEndOfPackage{\ExplSyntaxOff}
634 \tex_def:D\name_pop_stack:w{}

635 \tex_else:D

```

But if traditional T_EX code is disabled, do this...

As mentioned above, The L^AT_EX 2_ε package mechanism will insert some code to handle the filename stack, and reset the package options, this code will die if the T_EX primitives have gone, so skip past it and insert some equivalent code that will work.

First a version of `\ProvidesPackage` that can cope.

```

636 \tex_def:D\ProvidesPackage{
637   \tex_begingroup:D
638   \ExplSyntaxOff
639   \package_provides:w}

640 \tex_def:D\package_provides:w#1#2[#3]{
641   \tex_endgroup:D
642   \tex_immediate:D\tex_write:D-1{Package:~#1#2~#3}
643   \tex_expandafter:D\tex_xdef:D
644   \tex_csname:D ver@#1.sty\tex_endcsname:D{#1}}

```

In this case the catcode preserving stack is not maintained and `\CodeStart` conventions stay in force once on. You'll need to turn then off explicitly with `\CodeStop` (although as currently built on 2e, nothing except very experimental code will run in this mode!) Also note that `\RequirePackage` is a simple definition, just for one file, with no options.

```

645 \tex_def:D\name_pop_stack:w#1\relax{%
646   \ExplSyntaxOff
647   \tex_expandafter:D\@p@pfilename\@currnamestack\@nil
648   \tex_let:D\default@ds\@unknownoptionerror
649   \tex_global:D\tex_let:D\ds\@empty
650   \tex_global:D\tex_let:D\@declaredoptions\@empty}

651 \tex_def:D\@p@pfilename#1#2#3#4\@nil{%
652   \tex_gdef:D\@currname{#1}%
653   \tex_gdef:D\@current{#2}%
654   \tex_catcode:D'\@#3%
655   \tex_gdef:D\@currnamestack{#4}}

```

```

656 \tex_def:D\NeedsTeXFormat#1{}
657 \tex_def:D\RequirePackage#1{
658   \tex_expandafter:D\tex_ifx:D
659     \tex_csname:D ver@#1.sty\tex_endcsname:D\tex_relax:D
660     \ExplSyntaxOn
661     \tex_input:D#1.sty\tex_relax:D
662   \tex_fi:D}
663 \tex_fi:D

```

The `\futurelet` just forces the special end of file marker to vanish, so the argument of `\name_pop_stack:w` does not cause an end-of-file error. (Normally I use `\expandafter` for this trick, but here the next token is in fact `\let` and that may be undefined.)

```

664 \tex_futurelet:D\name_tmp:\name_pop_stack:w
665 \endpackage

```

As the name suggest this package holds some basic definitions which are needed by most or all other packages in this set.

5 Basics

Here we describe those functions that used all over the place. With that we mean functions dealing with the construction and testing of control sequences. Furthermore the basic parts of conditional processing are covered; conditional processing dealing with specific data types is described in the modules specific for the respective data types.

5.1 Predicates and conditionals

5.1.1 Primitive conditionals

The ε -TeX engine itself provides many different conditionals. Some expand whatever comes after them and others don't. Hence the names for these underlying functions will often contain a `:w` part but higher level functions are often available. See for instance `\int_compare_p:nNn` which is a wrapper for `\if_num:w`.

Certain conditionals deal with specific data types like boxes and fonts and are described there. The ones described below are either the universal conditionals or deal with control sequences. We will prefix primitive conditionals with `\if_`.

<pre> \if_true: \if_false: \else: \fi: \reverse_if:N </pre>	<pre> \if_true: <true code> \else: <false code> \fi: \if_false: <true code> \else: <false code> \fi: \reverse_if:N <primitive conditional> </pre>
---	---

`\if_true:` always executes *<true code>*, while `\if_false:` always executes *<false code>*. `\reverse_if:N` reverses any two-way primitive conditional. `\else:` and `\fi:` delimit the branches of the conditional.

	<code>\if_meaning:NN <cs1> <cs2> <true code> \else: <false code></code>
	<code>\fi:</code>
	<code>\if_cs_meaning_eq:NN <cs1> <cs2> <true code> \else:</code>
	<code><false code> \fi:</code>
<code>\if_meaning:NN</code>	<code>\if_token_eq:NN <token1> <token2> <true code> \else:</code>
<code>\if_cs_meaning_eq:NN</code>	<code><false code> \fi:</code>
<code>\if_token_eq:NN</code>	

`\if_meaning:NN` executes `<true code>` when the replacement text, i.e., the expansion of `<cs1>` and `<cs2>` are the same, otherwise it executes `<false code>`. However this name isn't really that good. What the TeX primitive does is compare two tokens to see if they are equal. Hence this is actually a `token` functions. A similar argument applies to the situation where it is used to compare control sequences, where it is the meaning being compared. Something to be cleaned up at some point.

	<code>\if:w <token1> <token2> <true code> \else: <false code></code>
	<code>\fi:</code>
<code>\if:w</code>	<code>\if_catcode:w <token1> <token2> <true code> \else: <false code> \fi:</code>
<code>\if_charcode:w</code>	
<code>\if_catcode:w</code>	

These conditionals will expand any following tokens until two unexpandable tokens are left. If you wish to prevent this expansion, prefix the token in question with `\exp_not:N`. `\if_catcode:w` tests if the category codes of the two tokens are the same whereas `\if:w` tests if the character codes are identical. `\if_charcode:w` is an alternative name for `\if:w`.

	<code>\if_cs_exist:N <cs> <true code> \else: <false code> \fi:</code>
	<code>\if_cs_exist:w <tokens> \cs_end: <true code> \else:</code>
<code>\if_cs_exist:N</code>	<code><false code> \fi:</code>
<code>\if_cs_exist:w</code>	

Check if `<cs>` appears in the hash table or if the control sequence that can be formed from `<tokens>` appears in the hash table. The latter function does not turn the control sequence in question into `\scan_stop:!` This can be useful when dealing with control sequences which cannot be entered as a single token.

<code>\if_mode_horizontal:</code>	
<code>\if_mode_vertical:</code>	
<code>\if_mode_math:</code>	
<code>\if_mode_inner:</code>	<code>\if_horizontal_mode: <true code> \else: <false code> \fi:</code>

Execute `<true code>` if currently in horizontal mode, otherwise execute `<false code>`. Similar for the other functions.

5.1.2 Non-primitive conditionals

<code>\cs_if_eq_p:NN</code>	<code>\cs_if_eq_p:NN <cs1> <cs2></code>
-----------------------------	---

Returns 'true' if `<cs1>` and `<cs2>` are textually the same, i.e. have the same name, otherwise it returns 'false'.

<code>\cs_if_eq:NNTF</code>
<code>\cs_if_eq:NNT</code>
<code>\cs_if_eq:NNF</code>
<code>\cs_if_eq:cNTF</code>
<code>\cs_if_eq:cNT</code>
<code>\cs_if_eq:cNF</code>
<code>\cs_if_eq:NcTF</code>
<code>\cs_if_eq:NcT</code>
<code>\cs_if_eq:NcF</code>
<code>\cs_if_eq:ccTF</code>
<code>\cs_if_eq:ccT</code>
<code>\cs_if_eq:ccF</code>

`\cs_if_eq:NNTF <cs1> <cs2> {\true code}{\false code}`

These functions check if `<cs1>` and `<cs2>` have same meaning and then execute either `<true code>` or `<false code>`.

<code>\cs_if_free_p:N</code>

`\cs_if_free_p:N <cs>`

Returns ‘true’ if `<cs>` is either undefined or equal to `\scan_stop:.` However, it returns ‘false’ if `<cs>` is textually `\c_undefined` (the constantly undefined function), or textually `\scan_stop:.`

<code>\cs_if_free:NTF</code>
<code>\cs_if_free:NT</code>
<code>\cs_if_free:NF</code>
<code>\cs_if_free:cTF</code>
<code>\cs_if_free:cT</code>
<code>\cs_if_free:cF</code>

`\cs_if_free:NTF <cs> {\true code}{\false code}`

These functions check if `<cs>` is free and then execute either `<true code>` or `<false code>`.

T_EXhackers note: The conditional `\cs_if_free:cTF` is the L^AT_EX3 implementation of the L^AT_EX2 function `\@ifundefined`. The other functions haven’t been around before.

<code>\cs_if_really_free:cTF</code>
<code>\cs_if_really_free:cF</code>
<code>\cs_if_really_free:cT</code>

`\cs_if_really_free:cTF {\tokens} {\true code} {\false code}`

Similar to `\cs_if_free:cTF` but does not put anything previously undefined into the hash table. Useful for special control sequences like `\foo/bar` which cannot be entered as one token.

<code>\cs_if_exist_p:N</code>

`\cs_if_exist_p:N <cs>`

This function does the opposite of `\cs_if_free_p:N`.

<code>\cs_if_exist:NTF</code>
<code>\cs_if_exist:NT</code>
<code>\cs_if_exist:NF</code>
<code>\cs_if_exist:cTF</code>
<code>\cs_if_exist:cT</code>
<code>\cs_if_exist:cF</code>

`\cs_if_exist:NTF <cs> {\true code}{\false code}`

These functions check if $\langle cs \rangle$ exists and then execute either $\langle true\ code \rangle$ or $\langle false\ code \rangle$. Exactly the opposite of `\cs_if_free:NTF`.

<code>\cs_if_really_exist:cTF</code>	<code>\cs_if_really_exist:cTF {$\langle tokens \rangle$} {$\langle true\ code \rangle$} {$\langle false\ code \rangle$}</code>
<code>\cs_if_really_exist:cF</code>	
<code>\cs_if_really_exist:cT</code>	

The opposite of `\cs_if_really_free:cTF`.

<code>\chk_new_cs:N</code>	<code>\chk_new_cs:N $\langle cs \rangle$</code>
----------------------------	--

This function checks that $\langle cs \rangle$ is so far either undefined or equals `\scan_stop:` (the function that is assigned to newly created control sequences by \TeX when `\cs:w ... \cs_end:` is used).

<code>\chk_exist_cs:N</code>	<code>\chk_exist_cs:N $\langle cs \rangle$</code>
<code>\chk_exist_cs:c</code>	

This function checks that $\langle cs \rangle$ is defined. If it is not an error is generated.

<code>\c_true</code>
<code>\c_false</code>

Constants that represent ‘true’ or ‘false’, respectively. Used to implement predicates.

5.2 Selecting and discarding tokens from the input stream

The conditional processing could not have been implemented without being able to gobble and select which tokens to use from the input stream.

<code>\use_none:n</code>	<code>\use_none:n {$\langle arg1 \rangle$}</code> <code>\use_none:nn {$\langle arg1 \rangle$}{$\langle arg2 \rangle$}</code>
<code>\use_none:nn</code>	
<code>\use_none:nnn</code>	
<code>\use_none:nnnn</code>	
<code>\use_none:nnnnn</code>	
<code>\use_none:nnnnnn</code>	
<code>\use_none:nnnnnnn</code>	
<code>\use_none:nnnnnnnn</code>	
<code>\use_none:nnnnnnnnn</code>	

These functions gobble the tokens or brace groups from the input stream.

<code>\use_arg_i:n</code>	<code>\use_arg_i:n { $\langle code1 \rangle$ }</code>
---------------------------	--

Function that executes the next argument after removing the surrounding braces. Used to implement conditionals.

<code>\use_arg_i:nn</code>	<code>\use_arg_i:nn { $\langle code1 \rangle$ }{ $\langle code2 \rangle$ }</code>
<code>\use_arg_ii:nn</code>	

Functions that execute the first or second argument respectively, after removing the surrounding braces. Primarily used to implement conditionals.

<code>\use_arg_i:nnn</code> <code>\use_arg_ii:nnn</code> <code>\use_arg_iii:nnn</code>	<code>\use_arg_i:nnn {⟨arg1⟩}{⟨arg2⟩}{⟨arg3⟩}</code>
--	--

Functions that pick up one of three arguments and execute them after removing the surrounding braces. Should be described somewhere else.

<code>\use_arg_i:nnnn</code> <code>\use_arg_ii:nnnn</code> <code>\use_arg_iii:nnnn</code> <code>\use_arg_iv:nnnn</code>	<code>\use_arg_i:nnnn {⟨arg1⟩}{⟨arg2⟩}{⟨arg3⟩}{⟨arg4⟩}</code>
--	---

Functions that pick up one of four arguments and execute them after removing the surrounding braces.

A different kind of functions for selecting tokens from the token stream are those that use delimited arguments.

<code>\use_none_delimit_by_q_nil:w</code> <code>\use_none_delimit_by_q_stop:w</code>	<code>\use_none_delimit_by_q_nil:w ⟨balanced text⟩ \q_nil</code>
---	--

Gobbles *⟨balanced text⟩*. Useful in gobbling the remainder in a list structure.

<code>\use_arg_i_delimit_by_q_nil:nw</code> <code>\use_arg_i_delimit_by_q_stop:nw</code>	<code>\use_arg_i_delimit_by_q_nil:nw {⟨arg⟩} ⟨balanced text⟩ \q_nil</code>
---	--

Gobbles *⟨balanced text⟩* and executes *⟨arg⟩* afterwards. This can also be used to get the first item in a token list.

<code>\use_arg_i_after_fi:nw</code> <code>\use_arg_i_after_else:nw</code> <code>\use_arg_i_after_or:nw</code> <code>\use_arg_i_after_orelse:nw</code>	<code>\use_arg_i_after_fi:nw {⟨arg⟩} \fi:</code> <code>\use_arg_i_after_else:nw {⟨arg⟩} \else: ⟨balanced text⟩ \fi:</code> <code>\use_arg_i_after_or:nw {⟨arg⟩} \or: ⟨balanced text⟩ \fi:</code> <code>\use_arg_i_after_orelse:nw {⟨arg⟩} \or:/\else: ⟨balanced text⟩ \fi:</code>
--	--

Executes *⟨arg⟩* after executing closing out `\fi:`. `\use_arg_i_after_orelse:nw` can be used anywhere where `\use_arg_i_after_else:nw` or `\use_arg_i_after_or:nw` are used.

5.3 Internal functions

<code>\cs:w</code> <code>\cs_end:</code>	<code>\cs:w ⟨tokens⟩ \cs_end:</code>
---	--------------------------------------

This is the \TeX internal way of generating a control sequence from some token list. *⟨tokens⟩* get expanded and must ultimately result in a sequence of characters.

\TeX hackers note: These functions are the primitives `\csname` and `\endcsname`. `\cs:w` is considered weird because it expands tokens until it reaches `\cs_end:`.

<code>\pref_global:D</code> <code>\pref_long:D</code> <code>\pref_protected:D</code>	<code>\pref_global:D \def:Npn</code>
--	--------------------------------------

Prefix functions that can be used in front of some definition functions (namely ...). The result of prefixing a function definition with `\pref_global:D` makes the definition global, `\pref_long:D` change the argument scanning mechanism so that it allows `\par` tokens in the argument of the prefixed function, and `\pref_protected:D` makes the definition robust in `\writes` etc.

None of these internal functions should be used by a programmer since the necessary combinations are all available as separate function, e.g., `\def_long:Npn` is internally implemented as `\pref_long:D \def:Npn`.

T_EXhackers note: These prefixes are the primitives `\global`, `\long`, and `\protected`. The `\outer` isn't used at all within L^AT_EX3 because ...

<code>\io_put_log:x</code> <code>\io_put_term:x</code> <code>\io_put_deferred:Nx</code>	<code>\io_put_log:x {<message>}</code> <code>\io_put_deferred:Nx <write_stream> {<message>}</code>
---	---

Writes `<message>` to either to log or the terminal.

5.4 Defining functions

There are two types of function definitions in L^AT_EX3: versions that check if the function name is still unused, and versions that simply make the definition. The later are used for internal scratch functions that get new meanings all over the place.

For each type there is an additional choice to be made: Does the function to be defined contain delimited arguments? The answer in 99% of the cases is no, so in most cases the programmer just want to input the number of arguments, which is basically how `\newcommand` in L^AT_EX 2_ε works. Therefore we provide functions that expect a number as the primary type and later on in this module you can find the ones with the more primitive syntax.

A definition of a new function can be done locally and globally. Currently nearly all function definitions are done locally on top level, in other words they are global but don't show it. Therefore I think it may be better to remove the local variants in the future and declare all checked function definitions global.

T_EXhackers note: While T_EX makes all definition functions directly available to the user L^AT_EX3 hides them very carefully to avoid the problems with definitions that are overwritten accidentally. Many functions that are in T_EX a combination of prefixes and definition functions are provided as individual functions.

5.4.1 Defining new functions

Firstly comes to variants most used namely those taking a number to denote the number of arguments.

<code>\def_new:NNn</code>
<code>\def_new:NNx</code>
<code>\def_new:cNn</code>
<code>\def_new:cNx</code>

`\def_new:NNn <cs> <num> { <code> }`

Defines a new function, making sure that `<cs>` is unused so far. `<num>` is the number of arguments which is in the interval `[0,9]` otherwise an error is raised. It is under the responsibility of the programmer to name the new function according to the rules laid out in the previous section. `<code>` is either passed literally or may be subject to expansion (under the x variants).

<code>\gdef_new:NNn</code>
<code>\gdef_new:cNn</code>
<code>\gdef_new:NNx</code>
<code>\gdef_new:cNx</code>

`\gdef_new:NNn <cs> <num> { <code> }`

Like `\def_new:NNn` but defines the new function globally.

<code>\def_long_new:NNn</code>
<code>\def_long_new:NNx</code>
<code>\def_long_new:cNn</code>
<code>\def_long_new:cNx</code>

`\def_long_new:NNn <cs> <num> { <code> }`

Defines a function that may contain `\par` tokens in the argument(s) when called. This is not allowed for normal functions.

<code>\gdef_long_new:NNn</code>
<code>\gdef_long_new:NNx</code>
<code>\gdef_long_new:cNn</code>
<code>\gdef_long_new:cNx</code>

`\gdef_long_new:NNn <cs> <num> { <code> }`

Global versions of the above functions.

<code>\def_protected_new:NNn</code>
<code>\def_protected_new:NNx</code>
<code>\def_protected_new:cNn</code>
<code>\def_protected_new:cNx</code>

`\def_protected_new:NNn <cs> <num> { <code> }`

Defines a function that does not expand when inside an x type expansion.

<code>\gdef_protected_new:NNn</code>
<code>\gdef_protected_new:NNx</code>
<code>\gdef_protected_new:cNn</code>
<code>\gdef_protected_new:cNx</code>

`\gdef_protected_new:NNn <cs> <num> { <code> }`

Global versions of the above functions.

<code>\def_protected_long_new:NNn</code>
<code>\def_protected_long_new:NNx</code>
<code>\def_protected_long_new:cNn</code>
<code>\def_protected_long_new:cNx</code>

`\def_protected_long_new:NNn <cs> <num> { <code> }`

Defines a function that is both robust and may contain `\par` tokens in the argument(s) when called.

<code>\gdef_protected_long_new:NNn</code> <code>\gdef_protected_long_new:NNx</code> <code>\gdef_protected_long_new:cNn</code> <code>\gdef_protected_long_new:cNx</code>	<code>\gdef_protected_long_new:NNn <cs> <num> { <code> }</code>
--	---

Global versions of the above functions.

Secondly comes the ones where the programmer can use delimited arguments. Rarely needed outside the kernel.

<code>\def_new:Npn</code> <code>\def_new:Npx</code> <code>\def_new:cpn</code> <code>\def_new:cpx</code>	<code>\def_new:Npn <cs> <parms> { <code> }</code>
--	---

Defines a new function, making sure that `<cs>` is unused so far. `<parms>` may consist of arbitrary parameter specification in T_EX syntax. It is under the responsibility of the programmer to name the new function according to the rules laid out in the previous section. `<code>` is either passed literally or may be subject to expansion (under the `x` variants).

<code>\gdef_new:Npn</code> <code>\gdef_new:cpn</code> <code>\gdef_new:Npx</code> <code>\gdef_new:cpx</code>	<code>\gdef_new:Npn <cs> <parms> { <code> }</code>
--	--

Like `\def_new:Npn` but defines the new function globally. See comments above.

<code>\def_long_new:Npn</code> <code>\def_long_new:Npx</code> <code>\def_long_new:cpn</code> <code>\def_long_new:cpx</code>	<code>\def_long_new:Npn <cs> <parms> { <code> }</code>
--	--

Defines a function that may contain `\par` tokens in the argument(s) when called. This is not allowed for normal functions.

<code>\gdef_long_new:Npn</code> <code>\gdef_long_new:Npx</code> <code>\gdef_long_new:cpn</code> <code>\gdef_long_new:cpx</code>	<code>\gdef_long_new:Npn <cs> <parms> { <code> }</code>
--	---

Global versions of the above functions.

<code>\def_protected_new:Npn</code> <code>\def_protected_new:Npx</code> <code>\def_protected_new:cpn</code> <code>\def_protected_new:cpx</code>	<code>\def_protected_new:Npn <cs> <parms> { <code> }</code>
--	---

Defines a function that does not expand when inside an `x` type expansion.

<code>\gdef_protected_new:Npn</code>
<code>\gdef_protected_new:Npx</code>
<code>\gdef_protected_new:cpn</code>
<code>\gdef_protected_new:cpx</code>

`\gdef_protected_new:Npn <cs> <parms> { <code> }`

Global versions of the above functions.

<code>\def_protected_long_new:Npn</code>
<code>\def_protected_long_new:Npx</code>
<code>\def_protected_long_new:cpn</code>
<code>\def_protected_long_new:cpx</code>

`\def_protected_long_new:Npn <cs> <parms> { <code> }`

Defines a function that is both robust and may contain `\par` tokens in the argument(s) when called.

<code>\gdef_protected_long_new:Npn</code>
<code>\gdef_protected_long_new:Npx</code>
<code>\gdef_protected_long_new:cpn</code>
<code>\gdef_protected_long_new:cpx</code>

`\gdef_protected_long_new:Npn <cs> <parms> { <code> }`

Global versions of the above functions.

<code>\let_new:NN</code>
<code>\let_new:cN</code>
<code>\let_new:Nc</code>
<code>\let_new:cc</code>
<code>\glet_new:NN</code>
<code>\glet_new:cN</code>
<code>\glet_new:Nc</code>
<code>\glet_new:cc</code>

`\let_new:NN <cs1> <cs2>`

Gives the function `<cs1>` the current meaning of `<cs2>`. Again, we may do this always globally.

5.4.2 Undefining functions

<code>\cs_gundefine:N</code>

`\cs_gundefine:N <cs>`

Undefines the control sequence.

5.4.3 Defining internal functions (no checks)

Besides the function definitions that check whether or not their argument is an unused function we need function definitions that overwrite currently used definitions. The following functions are provided for this purpose.

First comes the versions expecting a number to denote the number of arguments.

<code>\def:NNn</code>
<code>\def:NNx</code>
<code>\def:cNn</code>
<code>\def:cNx</code>

`\def:NNn <cs> <num> { <code> }`

Like `\def_new:NNn` etc. but does not check the `<cs>` name.

<code>\gdef:NNn</code>
<code>\gdef:NNx</code>
<code>\gdef:cNn</code>
<code>\gdef:cNx</code>

`\gdef:NNn <cs> <num> { <code> }`

Like `\def:NNn` but defines the `<cs>` globally.

<code>\def_long:NNn</code>
<code>\def_long:NNx</code>
<code>\def_long:cNn</code>
<code>\def_long:cNx</code>

`\def_long:NNn <cs> <num> { <code> }`

Like `\def:NNn` but allows `\par` tokens in the arguments of the function being defined.

<code>\gdef_long:NNn</code>
<code>\gdef_long:NNx</code>
<code>\gdef_long:cNn</code>
<code>\gdef_long:cNx</code>

`\gdef_long:NNn <cs> <num> { <code> }`

Global variant of `\def_long:NNn`.

<code>\def_protected:NNn</code>
<code>\def_protected:cNn</code>
<code>\def_protected:NNx</code>
<code>\def_protected:cNx</code>

`\def_protected:NNn <cs> <num> { <code> }`

Naturally robust macro that won't expand in an `x` type argument. This also comes as a long version. If you for some reason want to expand it inside an `x` type expansion, prefix it with `\exp_after:NN \use_noop:`.

<code>\gdef_protected:NNn</code>
<code>\gdef_protected:cNn</code>
<code>\gdef_protected:NNx</code>
<code>\gdef_protected:cNx</code>

`\gdef_protected:NNn <cs> <num> { <code> }`

Global versions of the above functions.

<code>\def_protected_long:NNn</code>
<code>\def_protected_long:cNn</code>
<code>\def_protected_long:NNx</code>
<code>\def_protected_long:cNx</code>

`\def_protected_long:NNn <cs> <num> { <code> }`

Naturally robust macro that won't expand in an `x` type argument. These varieties also allow `\par` tokens in the arguments of the function being defined.

<code>\gdef_protected_long:NNn</code> <code>\gdef_protected_long:cNn</code> <code>\gdef_protected_long:NNx</code> <code>\gdef_protected_long:cNx</code>	<code>\gdef_protected_long:NNn <cs> <num> { <code> }</code>
--	---

Global versions of the above functions.

Secondly the ones that use the primitive parameter build-up:

<code>\def:Npn</code> <code>\def:Npx</code> <code>\def:cpn</code> <code>\def:cpx</code>	<code>\def:Npn <cs> <parms> { <code> }</code>
--	---

Like `\def_new:Npn` etc. but does not check the `<cs>` name.

T_EXhackers note: `\def:Npn` is the L^AT_EX3 name for T_EX's `\def` and `\def:Npx` corresponds to the primitive `\edef`. The `\def:cpn` function was known in L^AT_EX2 as `\@namedef`. `\def:cpx` has no equivalent.

<code>\gdef:Npn</code> <code>\gdef:Npx</code> <code>\gdef:cpn</code> <code>\gdef:cpx</code>	<code>\gdef:Npn <cs> <parms> { <code> }</code>
--	--

Like `\def:Npn` but defines the `<cs>` globally.

T_EXhackers note: `\gdef:Npn` and `\gdef:Npx` are known to T_EXhackers as `\gdef` and `\xdef`.

<code>\def_long:Npn</code> <code>\def_long:Npx</code> <code>\def_long:cpn</code> <code>\def_long:cpx</code>	<code>\def_long:Npn <cs> <parms> { <code> }</code>
--	--

Like `\def:Npn` but allows `\par` tokens in the arguments of the function being defined.

<code>\gdef_long:Npn</code> <code>\gdef_long:Npx</code> <code>\gdef_long:cpn</code> <code>\gdef_long:cpx</code>	<code>\gdef_long:Npn <cs> <parms> { <code> }</code>
--	---

Global variant of `\def_long:Npn`.

<code>\def_protected:Npn</code> <code>\def_protected:cpn</code> <code>\def_protected:Npx</code> <code>\def_protected:cpx</code>	<code>\def_protected:Npn <cs> <parms> { <code> }</code>
--	---

Naturally robust macro that won't expand in an x type argument. This also comes as a

long version. If you for some reason want to expand it inside an `x` type expansion, prefix it with `\exp_after:NN \use_noop:`.

<code>\gdef_protected:Npn</code>
<code>\gdef_protected:cpn</code>
<code>\gdef_protected:Npx</code>
<code>\gdef_protected:cpx</code>

`\gdef_protected:Npn <cs> <parms> { <code> }`

Global versions of the above functions.

<code>\def_protected_long:Npn</code>
<code>\def_protected_long:cpn</code>
<code>\def_protected_long:Npx</code>
<code>\def_protected_long:cpx</code>

`\def_protected_long:Npn <cs> <parms> { <code> }`

Naturally robust macro that won't expand in an `x` type argument. These varieties also allow `\par` tokens in the arguments of the function being defined.

<code>\gdef_protected_long:Npn</code>
<code>\gdef_protected_long:cpn</code>
<code>\gdef_protected_long:Npx</code>
<code>\gdef_protected_long:cpx</code>

`\gdef_protected_long:Npn <cs> <parms> { <code> }`

Global versions of the above functions.

<code>\let:NN</code>
<code>\let:cN</code>
<code>\let:Nc</code>
<code>\let:cc</code>
<code>\glet:NN</code>
<code>\glet:cN</code>
<code>\glet:Nc</code>
<code>\glet:cc</code>

`\let:cN <cs1> <cs2>`

Gives the function `<cs1>` the current meaning of `<cs2>`. Again, we may always do this globally.

<code>\let:NwN</code>
<code>\let:NwN</code>

`\let:NwN <cs1> <cs2>`
`\let:NwN <cs1> = <cs2>`

These functions assign the meaning of `<cs2>` locally or globally to the function `<cs1>`. Because the `TEX` primitive operation is being used which may have an equal sign and (a certain number of) spaces between `<cs1>` and `<cs2>` the name contains a `w`. (Not happy about this convention!).

T_EXhackers note: `\let:NwN` is the L^AT_EX3 name for T_EX's `\let`.

5.5 Defining test functions

```
\def_test_function:npn
\def_long_test_function:npn
\def_test_function_new:npn
\def_long_test_function_new:npn \def_test_function_new:npn {name} {parms} {<test>}
```

Define all the common test cases for a simple test to reduce the risk of typos. As an example here's how we defined the functions `\cs_free:cTF`, `\cs_free:cT` and `\cs_free:cF`. You just have to fill in the test.

```
\def_test_function:npn{cs_free:c} #1 {
  \exp_after:NN \if_meaning:NN \cs:w#1\cs_end: \scan_stop:}
```

Be careful not to use this function inside some primitive conditional as \TeX will most likely get confused because of the unmatched conditionals.

5.6 The innards of a function

```
\cs_to_str:N \cs_to_str:N <cs>
```

This function return the name of $\langle cs \rangle$ as a sequence of letters with the escape character removed.

```
\token_to_string:N \token_to_string:N <arg>
```

This function return the name of $\langle arg \rangle$ as a sequence of letters including the escape character.

```
\token_to_meaning:N \token_to_meaning:N <arg>
```

This function returns the type and definition of $\langle arg \rangle$ as a sequence of letters.

Other functions regarding arbitrary tokens can be found in the `l3token` module.

5.7 Grouping and scanning

```
\scan_stop: \scan_stop:
```

This function stops \TeX 's scanning ahead when ending a number.

\TeX hackers note: This is the \TeX primitive `\relax` renamed.

```
\group_begin:
\group_end: \group_begin: <...> \group_end:
```

Encloses $\langle \dots \rangle$ inside a group.

\TeX hackers note: These are the \TeX primitives `\begingroup` and `\endgroup` renamed.

5.8 Engine specific definitions

```
\engine_if_aleph:TF \engine_if_aleph:TF {\true code} {\false code}
```

This function detects if we're running an Aleph based format. This is particularly useful when allocating registers.

5.9 The Implementation

We start by ensuring that the required packages are loaded. We need `l3names` to get things going but we actually need it very early on, so it is loaded at the very top of this file. Also, most of the code below won't run until `l3expan` has been loaded.

5.9.1 Renaming some T_EX primitives (again)

`\let:NwN` Having given all the tex primitives a consistent name, we need to give sensible names to the ones we actually want to use. These will be defined as needed in the appropriate modules, but do a few now, just to get started.³

```
666 <package>\ProvidesExplPackage
667 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
668 <*initex | package>
669 \tex_let:D \let:NwN \tex_let:D
```

```
\if_true: Then some conditionals.
\if_false:
  \else: 670 \let:NwN \if_true: \tex_iftrue:D
  \fi: 671 \let:NwN \if_false: \tex_iffalse:D
672 \let:NwN \else: \tex_else:D
\reverse_if:N 673 \let:NwN \fi: \tex_fi:D
  \if:w 674 \let:NwN \reverse_if:N \etex_unless:D
\if_charcode:w 675 \let:NwN \if:w \tex_if:D
\if_catcode:w 676 \let:NwN \if_charcode:w \tex_if:D
677 \let:NwN \if_catcode:w \tex_ifcat:D
```

```
\if_meaning:NN Some different names for \ifx.4
\if_token_eq:NN
\if_cs_meaning_eq:NN 678 \let:NwN \if_meaning:NN \tex_ifx:D
679 \let:NwN \if_token_eq:NN \tex_ifx:D
680 \let:NwN \if_cs_meaning_eq:NN \tex_ifx:D
```

```
\if_mode_math: TEX lets us detect some if its modes.
\if_mode_horizontal:
\if_mode_vertical: 681 \let:NwN \if_mode_math: \tex_ifmmode:D
682 \let:NwN \if_mode_horizontal: \tex_ifhmode:D
\if_mode_inner: 683 \let:NwN \if_mode_vertical: \tex_ifvmode:D
684 \let:NwN \if_mode_inner: \tex_ifinner:D
```

³This renaming gets expensive in terms of csname usage, an alternative scheme would be to just use the “tex...D” name in the cases where no good alternative exists.

⁴MH: Clean up at some point


```

\if_cs_exist:N
\if_cs_exist:w
685 \let:NwN \if_cs_exist:N \etex_ifdefined:D
686 \let:NwN \if_cs_exist:w \etex_ifcsname:D

```

```

\exp_after:NN The three \exp_ functions are used in the l3expan module where they are described.
\exp_not:N
\exp_not:n
687 \let:NwN \exp_after:NN \tex_expandafter:D
688 \let:NwN \exp_not:N \tex_noexpand:D
689 \let:NwN \exp_not:n \tex_unexpanded:D

```

```

\io_put_deferred:Nx
\token_to_meaning:N
\token_to_string:N
690 \let:NwN \io_put_deferred:Nx \tex_write:D
691 \let:NwN \token_to_meaning:N \tex_meaning:D
692 \let:NwN \token_to_string:N \tex_string:D
693 \let:NwN \cs:w \tex_csname:D
\cs_meaning:N
694 \let:NwN \cs_end: \tex_endcsname:D
\cs_meaning:c
695 \let:NwN \cs_meaning:N \tex_meaning:D
\cs_show:N
696 \tex_def:D \cs_meaning:c #1{\exp_after:NN\cs_meaning:N\cs:w #1\cs_end:}
\cs_show:c
697 \let:NwN \cs_show:N \tex_show:D
698 \tex_def:D \cs_show:c #1{\exp_after:NN\cs_show:N\cs:w #1\cs_end:}

```

```

\scan_stop: The next three are basic functions for which there also exist versions that are safe inside
\group_begin: alignments. These safe versions are defined in the l3prg module.
\group_end:
699 \let:NwN \scan_stop: \tex_relax:D
700 \let:NwN \group_begin: \tex_begingroup:D
701 \let:NwN \group_end: \tex_endgroup:D

```

```

\group_execute_after:N
702 \let:NwN \group_execute_after:N \tex_aftergroup:D

```

```

\the_internal:D These following names are temporary and should be removed as soon as possible (April
\pref_global:D 1998).
\pref_long:D
\pref_protected:D
703 \let:NwN \the_internal:D \tex_the:D
704 \let:NwN \pref_global:D \tex_global:D

```

\pref_long:D has been documented for years but didn't exist... Added it and the robustness prefix.

```

705 \let:NwN \pref_long:D \tex_long:D
706 \let:NwN \pref_protected:D \etex_protected:D

```

5.9.2 Defining functions

We start by providing functions for the typical definition functions. First the local ones.

```

\def:Npn All assignment functions in LATEX3 should be naturally robust; after all, the TEX primi-
\def:Npx tives for assignments are and it can be a cause of problems if others aren't.
\def_long:Npn
\def_long:Npx 707 \let:NwN \def:Npn \tex_def:D
\def_protected:Npn 708 \let:NwN \def:Npx \tex_edef:D
\def_protected:Npx 709 \pref_protected:D \def:Npn \def_long:Npn {\pref_long:D \def:Npn}
\def_protected:Npx 710 \pref_protected:D \def:Npn \def_long:Npx {\pref_long:D \def:Npx}
\def_protected_long:Npn 711 \pref_protected:D \def:Npn \def_protected:Npn {\pref_protected:D \def:Npn}
\def_protected_long:Npx 712 \pref_protected:D \def:Npn \def_protected:Npx {\pref_protected:D \def:Npx}
713 \def_protected:Npn \def_protected_long:Npn {
714 \pref_protected:D \pref_long:D \def:Npn
715 }
716 \def_protected:Npn \def_protected_long:Npx {
717 \pref_protected:D \pref_long:D \def:Npx
718 }

\gdef:Npn Global versions of the above functions.
\gdef:Npx
\gdef_long:Npn 719 \let:NwN \gdef:Npn \tex_gdef:D
\gdef_long:Npx 720 \let:NwN \gdef:Npx \tex_xdef:D
\gdef_protected:Npn 721 \def_protected:Npn \gdef_long:Npn {\pref_long:D \gdef:Npn}
\gdef_protected:Npx 722 \def_protected:Npn \gdef_long:Npx {\pref_long:D \gdef:Npx}
\gdef_protected:Npn 723 \def_protected:Npn \gdef_protected:Npn {\pref_protected:D \gdef:Npn}
\gdef_protected_long:Npn 724 \def_protected:Npn \gdef_protected:Npx {\pref_protected:D \gdef:Npx}
\gdef_protected_long:Npx 725 \def_protected:Npn \gdef_protected_long:Npn {
726 \pref_protected:D \pref_long:D \gdef:Npn
727 }
728 \def_protected:Npn \gdef_protected_long:Npx {
729 \pref_protected:D \pref_long:D \gdef:Npx
730 }

```

5.9.3 Predicate implementation

I think Michael originated the idea of expandable boolean tests. At first these were supposed to expand into either TT or TF to be tested using `\if:w` but this was later changed to 00 and 01, so they can be used in logical operations (see the `l3prg` module). We need this from the get-go.

```

\c_true Here are the canonical boolean values.
\c_false
731 \def:Npn \c_true {00}
732 \def:Npn \c_false {01}

```

5.9.4 Defining and checking (new) functions

```

\c_minus_one We need the constants \c_minus_one and \c_sixteen now for writing information to
\c_sixteen the log and the terminal but the rest are defined in the l3num module – at least for the
ones that can be defined with \tex_chardef:D or \tex_mathchardef:D. Otherwise the
l3int module is required but it can't be used until the allocation has been set up properly!
The actual allocation mechanism is in l3alloc and as TEX wants to reserve count registers
0–9, the first available one is 10 so we use that for \c_minus_one.

```

```

733 <!*initex>
734 \let:NwN \c_minus_one\m@ne
735 </!initex>
736 <!*package>
737 \tex_countdef:D \c_minus_one = 10 \scan_stop:
738 \c_minus_one = -1 \scan_stop:
739 </!package>
740 \tex_chardef:D \c_sixteen = 16\scan_stop:

```

We provide two kinds of functions that can be used to define control sequences. On the one hand we have functions that check if their argument doesn't already exist, they are called `\..._new`. The second type of defining functions doesn't check if the argument is already defined.

Before we can define them, we need some auxiliary macros that allow us to generate error messages. The definitions here are only temporary, they will be redefined later on.

`\io_put_log:x` We define a routine to write only to the log file. And a similar one for writing to both the log file and the terminal.

```

741 \def:Npn \io_put_log:x{
742   \tex_immediate:D\io_put_deferred:Nx \c_minus_one }
743 \def:Npn \io_put_term:x{
744   \tex_immediate:D\io_put_deferred:Nx \c_sixteen }

```

`\err_latex_bug:x` This will show internal errors.

```

745 \def:Npn\err_latex_bug:x#1{
746   \io_put_term:x{This~is~a~LaTeX~bug!~Check~coding!}\tex_errmessage:D{#1}}

```

`\cs_record_meaning:N` This macro will be used later on for tracing purposes. But we need some more modules to define it, so we just give some dummy definition here.

```

747 <*trace>
748 \def:Npn \cs_record_meaning:N#1{
749 </trace>

```

We need these two to make `\chk_new_cs:N` bulletproof.

```

750 \def_long:Npn \use_none:n #1{}
751 \def_long:Npn \use_arg_i:n #1{#1}

```

`\chk_new_cs:N` This command is called by `\def_new:Npn` and `\let_new:NN` etc. to make sure that the argument sequence is not already in use. If it is, an error is signalled. It checks if `<csname>` is undefined or `\scan_stop:.` Otherwise an error message is issued. We have to make sure we don't put the argument into the conditional processing since it may be an `\if...` type function!

```

752 \def:Npn \chk_new_cs:N #1{
753   \if:w \cs_if_free_p:N #1
754     \exp_after:NN \use_none:n
755   \else:

```

```

756   \exp_after:NN \use_arg_i:n
757   \fi:
758   {
759     \err_latex_bug:x {Command~name~'\token_to_string:N #1'~
760                       already~defined!~
761                       Current~meaning:~\token_to_meaning:N #1
762                       }
763   }
764   \*trace>
765   \cs_record_meaning:N#1
766   % \io_put_term:x{Defining~\token_to_string:N #1~on~}%}
767   \io_put_log:x{Defining~\token_to_string:N #1~on~
768               line~\tex_the:D \tex_inputlineno:D}
769   \*trace<
770 }

```

On 2005/11/20 Morten said: I think names for testing if a certain condition is true or false should always contain `if` to avoid confusion. This is what we do for lots of other types of test functions. For now I have defined both names for the functions checking names of control sequences.

`\cs_if_exist_p:N` Expands into `\c_true` if the control sequence given as its argument *is* in use.

```

771 \def:Npn \cs_if_exist_p:N #1{
772   \if:w \cs_if_free_p:N #1
773     \c_false
774   \else:
775     \c_true \fi:}

```

`\chk_if_exist_cs:N` This function issues a warning message when the control sequence in its argument does not exist.
`\chk_if_exist_cs:c`

```

776 \def:Npn \chk_if_exist_cs:N #1 {
777   \if:w \cs_if_exist_p:N #1
778   \else:
779     \err_latex_bug:x {Command~ '\token_to_string:N #1'~
780                     not~ yet~ defined!}
781   \fi:}
782 \def:Npn \chk_if_exist_cs:c #1 {
783   \exp_after:NN \chk_if_exist_cs:N \cs:w #1\cs_end: }

```

`\cs_if_free_p:N` Expands into `\c_true` if the control sequence given as its argument is not yet in use. Note that we make sure to expand into `\c_false` if the control sequence is textually `\c_undefined` or `\scan_stop:`, so that we don't end up (re)defining them.

```

784 \def:Npn \cs_if_free_p:N #1{
785   \if_cs_exist:N #1
786     \if_meaning:NN#1\scan_stop:
787       \if:w\cs_if_eq_name_p:NN #1\scan_stop:
788         \c_false \else: \c_true \fi:
789     \else:
790       \c_false
791   \fi:

```

```

792 \else:
793   \if:w \cs_if_eq_name_p:NN #1\c_undefined
794     \c_false \else: \c_true \fi:
795 \fi:
796 }
797 \let:NwN \cs_free_p:N \cs_if_free_p:N

```

`\str_if_eq_p:nn` Takes 2 lists of characters as arguments and expands into `\c_true` if they are equal, and `\str_if_eq_p_aux:w` `\c_false` otherwise. Note that in the current implementation spaces in these strings are ignored.⁵

```

798 \def:Npn \str_if_eq_p:nn #1#2{
799   \str_if_eq_p_aux:w #1\scan_stop:\#2\scan_stop:\
800 }
801 \def:Npn \str_if_eq_p_aux:w #1#2\#3#4\#{
802   \if_meaning:NN#1#3
803     \if_meaning:NN#1\scan_stop:\c_true \else:
804     \if_meaning:NN#3\scan_stop:\c_false \else:
805     \str_if_eq_p_aux:w #2\#4\#3\fi:\fi:
806   \else:\c_false \fi:}

```

`\cs_if_eq_name_p:NN` An application of the above function, already streamlined for speed, so I put it in here. It takes two control sequences as arguments and expands into true iff they have the same name. We make it long in case one of them is `\par`!

```

807 \def_long:Npn \cs_if_eq_name_p:NN #1#2{
808   \exp_after:NN\exp_after:NN
809   \exp_after:NN\str_if_eq_p_aux:w
810   \exp_after:NN\token_to_string:N
811   \exp_after:NN#1
812   \exp_after:NN\scan_stop:
813   \exp_after:NN\
814   \token_to_string:N#2\scan_stop:\}

```

`\str_if_eq_var_p:nf` A variant of `\str_if_eq_p:nn` which has the advantage of obeying spaces in at least the second argument. See `l3quark` for an application. From the hand of David Kastrup with slight modifications to make it fit with the remainder of the `expl3` language.

The macro builds a string of `\if:w \fi:` pairs from the first argument. The idea is to turn the comparison of `ab` and `cde` into

```

\tex_number:D
  \if:w \scan_stop: \if:w b\if:w a cde\scan_stop: '\fi: \fi: \fi:
13

```

The `'` is important here. If all tests are true, the `'` is read as part of the number in which case the returned number is 13 in octal notation so `\tex_number:D` returns 11. If one test returns false the `'` is never seen and then we get just 13. We wrap the whole process in an external `\if:w` in order to make it return either `\c_true` or `\c_false` since some parts of `l3prg` expect a predicate to return one of these two tokens.

⁵This is a function which could use `\tlist_compare:xx`.

```

815 \def:Npn \str_if_eq_var_p:nf#1{
816   \if:w \tex_number:D\str_if_eq_var_start:nnN{}{}#1\scan_stop:
817 }
818 \def:Npn\str_if_eq_var_start:nnN#1#2#3{
819   \if:w#3\scan_stop:\exp_after:NN\str_if_eq_var_stop:w\fi:
820   \str_if_eq_var_start:nnN{\if:w#3#1}{#2\fi:}
821 }
822 \def:Npn\str_if_eq_var_stop:w\str_if_eq_var_start:nnN#1#2#3{
823   #1#3\scan_stop:'#213~\c_true\else:\c_false\fi:
824 }

```

5.9.5 More new definitions

<pre> \def_new:Npn \def_new:Npx \def_long_new:Npn \def_long_new:Npx \def_protected_new:Npn \def_protected_new:Npx \def_protected_long_new:Npn \def_protected_long_new:Npx </pre>	<p>These are like <code>\def:Npn</code> and <code>\let:NN</code>, but they first check that the argument command is not already in use. You may use <code>\pref_global:D</code>, <code>\pref_long:D</code>, <code>\pref_protected:D</code>, and <code>\tex_outer:D</code> as prefixes.</p> <pre> 825 \def_protected:Npn \def_new:Npn #1{\chk_new_cs:N #1 826 \def:Npn #1} 827 \def_protected:Npn \def_new:Npx #1{\chk_new_cs:N #1 828 \def:Npx #1} 829 \def_protected:Npn \def_long_new:Npn #1{\chk_new_cs:N #1 830 \def_long:Npn #1} 831 \def_protected:Npn \def_long_new:Npx #1{\chk_new_cs:N #1 832 \def_long:Npx #1} 833 \def_protected:Npn \def_protected_new:Npn #1{\chk_new_cs:N #1 834 \def_protected:Npn #1} 835 \def_protected:Npn \def_protected_new:Npx #1{\chk_new_cs:N #1 836 \def_protected:Npx #1} 837 \def_protected:Npn \def_protected_long_new:Npn #1{\chk_new_cs:N #1 838 \def_protected_long:Npn #1} 839 \def_protected:Npn \def_protected_long_new:Npx #1{\chk_new_cs:N #1 840 \def_protected_long:Npx #1} </pre>
<pre> \gdef_new:Npn \gdef_new:Npx \gdef_long_new:Npn \gdef_long_new:Npx \gdef_protected_new:Npn \gdef_protected_new:Npx \gdef_protected_long_new:Npn \gdef_protected_long_new:Npx </pre>	<p>Global versions of the above functions.</p> <pre> 841 \def_protected_new:Npn \gdef_new:Npn #1{\chk_new_cs:N #1 842 \gdef:Npn #1} 843 \def_protected_new:Npn \gdef_new:Npx #1{\chk_new_cs:N #1 844 \gdef:Npx #1} 845 \def_protected_new:Npn \gdef_long_new:Npn #1{\chk_new_cs:N #1 846 \gdef_long:Npn #1} 847 \def_protected_new:Npn \gdef_long_new:Npx #1{\chk_new_cs:N #1 848 \gdef_long:Npx #1} 849 \def_protected_new:Npn \gdef_protected_new:Npn #1{\chk_new_cs:N #1 850 \gdef_protected:Npn #1} 851 \def_protected_new:Npn \gdef_protected_new:Npx #1{\chk_new_cs:N #1 852 \gdef_protected:Npx #1} 853 \def_protected_new:Npn \gdef_protected_long_new:Npn #1{\chk_new_cs:N #1 854 \gdef_protected_long:Npn #1} 855 \def_protected_new:Npn \gdef_protected_long_new:Npx #1{\chk_new_cs:N #1 856 \gdef_protected_long:Npx #1} </pre>

`\def:cpn` Like `\def:Npn` and `\def_new:Npn`, except that the first argument consists of the sequence of characters that should be used to form the name of the desired control sequence (the `c` stands for `cname` argument, see the expansion module.). Global versions are also provided.
`\def:cpx`
`\gdef:cpn`
`\gdef:cpx`
`\def_new:cpn` `\def:cpn⟨string⟩⟨rep-text⟩` will turn `⟨string⟩` into a `cname` and then assign `⟨rep-text⟩` to it by using `\def:Npn`. This means that there might be a parameter string between the two arguments.
`\def_new:cpx`
`\gdef_new:cpn`
`\gdef_new:cpx`

```

857 \def_new:Npn \def:cpn #1{\exp_after:NN \def:Npn \cs:w #1\cs_end:}
858 \def_new:Npn \def:cpx #1{\exp_after:NN \def:Npx \cs:w #1\cs_end:}
859 \def_new:Npn \gdef:cpn #1{\exp_after:NN \gdef:Npn \cs:w #1\cs_end:}
860 \def_new:Npn \gdef:cpx #1{\exp_after:NN \gdef:Npx \cs:w #1\cs_end:}
861 \def_new:Npn \def_new:cpn #1{\exp_after:NN \def_new:Npn \cs:w #1\cs_end:}
862 \def_new:Npn \def_new:cpx #1{\exp_after:NN \def_new:Npx \cs:w #1\cs_end:}
863 \def_new:Npn \gdef_new:cpn #1{\exp_after:NN \gdef_new:Npn \cs:w #1\cs_end:}
864 \def_new:Npn \gdef_new:cpx #1{\exp_after:NN \gdef_new:Npx \cs:w #1\cs_end:}

```

`\def_long:cpn` Variants of the `\def_long:Npn` versions which make a `cname` out of the first arguments.
`\def_long:cpx` We may also do this globally.

```

\gdef_long:cpn
\gdef_long:cpx 865 \def_new:Npn \def_long:cpn #1{\exp_after:NN \def_long:Npn \cs:w #1\cs_end:}
\def_long_new:cpn 866 \def_new:Npn \def_long:cpx #1{
\def_long_new:cpx 867   \exp_after:NN\def_long:Npx\cs:w #1\cs_end:}
\gdef_long_new:cpn 868 \def_new:Npn \gdef_long:cpn #1{
\gdef_long_new:cpx 869   \exp_after:NN \gdef_long:Npn \cs:w #1\cs_end:}
870 \def_new:Npn \gdef_long:cpx #1{
871   \exp_after:NN\gdef_long:Npx\cs:w #1\cs_end:}
872 \def_new:Npn \def_long_new:cpn #1{
873   \exp_after:NN \def_long_new:Npn \cs:w #1\cs_end:}
874 \def_new:Npn \def_long_new:cpx #1{
875   \exp_after:NN \def_long_new:Npx \cs:w #1\cs_end:}
876 \def_new:Npn \gdef_long_new:cpn #1{
877   \exp_after:NN \gdef_long_new:Npn \cs:w #1\cs_end:}
878 \def_new:Npn \gdef_long_new:cpx #1{
879   \exp_after:NN \gdef_long_new:Npx \cs:w #1\cs_end:}

```

`\def_protected:cpn` Variants of the `\def_protected:Npn` versions which make a `cname` out of the first arguments. We may also do this globally.

```

\def_protected:cpx
\gdef_protected:cpn
\gdef_protected:cpx 880 \def_new:Npn \def_protected:cpn #1{
\def_protected_new:cpn 881   \exp_after:NN \def_protected:Npn \cs:w #1\cs_end:}
\def_protected_new:cpx 882 \def_new:Npn \def_protected:cpx #1{
\gdef_protected_new:cpn 883   \exp_after:NN\def_protected:Npx\cs:w #1\cs_end:}
\gdef_protected_new:cpx 884 \def_new:Npn \gdef_protected:cpn #1{
885   \exp_after:NN \gdef_protected:Npn \cs:w #1\cs_end:}
886 \def_new:Npn \gdef_protected:cpx #1{
887   \exp_after:NN\gdef_protected:Npx\cs:w #1\cs_end:}
888 \def_new:Npn \def_protected_new:cpn #1{
889   \exp_after:NN \def_protected_new:Npn \cs:w #1\cs_end:}
890 \def_new:Npn \def_protected_new:cpx #1{
891   \exp_after:NN \def_protected_new:Npx \cs:w #1\cs_end:}
892 \def_new:Npn \gdef_protected_new:cpn #1{
893   \exp_after:NN \gdef_protected_new:Npn \cs:w #1\cs_end:}

```

```

894 \def_new:Npn \gdef_protected_new:cpx #1{
895   \exp_after:NN \gdef_protected_new:Npx \cs:w #1\cs_end:}

```

\def_protected_long:cpn Variants of the \def_protected_long:Npn versions which make a csname out of the first arguments. We may also do this globally.

```

\def_protected_long:cpx
\gdef_protected_long:cpn
\gdef_protected_long:cpx
896 \def_new:Npn \def_protected_long:cpn #1{
897   \exp_after:NN \def_protected_long:Npn \cs:w #1\cs_end:}
\def_protected_long_new:cpn
\def_protected_long_new:cpx
898 \def_new:Npn \def_protected_long:cpx #1{
899   \exp_after:NN \def_protected_long:Npx \cs:w #1\cs_end:}
\gdef_protected_long_new:cpn
\gdef_protected_long_new:cpx
900 \def_new:Npn \gdef_protected_long:cpn #1{
901   \exp_after:NN \gdef_protected_long:Npn \cs:w #1\cs_end:}
902 \def_new:Npn \gdef_protected_long:cpx #1{
903   \exp_after:NN \gdef_protected_long:Npx \cs:w #1\cs_end:}
904 \def_new:Npn \def_protected_long_new:cpn #1{
905   \exp_after:NN \def_protected_long_new:Npn \cs:w #1\cs_end:}
906 \def_new:Npn \def_protected_long_new:cpx #1{
907   \exp_after:NN \def_protected_long_new:Npx \cs:w #1\cs_end:}
908 \def_new:Npn \gdef_protected_long_new:cpn #1{
909   \exp_after:NN \gdef_protected_long_new:Npn \cs:w #1\cs_end:}
910 \def_new:Npn \gdef_protected_long_new:cpx #1{
911   \exp_after:NN \gdef_protected_long_new:Npx \cs:w #1\cs_end:}

```

\def_aux_0:NNn Defining a function with n arguments. First some helper functions.

```

\def_aux_1:NNn
\def_aux_2:NNn 912 \def_new:cpn {def_aux_0:NNn} #1#2 {#1 #2 }
\def_aux_3:NNn 913 \def_new:cpn {def_aux_1:NNn} #1#2 {#1 #2 ##1 }
\def_aux_4:NNn 914 \def_new:cpn {def_aux_2:NNn} #1#2 {#1 #2 ##1##2 }
\def_aux_5:NNn 915 \def_new:cpn {def_aux_3:NNn} #1#2 {#1 #2 ##1##2##3 }
\def_aux_6:NNn 916 \def_new:cpn {def_aux_4:NNn} #1#2 {#1 #2 ##1##2##3##4 }
\def_aux_7:NNn 917 \def_new:cpn {def_aux_5:NNn} #1#2 {#1 #2 ##1##2##3##4##5 }
\def_aux_8:NNn 918 \def_new:cpn {def_aux_6:NNn} #1#2 {#1 #2 ##1##2##3##4##5##6 }
\def_aux_9:NNn 919 \def_new:cpn {def_aux_7:NNn} #1#2 {#1 #2 ##1##2##3##4##5##6##7 }
\def_aux:NNnn 920 \def_new:cpn {def_aux_8:NNn} #1#2 {#1 #2 ##1##2##3##4##5##6##7##8 }
\def_aux:NNnn 921 \def_new:cpn {def_aux_9:NNn} #1#2 {#1 #2 ##1##2##3##4##5##6##7##8##9 }
\def_aux:Ncnn

```

\def_arg_number_error_msg:Nn Then the function itself which checks for the existence of such a helper function. If it doesn't exist, return an error. Otherwise call it to define #2 with the correct number of arguments.

```

922 \def_protected_long_new:Npn \def_aux:NNnn #1#2#3#4 {
923   \cs_if_really_exist:cTF {def_aux_\tex_the:D\etex_numexpr:D #3 :NNn}
924   {
925     \cs_use:c {def_aux_\tex_the:D\etex_numexpr:D #3 :NNn} #1 #2 {#4}
926   }
927   { \def_arg_number_error_msg:Nn #2{#3} }
928 }
929 \def_new:Npn \def_aux:Ncnn #1#2{
930   \exp_after:NN \def_aux:NNnn \exp_after:NN #1 \cs:w #2\cs_end:}

```

The error message.

```

931 \def_new:Npn \def_arg_number_error_msg:Nn #1#2 {
932   \err_latex_bug:x{

```



```

933   You're~ trying~ to~ define~ the~ command~ '\token_to_string:N #1'~
934   with~ \use_arg_i:n{\tex_the:D\etex_numexpr:D #2\scan_stop:} ~
935   arguments~ but~ I~ only~ allow~ 0-9~ arguments.~ I~ can~ probably~
936   not~ help~ you~ here
937 }
938 }

```

```

\def_aux_use_0_parameter: Something similar to \def_aux_9:NNn but for using the parameters.
\def_aux_use_1_parameter:
\def_aux_use_2_parameter: 939 \def:cpn{def_aux_use_0_parameter:}{ }
\def_aux_use_3_parameter: 940 \def:cpn{def_aux_use_1_parameter:}{\{##1\}}
\def_aux_use_4_parameter: 941 \def:cpn{def_aux_use_2_parameter:}{\{##1\}{##2\}}
\def_aux_use_5_parameter: 942 \def:cpn{def_aux_use_3_parameter:}{\{##1\}{##2\}{##3\}}
\def_aux_use_6_parameter: 943 \def:cpn{def_aux_use_4_parameter:}{\{##1\}{##2\}{##3\}{##4\}}
\def_aux_use_7_parameter: 944 \def:cpn{def_aux_use_5_parameter:}{\{##1\}{##2\}{##3\}{##4\}{##5\}}
\def_aux_use_8_parameter: 945 \def:cpn{def_aux_use_6_parameter:}{\{##1\}{##2\}{##3\}{##4\}{##5\}{##6\}}
\def_aux_use_9_parameter: 946 \def:cpn{def_aux_use_7_parameter:}{\{##1\}{##2\}{##3\}{##4\}{##5\}{##6\}{##7\}}
947 \def:cpn{def_aux_use_8_parameter:}{
948   {\{##1\}{##2\}{##3\}{##4\}{##5\}{##6\}{##7\}{##8\}}
949 \def:cpn{def_aux_use_9_parameter:}{
950   {\{##1\}{##2\}{##3\}{##4\}{##5\}{##6\}{##7\}{##8\}{##9\}}

```

```

\def:NNn Defining macros without delimited arguments is now relatively easy. First local and
\def:NNx global versions of the usual \def:Npn operation.
\def:cNn
\def:cNx 951 \def_new:Npn \def:NNn { \def_aux:NNnn \def:Npn }
\gdef:NNn 952 \def_new:Npn \def:NNx { \def_aux:NNnn \def:Npx }
\gdef:NNx 953 \def_new:Npn \def:cNn { \def_aux:Ncnn \def:Npn }
\gdef:cNn 954 \def_new:Npn \def:cNx { \def_aux:Ncnn \def:Npx }
\gdef:cNx 955 \def_new:Npn \gdef:NNn { \def_aux:NNnn \gdef:Npn }
956 \def_new:Npn \gdef:NNx { \def_aux:NNnn \gdef:Npx }
\def_new:NNn 957 \def_new:Npn \gdef:cNn { \def_aux:Ncnn \gdef:Npn }
\def_new:NNx 958 \def_new:Npn \gdef:cNx { \def_aux:Ncnn \gdef:Npx }
\def_new:cNn 959 \def_new:Npn \def_new:NNn { \def_aux:NNnn \def_new:Npn }
\def_new:cNx 960 \def_new:Npn \def_new:NNx { \def_aux:NNnn \def_new:Npx }
\gdef_new:NNn 961 \def_new:Npn \def_new:cNn { \def_aux:Ncnn \def_new:Npn }
\gdef_new:NNx 962 \def_new:Npn \def_new:cNx { \def_aux:Ncnn \def_new:Npx }
\gdef_new:cNn 963 \def_new:Npn \gdef_new:NNn { \def_aux:NNnn \gdef_new:Npn }
\gdef_new:cNx 964 \def_new:Npn \gdef_new:NNx { \def_aux:NNnn \gdef_new:Npx }
965 \def_new:Npn \gdef_new:cNn { \def_aux:Ncnn \gdef_new:Npn }
966 \def_new:Npn \gdef_new:cNx { \def_aux:Ncnn \gdef_new:Npx }

```

```

\def_long:NNn Long versions of the above.
\def_long:NNx
\def_long:cNn 967 \def_new:Npn \def_long:NNn { \def_aux:NNnn \def_long:Npn }
968 \def_new:Npn \def_long:NNx { \def_aux:NNnn \def_long:Npx }
\def_long:cNx 969 \def_new:Npn \def_long:cNn { \def_aux:Ncnn \def_long:Npn }
\gdef_long:NNn 970 \def_new:Npn \def_long:cNx { \def_aux:Ncnn \def_long:Npx }
\gdef_long:NNx 971 \def_new:Npn \gdef_long:NNn { \def_aux:NNnn \gdef_long:Npn }
\gdef_long:cNn 972 \def_new:Npn \gdef_long:NNx { \def_aux:NNnn \gdef_long:Npx }
\gdef_long:cNx 973 \def_new:Npn \gdef_long:cNn { \def_aux:Ncnn \gdef_long:Npn }
\def_long_new:NNn 974 \def_new:Npn \gdef_long:cNx { \def_aux:Ncnn \gdef_long:Npx }
\def_long_new:NNx 975 \def_new:Npn \def_long_new:NNn { \def_aux:NNnn \def_long_new:Npn }
\def_long_new:cNn 976 \def_new:Npn \def_long_new:NNx { \def_aux:NNnn \def_long_new:Npx }
\def_long_new:cNx
\gdef_long_new:NNn
\gdef_long_new:NNx
\gdef_long_new:cNn
\gdef_long_new:cNx

```

```

977 \def_new:Npn \def_long_new:cNn { \def_aux:Ncnn \def_long_new:Npn }
978 \def_new:Npn \def_long_new:cNx { \def_aux:Ncnn \def_long_new:Npx }
979 \def_new:Npn \gdef_long_new:NNn { \def_aux:NNnn \gdef_long_new:Npn }
980 \def_new:Npn \gdef_long_new:NNx { \def_aux:NNnn \gdef_long_new:Npx }
981 \def_new:Npn \gdef_long_new:cNn { \def_aux:Ncnn \gdef_long_new:Npn }
982 \def_new:Npn \gdef_long_new:cNx { \def_aux:Ncnn \gdef_long_new:Npx }

```

\def_protected:NNn Protected versions of the above.

```

\def_protected:NNx
\def_protected:cNn 983 \def_new:Npn \def_protected:NNn { \def_aux:NNnn \def_protected:Npn }
\def_protected:cNx 984 \def_new:Npn \def_protected:NNx { \def_aux:NNnn \def_protected:Npx }
\gdef_protected:NNn 985 \def_new:Npn \def_protected:cNn { \def_aux:Ncnn \def_protected:Npn }
\gdef_protected:NNx 986 \def_new:Npn \def_protected:cNx { \def_aux:Ncnn \def_protected:Npx }
\gdef_protected:cNn 987 \def_new:Npn \gdef_protected:NNn { \def_aux:NNnn \gdef_protected:Npn }
\gdef_protected:cNx 988 \def_new:Npn \gdef_protected:NNx { \def_aux:NNnn \gdef_protected:Npx }
\def_protected_new:NNn 989 \def_new:Npn \gdef_protected:cNn { \def_aux:Ncnn \gdef_protected:Npn }
\def_protected_new:NNx 990 \def_new:Npn \gdef_protected:cNx { \def_aux:Ncnn \gdef_protected:Npx }
\def_protected_new:cNn 991 \def_new:Npn \def_protected_new:NNn { \def_aux:NNnn \def_protected_new:Npn }
\def_protected_new:cNx 992 \def_new:Npn \def_protected_new:NNx { \def_aux:NNnn \def_protected_new:Npx }
\gdef_protected_new:NNn 993 \def_new:Npn \def_protected_new:cNn { \def_aux:Ncnn \def_protected_new:Npn }
\gdef_protected_new:NNx 994 \def_new:Npn \def_protected_new:cNx { \def_aux:Ncnn \def_protected_new:Npx }
\gdef_protected_new:cNn 995 \def_new:Npn \gdef_protected_new:NNn { \def_aux:NNnn \gdef_protected_new:Npn }
\gdef_protected_new:cNx 996 \def_new:Npn \gdef_protected_new:NNx { \def_aux:NNnn \gdef_protected_new:Npx }
997 \def_new:Npn \gdef_protected_new:cNn { \def_aux:Ncnn \gdef_protected_new:Npn }
998 \def_new:Npn \gdef_protected_new:cNx { \def_aux:Ncnn \gdef_protected_new:Npx }

```

\def_protected_long:NNn And finally both long and protected.

```

\def_protected_long:NNx
\def_protected_long:cNn 999 \def_new:Npn \def_protected_long:NNn { \def_aux:NNnn \def_protected_long:Npn }
\def_protected_long:cNx 1000 \def_new:Npn \def_protected_long:NNx { \def_aux:NNnn \def_protected_long:Npx }
\gdef_protected_long:NNn 1001 \def_new:Npn \def_protected_long:cNn { \def_aux:Ncnn \def_protected_long:Npn }
\gdef_protected_long:NNx 1002 \def_new:Npn \def_protected_long:cNx { \def_aux:Ncnn \def_protected_long:Npx }
\gdef_protected_long:cNn 1003 \def_new:Npn \gdef_protected_long:NNn { \def_aux:NNnn \gdef_protected_long:Npn }
\gdef_protected_long:cNx 1004 \def_new:Npn \gdef_protected_long:NNx { \def_aux:NNnn \gdef_protected_long:Npx }
\def_protected_long_new:NNn 1005 \def_new:Npn \gdef_protected_long:cNn { \def_aux:Ncnn \gdef_protected_long:Npn }
\def_protected_long_new:NNx 1006 \def_new:Npn \gdef_protected_long:cNx { \def_aux:Ncnn \gdef_protected_long:Npx }
\def_protected_long_new:cNn 1007 \def_new:Npn \def_protected_long_new:NNn {
\def_protected_long_new:cNx 1008 \def_aux:NNnn \def_protected_long_new:Npn }
\gdef_protected_long_new:NNn 1009 \def_new:Npn \def_protected_long_new:NNx {
\gdef_protected_long_new:NNx 1010 \def_aux:NNnn \def_protected_long_new:Npx }
\gdef_protected_long_new:cNn 1011 \def_new:Npn \def_protected_long_new:cNn {
\gdef_protected_long_new:cNx 1012 \def_aux:Ncnn \def_protected_long_new:Npn }
1013 \def_new:Npn \def_protected_long_new:cNx {
1014 \def_aux:Ncnn \def_protected_long_new:Npx }
1015 \def_new:Npn \gdef_protected_long_new:NNn {
1016 \def_aux:NNnn \gdef_protected_long_new:Npn }
1017 \def_new:Npn \gdef_protected_long_new:NNx {
1018 \def_aux:NNnn \gdef_protected_long_new:Npx }
1019 \def_new:Npn \gdef_protected_long_new:cNn {
1020 \def_aux:Ncnn \gdef_protected_long_new:Npn }
1021 \def_new:Npn \gdef_protected_long_new:cNx {
1022 \def_aux:Ncnn \gdef_protected_long_new:Npx }

```

```

\let:NN These macros allow us to copy the definition of a control sequence to another control
\let:cN sequence.
\let:Nc
\let:cc 1023 \def_protected_long_new:Npn \let:NN #1{
\let_new:NN The = sign allows us to define funny char tokens like .= itself or \_ with this function. For
\let_new:cN the definition of \c_space_chartok{~} to work we need the ~ after the =
\let_new:Nc
\let_new:cc 1024 \let:NwN #1=~}
1025 \def_new:Npn\let:cN #1 {\exp_after:NN\let:NN\cs:w#1\cs_end:}
1026 \def_new:Npn\let:Nc{\exp_args:NNc\let:NN}
1027 \def_new:Npn\let:cc{\exp_args:Ncc\let:NN}
1028 \def_new:Npn \let_new:NN #1{\chk_new_cs:N #1
1029 \let:NN #1}
1030 \def_new:Npn \let_new:cN {\exp_args:Nc \let_new:NN}
1031 \def_new:Npn \let_new:Nc {\exp_args:NNc \let_new:NN}
1032 \def_new:Npn \let_new:cc {\exp_args:Ncc \let_new:NN}

\glet:NN These are global versions of some of the previously defined functions.
\glet:cN
\glet:Nc 1033 \def_protected_new:Npn \glet:NN {\pref_global:D \let:NN}
\glet:cc 1034 \def_protected_new:Npn \glet:Nc {\exp_args:NNc \glet:NN}
1035 \def_protected_new:Npn \glet:cN {\exp_args:Nc \glet:NN}
\glet_new:NN 1036 \def_new:Npn \glet:cc {\exp_args:Ncc \glet:NN}
\glet_new:cN 1037 \def_new:Npn \glet_new:NN #1{\chk_new_cs:N #1
\glet_new:Nc 1038 \tex_global:D\let:NN #1}
\glet_new:cc 1039 \def_new:Npn \glet_new:cN {\exp_args:Nc \glet_new:NN}
1040 \def_new:Npn \glet_new:Nc {\exp_args:NNc \glet_new:NN}
1041 \def_new:Npn \glet_new:cc {\exp_args:Ncc \glet_new:NN}

\def:No \def:No expands its second argument one time before making the definition.
\gdef:No
1042 \def_new:Npn \def:No{\exp_args:NNo\def:Npn}
1043 \def_new:Npn \gdef:No {\exp_args:NNo\gdef:Npn}

\def_test_function_aux:Nnnn We will often be defining several almost identical TF, T and F type functions so it makes
\def_test_function_aux:Nnnx sense for us to define a small function that will do this for us so that we are less likely to
\def_test_function:npx introduce typos (it does tend to happen). By doing it in two steps as below we can still
\def_test_function:npx retain a simple interface where you write the TEX parameters as usual. Just don't do it
\def_long_test_function:npx when you're already within a conditional!
\def_test_function_new:npx I think the ways of exiting conditionals below are as fast as they get. Using
\def_test_function_new:npx \reverse_if:N instead of \else: didn't give any difference I could measure.
\def_test_function_new:npx
\def_long_test_function_new:npx 1044 \def_long_new:Npn \def_test_function_aux:Nnnn #1#2#3#4{
\def_long_test_function_new:npx 1045 #1 {#2TF} #3 {#4
1046 \exp_after:NN\use_arg_i:nn\else:\exp_after:NN\use_arg_ii:nn\fi:}
1047 #1 {#2FT} #3 {#4
1048 \exp_after:NN\use_arg_ii:nn\else:\exp_after:NN\use_arg_i:nn\fi:}
1049 #1 {#2T} #3 {#4
1050 \else:\exp_after:NN\use_none:nn\fi:\use_arg_i:n}
1051 #1 {#2F} #3 {#4
1052 \exp_after:NN\use_none:nn\fi:\use_arg_i:n}}
1053 \def_long_new:Npn \def_test_function_aux:Nnnx #1#2#3#4{

```

```

1054 #1 {#2TF} #3 {#4
1055   \exp_not:n{\exp_after:NN\use_arg_i:nn\else:\exp_after:NN\use_arg_ii:nn\fi:}}
1056 #1 {#2FT} #3 {#4
1057   \exp_not:n{\exp_after:NN\use_arg_ii:nn\else:\exp_after:NN\use_arg_i:nn\fi:}}
1058 #1 {#2T} #3 {#4
1059   \exp_not:n{\else:\exp_after:NN\use_none:nn\fi:\use_arg_i:n}}
1060 #1 {#2F} #3 {#4
1061   \exp_not:n{\exp_after:NN\use_none:nn\fi:\use_arg_i:n}}
1062 \def_long_new:Npn \def_test_function:npx #1#2#{
1063   \def_test_function_aux:Nnnn \def:cpn {#1}{#2}
1064 }
1065 \def_long_new:Npn \def_test_function:npx #1#2#{
1066   \def_test_function_aux:Nnnx \def:cpx {#1}{#2}
1067 }
1068 \def_long_new:Npn \def_long_test_function:npx #1#2#{
1069   \def_test_function_aux:Nnnn \def_long:cpn {#1}{#2}
1070 }
1071 \def_long_new:Npn \def_long_test_function:npx #1#2#{
1072   \def_test_function_aux:Nnnx \def_long:cpx {#1}{#2}
1073 }
1074 \def_long_new:Npn \def_test_function_new:npx #1#2#{
1075   \def_test_function_aux:Nnnn \def_new:cpn {#1}{#2}
1076 }
1077 \def_long_new:Npn \def_long_test_function_new:npx #1#2#{
1078   \def_test_function_aux:Nnnn \def_long_new:cpn {#1}{#2}
1079 }
1080 \def_long_new:Npn \def_test_function_new:npx #1#2#{
1081   \def_test_function_aux:Nnnx \def_new:cpx {#1}{#2}
1082 }
1083 \def_long_new:Npn \def_long_test_function_new:npx #1#2#{
1084   \def_test_function_aux:Nnnx \def_long_new:cpx {#1}{#2}
1085 }

```

5.9.6 Further checking

`\cs_if_free:N`TF The old `\@ifundefined` of L^AT_EX 2.09 is re-implemented in the function `\cs_free:cTF`, again in a way that `\else:` and `\fi:` are removed. In this implementation this is absolutely necessary because functions inside the conditional parts expect to read further input from outside the conditional. Actually, the first part of the code below is more general, since it checks `\csnames` directly and therefore allows both `\scan_stop:` and `\c_undefined`.

```

1086 \def_long_test_function_new:npx {cs_if_free:N}#1{\if:w\cs_if_free_p:N #1}
1087 \let:NN \cs_free:NTF \cs_if_free:NTF
1088 \let:NN \cs_free:NT \cs_if_free:NT
1089 \let:NN \cs_free:NF \cs_if_free:NF

```

`\cs_if_free:cTF` We have to implement the `c` variants ‘by hand’ because a different test is necessary and I don’t want the overhead for the test with `\if:w`. What a mistake Don made by making this a feature of `\cs:w`. If I’m not totally mistaken this feature alone has cost him more than 600\$ for bug-checks.

```

1090 \def_long_test_function_new:npn {cs_if_free:c}#1{
1091   \exp_after:NN \if_meaning:NN \cs:w#1\cs_end: \scan_stop:}
1092 \let:NN \cs_free:cTF \cs_if_free:cTF
1093 \let:NN \cs_free:cT \cs_if_free:cT
1094 \let:NN \cs_free:cF \cs_if_free:cF

```

\cs_if_really_free:cTF These versions are for special control sequences that can only be formed through
\cs_if_really_free:cT \cs:w ... \cs_end:. They do not turn the control sequence formed into \scan_stop:.
\cs_if_really_free:cF

```

1095 \def_long_test_function_new:npn {cs_if_really_free:c}#1{
1096   \reverse_if:N\if_cs_exist:w #1\cs_end:}
1097 \let:NN \cs_really_free:cTF \cs_if_really_free:cTF
1098 \let:NN \cs_really_free:cT \cs_if_really_free:cT
1099 \let:NN \cs_really_free:cF \cs_if_really_free:cF

```

\cs_if_exist:NTF Now the same functions but with reverse logic: test if the control sequence exists.

```

\cs_if_exist:NT
\cs_if_exist:NF 1100 \def_long_test_function_new:npn {cs_if_exist:N}#1{\if:w\cs_if_exist_p:N #1}
\cs_if_exist:cTF 1101 \def_long_test_function_new:npn {cs_if_exist:c}#1{
\cs_if_exist:cT 1102   \exp_after:NN\reverse_if:N
\cs_if_exist:cF 1103   \exp_after:NN \if_meaning:NN \cs:w#1\cs_end: \scan_stop:}
\cs_if_really_exist:cTF 1104 \def_long_test_function_new:npn {cs_if_really_exist:c}#1{
\cs_if_really_exist:cT 1105   \if_cs_exist:w #1\cs_end:}
\cs_if_really_exist:cF

```

5.9.7 Freeing memory

\cs_gundefine:N The following function is used to free the main memory from the definition of some function that isn't in use any longer.

```

1106 \def_new:Npn \cs_gundefine:N #1{\glet:NN #1\c_undefined}

```

5.9.8 Engine specific definitions

\engine_if_aleph:TF In some cases it will be useful to know which engine we're running.

```

1107 \def_test_function_new:npn {engine_if_aleph:}\if_cs_exist:N \aleph_textdir:D}

```

5.9.9 Selecting tokens

\use_arg_i:n This macro grabs its argument and returns it back to the input (with outer braces removed).

```

1108 %\def_long_new:Npn \use_arg_i:n #1{#1}% moved earlier

```

\use:c This macro grabs its argument and returns a csname from it.

```

\cs_use:c
\use:cc 1109 \def_new:Npn \use:c #1{\cs:w #1\cs_end:}
1110 \def_new:Npn \cs_use:c #1 { \cs:w#1\cs_end: }

```

THE NAME IS COMPLETELY WRONG!!!! Morten says: Perhaps this is really `\exp_args:cc` instead?

```
1111 \def_new:Npn \use:cc #1#2
1112 {\cs:w #1\exp_after:NN\cs_end:\cs:w #2\cs_end:}
```

`\use_arg_i:nn` These macros are needed to provide functions with true and false cases, as introduced by Michael some time ago. By using `\exp_after:NN \use_arg_i:nn \else:` constructions it is possible to write code where the true or false case is able to access the following tokens from the input stream, which is not possible if the `\c_true` syntax is used.

```
1113 \def_long_new:Npn \use_arg_i:nn #1#2{#1}
1114 \def_long_new:Npn \use_arg_ii:nn #1#2{#2}
```

`\use_arg_i:nnn` We also need something for picking up arguments from a longer list.

```
\use_arg_ii:nnn
\use_arg_iii:nnn 1115 \def_long_new:NNn \use_arg_i:nnn 3{#1}
\use_arg_i:nnnn 1116 \def_long_new:NNn \use_arg_ii:nnn 3{#2}
\use_arg_ii:nnnn 1117 \def_long_new:NNn \use_arg_iii:nnn 3{#3}
\use_arg_iii:nnnn 1118 \def_long_new:NNn \use_arg_i:nnnn 4{#1}
\use_arg_iiii:nnnn 1119 \def_long_new:NNn \use_arg_ii:nnnn 4{#2}
\use_arg_iv:nnnn 1120 \def_long_new:NNn \use_arg_iii:nnnn 4{#3}
1121 \def_long_new:NNn \use_arg_iv:nnnn 4{#4}
```

`\use_arg_i_ii:nn` And a function for grabbing two arguments and returning them again.

```
1122 \def_long_new:NNn\use_arg_i_ii:nn 2{#1#2}
```

`\use_none_delimit_by_q_nil:w` Functions that gobble everything until they see either `\q_nil` or `\q_stop` resp.

```
\use_none_delimit_by_q_stop:w
1123 \def_long_new:Npn \use_none_delimit_by_q_nil:w #1\q_nil{}
1124 \def_long_new:Npn \use_none_delimit_by_q_stop:w #1\q_stop{}
```

`\use_arg_i_delimit_by_q_nil:nw` Same as above but execute first argument after gobbling. Very useful when you need to skip the rest of a mapping sequence but want an easy way to control what should be expanded next.

```
1125 \def_long_new:Npn \use_arg_i_delimit_by_q_nil:nw #1#2\q_nil{#1}
1126 \def_long_new:Npn \use_arg_i_delimit_by_q_stop:nw #1#2\q_stop{#1}
```

`\use_arg_i_after_fi:nw` Returns the first argument after ending the conditional.

```
\use_arg_i_after_else:nw
\use_arg_i_after_or:nw 1127 \def_long_new:Npn \use_arg_i_after_fi:nw #1\fi:{\fi: #1}
\use_arg_i_after_or:nw 1128 \def_long_new:Npn \use_arg_i_after_else:nw #1\else:#2\fi:{\fi: #1}
\use_arg_i_after_or:nw 1129 \def_long_new:Npn \use_arg_i_after_or:nw #1\or: #2\fi: {\fi:#1}
1130 \def_long_new:Npn \use_arg_i_after_orelse:nw #1 #2#3\fi: {\fi:#1}
```

5.9.10 Gobbling tokens from input

`\use_none:n` To gobble tokens from the input we use a standard naming convention: the number of
`\use_none:nn` tokens gobbled is given by the number of `n`'s following the `:` in the name. Although
`\use_none:nnn` defining `\use_none:nnn` and above as separate calls of `\use_none:n` and `\use_none:nn`
`\use_none:nnnn` is slightly faster, this is very non-intuitive to the programmer who will assume that
`\use_none:nnnnn` expanding such a function once will take care of gobbling all the tokens in one go.
`\use_none:nnnnnn`
`\use_none:nnnnnnn` 1131 `%\def_long_new:NNn \use_none:n 1{}\%` moved earlier
`\use_none:nnnnnnnn` 1132 `\def_long_new:NNn \use_none:nn 2{}\`
`\use_none:nnnnnnnnn` 1133 `\def_long_new:NNn \use_none:nnn 3{}\`
`\use_none:nnnnnnnnnn` 1134 `\def_long_new:NNn \use_none:nnnn 4{}\`
1135 `\def_long_new:NNn \use_none:nnnnn 5{}\`
1136 `\def_long_new:NNn \use_none:nnnnnn 6{}\`
1137 `\def_long_new:NNn \use_none:nnnnnnn 7{}\`
1138 `\def_long_new:NNn \use_none:nnnnnnnn 8{}\`
1139 `\def_long_new:NNn \use_none:nnnnnnnnn 9{}\`

5.9.11 Scratch functions

`\gtmp:w` This function is for global scratch definitions that are used immediately afterwards. It should be used when we need a function that operates on input, i.e. has arguments. If we want to save only some tokens for later use, token-list scratch variables should be used.

1140 `\def_new:Npn \gtmp:w {}`

`\tmp:w` This is a local version of the previous function.

1141 `\def_new:Npn \tmp:w {}`

`\use_noop:` I don't think this function belongs here, but one place is as good as any other. I want to use this function when I want to express 'no operation'.

1142 `\def_new:Npn \use_noop: {}`

5.9.12 Strings and input stream token lists

`\cs_to_str:N` This converts a control sequence into the character string of its name, removing the leading escape character.

1143 `\def_new:Npn \cs_to_str:N {\exp_after:NN\use_none:n \token_to_string:N}`

`\cs_if_eq:NNTF` Check if two control sequences are identical.

`\cs_if_eq:NNT` 1144 `\def_test_function_new:npn {cs_if_eq:NN} #1#2{\if_meaning:NN #1#2}`
`\cs_if_eq:NNF` 1145 `\def_new:Npn \cs_if_eq:cNTF {\exp_args:Nc \cs_if_eq:NNTF}`
`\cs_if_eq:cNTF` 1146 `\def_new:Npn \cs_if_eq:cNT {\exp_args:Nc \cs_if_eq:NNT}`
`\cs_if_eq:cNT` 1147 `\def_new:Npn \cs_if_eq:cNF {\exp_args:Nc \cs_if_eq:NNF}`
`\cs_if_eq:cNF` 1148 `\def_new:Npn \cs_if_eq:NcTF {\exp_args:NNc \cs_if_eq:NNTF}`
`\cs_if_eq:NcTF` 1149 `\def_new:Npn \cs_if_eq:NcT {\exp_args:NNc \cs_if_eq:NNT}`
`\cs_if_eq:NcT` 1150 `\def_new:Npn \cs_if_eq:NcF {\exp_args:NNc \cs_if_eq:NNF}`
`\cs_if_eq:NcF` 1151 `\def_new:Npn \cs_if_eq:ccTF {\exp_args:Ncc \cs_if_eq:NNTF}`
`\cs_if_eq:ccTF` 1152 `\def_new:Npn \cs_if_eq:ccT {\exp_args:Ncc \cs_if_eq:NNT}`
`\cs_if_eq:ccT` 1153 `\def_new:Npn \cs_if_eq:ccF {\exp_args:Ncc \cs_if_eq:NNF}`
`\cs_if_eq:ccF`

Finally some code that is needed as we do not distribute the file module at the moment (so we simply define the needed function via an existing \LaTeX command) and some other stuff which was set up elsewhere, in undistributed modules.

```
1154 \def_new:Npn\file_not_found:nTF #1#2#3{\IfFileExists{#1}{#3}{#2}}
```

5.9.13 Predicates and conditionals

\LaTeX 3 has three concepts for conditional flow processing:

1. Functions that carry out a test and then execute, depending on its result, either the code supplied in the $\langle true\ arg \rangle$ or the $\langle false\ arg \rangle$. These arguments are denoted with T and F respectively. An example would be

```
\cs_free:cTF{abc}{...}{...}
```

a function that will turn the first argument into a control sequence (since its marked as c) then checks whether this control sequence is still free and then depending on the result carry out the code in the second argument (true case) or in the third argument (false case).

2. Functions that return a special type of boolean value which can be tested by the function \if:w . All functions of this type have names that end with $_p$ in the description part. For example

```
\cs_free_p:N
```

would be a predicate function for the same type of test as the function above. It would return ‘true’ if its argument (a single token denoted by N) is still free for definition. It would be used in constructions like

```
\if:w \cs_free_p:N \l_foo_bar ... \else: ... \fi:
```

3. Actually there is a third one, namely the original concept used in plain \TeX . This belongs to the second form but needs further thoughts.

Important to note is that conditionals with $\langle true\ code \rangle$ and/or $\langle false\ code \rangle$ are always defined in a way that the code of the chosen alternative can operate on following tokens in the input stream while the predicate implementations always have an \else: or \fi: interfering. This can be important in scanner implementations.

```
1155 \</initex | package>
1156 \<*showmemory>
1157 \showMemUsage
1158 \</showmemory>
```

6 The chk module

To ensure that functions and variables are properly used certain checking functions are implemented that may or may not be compiled into the final format.

6.1 Functions

<code>\chk_var_or_const:N</code>
<code>\chk_var_or_const:c</code>

`\chk_var_or_const:N <cs>`

Checks that `<cs>` is a proper variable or constant which means that its name starts out with `\L`, `\l`, `\G`, `\g`, `\R`, `\C`, `\c`, or `\q`.

<code>\chk_local:N</code>
<code>\chk_local:c</code>
<code>\chk_global:N</code>
<code>\chk_global:c</code>

`\chk_local:N <cs>`

Checks that `<cs>` is a proper local or global variable. This means that its name starts out with `\L`, `\l`, or `\G`, `\g` respectively.

<code>\chk_local_or_pref_global:N</code>
<code>\pref_global_chk:</code>

To allow implementations where we precede some function with `\pref_global:D` without loosing the possibility to check for the correct variable type the following helper functions can be used: `\chk_local_or_pref_global:N <cs>` is the variable check which is usually let to `\chk_local:N`, i.e. it will check that its argument is a local variable. This behavior will be changed by `\pref_global_chk:`. This function first changes `\chk_local_or_pref_global:N` to check for global variables then it issues a `\pref_global:D`. After use `\chk_local_or_pref_global:N` will restore itself to `\chk_local:N`. So, if we use `\chk_local_or_pref_global:N` inside some function `\foo_bar:n` we can implement a global version `\foo_gbar:n` by defining

```
\def_new:Npn \foo_gbar:n {\pref_global_chk: \foo_bar:n }
```

provided that `\foo_bar:n` is built in a way that prefixing it with `\pref_global:D` turns its operation into a global one. See implementation for details.

6.2 Constants

<code>\c_undefined</code>

This constant is always undefined and therefore can be used to check for free function names.

6.3 Internal functions

<code>\chk_global_aux:w</code>
<code>\chk_local_aux:w</code>
<code>\chk_var_or_const_aux:w</code>

Helper functions that implement the checking.

6.4 The Implementation

We start by ensuring that the required packages are loaded.

```

1159 <package>\ProvidesExplPackage
1160 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
1161 <package>\RequirePackage{l3basics}
1162 <package>\RequirePackage{l3int,l3prg}
1163 <*initex | package>

```

The `\chk` module contains those functions that are used primarily during the development of L^AT_EX3 for checking that things are not mixed up too badly. All these functions are of type ‘e’ since they issue error messages if certain conditions are violated.

6.4.1 Checking variable assignments

`\chk_local:N` This function checks that its argument is a proper local variable, i.e. its name starts with `\l` or `\L`. It is not allowed that the name starts with `\g` or `\G`, which means that we do not allow to update global variables locally. But see `\pref_global_chk:` below for the encoding of functions that might accept global variables in certain situations. Not checked is the case that the argument isn’t a variable at all, i.e. it doesn’t have a `_` as second letter. Maybe we should add this for safety during the implementation since it will find certain errors involving wrong expansion in earlier stage.

```

1164 \def_new:Npn \chk_local:N #1{
1165   \exp_after:NN \chk_local_aux:w \token_to_string:N#1\q_stop}
1166
1167 \def_new:Npn \chk_local_aux:w #1#2#3\q_stop{
1168   \if_num:w\tex_uccode:D'#2='G\scan_stop:
1169     \err_latex_bug:x{Local~mismatch:~local~function~called~with~
1170                       global~variable:~^~J\text_put_four_sp: #1#2#3~
1171                       on~line~\tex_the:D\tex_inputlineno:D}
1172   \else:
1173     \if_num:w\tex_uccode:D'#2='L\scan_stop:
1174     \else:
1175       \err_latex_bug:x{Variable~mismatch:~function~not~called~with~
1176                       proper~variable:~^~J\text_put_four_sp: #1#2#3~
1177                       on~line~\tex_the:D\tex_inputlineno:D}\fi:
1178   \fi:}

```

We set the `\l_iow_new_line_code` at this point, just in case we run into errors.

```

1179 \tex_newlinechar:D='^^J

```

`\chk_global:N` `\chk_global:N` is similar to `\chk_local:N` but looks only for global variables.

```

\chk_global_aux:w
1180 \def_new:Npn \chk_global:N #1{\exp_after:NN
1181   \chk_global_aux:w \token_to_string:N#1\q_stop}
1182 \def_new:Npn \chk_global_aux:w #1#2#3\q_stop{
1183   \if_num:w\tex_uccode:D'#2='L\scan_stop:
1184     \err_latex_bug:x{Global~mismatch:~global~function~called~with~
1185                       local~variable:~#1#2#3~
1186                       on~line~\tex_the:D\tex_inputlineno:D}

```

```

1187 \else:
1188 \if_num:w\tex_uccode:D'#2='G\scan_stop:
1189 \else:
1190 \err_latex_bug:x{Variable~mismatch:~function~not~called~with~
1191 proper~variable:~#1#2#3~
1192 on~line~\tex_the:D\tex_inputlineno:D}\fi:\fi:}

```

`\pref_global_chk:` To allow implementations where we precede some function with `\pref_global:D` without losing the possibility to check for the correct type of a variable, the following helper functions can be used: `\chk_local_or_pref_global:N` *<variable>* is the variable check which is usually `\let:NN` to `\chk_local:N`, i.e. it will check for local variables. This behavior will be changed by `\pref_global_chk:`. This function first changes `\chk_local_or_pref_global:N` to check for global variables, then issues a `\pref_global:D`. After being used, `\chk_local_or_pref_global:N` will restore itself to `\chk_local:N`. So, if we use `\chk_local_or_pref_global:N` inside some function `\foo_bar:n` we can implement a global version `\foo_gbar:n` by defining

```
\def_new:Npn \foo_gbar:n {\pref_global_chk: \foo_bar:n }
```

provided of course, that `\foo_bar:n` is defined in a way that a `\pref_global:D` does work. Such a scheme has to be used carefully, but its advantage is that the checking version has the same structure as a streamlined version, we only have to change `\pref_global_chk:` into `\pref_global:D` and omit the `\chk_local_or_pref_global:N` function in the body.

```

1193 \def_new:Npn \pref_global_chk: {
1194 \gdef:Npn \chk_local_or_pref_global:N ##1{
1195 \chk_global:N ##1
1196 \glet:NN \chk_local_or_pref_global:N \chk_local:N}
1197 \pref_global:D}
1198 \let_new:NN \chk_local_or_pref_global:N \chk_local:N

```

`\chk_var_or_const:N` `\chk_var_or_const:N` is used in situations where we want to check that we have a variable (or a constant) but do not care whether or not it is global (for example, in allocation routines).

```

1199 \def_new:Npn \chk_var_or_const:N #1{\exp_after:NN
1200 \chk_var_or_const_aux:w \token_to_string:N#1\q_stop }
1201 \def_new:Npn \chk_var_or_const_aux:w #1#2#3\q_stop {
1202 \if_num:w\tex_uccode:D'#2='L\scan_stop:
1203 \else:
1204 \if_num:w\tex_uccode:D'#2='G\scan_stop:
1205 \else:
1206 \if_num:w\tex_uccode:D'#2='C\scan_stop:
1207 \else:

```

We also allow the beast to be a quark, i.e. to start with `\q`.

```

1208 \if_charcode:w#2q\scan_stop:
1209 \else:

```

We might also want to allow that it is a user definable variable which means that its name consists of letters only. We could check this by testing that there is no `_`, but this is not implemented so far.

```

1210          \err_latex_bug:x{Variable~mismatch:~function~not~called~with~
1211                  proper~variable:^^J\text_put_four_sp: #1#2#3~
1212                  on~line~\tex_the:D\tex_inputlineno:D}\fi:\fi:\fi:
1213      \fi:}

```

6.4.2 Doing some tracing

`\tracingall` During `\tracingall` we don't want to see all this code coming from the checking functions
`\absolutelytracingall` since something more substantial is probably wrong. Therefore this definition of it turns
`\donotcheck` the functions of as far as possible.

```

1214 \def_new:Npn\donotcheck{
1215   \let:NN \chk_global:N \use_none:n
1216   \let:NN \chk_local:N \use_none:n
1217   \let:NN \chk_local_or_pref_global:N \use_none:n
1218   \let:NN \pref_global_chk: \pref_global:D
1219   \let:NN \chk_new_cs:N \use_none:n
1220   \let:NN \chk_exist_cs:N \use_none:n
1221   \let:NN \chk_var_or_const:N \use_none:n
1222   \let:NN \cs_record_name:N \use_none:n
1223   \let:NN \cs_record_name:c \use_none:n
1224   \let:NN \cs_record_meaning:N \use_none:n
1225   \let:NN \register_record_name:N \use_none:n
1226 }
1227 \def_new:Npn\absolutelytracingall{

```

We do the settings by hand to avoid uninteresting lines in the log file as much as possible.

```

1228   \pref_global:D\g_trace_commands_status\c_two
1229   \pref_global:D\g_trace_statistics_status\c_two
1230   \pref_global:D\g_trace_pages_status\c_one
1231   \pref_global:D\g_trace_output_status\c_one
1232   \pref_global:D\g_trace_chars_status\c_one
1233   \pref_global:D\g_trace_macros_status\c_two
1234   \pref_global:D\g_trace_paragraphs_status\c_one
1235   \pref_global:D\g_trace_restores_status\c_one
1236   \pref_global:D\g_trace_box_breadth_int\c_ten_thousand
1237   \pref_global:D\g_trace_box_depth_int\c_ten_thousand
1238   \pref_global:D\g_trace_online_status\c_one
1239   \tex_errorstopmode:D}
1240 %
1241 % Use LaTeX2e definition for now.
1242 %\def_new:Npn\tracingall{
1243 %   \donotcheck
1244 %   \absolutelytracingall
1245 %}

```

`\tracingoff` This macro turns off all tracing.

```

1246 \def_new:Npn\tracingoff{

```

First we turn off `\g_trace_online_status` so that we get minimal rubbish on the terminal. Of course, in the log file all assignments are shown.

```

1247 \pref_global:D\g_trace_online_status\c_zero
1248 \pref_global:D\g_trace_commands_status\c_zero
1249 \pref_global:D\g_trace_statistics_status\c_zero
1250 \pref_global:D\g_trace_pages_status\c_zero
1251 \pref_global:D\g_trace_output_status\c_zero
1252 \pref_global:D\g_trace_chars_status\c_zero
1253 \pref_global:D\g_trace_macros_status\c_zero
1254 \pref_global:D\g_trace_paragraphs_status\c_zero
1255 \pref_global:D\g_trace_restores_status\c_zero
1256 \pref_global:D\g_trace_box_breadth_int\c_zero
1257 \pref_global:D\g_trace_box_depth_int\c_zero
1258 }

```

6.4.3 Tracing modules

`\traceon` Turning the tracing of modules on or off. (primitive version).

```

\traceoff
1259 <*trace>
1260 \def_new:Npn\traceon#1{
1261   \clist_map_inline:nn {#1}{
1262     \cs_free:cF{g_trace_ ##1 _status}
1263     {\int_gincr:c{g_trace_ ##1 _status}}
1264   }
1265 }
1266 \def_new:Npn\traceoff#1{
1267   \clist_map_inline:nn {#1}{
1268     \cs_free:cF{g_trace_ ##1 _status}
1269     {\int_gdecr:c{g_trace_ ##1 _status}}
1270   }
1271 }
1272 </trace>
1273 <-trace>\let_new:NN\traceon\use_none:n
1274 <-trace>\let_new:NN\traceoff\use_none:n
1275 </initex | package>

```

Show token usage:

```

1276 <*showmemory>
1277 \showMemUsage
1278 </showmemory>

```

7 Token list pointers

L^AT_EX3 stores token lists in so called ‘token list pointers’. Variables of this type get the suffix `tlp` and functions of this type have the prefix `tlp`. To use a token list pointer you simply call the corresponding variable.

Often you find yourself with not a token list pointer but an arbitrary token list which has to undergo certain tests. We will prefix these functions with `tlist`. While token list

pointers are always single tokens, token lists are always surrounded by braces. Perhaps these token lists should have their own module but for now I decided to put them here because there is quite a bit of overlap with token list pointers.

7.1 Functions

<code>\tlp_new:N</code>
<code>\tlp_new:c</code>
<code>\tlp_new:Nn</code>
<code>\tlp_new:cn</code>
<code>\tlp_new:Nx</code>

`\tlp_new:Nn <tlp> { <initial token list> }`

Defines $\langle tlp \rangle$ to be a new variable (or constant) of type token list pointer. $\langle initial\ token\ list \rangle$ is the initial value of $\langle tlp \rangle$. This makes it possible to assign values to a constant token list pointer.

The form `\tlp_new:N` initializes the token list pointer with an empty value.

<code>\tlp_use:N</code>
<code>\tlp_use:c</code>

`\tlp_use:N <tlp>`

Function that inserts the $\langle tlp \rangle$ into the processing stream. Instead of `\tlp_use:N` simply placing the $\langle tlp \rangle$ into the input stream is also supported. `\tlp_use:c` will complain if the $\langle tlp \rangle$ hasn't been declared previously!

<code>\tlp_set:Nn</code>
<code>\tlp_set:Nc</code>
<code>\tlp_set:No</code>
<code>\tlp_set:Nd</code>
<code>\tlp_set:Nf</code>
<code>\tlp_set:Nx</code>
<code>\tlp_gset:Nn</code>
<code>\tlp_gset:Nc</code>
<code>\tlp_gset:No</code>
<code>\tlp_gset:Nd</code>
<code>\tlp_gset:Nx</code>
<code>\tlp_gset:cn</code>
<code>\tlp_gset:cx</code>

`\tlp_set:Nn <tlp> { <token list> }`

Defines $\langle tlp \rangle$ to hold the token list $\langle token\ list \rangle$. Global variants of this command assign the value globally the other variants expand the $\langle token\ list \rangle$ up to a certain level before the assignment or interpret the $\langle token\ list \rangle$ as a character list and form a control sequence out of it.

<code>\tlp_clear:N</code>
<code>\tlp_clear:c</code>
<code>\tlp_gclear:N</code>
<code>\tlp_gclear:c</code>

`\tlp_clear:N <tlp>`

The $\langle tlp \rangle$ is locally or globally cleared. The `c` variants will generate a control sequence name which is then interpreted as $\langle tlp \rangle$ before clearing.

<code>\tlp_clear_new:N</code>
<code>\tlp_clear_new:c</code>
<code>\tlp_gclear_new:N</code>
<code>\tlp_gclear_new:c</code>

`\tlp_clear_new:N <tlp>`

These functions check if $\langle tlp \rangle$ exists. If it does it will be cleared; if it doesn't it will be allocated.

<code>\tlp_put_left:Nn</code>
<code>\tlp_put_left:No</code>
<code>\tlp_put_left:Nx</code>
<code>\tlp_gput_left:Nn</code>
<code>\tlp_gput_left:No</code>
<code>\tlp_gput_left:Nx</code>
<code>\tlp_put_right:Nn</code>
<code>\tlp_put_right:No</code>
<code>\tlp_put_right:Nx</code>
<code>\tlp_put_right:cc</code>
<code>\tlp_gput_right:Nn</code>
<code>\tlp_gput_right:No</code>
<code>\tlp_gput_right:Nx</code>
<code>\tlp_gput_right:cn</code>
<code>\tlp_gput_right:co</code>

`\tlp_put_left:Nn <tlp> { <token list> }`

These functions will append $\langle token list \rangle$ to the left or right of $\langle tlp \rangle$. Assignment is done either locally or globally and $\langle token list \rangle$ might be subject to expansion before assignment.

A word of warning is appropriate here: Token list pointers are implemented as macros and as such currently inherit some of the peculiarities of how T_EX handles $\#s$ in the argument of macros. In particular, the following actions are legal

```
\tlp_set:Nn \l_tmpa_tlp{##1}
\tlp_put_right:Nn \l_tmpa_tlp{##2}
\tlp_set:No \l_tmpb_tlp{\l_tmpa_tlp ##3}
```

x type expansions where macros being expanded contain $\#s$ do not work and will not work until there is an `\expanded` primitive in the engine. If you want them to work you must double $\#s$ another level.

<code>\tlp_set_eq:NN</code>
<code>\tlp_set_eq:Nc</code>
<code>\tlp_set_eq:cN</code>
<code>\tlp_set_eq:cc</code>
<code>\tlp_gset_eq:NN</code>
<code>\tlp_gset_eq:Nc</code>
<code>\tlp_gset_eq:cN</code>
<code>\tlp_gset_eq:cc</code>

`\tlp_set_eq:NN <tlp1> <tlp2>`

Fast form for `\tlp_set:No <tlp1> { <tlp2> }`

when $\langle tlp2 \rangle$ is known to be a variable of type token list pointer.

<code>\tlp_to_str:N</code> <code>\tlp_to_str:c</code>	<code>\tlp_to_str:N <tlp></code>
--	--

This function returns the token list kept in $\langle tlp \rangle$ as a string list with all characters catcoded to ‘other’.

7.2 Predicates and conditionals

<code>\tlp_if_empty_p:N</code> <code>\tlp_if_empty_p:c</code>	<code>\tlp_if_empty_p:N <tlp></code>
--	--

This predicate returns ‘true’ if $\langle tlp \rangle$ is ‘empty’ i.e., doesn’t contain any tokens.

<code>\tlp_if_empty:NTF</code> <code>\tlp_if_empty:NT</code> <code>\tlp_if_empty:NF</code> <code>\tlp_if_empty:cTF</code> <code>\tlp_if_empty:cT</code> <code>\tlp_if_empty:cF</code>	<code>\tlp_if_empty:NTF <tlp> {\true code}\{false code}</code>
--	--

Execute $\langle true\ code \rangle$ if $\langle tlp \rangle$ is empty and $\langle false\ code \rangle$ if it contains any tokens.

<code>\tlp_if_eq_p:NN</code> <code>\tlp_if_eq_p:cN</code> <code>\tlp_if_eq_p:Nc</code> <code>\tlp_if_eq_p:cc</code>	<code>\tlp_if_eq_p:NN <tlp1> <tlp2></code>
--	--

Predicate function which returns ‘true’ if the two token list pointers are identical and ‘false’ otherwise.

<code>\tlp_if_eq:NNTF</code> <code>\tlp_if_eq:NNT</code> <code>\tlp_if_eq:NNF</code> <code>\tlp_if_eq:cNTF</code> <code>\tlp_if_eq:cNT</code> <code>\tlp_if_eq:cNF</code> <code>\tlp_if_eq:NcTF</code> <code>\tlp_if_eq:NcT</code> <code>\tlp_if_eq:NcF</code> <code>\tlp_if_eq:ccTF</code> <code>\tlp_if_eq:ccT</code> <code>\tlp_if_eq:ccF</code>	<code>\tlp_if_eq:NNTF <tlp1> <tlp2> {\true code}\{false code}</code>
--	--

Execute $\langle true\ code \rangle$ if $\langle tlp1 \rangle$ holds the same token list as $\langle tlp2 \rangle$ and $\langle false\ code \rangle$ otherwise.

7.3 Token lists

<code>\tlist_if_eq:nnTF</code> <code>\tlist_if_eq:nnT</code> <code>\tlist_if_eq:nnF</code> <code>\tlist_if_eq:noTF</code> <code>\tlist_if_eq:noT</code> <code>\tlist_if_eq:noF</code>	<code>\tlist_if_eq:nnTF {<tlst1>} {<tlst2>} {<true code>} {<false code>}</code>
--	---

Execute *<true code>* if the two token lists *<tlst1>* and *<tlst2>* are identical.

<code>\tlist_if_empty_p:n</code> <code>\tlist_if_empty_p:o</code> <code>\tlist_if_empty:nTF</code> <code>\tlist_if_empty:nT</code> <code>\tlist_if_empty:nF</code> <code>\tlist_if_empty:oTF</code> <code>\tlist_if_empty:oT</code> <code>\tlist_if_empty:oF</code>	<code>\tlist_if_empty:nTF {<tlst>} {<true code>} {<false code>}</code>
--	--

Execute *<true code>* if *<tlst>* doesn't contain any tokens and *<false code>* otherwise.

<code>\tlist_if_blank_p:n</code> <code>\tlist_if_blank:nTF</code> <code>\tlist_if_blank:nT</code> <code>\tlist_if_blank:nF</code> <code>\tlist_if_blank_p:o</code> <code>\tlist_if_blank:oTF</code> <code>\tlist_if_blank:oT</code> <code>\tlist_if_blank:oF</code>	<code>\tlist_if_blank:nTF {<tlst>} {<true code>} {<false code>}</code>
--	--

Execute *<true code>* if *<tlst>* is blank meaning that it is either empty or contains only blank spaces.

<code>\tlist_to_lowercase:n</code> <code>\tlist_to_uppercase:n</code>	<code>\tlist_to_lowercase:n {<tlst>}</code>
--	---

`\tlist_to_lowercase:n` converts all tokens in *<tlst>* to their lower case representation. Similar for `\tlist_to_uppercase:n`.

T_EXhackers note: These are the T_EX primitives `\lowercase` and `\uppercase` renamed.

<code>\tlist_to_str:n</code>	<code>\tlist_to_str:n {<tlst>}</code>
------------------------------	---

This function turns its argument into a string where all characters have catcode 'other'.

T_EXhackers note: This is the ε -T_EX primitive `\detokenize`.

<code>\tlist_map_function:nN</code>	
<code>\tlist_map_function:NN</code>	<code>\tlist_map_function:nN {<tlst>} <function></code>
<code>\tlist_map_function:cN</code>	<code>\tlist_map_function:NN <tlp> <function></code>

Runs through all elements in a `tlst` from left to right and places `<function>` in front of each element. As this function will also pick up elements in brace groups, the element is returned with braces and hence `<function>` should be a function with a `:n` suffix even though it may very well only deal with a single token. This function uses a purely expandable loop function and will stay so as long as `<function>` is expandable too.

<code>\tlist_map_inline:nn</code>	
<code>\tlist_map_inline:Nn</code>	<code>\tlist_map_inline:nn {<tlst>} {<inline function>}</code>
<code>\tlist_map_inline:cn</code>	<code>\tlist_map_inline:Nn <tlp> {<inline function>}</code>

Allows a syntax like `\tlist_map_inline:nn {<tlst>} {\token_to_string:N ##1}`. This renders it non-expandable though. Remember to double the `#`s for each level.

<code>\tlist_map_variable:nNn</code>	
<code>\tlist_map_variable:NNn</code>	<code>\tlist_map_variable:nNn {<tlst>} <temp> {<action>}</code>
<code>\tlist_map_variable:cNn</code>	<code>\tlist_map_variable:NNn <tlp> <temp> {<action>}</code>

Assigns `<temp>` to each element on `<tlst>` and executes `<action>`. As there is an assignment in this process it is not expandable.

TeXhackers note: This is the L^AT_EX2 function `\@tfor` but with a more sane syntax. Also it works by tail recursion and so is faster as lists grow longer.

<code>\tlist_map_break:w</code>	
<code>\tlist_map_break:w</code>	<code>\tlist_map_break:w</code>

For breaking out of a loop. You should take note of the `:w` as its usage must be precise!

<code>\tlist_reverse:n</code>	<code>\tlist_reverse:n {<token_{12m}</code>
-------------------------------	--

Reverse the token list to result in `<tokenm21. Note that spaces in this token list are gobbled in the process.`

<code>\tlist_elt_count:n</code>	
<code>\tlist_elt_count:o</code>	<code>\tlist_elt_count:n {<token list>}</code>
<code>\tlist_elt_count:N</code>	<code>\tlist_elt_count:N <tlp></code>

Returns the number of elements in the token list. Brace groups encountered count as one element. Note that spaces in this token list are gobbled in the process.

7.3.1 Internal functions

<code>\tlist_map_function_aux:Nn</code>
<code>\tlist_map_inline_aux:Nn</code>
<code>\tlist_map_variable_aux:Nnn</code>

Internal helper functions for the `<tlst>` loops.

`\tlist_if_blank_p_aux:w`

7.4 Variables and constants

`\C_job_name_tlp` Constant that gets the ‘job name’ assigned when $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ starts.

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ hackers note: This is the new name for the primitive `\jobname`. It is a constant that will be set by $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ and can not be overwritten by the package. Therefore the `\C`

`\c_empty_tlp` Constant that is always empty.

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ hackers note: This was named `\@empty` in $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}2$ and `\empty` in plain $\mathrm{T}_{\mathrm{E}}\mathrm{X}$.

`\c_relax_tlp` Constant holding the token that is assigned to a newly created control sequence by $\mathrm{T}_{\mathrm{E}}\mathrm{X}$.

`\l_tmpa_tlp`
`\l_tmpb_tlp`
`\g_tmpa_tlp`
`\g_tmpb_tlp`

Scratch register for immediate use. They are not used by conditionals or predicate functions.

7.4.1 Internal functions

`\l_replace_tlp` Internal register used in the replace functions.

`\l_testa_tlp`
`\l_testb_tlp`
`\g_testa_tlp`
`\g_testb_tlp`

Registers used for conditional processing if the engine doesn’t support arbitrary string comparison.

`\tlp_to_str_aux:w` Function used to implement `\tlp_to_str:N`.

7.5 Search and replace

<pre> \tlp_if_in:NnTF \tlp_if_in:cnTF \tlp_if_in:NnT \tlp_if_in:cnT \tlp_if_in:NnF \tlp_if_in:cnF \tlist_if_in:nnTF \tlist_if_in:onTF </pre>	<pre> \tlp_if_in:NnTF <tlp> { <item> }{ <true code> }{ <false code> } </pre>
--	--

Function that tests if $\langle item \rangle$ is in $\langle tlp \rangle$. Depending on the result either $\langle true code \rangle$ or $\langle false code \rangle$ is executed. Note that $\langle item \rangle$ cannot contain brace groups.

<pre> \tlp_replace_in:Nnn \tlp_replace_in:cnn \tlp_greplace_in:Nnn \tlp_greplace_in:cnn </pre>	<pre> \tlp_replace_in:Nnn <tlp> { <item1> }{ <item2> } </pre>
--	---

Replaces the leftmost occurrence of $\langle item1 \rangle$ in $\langle tlp \rangle$ with $\langle item2 \rangle$ if present, otherwise the $\langle tlp \rangle$ is left untouched.

<pre> \tlp_replace_all_in:Nnn \tlp_replace_all_in:cnn \tlp_greplace_all_in:Nnn \tlp_greplace_all_in:cnn </pre>	<pre> \tlp_replace_all_in:Nnn <tlp> { <item1> }{ <item2> } </pre>
--	---

Replaces *all* occurrences of $\langle item1 \rangle$ in $\langle tlp \rangle$ with $\langle item2 \rangle$.

<pre> \tlp_remove_in:Nn \tlp_remove_in:cn \tlp_gremove_in:Nn \tlp_gremove_in:cn </pre>	<pre> \tlp_remove_in:Nn <tlp> { <item> } </pre>
--	---

Removes the leftmost occurrence of $\langle item \rangle$ from $\langle tlp \rangle$ if present.

<pre> \tlp_remove_all_in:Nn \tlp_remove_all_in:cn \tlp_gremove_all_in:Nn \tlp_gremove_all_in:cn </pre>	<pre> \tlp_remove_all_in:Nn <tlp> { <item> } </pre>
--	---

Removes *all* occurrences of $\langle item \rangle$ from $\langle tlp \rangle$.

7.6 Heads or tails?

Here are some functions for grabbing either the head or tail of a list and perform some tests on it.

<code>\tlist_head:n</code>	
<code>\tlist_tail:n</code>	
<code>\tlist_head_iii:n</code>	
<code>\tlist_head_iii:f</code>	
<code>\tlist_head:w</code>	
<code>\tlist_tail:w</code>	<code>\tlist_head:n { <token1><token2>...<token-n> }</code>
<code>\tlist_head_iii:w</code>	<code>\tlist_tail:n { <token1><token2>...<token-n> }</code>

These functions return either the head or the tail of a list, thus in the above example `\tlist_head:n` would return `<token1>` and `\tlist_tail:n` would return `<token2>...<token-n>`. `\tlist_head_iii:n` returns the first three tokens. The `:w` versions require some care as they use a delimited argument internally.

TeXhackers note: These are the Lisp functions `car` and `cdr` but with L^AT_EX3 names.

<code>\tlist_if_head_eq_meaning_p:nN</code>	
<code>\tlist_if_head_eq_meaning:nNTF</code>	
<code>\tlist_if_head_eq_meaning:nNTF</code>	<code>\tlist_if_head_eq_meaning:nNTF { <token list> } <token></code>
<code>\tlist_if_head_eq_meaning:nNTF</code>	<code>{<true>}{<false>}</code>

Returns `<true>` if the first token in `<token list>` is equal to `<token>` and `<false>` otherwise. The `meaning` version compares the two tokens with `\if_meaning:NN`.

<code>\tlist_if_head_eq_charcode_p:nN</code>	
<code>\tlist_if_head_eq_charcode:nNTF</code>	
<code>\tlist_if_head_eq_charcode:nNTF</code>	<code>\tlist_if_head_eq_charcode:nNTF { <token list> } <token></code>
<code>\tlist_if_head_eq_charcode:nNTF</code>	<code>{<true>}{<false>}</code>

Returns `<true>` if the first token in `<token list>` is equal to `<token>` and `<false>` otherwise. The `meaning` version compares the two tokens with `\if_charcode:w` but it prevents expansion of them. If you want them to expand, you can use an `f` type expansion first (define `\tlist_if_head_eq_charcode:fNTF` or similar).

<code>\tlist_if_head_eq_catcode_p:nN</code>	
<code>\tlist_if_head_eq_catcode:nNTF</code>	
<code>\tlist_if_head_eq_catcode:nNTF</code>	<code>\tlist_if_head_eq_catcode:nNTF { <token list> } <token></code>
<code>\tlist_if_head_eq_catcode:nNTF</code>	<code>{<true>}{<false>}</code>

Returns `<true>` if the first token in `<token list>` is equal to `<token>` and `<false>` otherwise. This version uses `\if_catcode:w` for the test but is otherwise identical to the `charcode` version.

7.7 The Implementation

We start by ensuring that the required packages are loaded.

```
1279 <package>\ProvidesExplPackage
```

```

1280 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
1281 <package&!check>\RequirePackage{l3basics}
1282 <package & check>\RequirePackage{l3chk}

1283 <*initex | package>

```

A token list pointer is a control sequence that holds tokens. The interface is similar to that for token registers, but beware that the behavior vis á vis `\def:Npx` etc. ... is different. (You see this comes from Denys' implementation.)

```

\tlp_new:N We provide one allocation function (which checks that the name is not used) and two
\tlp_new:c clear functions that locally or globally clear the token list. The allocation function has
\tlp_new:Nn two arguments to specify an initial value. This is the only way to give values to constants.
\tlp_new:cn
\tlp_new:Nx 1284 \def_long_new:Npn \tlp_new:Nn #1#2{
1285   \chk_new_cs:N #1

```

If checking we don't allow constants to be defined.

```

1286 <*check>
1287   \chk_var_or_const:N #1
1288 </check>

```

Otherwise any variable type is allowed.

```

1289   \gdef:Npn #1{#2}
1290 }
1291 \def_new:Npn \tlp_new:cn {\exp_args:Nc \tlp_new:Nn }
1292 \def_long_new:Npn \tlp_new:Nx #1#2{
1293   \chk_new_cs:N #1
1294 <check> \chk_var_or_const:N #1
1295   \gdef:Npx #1{#2}
1296 }
1297 \def_new:Npn \tlp_new:N #1{\tlp_new:Nn #1{}}
1298 \def_new:Npn \tlp_new:c #1{\tlp_new:cn {#1}{}}

```

```

\tlp_use:N Perhaps this should just be enabled when checking?
\tlp_use:c

```

```

1299 \def_new:Npn \tlp_use:N #1 {
1300   \if_meaning:NN #1 \scan_stop:

```

If `<tlp>` equals `\scan_stop:` it is probably stemming from a `\cs:w ... \cd_end:` that was created by mistake somewhere.

```

1301     \err_latex_bug:x {Token~list~pointer~ '\token_to_string:N #1'~
1302                      has~ an~ erroneous~ structure!}
1303   \else:
1304     \exp_after:NN #1
1305   \fi:
1306 }
1307 \def_new:Npn \tlp_use:c {\exp_args:Nc \tlp_use:N}

```

`\tlp_set:Nn` To set token lists to a specific value to type of functions are available: `\tlp_set_eq:NN`
`\tlp_set:No` takes two token-lists as its arguments assign the first the contents of the second;
`\tlp_set:Nd` `\tlp_set:Nn` has as its second argument a ‘real’ list of tokens. One can view
`\tlp_set:Nf` `\tlp_set_eq:NN` as a special form of `\tlp_set:No`. Both functions have global counter-
`\tlp_set:Nx` parts.
`\tlp_set:cn` During development we check if the token list that is being assigned to exists. If not, a
`\tlp_set:co` warning will be issued.
`\tlp_set:cx`
`\tlp_gset:Nn` 1308 `*check`
`\tlp_gset:No` 1309 `\def_long_new:Npn \tlp_set:Nn #1#2{`
`\tlp_gset:Nd` 1310 `\chk_exist_cs:N #1 \def:Npn #1{#2}`
`\tlp_gset:Nx` We use `\chk_local_or_pref_global:N` after the assignment to allow constructs with
`\tlp_gset:cn` `\pref_global_chk:.` But one should note that this is less efficient then using the real
`\tlp_gset:cx` global variant since they are built-in.

```

1311 \chk_local_or_pref_global:N #1
1312 }
1313 \def_long_new:Npn \tlp_set:Nx #1#2{
1314 \chk_exist_cs:N #1 \def:Npx #1{#2} \chk_local:N #1
1315 }

```

The the global versions.

```

1316 \def_long_new:Npn \tlp_gset:Nn #1#2{
1317 \chk_exist_cs:N #1 \gdef:Npn #1{#2} \chk_global:N #1
1318 }
1319 \def_long_new:Npn \tlp_gset:Nx #1#2{
1320 \chk_exist_cs:N #1 \gdef:Npx #1{#2} \chk_global:N #1
1321 }
1322 \*check

```

For some functions like `\tlp_set:Nn` we need to define the ‘non-check’ version with arguments since we want to allow constructions like `\tlp_set:Nn\l_tmpa_tlp\foo` and so we can’t use the primitive `\TeX` command.

```

1323 \*!check
1324 \def_long_new:Npn \tlp_set:Nn#1#2{\def:Npn#1{#2}}
1325 \def_long_new:Npn \tlp_set:Nx#1#2{\def:Npx#1{#2}}
1326 \def_long_new:Npn \tlp_gset:Nn#1#2{\gdef:Npn#1{#2}}
1327 \def_long_new:Npn \tlp_gset:Nx#1#2{\gdef:Npx#1{#2}}
1328 \*!check

```

The remaining functions can just be defined with help from the expansion module.

```

1329 \def_new:Npn \tlp_set:No {\exp_args:NNo \tlp_set:Nn}
1330 \def_new:Npn \tlp_set:Nd {\exp_args:NNd \tlp_set:Nn}
1331 \def_new:Npn \tlp_set:Nf {\exp_args:NNf \tlp_set:Nn}
1332 \def_new:Npn \tlp_set:cn {\exp_args:Nc \tlp_set:Nn}
1333 \def_new:Npn \tlp_set:co {\exp_args:Nco \tlp_set:Nn}
1334 \def_new:Npn \tlp_set:cx {\exp_args:Nc \tlp_set:Nx}
1335 \def_new:Npn \tlp_gset:No {\exp_args:NNo \tlp_gset:Nn}
1336 \def_new:Npn \tlp_gset:Nd {\exp_args:NNd \tlp_gset:Nn}
1337 \def_new:Npn \tlp_gset:cn {\exp_args:Nc \tlp_gset:Nn}
1338 \def_new:Npn \tlp_gset:cx {\exp_args:Nc \tlp_gset:Nx}

```

```

\tlp_set_eq:NN For setting token list pointers equal to each other. First checking:
\tlp_set_eq:Nc
\tlp_set_eq:cN 1339 <{*check>
\tlp_set_eq:cc 1340 \def_new:Npn \tlp_set_eq:NN #1#2{
\tlp_set_eq:cc 1341 \chk_exist_cs:N #1 \let:NN #1#2
\tlp_gset_eq:NN 1342 \chk_local_or_pref_global:N #1 \chk_var_or_const:N #2
\tlp_gset_eq:Nc 1343 }
\tlp_gset_eq:cN 1344 \def_new:Npn \tlp_gset_eq:NN #1#2{
\tlp_gset_eq:cc 1345 \chk_exist_cs:N #1 \glet:NN #1#2
1346 \chk_global:N #1 \chk_var_or_const:N #2
1347 }
1348 </check>

```

Non-checking versions are easy.

```

1349 <{*!check>
1350 \let_new:NN \tlp_set_eq:NN \let:NN
1351 \let_new:NN \tlp_gset_eq:NN \glet:NN
1352 </!check>

```

The rest again with the expansion module.

```

1353 \def_new:Npn \tlp_set_eq:Nc {\exp_args:Nnc \tlp_set_eq:NN}
1354 \def_new:Npn \tlp_set_eq:cN {\exp_args:Nc \tlp_set_eq:NN}
1355 \def_new:Npn \tlp_set_eq:cc {\exp_args:Ncc \tlp_set_eq:NN}
1356 \def_new:Npn \tlp_gset_eq:Nc {\exp_args:Nnc \tlp_gset_eq:NN}
1357 \def_new:Npn \tlp_gset_eq:cN {\exp_args:Nc \tlp_gset_eq:NN}
1358 \def_new:Npn \tlp_gset_eq:cc {\exp_args:Ncc \tlp_gset_eq:NN}

```

\tlp_clear:N Clearing a token list pointer.

```

\tlp_clear:c
\tlp_gclear:N 1359 \def_new:Npn \tlp_clear:N #1{\tlp_set_eq:NN #1\c_empty_tlp}
\tlp_gclear:c 1360 \def_new:Npn \tlp_clear:c {\exp_args:Nc \tlp_clear:N}
\tlp_gclear:c 1361 \def_new:Npn \tlp_gclear:N #1{\tlp_gset_eq:NN #1\c_empty_tlp}
1362 \def_new:Npn \tlp_gclear:c {\exp_args:Nc \tlp_gclear:N}

```

\tlp_clear_new:N These macros check whether a token list exists. If it does it is cleared, if it doesn't it is allocated.

```

1363 <{*check>
1364 \def_new:Npn \tlp_clear_new:N #1{
1365 \chk_var_or_const:N #1
1366 \if:w \cs_exist_p:N #1
1367 \tlp_clear:N #1
1368 \else:
1369 \tlp_new:Nn #1{}
1370 \fi:
1371 }
1372 </check>
1373 <-check>\let_new:NN \tlp_clear_new:N \tlp_clear:N
1374 \def_new:Npn \tlp_clear_new:c {\exp_args:Nc \tlp_clear_new:N}

```

\tlp_gclear_new:N These are the global versions of the above.

\tlp_gclear_new:c


```

1375 <*check>
1376 \def_new:Npn \tlp_gclear_new:N #1{
1377   \chk_var_or_const:N #1
1378   \if:w \cs_exist_p:N #1
1379     \tlp_gclear:N #1
1380   \else:
1381     \tlp_new:Nn #1{}
1382   \fi:}
1383 </check>
1384 <-check>\let_new:NN \tlp_gclear_new:N \tlp_gclear:N
1385 \def_new:Npn \tlp_gclear_new:c {\exp_args:Nc \tlp_gclear_new:N}

```

```

\tlp_put_left:Nn We can add tokens to the left (either globally or locally). It is not quite as easy as we
\tlp_put_left:No would like because we have to ensure the assignments
\tlp_put_left:Nx
\tlp_gput_left:Nn \tlp_set:Nn \l_tmpa_tlp{##1abc##2def}
\tlp_gput_left:No \tlp_set:Nn \l_tmpb_tlp{##1abc}
\tlp_gput_left:Nx \tlp_put_right:Nn \l_tmpb_tlp {##2def}

```

cause `\l_tmpa_tlp` and `\l_tmpb_tlp` to be identical. The old code did not succeed in doing this (it gave an error) and so we use a different technique where the item(s) to be added are first stored in a temporary pointer and then added using an `x` type expansion combined with the appropriate level of non-expansion. Putting the tokens directly into one assignment does not work unless we want full expansion. Note (according to the warning earlier) \TeX does not allow us to treat `#s` the same in all cases. Tough.

```

1386 \def_long_new:Npn \tlp_put_left:Nn #1#2{
1387   \tlp_set:Nn \l_exp_tlp{#2}
1388   \tlp_set:Nx #1{\exp_not:o{\l_exp_tlp}\exp_not:o{#1}}
1389 <check> \chk_local_or_pref_global:N #1
1390 }
1391 \def_long_new:Npn \tlp_put_left:No #1#2{
1392   \tlp_set:Nn \l_exp_tlp{#2}
1393   \tlp_set:Nx #1{\exp_not:d{\l_exp_tlp}\exp_not:o{#1}}
1394 <check> \chk_local_or_pref_global:N #1
1395 }
1396 \def_long_new:Npn \tlp_put_left:Nx #1#2{
1397   \tlp_set:Nx #1{#2\exp_not:o{#1}}
1398 <check> \chk_local_or_pref_global:N #1
1399 }
1400 \def_long_new:Npn \tlp_gput_left:Nn #1#2{
1401   \tlp_set:Nn \l_exp_tlp{#2}
1402   \tlp_gset:Nx #1{\exp_not:o{\l_exp_tlp}\exp_not:o{#1}}
1403 <check> \chk_local_or_pref_global:N #1
1404 }
1405 \def_long_new:Npn \tlp_gput_left:No #1#2{
1406   \tlp_set:Nn \l_exp_tlp{#2}
1407   \tlp_gset:Nx #1{\exp_not:d{\l_exp_tlp}\exp_not:o{#1}}
1408 <check> \chk_local_or_pref_global:N #1
1409 }
1410 \def_long_new:Npn \tlp_gput_left:Nx #1#2{
1411   \tlp_gset:Nx #1{#2\exp_not:o{#1}}

```

```

1412 <check> \chk_local_or_pref_global:N #1
1413 }
1414 \def_long_new:Npn \tlp_put_left:cn{\exp_args:Nc\tlp_put_left:Nn}
1415 \def_long_new:Npn \tlp_put_left:co{\exp_args:Nc\tlp_put_left:No}
1416 \def_long_new:Npn \tlp_put_left:cx{\exp_args:Nc\tlp_put_left:Nx}
1417 \def_long_new:Npn \tlp_gput_left:cn{\exp_args:Nc\tlp_gput_left:Nn}
1418 \def_long_new:Npn \tlp_gput_left:co{\exp_args:Nc\tlp_gput_left:No}
1419 \def_long_new:Npn \tlp_gput_left:cx{\exp_args:Nc\tlp_gput_left:Nx}

```

\tlp_put_right:Nn These are variants of the functions above, but for adding tokens to the right.

```

\tlp_put_right:No
\tlp_put_right:Nx 1420 \def_long_new:Npn \tlp_put_right:Nn #1#2{
\tlp_put_right:Nx 1421 \tlp_set:Nn \l_exp_tlp{#2}
\tlp_put_right:cc 1422 \tlp_set:Nx #1{\exp_not:o{#1}\exp_not:o{\l_exp_tlp}}
\tlp_gput_right:Nn 1423 <check> \chk_local_or_pref_global:N #1
\tlp_gput_right:No 1424 }
\tlp_gput_right:cn 1425 \def_long_new:Npn \tlp_gput_right:Nn #1#2{
\tlp_gput_right:co 1426 \tlp_set:Nn \l_exp_tlp{#2}
\tlp_gput_right:Nx 1427 \tlp_gset:Nx #1{\exp_not:o{#1}\exp_not:o{\l_exp_tlp}}
1428 <check> \chk_local_or_pref_global:N #1
1429 }
1430 \def_long_new:Npn \tlp_put_right:No #1#2{
1431 \tlp_set:Nn \l_exp_tlp{#2}
1432 \tlp_set:Nx #1{\exp_not:o{#1}\exp_not:d{\l_exp_tlp}}
1433 <check> \chk_local_or_pref_global:N #1
1434 }
1435 \def_long_new:Npn \tlp_gput_right:No #1#2{
1436 \tlp_set:Nn \l_exp_tlp{#2}
1437 \tlp_gset:Nx #1{\exp_not:o{#1}\exp_not:d{\l_exp_tlp}}
1438 <check> \chk_local_or_pref_global:N #1
1439 }
1440 \def_long:Npn \tlp_put_right:Nx #1#2{
1441 \tlp_set:Nx #1{\exp_not:o{#1}#2}
1442 <check> \chk_local_or_pref_global:N #1
1443 }
1444 \def_long:Npn \tlp_gput_right:Nx #1#2{
1445 \tlp_gset:Nx #1{\exp_not:o{#1}#2}
1446 <check> \chk_local_or_pref_global:N #1
1447 }
1448 \def_new:Npn \tlp_gput_right:cn {\exp_args:Nc \tlp_gput_right:Nn}
1449 \def_new:Npn \tlp_gput_right:co {\exp_args:Nc \tlp_gput_right:No}
1450 \def_new:Npn \tlp_put_right:cc {\exp_args:Ncc \tlp_put_right:Nn}

```

\tlp_gset:Nc These two functions are included because they are necessary in Denys' implementations.

\tlp_set:Nc The :Nc convention (see the expansion module) is very unusual at first sight, but it works nicely over all modules, so we would like to keep it.

Construct a control sequence on the fly from #2 and save it in #1.

```

1451 \def_new:Npn \tlp_gset:Nc {
1452 *check>
1453 \pref_global_chk:
1454 </check>
1455 <-check> \pref_global:D
1456 \tlp_set:Nc}

```

`\pref_global_chk`: will turn the variable check in `\tlp_set:No` into a global check.

```
1457 \def_new:Npn \tlp_set:Nc #1#2{\tlp_set:No #1{\cs:w#2\cs_end:}}
```

We also provide a few conditionals, both in expandable form (with `\c_true`) and in ‘brace-form’, the latter are denoted by TF at the end, as explained elsewhere.

`\tlp_if_empty_p:N` Returns `\c_true` iff the token list in the argument is empty.

```
\tlp_if_empty_p:c
1458 \def_new:Npn \tlp_if_empty_p:N #1{
1459   \if_meaning:NN#1\c_empty_tlp \c_true \else: \c_false \fi:}
1460 \def_new:Npn \tlp_if_empty_p:c {\exp_args:Nc\tlp_if_empty_p:N}
```

`\tlp_if_empty:N` These functions check whether the token list in the argument is empty and execute the proper code from their argument(s).

```
\tlp_if_empty:N
1461 \def_test_function_new:npn {tlp_if_empty:N} #1{
\tlp_if_empty:cTF 1462   \if_meaning:NN#1\c_empty_tlp}
\tlp_if_empty:cT 1463 \def_new:Npn \tlp_if_empty:cTF {\exp_args:Nc \tlp_if_empty:N}TF}
\tlp_if_empty:cF 1464 \def_new:Npn \tlp_if_empty:cT {\exp_args:Nc \tlp_if_empty:N}T}
1465 \def_new:Npn \tlp_if_empty:cF {\exp_args:Nc \tlp_if_empty:N}F}
```

`\tlp_if_eq_p:NN` Returns `\c_true` iff the two token list pointers are equal.

```
\tlp_if_eq_p:Nc
1466 \def_new:Npn \tlp_if_eq_p:NN #1#2{
\tlp_if_eq_p:cN 1467   \if_meaning:NN#1#2 \c_true \else: \c_false \fi:}
\tlp_if_eq_p:cc 1468 \def_new:Npn \tlp_if_eq_p:Nc {\exp_args:Nnc\tlp_if_empty_p:NN}
1469 \def_new:Npn \tlp_if_eq_p:cN {\exp_args:Nc\tlp_if_empty_p:NN}
1470 \def_new:Npn \tlp_if_eq_p:cc {\exp_args:Ncc\tlp_if_empty_p:NN}
```

`\tlp_if_eq:NNTF` These function tests whether the token list pointers that are in its first two arguments are equal.

```
\tlp_if_eq:NNTF 1471 \def_test_function_new:npn {tlp_if_eq:NN} #1#2{\if_meaning:NN#1#2}
\tlp_if_eq:NcTF 1472 \def_new:Npn \tlp_if_eq:cNTF{\exp_args:Nc \tlp_if_eq:NNTF}
\tlp_if_eq:NcT 1473 \def_new:Npn \tlp_if_eq:cNT {\exp_args:Nc \tlp_if_eq:NNT}
\tlp_if_eq:NcF 1474 \def_new:Npn \tlp_if_eq:cNF {\exp_args:Nc \tlp_if_eq:NNTF}
\tlp_if_eq:cNTF 1475 \def_new:Npn \tlp_if_eq:NcTF{\exp_args:Nnc \tlp_if_eq:NNTF}
\tlp_if_eq:cNT 1476 \def_new:Npn \tlp_if_eq:NcT {\exp_args:Nnc \tlp_if_eq:NNT}
\tlp_if_eq:cNF 1477 \def_new:Npn \tlp_if_eq:NcF {\exp_args:Nnc \tlp_if_eq:NNTF}
\tlp_if_eq:ccTF 1478 \def_new:Npn \tlp_if_eq:ccTF{\exp_args:Ncc \tlp_if_eq:NNTF}
\tlp_if_eq:ccT 1479 \def_new:Npn \tlp_if_eq:ccT {\exp_args:Ncc \tlp_if_eq:NNT}
\tlp_if_eq:ccF 1480 \def_new:Npn \tlp_if_eq:ccF {\exp_args:Ncc \tlp_if_eq:NNTF}
```

`\c_empty_tlp` Two constants which are often used.

```
\c_relax_tlp
1481 \tlp_new:Nn \c_empty_tlp {}
1482 \tlp_new:Nn \c_relax_tlp {\scan_stop:}
```

`\g_tmpa_tlp` Global temporary token list pointers. They are supposed to be set and used immediately, with no delay between the definition and the use because you can’t count on other macros not to redefine them from under you.

```
1483 \tlp_new:Nn \g_tmpa_tlp{}
1484 \tlp_new:Nn \g_tmpb_tlp{}
```

\l_testa_tlp Global and local temporaries. These are the ones for test routines. This means that one
 \l_testb_tlp can safely use other temporaries when calling test routines.

```
\g_testa_tlp
\g_testb_tlp 1485 \tlp_new:Nn \l_testa_tlp {}
              1486 \tlp_new:Nn \l_testb_tlp {}
              1487 \tlp_new:Nn \g_testa_tlp {}
              1488 \tlp_new:Nn \g_testb_tlp {}
```

\l_tmpa_tlp These are local temporary token list pointers.

```
\l_tmpb_tlp
              1489 \tlp_new:Nn \l_tmpa_tlp{}
              1490 \tlp_new:Nn \l_tmpb_tlp{}
```

\tlp_to_str:N These functions return the replacement text of a token list as a string list with all char-
 \tlp_to_str:c acters catcoded to ‘other’.

```
\tlp_to_str_aux:w
              1491 \def_new:Npn \tlp_to_str:N {\exp_after:NN\tlp_to_str_aux:w
              1492   \token_to_meaning:N}
              1493 \def_new:Npn \tlp_to_str_aux:w #1>{}
              1494 \def_new:Npn\tlp_to_str:c{\exp_args:Nc\tlp_to_str:N}
```

\tlist_if_empty_p:n It would be tempting to just use \if_meaning:NN\q_nil#1\q_nil as a test since this
 works really well. However it fails on a token list starting with \q_nil of course but
 more troubling is the case where argument is a complete conditional such as \if_true:
 a \else: b \fi: because then \if_true: is used by \if_meaning:NN, the test turns
 out false, the \else: executes the false branch, the \fi: ends it and the \q_nil at the
 end starts executing... A safer route is to convert the entire token list into harmless
 characters first and then compare that. This way the test will even accept \q_nil as the
 first token.

```
1495 \def_long_new:Npn \tlist_if_empty_p:n #1{
1496   \exp_after:NN\if_meaning:NN\exp_after:NN\q_nil\tlist_to_str:n{#1}\q_nil
1497   \c_true
1498   \else:
1499   \c_false
1500   \fi:
1501 }
```

\tlist_if_empty_p:o

\tlist_if_empty:nTF

```
\tlist_if_empty:nT 1502 \def_new:Npn \tlist_if_empty_p:o {\exp_args:No\tlist_if_empty_p:n}
```

```
\tlist_if_empty:nF 1503 \def_long_test_function_new:npn\tlist_if_empty:n}{#1{
```

```
\tlist_if_empty:nF 1504   \if:w\tlist_if_empty_p:n{#1}}
```

```
\tlist_if_empty:oTF 1505 \def_long_test_function_new:npn\tlist_if_empty:o}{#1{
```

```
\tlist_if_empty:oT 1506   \if:w\tlist_if_empty_p:o{#1}}
```

\tlist_if_empty:oF

\tlist_if_blank_p:n This is based on the answers in “Around the Bend No 2” but is safer as the tests listed
 \tlist_if_blank_p_aux:w there all have one small flaw: If the input in the test is two tokens with the same
 meaning as the internal delimiter, they will fail since one of them is mistaken for the actual
 delimiter. In our version below we make sure to pass the input through \tlist_to_str:n
 which ensures that all the tokens are converted to catcode 12. However we use an a with

catcode 11 as delimiter so we can *never* get into the same problem as the solutions in “Around the Bend No 2”.

```
1507 \def_long_new:Npn \tlist_if_blank_p:n #1{
1508   \exp_after:NN\tlist_if_blank_p_aux:w\tlist_to_str:n{#1}aa..\q_nil
1509 }
1510 \def_new:Npn \tlist_if_blank_p_aux:w #1#2a#3#4\q_nil{
1511   \if_meaning:NN #3#4\c_true\else:\c_false\fi:}
```

```
\tlist_if_blank:nTF Variations on the original function above.
\tlist_if_blank:nT
\tlist_if_blank:nF 1512 \def_long_test_function_new:npn{\tlist_if_blank:n}#1{
\tlist_if_blank_p:o 1513   \if:w\tlist_if_blank_p:n{#1}}
\tlist_if_blank:oTF 1514 \def:Npn \tlist_if_blank_p:o{\exp_args:No\tlist_if_blank_p:n}
\tlist_if_blank:oT 1515 \def_long_test_function_new:npn{\tlist_if_blank:o}#1{
\tlist_if_blank:oF 1516   \if:w\tlist_if_blank_p:o{#1}}
```

```
\tlist_to_lowercase:n Just some names for a few primitives.
\tlist_to_uppercase:n
1517 \let_new:NN \tlist_to_lowercase:n \tex_lowercase:D
1518 \let_new:NN \tlist_to_uppercase:n \tex_uppercase:D
```

```
\tlist_to_str:n Another name for a primitive.
1519 \let_new:NN \tlist_to_str:n \etex_detokenize:D
```

```
\tlist_map_function:nN Expandable loop macro for tlists. These have the advantage of not needing to test if
\tlp_map_function:NN the argument is empty, because if it is, the stop marker will be read immediately and the
\tlp_map_function:cN loop terminated.
\tlist_map_function_aux:NN
1520 \def_long_new:Npn \tlist_map_function:nN #1#2{
1521   \tlist_map_function_aux:Nn #2 #1 \q_recursion_tail \q_recursion_stop
1522 }
1523 \def_new:Npn \tlp_map_function:NN #1#2{
1524   \exp_after:NN \tlist_map_function_aux:Nn
1525   \exp_after:NN #2 #1 \q_recursion_tail \q_recursion_stop
1526 }
1527 \def_long_new:Npn \tlist_map_function_aux:Nn #1#2{
1528   \quark_if_recursion_tail_stop:n{#2}
1529   #1{#2} \tlist_map_function_aux:Nn #1
1530 }
1531 \def_new:Npn\tlp_map_function:cN{\exp_args:Nc\tlp_map_function:NN}
```

```
\tlist_map_inline:nn The inline functions are straight forward by now. We use a little a trick with the fake
\tlp_map_inline:Nn counter \g_tlp_inline_level_num to make them nestable.6 We can also make use of
\tlp_map_inline:cN \tlist_map_function:Nn from before.
\tlist_map_inline_aux:n
\g_tlp_inline_level_num 1532 \def_long_new:Npn \tlist_map_inline:nn #1#2{
1533   \num_gincr:N \g_tlp_inline_level_num
1534   \gdef_long:cpn {tlist_map_inline_ \num_use:N \g_tlp_inline_level_num :n}
1535   ##1{#2}
```

⁶This should be a proper integer, but I don’t want to mess with the dependencies right now...

```

1536 \exp_args:Nc \tlist_map_function_aux:Nn
1537 {tlist_map_inline_ \num_use:N \g_tlp_inline_level_num :n}
1538 #1 \q_recursion_tail\q_recursion_stop
1539 \num_gdecr:N \g_tlp_inline_level_num
1540 }
1541 \def_long_new:Npn \tlist_map_inline:Nn #1#2{
1542 \num_gincr:N \g_tlp_inline_level_num
1543 \gdef_long:cpn {tlist_map_inline_ \num_use:N \g_tlp_inline_level_num :n}
1544 ##1{#2}
1545 \exp_args:NcE \tlist_map_function_aux:Nn
1546 {tlist_map_inline_ \num_use:N \g_tlp_inline_level_num :n}
1547 #1 \q_recursion_tail\q_recursion_stop
1548 \num_gdecr:N \g_tlp_inline_level_num
1549 }
1550 \def_new:Npn\tlist_map_inline:cN{\exp_args:Nc\tlist_map_inline:NN}
1551 \tlist_map_new:Nn \g_tlp_inline_level_num{0}

```

\tlist_map_variable:nNn \tlist_map:nNn <tlist> <temp> <action> assigns <temp> to each element and executes <action>.

```

\tlist_map_variable:cNn
1552 \def_long_new:Npn \tlist_map_variable:nNn #1#2#3{
1553 \tlist_map_variable_aux:Nnn #2 {#3} #1 \q_recursion_tail \q_recursion_stop
1554 }
1555 \def_new:Npn \tlist_map_variable:NNn {\exp_args:No \tlist_map_variable:nNn}
1556 \def_new:Npn \tlist_map_variable:cNn {\exp_args:Nc \tlist_map_variable:NNn}

```

\tlist_map_variable_aux:Nnn The general loop. Assign the temp variable #1 to the current item #3 and then check if that's the stop marker. If it is, break the loop. If not, execute the action #2 and continue.

```

1557 \def_long_new:Npn \tlist_map_variable_aux:Nnn #1#2#3{
1558 \tlist_map_set:Nn #1{#3}
1559 \quark_if_recursion_tail_stop:N #1
1560 #2 \tlist_map_variable_aux:Nnn #1{#2}
1561 }

```

\tlist_map_break:w The break statement.

```

\tlist_map_break:w
1562 \let_new:NN \tlist_map_break:w \use_none_delimit_by_q_recursion_stop:w
1563 \let_new:NN \tlist_map_break:w \tlist_map_break:w

```

\tlist_elt_count:n Count number of elements within a token list or token list pointer. Brace groups within

\tlist_elt_count:o the list are read as a single element. \num_elt_count:n grabs the element and replaces

\tlist_elt_count:n it by +1. The 0 to ensure it works on an empty list.

```

1564 \def_long_new:Npn \tlist_elt_count:n #1{
1565 \num_value:w \num_eval:w 0
1566 \tlist_map_function:nN {#1}\num_elt_count:n
1567 \num_eval_end:
1568 }
1569 \def_new:Npn \tlist_elt_count:o {\exp_args:No\tlist_elt_count:n}
1570 \def_new:Npn \tlist_elt_count:N #1{
1571 \num_value:w \num_eval:w 0
1572 \tlist_map_function:NN #1 \num_elt_count:n
1573 \num_eval_end:
1574 }

```

`\tlist_if_eq:nnTF` Test if two token lists are identical. pdfTeX contains a most interesting primitive for expandable string comparison so we make use of it if available. Presumably it will be in the final version.

Firstly we give it an appropriate name. Note that this primitive actually performs an `x` type expansion but it is still expandable! Hence we must program these functions backwards to add `\exp_not:n`. We provide the combinations for the types `n`, `o` and `x`.

```

1575 \let_new:NN \tlist_compare:xx \pdfstrcmp
1576 \def_long_new:NNn \tlist_compare:nn 2{
1577   \tlist_compare:xx{\exp_not:n{#1}}{\exp_not:n{#2}}
1578 }
1579 \def_long_new:NNn \tlist_compare:nx 1{
1580   \tlist_compare:xx{\exp_not:n{#1}}
1581 }
1582 \def_long_new:NNn \tlist_compare:xn 2{
1583   \tlist_compare:xx{#1}{\exp_not:n{#2}}
1584 }
1585 \def_long_new:NNn \tlist_compare:no 2{
1586   \tlist_compare:xx{\exp_not:n{#1}}{\exp_not:n\exp_after:NN{#2}}
1587 }
1588 \def_long_new:NNn \tlist_compare:on 2{
1589   \tlist_compare:xx{\exp_not:n\exp_after:NN{#1}}{\exp_not:n{#2}}
1590 }
1591 \def_long_new:NNn \tlist_compare:oo 2{
1592   \tlist_compare:xx{\exp_not:n\exp_after:NN{#1}}{\exp_not:n\exp_after:NN{#2}}
1593 }
1594 \def_long_new:NNn \tlist_compare:xo 2{
1595   \tlist_compare:xx{#1}{\exp_not:n\exp_after:NN{#2}}
1596 }
1597 \def_long_new:NNn \tlist_compare:ox 2{
1598   \tlist_compare:xx{\exp_not:n\exp_after:NN{#1}}{\exp_not:n{#2}}
1599 }

```

Since we have a lot of basically identical functions to define we define one to define the rest. Unfortunately we aren't quite set up to use the new `\tlist_map_inline:nn` function yet.

```

1600 \def:Npn \tmp:w #1{
1601   \def_long_new:cNx {tlist_if_eq_p:#1} 2{
1602     \exp_not:N \if_num:w
1603     \exp_after:NN \exp_not:N \cs:w tlist_compare:#1\cs_end:{##1}{##2}
1604     \exp_not:n{ =\c_zero \c_true \else: \c_false \fi: }
1605   }
1606   \def_long_test_function_new:npx{tlist_if_eq:#1}##1##2{
1607     \exp_not:N \if_num:w
1608     \exp_after:NN \exp_not:N \cs:w tlist_compare:#1\cs_end:{##1}{##2}
1609     \exp_not:n{ =\c_zero }
1610   }
1611 }
1612 \tmp:w{xx} \tmp:w{nn} \tmp:w{oo} \tmp:w{xn} \tmp:w{nx}
1613 \tmp:w{on} \tmp:w{no} \tmp:w{xo} \tmp:w{ox}

```

However all of this only makes sense if we actually have that primitive. Therefore we disable it again if it is not there and define `\tlist_if_eq:nn` the old fashioned (and

unexpandable) way.

```

1614 \cs_if_really_free:cT{pdf_strcmp:D}{
1615   \def_long_test_function:npn{tlist_if_eq:nn}#1#2{
1616     \tlp_set:Nx \l_testa_tlp {\exp_not:n{#1}}
1617     \tlp_set:Nx \l_testb_tlp {\exp_not:n{#2}}
1618     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1619   }
1620   \def_long_test_function:npn{tlist_if_eq:no}#1#2{
1621     \tlp_set:Nx \l_testa_tlp {\exp_not:n{#1}}
1622     \tlp_set:Nx \l_testb_tlp {\exp_not:o{#2}}
1623     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1624   }
1625   \def_long_test_function:npn{tlist_if_eq:nx}#1#2{
1626     \tlp_set:Nx \l_testa_tlp {\exp_not:n{#1}}
1627     \tlp_set:Nx \l_testb_tlp {#2}
1628     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1629   }
1630   \def_long_test_function:npn{tlist_if_eq:on}#1#2{
1631     \tlp_set:Nx \l_testa_tlp {\exp_not:o{#1}}
1632     \tlp_set:Nx \l_testb_tlp {\exp_not:n{#2}}
1633     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1634   }
1635   \def_long_test_function:npn{tlist_if_eq:oo}#1#2{
1636     \tlp_set:Nx \l_testa_tlp {\exp_not:o{#1}}
1637     \tlp_set:Nx \l_testb_tlp {\exp_not:o{#2}}
1638     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1639   }
1640   \def_long_test_function:npn{tlist_if_eq:ox}#1#2{
1641     \tlp_set:Nx \l_testa_tlp {\exp_not:o{#1}}
1642     \tlp_set:Nx \l_testb_tlp {#2}
1643     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1644   }
1645   \def_long_test_function:npn{tlist_if_eq:xn}#1#2{
1646     \tlp_set:Nx \l_testa_tlp {#1}
1647     \tlp_set:Nx \l_testb_tlp {\exp_not:n{#2}}
1648     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1649   }
1650   \def_long_test_function:npn{tlist_if_eq:xo}#1#2{
1651     \tlp_set:Nx \l_testa_tlp {#1}
1652     \tlp_set:Nx \l_testb_tlp {\exp_not:o{#2}}
1653     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1654   }
1655   \def_long_test_function:npn{tlist_if_eq:xx}#1#2{
1656     \tlp_set:Nx \l_testa_tlp {#1}
1657     \tlp_set:Nx \l_testb_tlp {#2}
1658     \if_meaning:NN\l_testa_tlp \l_testb_tlp
1659   }
1660 }

```

7.7.1 Checking for and replacing tokens

\tlp_if_in:NnTF	See the replace functions for further comments. In this part we don't care too much
\tlp_if_in:cnTF	about brace stripping since we are not interested in passing on the tokens which are split
\tlp_if_in:NnT	
\tlp_if_in:cnT	
\tlp_if_in:NnF	
\tlp_if_in:cnF	
\tlist_if_in:nnTF	
\tlist_if_in:onTF	

off in the process.

```

1661 \def_long:Npn \tlp_if_in:NnTF #1#2{
1662   \def_long:Npn\tmp:w ##1 #2 ##2\q_stop{
1663     \quark_if_no_value:nFT{##2}
1664   }
1665   \exp_after:NN \tmp:w #1 #2 \q_no_value \q_stop
1666 }
1667 \def_new:Npn \tlp_if_in:cnTF {\exp_args:Nc\tlp_if_in:NnTF}
1668 \def_long:Npn \tlp_if_in:NnT #1#2{
1669   \def_long:Npn\tmp:w ##1 #2 ##2\q_stop{
1670     \quark_if_no_value:nF{##2}
1671   }
1672   \exp_after:NN \tmp:w #1 #2 \q_no_value \q_stop
1673 }
1674 \def_new:Npn \tlp_if_in:cnT {\exp_args:Nc\tlp_if_in:NnT}
1675 \def_long:Npn \tlp_if_in:NnF #1#2{
1676   \def_long:Npn\tmp:w ##1 #2 ##2\q_stop{
1677     \quark_if_no_value:nT{##2}
1678   }
1679   \exp_after:NN \tmp:w #1 #2 \q_no_value \q_stop
1680 }
1681 \def_new:Npn \tlp_if_in:cnF {\exp_args:Nc\tlp_if_in:NnF}
1682 \def_long_new:Npn \tlist_if_in:nnTF #1#2{
1683   \def_long:Npn\tmp:w ##1 #2 ##2\q_stop{
1684     \quark_if_no_value:nFT{##2}
1685   }
1686   \tmp:w #1 #2 \q_no_value \q_stop
1687 }
1688 \def_new:Npn \tlist_if_in:onTF {\exp_args:No\tlist_if_in:nnTF}

```

`\l_tlp_replace_tlp` A temp variable for the replace operations.

```

1689 \tlp_new:Nn\l_tlp_replace_tlp{}

```

<code>\tlp_replace_in:Nnn</code> <code>\tlp_replace_in:cnn</code> <code>\tlp_greplace_in:Nnn</code> <code>\tlp_greplace_in:cnn</code> <code>\tlp_replace_in_aux:NNnn</code>	<p>Replacing the first item in a token list pointer goes like this: Define a temporary function with delimited arguments containing the search term and take a closer look at what is left.</p> <p>We append the expansion of the tlp with the search term plus the quark <code>\q_no_value</code>. If the search term isn't present this last one is found and the following token is the quark, so we test for that. If the search term is present we will have to split off the <code>#3\q_no_value</code> we had, so we define yet another function with delimited arguments to do this. The advantage here is that now we have a special end sequence so there is no problem if the search term appears more than once. Only problem left is to prevent brace stripping in both ends, so we prepend the expansion of the tlp with <code>\q_mark</code> later to be gobbled and also prepend the remainder of the first split operation with <code>\q_mark</code> also to be gobbled again later on.</p>
---	--

```

1690 \def_long_new:NNn \tlp_replace_in_aux:NNnn 4{
1691   \def_long:Npn \tmp:w ##1#3##2\q_stop{
1692     \quark_if_no_value:nF{##2}
1693   }

```

At this point `##1` starts with a `\q_mark` so remove it.

```

1694     \tlp_set:Nx\l_tlp_replace_tlp{\exp_not:o{\use_none:n##1#4}}
1695     \def_long:Npn \tmp:w ###1#3\q_no_value{
1696         \tlp_put_right:Nx \l_tlp_replace_tlp {\exp_not:o{\use_none:n ###1}}
1697     }
1698     \tmp:w \q_mark ##2
1699 }

```

Now all that is done is setting the token list pointer equal to the expansion of the token register.

```

1700     #1#2\l_tlp_replace_tlp
1701 }

```

Here is where we start the process. Note that the tlp might start with a space token so we use this little trick with \use_arg_i:n to prevent it from being removed.

```

1702 \use_arg_i:n{\exp_after:NN \tmp:w\exp_after:NN\q_mark}
1703 #2#3 \q_no_value\q_stop
1704 }

```

Now the various versions doing the replacement either globally or locally.

```

1705 \def_new:Npn \tlp_replace_in:Nnn {\tlp_replace_in_aux:NNnn \tlp_set_eq:NN}
1706 \def_new:Npn \tlp_replace_in:cnn{\exp_args:Nc\tlp_replace_in:Nnn}
1707 \def_new:Npn \tlp_greplace_in:Nnn {\tlp_replace_in_aux:NNnn \tlp_gset_eq:NN}
1708 \def_new:Npn \tlp_greplace_in:cnn{\exp_args:Nc\tlp_greplace_in:Nnn}

```

\tlp_replace_all_in:Nnn The version for replacing *all* occurrences of the search term is fairly easy since we just have to keep doing the replacement on the split-off part until all are replaced. Otherwise it is pretty much the same as above.

```

\tlp_greplace_all_in:cnn
\tlp_replace_all_in_aux:NNnn
1709 \def_long:NNn \tlp_replace_all_in_aux:NNnn 4{
1710     \tlp_clear:N \l_tlp_replace_tlp
1711     \def_long:Npn \tmp:w ##1#3##2\q_stop{
1712         \quark_if_no_value:nTF{##2}
1713         {
1714             \tlp_put_right:Nx \l_tlp_replace_tlp {\exp_not:o{\use_none:n##1}}
1715         }
1716         {
1717             \tlp_put_right:Nx \l_tlp_replace_tlp {\exp_not:o{\use_none:n##1 #4}}
1718             \tmp:w \q_mark##2 \q_stop
1719         }
1720     }
1721     \use_arg_i:n{\exp_after:NN \tmp:w\exp_after:NN\q_mark}
1722     #2#3 \q_no_value\q_stop
1723     #1#2\l_tlp_replace_tlp
1724 }

```

Now the various forms.

```

1725 \def_new:Npn \tlp_replace_all_in:Nnn {
1726     \tlp_replace_all_in_aux:NNnn \tlp_set_eq:NN}
1727 \def_new:Npn \tlp_replace_all_in:cnn{\exp_args:Nc\tlp_replace_all_in:Nnn}
1728 \def_new:Npn \tlp_greplace_all_in:Nnn {
1729     \tlp_replace_all_in_aux:NNnn \tlp_gset_eq:NN}
1730 \def_new:Npn \tlp_greplace_all_in:cnn{\exp_args:Nc\tlp_greplace_all_in:Nnn}

```

\tlist_remove_in:Nn Next comes a series of removal functions. I have just implemented them as subcases of
\tlist_remove_in:cn the replace functions for now (I'm lazy).
\tlist_gremove_in:Nn
\tlist_gremove_in:cn 1731 \def_long_new:NNn \tlist_remove_in:Nn 2{\tlist_replace_in:Nnn #1{#2}{}}
1732 \def_long_new:NNn \tlist_gremove_in:Nn 2{\tlist_greplace_in:Nnn #1{#2}{}}
1733 \def_new:Npn \tlist_remove_in:cn{\exp_args:Nc\tlist_remove_in:Nn}
1734 \def_new:Npn \tlist_gremove_in:cn{\exp_args:Nc\tlist_gremove_in:Nn}

\tlist_remove_all_in:Nn Same old, same old.
\tlist_remove_all_in:Nn
\tlist_gremove_all_in:Nn 1735 \def_long_new:Npn \tlist_remove_all_in:Nn #1#2{
\tlist_gremove_all_in:Nn 1736 \tlist_replace_all_in:Nnn #1{#2}{}
1737 }
1738 \def_long_new:Npn \tlist_gremove_all_in:Nn #1#2{
1739 \tlist_greplace_all_in:Nnn #1{#2}{}
1740 }
1741 \def_new:Npn \tlist_remove_all_in:cn{\exp_args:Nc\tlist_remove_all_in:Nn}
1742 \def_new:Npn \tlist_gremove_all_in:cn{\exp_args:Nc\tlist_gremove_all_in:Nn}

7.7.2 Heads or tails?

\tlist_head:n These functions pick up either the head or the tail of a list. \tlist_head_iii:n returns
\tlist_head_i:n the first three items on a list.
\tlist_tail:n
\tlist_tail:f 1743 \def_long_new:Npn \tlist_head:n #1{\tlist_head:w #1\q_nil}
1744 \let_new:NN \tlist_head_i:n \tlist_head:n
\tlist_head_iii:n 1745 \def_long_new:Npn \tlist_tail:n #1{\tlist_tail:w #1\q_nil}
\tlist_head_iii:f 1746 \def_new:Npn \tlist_tail:f {\exp_args:Nf \tlist_tail:n}
\tlist_head:w 1747 \def_long_new:Npn \tlist_head_iii:n #1{\tlist_head_iii:w #1\q_nil}
\tlist_tail:w 1748 \def_new:Npn \tlist_head_iii:f {\exp_args:Nf \tlist_head_iii:n}
\tlist_head_iii:w 1749 \let_new:NN \tlist_head:w \use_arg_i_delimit_by_q_nil:nw
1750 \def_long_new:Npn \tlist_tail:w #1#2\q_nil{#2}
1751 \def_long_new:Npn \tlist_head_iii:w #1#2#3#4\q_nil{#1#2#3}

\tlist_if_head_eq_meaning_p:nN When we want to check if the first token of a list equals something specific it is usu-
\tlist_if_head_eq_meaning:nNTF ally either to see if it is a control sequence or a character. Hence we make two differ-
\tlist_if_head_eq_meaning:nNT different functions as the internal test is different. \tlist_if_head_meaning_eq:nNTF uses
\tlist_if_head_eq_meaning:nNF \if_meaning:NN and will consider the tokens b₁₁ and b₁₂ different. \tlist_if_head_char_eq:nNTF
\tlist_if_head_eq_charcode_p:nNTF on the other hand only compares character codes so would regard b₁₁ and b₁₂ as equal
\tlist_if_head_eq_charcode:nNTF but would also regard two primitives as equal.
\tlist_if_head_eq_charcode:nNT 1752 \def_long_new:Npn \tlist_if_head_eq_meaning_p:nN #1#2{
\tlist_if_head_eq_charcode:nNF 1753 \exp_after:NN \if_meaning:NN \tlist_head:w #1\q_nil#2
\tlist_if_head_eq_catcode_p:nNTF 1754 \c_true
\tlist_if_head_eq_catcode:nNTF 1755 \else:
\tlist_if_head_eq_catcode:nNT 1756 \c_false
\tlist_if_head_eq_catcode:nNF 1757 \fi:
1758 }
1759 \def_long_test_function_new:npn {\tlist_if_head_eq_meaning:nN}#1#2{
1760 \if:w \tlist_if_head_eq_meaning_p:nN{#1}#2}

For the charcode and catcode versions we insert `\exp_not:N` in front of both tokens. If you need them to expand fully as \TeX does itself with these you can use an `f` type expansion.

```

1761 \def_long_new:Npn \tlist_if_head_eq_charcode_p:nN #1#2{
1762   \exp_after:NN\if_charcode:w \exp_after:NN\exp_not:N
1763     \tlist_head:w #1\q_nil\exp_not:N#2
1764   \c_true
1765   \else:
1766     \c_false
1767   \fi:
1768 }
1769 \def_long_test_function_new:nnpn {tlist_if_head_eq_charcode:nN}#1#2{
1770   \if:w\tlist_if_head_eq_charcode_p:nN{#1}#2}

```

Actually the default is already an `f` type expansion.

```

1771 \def_long_new:Npn \tlist_if_head_eq_charcode_p:fN #1#2{
1772   \exp_after:NN\if_charcode:w \tlist_head:w #1\q_nil\exp_not:N#2
1773   \c_true
1774   \else:
1775     \c_false
1776   \fi:
1777 }
1778 \def_long_test_function_new:nnpn {tlist_if_head_eq_charcode:fN}#1#2{
1779   \if:w\tlist_if_head_eq_charcode_p:fN{#1}#2}
1780 \def_long_new:Npn \tlist_if_head_eq_catcode_p:nN #1#2{
1781   \exp_after:NN\if_charcode:w \exp_after:NN\exp_not:N
1782     \tlist_head:w #1\q_nil\exp_not:N#2
1783   \c_true
1784   \else:
1785     \c_false
1786   \fi:
1787 }
1788 \def_long_test_function_new:nnpn {tlist_if_head_eq_catcode:nN}#1#2{
1789   \if:w\tlist_if_head_eq_catcode_p:nN{#1}#2}

```

`\tlist_reverse:n` Reversal of a token list is done by taking one token at a time and putting it in front of
`\tlist_reverse_aux:nN` the ones before it.

```

1790 \def_long_new:Npn \tlist_reverse:n #1{
1791   \tlist_reverse_aux:nN {} #1 \q_recursion_tail\q_recursion_stop
1792 }
1793 \def_long_new:Npn \tlist_reverse_aux:nN #1#2{
1794   \quark_if_recursion_tail_stop_do:nn {#2}{ #1 }
1795   \tlist_reverse_aux:nN {#2#1}
1796 }

```

As this package relies heavily on a lot of the expansion tricks used in `l3expan` we make sure to load it automatically at the end when used as a package. Probably not needed but I'm just such a nice guy...

```

1797 <package>\RequirePackage{l3expan}
1798 <package>\RequirePackage{l3num}\par

```

Show token usage:

```
1799 </initex | package>
1800 <*showmemory>
1801 \showMemUsage
1802 </showmemory>
```

8 L^AT_EX3 functions

All L^AT_EX3 functions contain a colon in their name. Characters following the colon are used to denote the number and the “type” of arguments that the function takes. An uppercase N is used to denote an argument that consists of a single token and a lowercase n is used when the argument can consist of several tokens surrounded by braces. In case of n arguments that consist of a single token the surrounding braces can be omitted in nearly all situations—functions that force the use of braces even for single token arguments are explicitly mentioned. For example, `\seq_gpush:Nn` is a function that takes two arguments, the first is a single token (the sequence) and the second may consist of several tokens surrounded by braces.

This concept of argument specification makes it easy to read the code and should be followed when defining new functions.

8.1 Expanding arguments of functions

Within code it is often necessary to expand or partially expand arguments before passing it on to some function. For example, if the token list pointer `\l_tmpa_tlp` contains the current file that should be pushed onto some stack, we can not write

```
\seq_gpush:Nn
  \g_file_name_stack
  \l_tmpa_tlp
```

since this would put the token `\l_tmpa_tlp` and not its contents on the stack. Instead a suitable number of `\exp_after:NN` would be necessary (together with extra braces) to change the order of execution, i.e.

```
\exp_after:NN
  \seq_gpush:Nn
\exp_after:NN
  \g_file_name_stack
\exp_after:NN
  {\l_tmpa_tlp}
```

The above example is probably the simplest case but is already shows how the code changes to something difficult to understand. Therefore L^AT_EX3 provides the programmer with a general scheme that keeps the code compact and easy to understand. To denote that some argument to a function needs special treatment one just uses different letters in the argument part of the function to mark the desired behavior. In the above example one would write

```

\seq_gpush:No
  \g_file_name_stack
  \l_tmpa_tlp

```

to achieve the desired effect. Here the `o` stands for expand this (the second) argument once before passing it to the function.

The following letters can be used to denote special treatment of arguments before passing it to the basic function:

- o** One time expanded token or token-list. In the latter case, effectively only the first token in the list gets expanded. Since the expansion might result in more than one token, the result is surrounded for further processing with braces.
- x** Fully expanded token or token-list. Like `o` but the argument is expanded using `\def:Npx` before it is passed on. This means that expansion takes place until only unexpandable tokens are left.
- f** Almost the same as the `x` type except here the token list is expanded fully until the first unexpandable token is found and the rest is left unchanged. Note that if this function finds a space at the beginning of the argument it will gobble it and not expand the next argument.
- N,O,X** Like `n`, `o`, `x` but the argument must be a single token without any braces around it.
- c** A character string or a token-list that ultimately expands to characters. This string (after expansion) is used to construct a command name that is eventually passed on.
- C** A character string or a token-list that ultimately expands to characters. From this string (after expansion) a command name is constructed and then this command name is expanded once (like `o`). The result of this is eventually passed on. In other words

```

\seq_gpush:NC
  \g_file_name_stack
  {l_tmpa_tlp}

```

Has the same effect as the example above.

Here are three new expansion types that may be useful but I'm not sure yet. Only time will tell... Proper documentation of these functions is postponed until later.

- d** This is pretty much like the `o` type except the token list get's expanded twice before being passed on. (`d` is for double.) It is often useful in conjunction with a forced expansion.
- E** Sometimes you need to unpack a token list or something else but you don't want it to add the braces that the `o` type does. This is where you usually wind up with a lot of `\exp_after:N`s and we would like to avoid that. This type works quite well with the other syntax but it won't work in certain circumstances: Since the generic expansion functions read their arguments when the expanded code is

shuffled around, this type will have a problem if the last token you want to expand once is `\token_to_str:N` and you're in an argument expansion process involving arguments in braces such as the `n` and `o` type arguments. If you stick to functions involving only `N` and `E` everything will work just fine. (`E` is for expanded, single token.)

e Same as above but the argument must be given in braces.

Due to memory constraints not all possible variations are implemented for every base function. Instead only those that are used within the L^AT_EX3 kernel or otherwise seem to be of general interest are implemented. Consult the module description to find out which functions are actually defined. The next section explains how to define missing variants.

8.2 Defining new variants

The definition of variant forms for base functions may be necessary when writing new functions or when applying a kernel function in a situation that we haven't thought of before.

Internally preprocessing of arguments is done with functions from the `\exp_` module. They all look alike, an example would be `\exp_args:NNo`. This function has three arguments, the first and the second are a single tokens the third argument gets expanded once. If `\seq_gpush:No` wouldn't be defined the example above could be coded in the following way:

```
\exp_args:NNo\seq_gpush:Nn
  \g_file_name_stack
  \l_tmpa_tlp
```

In other words, the first argument to `\exp_args:NNo` is the base function and the other arguments are preprocessed and then passed to this base function. In the example the first argument to the base function should be a single token which is left unchanged while the second argument is expanded once. From this example we can also see how the variants are defined. They just expand into the appropriate `\exp_` function followed by the desired base function, e.g.

```
\def_new:Npn\seq_gpush:No{\exp_args:NNo\seq_gpush:Nn}
```

Providing variants in this way in style files is uncritical as the `\def_new:Npn` function will silently accept definitions whenever the new definition is identical to an already given one. Therefore adding such definition to later releases of the kernel will not make such style files obsolete.

The `f` type is so special that it deserves an example. Let's pretend we want to set `\aaa` equal to the control sequence stemming from turning `b \l_tmpa_tlp b` into a control sequence. Furthermore we want to store the execution of it in a $\langle toks \rangle$ register. In this example we assume `\l_tmpa_tlp` contains the text string `lur`. The straight forward approach is

```
\toks_set:No \l_tmpa_toks {\let:Nc \aaa {b \l_tmpa_tlp b}}
```

Unfortunately this only puts `\exp_args:Nnc \let:NN \aaa {b \l_tmpa_tlp b}` into `\l_tmpa_toks` and not `\let:NwN \aaa = \blurb` as we probably wanted. Using `\toks_set:Nx` is not an option as that will die horribly. Instead we can do a

```
\toks_set:Nf \l_tmpa_toks {\let:Nc \aaa {b \l_tmpa_tlp b}}
```

which puts the desired result in `\l_tmpa_toks`. It requires `\toks_set:Nf` to be defined as

```
\def:Npn \toks_set:Nf {\exp_args:Nnf \toks_set:Nn}
```

If you use this type of expansion in conditional processing then you should stick to using TF type functions only as it does not try to finish any `\if... \fi`: itself!

The available internal functions for argument expansion come in to flavours, some of them are faster then others. Therefore it is usually best to follow the following guidelines when defining new functions that are supposed to come with variant forms:

- Arguments that might need expansion should come first in the list of arguments to make processing faster.
- Arguments that should consist of single tokens should come first.
- Arguments that need full expansion (i.e., are denoted with `x`) should be avoided if possible as they can not be processed very fast.
- In general `n`, `x`, and `o` (if not in the last position) will need special processing which is not fast and not expandable, i.e., functions of this type may not work correctly in arguments that are itself subject to `x` expansion. Therefore it is best to use the “expandable” functions (i.e., those that contain only `c`, `N`, `O`, `o` or `f` in the last position) whenever possible.

When pdfTeX 1.50 arrives, it will contain a primitive for performing the equivalent of an `x` expansion after only one expansion and most importantly: as an expandable operation.

```
\exp_arg:x \exp_arg:x { \arg }
```

`\arg` is expanded fully using an `x` expansion.

8.3 Manipulating the first argument

```
\exp_args:No \exp_args:No \fct \arg1 \arg2 ...
```

The first argument of `\fct` (i.e., `\arg1`) is expanded once, the result is surrounded by braces and passed to `\fct`. `\fct` may have more than one argument—all others are passed unchanged.

```
\exp_args:Nc \exp_args:Nc \fct \arg1 \arg2 ...
```

The first argument of `\fct` (i.e., `\arg1`) is expanded until only characters remain. (An

internal error occurs if something else is the result of this expansion.) Then the result is turned into a control sequence and passed to $\langle funct \rangle$ as the first argument. $\langle funct \rangle$ may have more than one argument—all others are passed unchanged.

$\boxed{\backslash exp_args:NC}$ $\backslash exp_args:Nc \langle funct \rangle \langle arg1 \rangle \langle arg2 \rangle \dots$

The first argument of $\langle funct \rangle$ (i.e., $\langle arg1 \rangle$) is expanded until only characters remain. (An internal error occurs if something else is the result of this expansion.) Then the result is turned into a control sequence which is then expanded once more. The result of this is then passed to $\langle funct \rangle$ as the first argument. $\langle funct \rangle$ may have more than one argument—all others are passed unchanged.

$\boxed{\backslash exp_args:Nx}$ $\backslash exp_args:Nx \langle funct \rangle \langle arg1 \rangle \langle arg2 \rangle \dots$

The first argument of $\langle funct \rangle$ (i.e., $\langle arg1 \rangle$) is fully expanded until only unexpandable tokens remain, the result is surrounded by braces and passed to $\langle funct \rangle$. $\langle funct \rangle$ may have more than one argument—all others are passed unchanged. As mentioned before, this type of function is relatively slow.

$\boxed{\backslash exp_args:Nf}$ $\backslash exp_args:Nf \langle funct \rangle \langle arg1 \rangle \langle arg2 \rangle \dots$

The first argument of $\langle funct \rangle$ (i.e., $\langle arg1 \rangle$) undergoes full expansion until the first unexpandable token is encountered, the result is surrounded by braces and passed to $\langle funct \rangle$. $\langle funct \rangle$ may have more than one argument—all others are passed unchanged. Beware of its special behavior as explained above.

8.4 Manipulating two arguments

$\boxed{\backslash exp_args:NNx}$
 $\backslash exp_args:Nnx$
 $\backslash exp_args:Ncx$
 $\backslash exp_args:Nox$
 $\backslash exp_args:Nxo$
 $\backslash exp_args:Nxx$ $\backslash exp_args:Nnx \langle funct \rangle \langle arg1 \rangle \langle arg2 \rangle \dots$

The above functions all manipulate the first two arguments of $\langle funct \rangle$. They are all slow and non-expandable.

$\boxed{\backslash exp_args:NNo}$
 $\backslash exp_args:NNf$
 $\backslash exp_args:Nno$
 $\backslash exp_args:NNc$
 $\backslash exp_args:Noo$
 $\backslash exp_args:NOo$
 $\backslash exp_args:NOc$
 $\backslash exp_args:Nco$
 $\backslash exp_args:Ncc$
 $\backslash exp_args:NNC$ $\backslash exp_args:NNo \langle funct \rangle \langle arg1 \rangle \langle arg2 \rangle \dots$

These are the fast and expandable functions for the first two arguments.

8.5 Manipulating three arguments

So far not all possible functions are provided and even the selection below may be reduced in the future as far as the non-expandable functions are concerned.

<code>\exp_args:Nnnx</code>
<code>\exp_args:Noox</code>
<code>\exp_args:Nnox</code>
<code>\exp_args:Ncnx</code>

`\exp_args:Nnnx <funct> <arg1> <arg2> <arg3> ...`

All the above functions are non-expandable.

<code>\exp_args:NnnN</code>
<code>\exp_args:Nnno</code>
<code>\exp_args:NNoo</code>
<code>\exp_args:NOoo</code>
<code>\exp_args:Nccc</code>
<code>\exp_args:NcNc</code>
<code>\exp_args:Nnnc</code>
<code>\exp_args:NcNo</code>
<code>\exp_args:Ncco</code>

`\exp_args:NNoo <funct> <arg1> <arg2> <arg3> ...`

These are the fast and expandable functions for the first three arguments.

8.6 Internal functions and variables

<code>\exp_after:NN</code>

`\exp_after:NN <token1> <token2>`

This will expand `<token2>` once before processing `<token1>`. This is similar to `\exp_args:No` except that no braces are put around the result of expanding `<token2>`.

TeXhackers note: This is the primitive `\expandafter` which was renamed to fit into the naming conventions of L^AT_EX3.

<code>\exp_not:N</code>
<code>\exp_not:c</code>
<code>\exp_not:n</code>

`\exp_not:N <token>`
`\exp_not:n { <token list> }`

This function will prohibit the expansion of `<token>` in situation where `<token>` would otherwise be replaced by its definition, e.g., inside an argument that is handled by the x convention.

TeXhackers note: `\exp_not:N` is the primitive `\noexpand` renamed and `\exp_not:n` is the ε -TeX primitive `\unexpanded`.

<code>\exp_not:o</code>
<code>\exp_not:d</code>
<code>\exp_not:f</code>

`\exp_not:o {<token list>}`

Same as `\exp_not:n` except `<token list>` is expanded once for the o type and twice for the d type and the result of this expansion is then prohibited from being expanded further.

`\exp_not:E` `\exp_not:E` $\langle token \rangle$

The name of this command is a lie. Perhaps it should be called “`\exp_perhaps_once`”. What it actually does is, it expands $\langle token \rangle$ and then issues an `\exp_not:N` to prohibit further expansion of the first token in the replacement text of $\langle token \rangle$. This means that if the replacement text of $\langle token \rangle$ consists of more than one token all further tokens are still subject to full expansion.

T_EXhackers note: This command has no equivalent.

`\exp_stop_f:` $\langle f \text{ expansion} \rangle \dots \exp_stop_f:$

This function stops an `f` type expansion. An example use is one such as

```
\tlp_set:Nf \l_tmpa_tlp {
  \if_case:w \l_tmpa_int
  \or:   \use_arg_i_after_orelse:nw{\exp_stop_f: \textbullet}
  \or:   \use_arg_i_after_orelse:nw{\exp_stop_f: \textendash}
  \else: \use_arg_i_after_orelse:nw{\exp_stop_f: else-item}
  \fi:
}
```

This ensures the expansion is stopped right after finishing the conditional but without expanding `\textbullet` etc.

T_EXhackers note: This function is a space token but it is better to distinguish this expansion stopping token from a desired space token when writing code.

`\l_exp_tlp`

The `\exp_` module has its private variables to temporarily store results of the argument expansion. This is done to avoid interference with other functions using temporary variables.

8.7 The Implementation

We start by ensuring that the required packages are loaded.

```
1803 <package>\ProvidesExplPackage
1804 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
1805 <package>\RequirePackage{l3tlp}
1806 <*initex | package>
```

8.7.1 General expansion

In this section a general mechanism for defining functions to handle argument handling is defined. These general expansion functions are expandable unless `\x` is used. (Any version

of `x` is going to have to use one of the L^AT_EX3 names for `\def:Npx` at some point, and so is never going to be expandable.⁷)

In a later section some common cases are coded by a more direct method, typically using calls to `\exp_after:NN`.

`\l_exp_tlp` We need a scratch token list pointer.

```
1807 \tlp_new:Nn\l_exp_tlp{}
```

This code uses internal functions with names that start with `\::` to perform the expansions. All macros are `long` as this turned out to be desirable since the tokens undergoing expansion may be arbitrary user input.

`\exp_arg_next:nnn` This is basically the same function as `\Dexp_arg_next:nnn`.

```
1808 \def_long_new:Npn\exp_arg_next:nnn#1#2#3{
1809   #2\:::{#3#1}
1810 }
```

`\::n` This function is used to skip an argument that doesn't need to be expanded.

```
1811 \def_long_new:Npn\::n#1\:::#2#3{
1812   #1\:::{#2{#3}}
1813 }
```

`\::N` This function is used to skip an argument that consists of a single token and doesn't need to be expanded.

```
1814 \def_long_new:Npn\::N#1\:::#2#3{
1815   #1\:::{#2#3}
1816 }
```

`\::c` This function is used to skip an argument that is turned into a control sequence without expansion.

```
1817 \def_long_new:Npn\::c#1\:::#2#3{
1818   \exp_after:NN\exp_arg_next:nnn\cs:w #3\cs_end:{#1}{#2}
1819 }
```

`\::o` This function is used to expand an argument once.

```
1820 \def_long_new:Npn\::o#1\:::#2#3{
1821   \exp_after:NN\exp_arg_next:nnn\exp_after:NN{\exp_after:NN{#3}}{#1}{#2}
1822 }
```

⁷However, some primitives have certain characteristics that means that their arguments undergo an `x` type expansion but the primitive is in fact still expandable. We shall make it very clear when such a function is expandable.

`\::f` This function is used to expand a token list until the first unexpandable token is found.

`\exp_stop_f:` The underlying `\int_to_roman:w -'0` expands everything in its way to find something terminating the number and thereby expands the function in front of it. This scanning procedure is terminated once the expansion hits something non-expandable or a space. We introduce `\exp_stop_f:` to mark such an end of expansion marker; in case the scanner hits a number, this number also terminates the scanning and is left untouched. In the example shown earlier the scanning was stopped once `TEX` had fully expanded `\let:Nc \aaa {b \l_tmpa_tlp b}` into `\let:NwN \aaa = \blurb` which then turned out to contain the non-expandable token `\let:NwN`. Since the expansion of `\int_to_roman:w -'0` is *null*, we wind up with a fully expanded list, only `TEX` has not tried to execute any of the non-expandable tokens. This is what differentiates this function from the `x` argument type.

```

1823 \def_long_new:Npn\::f#1\:::#2#3{
1824   \exp_after:NN\exp_arg_next:nnn
1825   \exp_after:NN{\exp_after:NN{\int_to_roman:w -'0 #3}}
1826   {#1}{#2}
1827 }
1828 \def_new:Npn \exp_stop_f: {~}

```

`\::x` This function is used to expand an argument fully. If the pdf`TEX` primitive `\expanded` is present, we use it.

```

1829 \let_new:NN \exp_arg:x \expanded % Move eventually.
1830 \cs_if_free:NTF\exp_arg:x{
1831   \def_long_new:Npn\::x#1\:::#2#3{
1832     % \tlp_set:Nx\l_exp_tlp{{{#3}}}
1833     \def:Npx \l_exp_tlp{{{#3}}}
1834     \exp_after:NN\exp_arg_next:nnn\l_exp_tlp{#1}{#2}
1835   }
1836 {
1837   \def_long_new:Npn\::x#1\:::#2#3{
1838     \exp_after:NN\exp_arg_next:nnn
1839     \exp_after:NN{\exp_arg:x{{{#3}}}}{#1}{#2}
1840   }
1841 }

```

`\:::` Just another name for the identity function.

```

1842 \def_long_new:Npn\:::#1{#1}

```

`\::C` This function creates a control sequence out of `#3` and expands that once before passing it on to `\exp_arg_next:nnn`.

```

1843 \def_long_new:Npn\:::C#1\:::#2#3{
1844   \exp_after:NN\exp_C_aux:nnn\cs:w #3\cs_end:{#1}{#2}

```

`\exp_C_aux:nnn` A helper function for `\::C` wich expands its argument before passing it on to `\exp_arg_next:nnn`.

```

1845 \def_long_new:Npn\exp_C_aux:nnn #1{
1846   \exp_after:NN

```

```

1847 \exp_arg_next:nnn
1848 \exp_after:NN
1849 {
1850 \exp_after:NN
1851 {#1}
1852 }
1853 }

```

Here are some that might not stay but let's see.

`\::E` This function is used to expand an argument once and return it *without* braces. Use this only when you feel pretty comfortable about your input! Actually this is pretty much just generic wrapper for `\exp_after:NN`.

```

1854 \def_long_new:Npn\::E#1\:::#2#3{
1855 \exp_after:NN\exp_arg_next:nnn \exp_after:NN{#3}{#1}{#2}
1856 }

```

`\::e` Same as `\::E` really but conceptually they are different.

```

1857 \def_long_new:Npn\::e#1\:::#2#3{
1858 \exp_after:NN\exp_arg_next:nnn \exp_after:NN{#3}{#1}{#2}
1859 }

```

`\::d` This function is used to expand an argument twice. Mostly useful for `toks` type things.

```

1860 \def_long_new:Npn\::d#1\:::#2#3{
1861 \exp_after:NN\exp_after:NN\exp_after:NN\exp_arg_next:nnn
1862 \exp_after:NN\exp_after:NN\exp_after:NN{
1863 \exp_after:NN\exp_after:NN\exp_after:NN{#3}}{#1}{#2}
1864 }

```

We do most of them by hand here. This also means that we get a name for `\exp_after:NN` that fits with the rest of the code.

```

1865 \let:NN \exp_args:NE \exp_after:NN
1866 \def:Npn \exp_args:NNE #1{\exp_args:NE#1\exp_args:NE}
1867 \def:Npn \exp_args:NNNE #1#2{\exp_args:NE#1\exp_args:NE#2\exp_args:NE}
1868 \def:Npn \exp_args:NEE #1{\exp_args:NE\exp_args:NE\exp_args:NE#1\exp_args:NE}
1869 \def:Npn \exp_args:NcE #1#2{\exp_after:NN #1\cs:w #2\exp_after:NN\cs_end:}
1870 \def:Npn \exp_args:Nd {\::d\:::}
1871 \def:Npn \exp_args:NNd {\::N\::d\:::}

```

`\exp_args:NC` Here are the actual function definitions, using the helper functions above.

```

\exp_args:Ncx
\exp_args:Ncco 1872 %\def:Npn \exp_args:NNNo {\::N\::o\:::}
\exp_args:Ncco 1873 %\def:Npn \exp_args:NNNo {\::N\::o\::o\:::}
\exp_args:Nccx 1874 %\def:Npn \exp_args:NNc {\::N\::c\:::}
\exp_args:Ncnx 1875 %\def:Npn \exp_args:NNc {\::N\::o\::\:::}
\exp_args:NcNc 1876 %\def:Npn \exp_args:NNc {\::o\::o\::o\:::}
\exp_args:Nf 1877 %\def:Npn \exp_args:Nc {\::o\::c\:::}
\exp_args:NNf 1878 %\def:Npn \exp_args:Nc {\::o\::o\:::}
\exp_args:Nfo 1879 %\def:Npn \exp_args:Nc {\::c\:::}
\exp_args:Nnf
\exp_args:NNno
\exp_args:NnnN
\exp_args:Nnno
\exp_args:Nnnx
\exp_args:Nno
\exp_args:Nnox
\exp_args:NNx
\exp_args:Nnx
\exp_args:Noo

```

```

1880 %\def:Npn \exp_args:Ncc {\::c\::c\:::}
1881 %\def:Npn \exp_args:Nccc {\::c\::c\::c\:::}
1882 %\def:Npn \exp_args:Nco {\::c\::o\:::}
1883 %\def:Npn \exp_args:No {\::o\:::}
1884
1885 \def:Npn \exp_args:NC {\::C\:::}
1886 \def:Npn \exp_args:NNC {\::N\::C\:::}
1887 \def:Npn \exp_args:NNf {\::N\::f\:::}
1888 \def:Npn \exp_args:NNno {\::N\::n\::o\:::}
1889 \def:Npn \exp_args:NNnx {\::N\::n\::x\:::} % new
1890 \def:Npn \exp_args:NNoo {\::N\::o\::o\:::} % new
1891 \def:Npn \exp_args:NNox {\::N\::o\::x\:::} % new
1892 \def:Npn \exp_args:NNx {\::N\::x\:::}
1893 \def:Npn \exp_args:NcNc {\::c\::N\::c\:::}
1894 \def:Npn \exp_args:NcNo {\::c\::N\::o\:::}
1895 \def:Npn \exp_args:Ncco {\::c\::c\::o\:::}
1896 \def:Npn \exp_args:Ncco {\::c\::c\::o\:::}
1897 \def:Npn \exp_args:Nccx {\::c\::c\::x\:::}
1898 \def:Npn \exp_args:Ncnx {\::c\::n\::x\:::}
1899 \def:Npn \exp_args:Ncx {\::c\::x\:::}
1900 \def:Npn \exp_args:Nf {\::f\:::}
1901 \def:Npn \exp_args:Nfo{\::f\::o\:::}
1902 \def:Npn \exp_args:Nnf {\::n\::f\:::}
1903 \def:Npn \exp_args:NnnN {\::n\::n\::N\:::} %% Strange one this one...
1904 \def:Npn \exp_args:Nnnc {\::n\::n\::c\:::}
1905 \def:Npn \exp_args:Nnno {\::n\::n\::o\:::}
1906 \def:Npn \exp_args:Nnnx {\::n\::n\::x\:::}
1907 \def:Npn \exp_args:Nno {\::n\::o\:::}
1908 \def:Npn \exp_args:Nnox {\::n\::o\::x\:::}
1909 \def:Npn \exp_args:Nnx {\::n\::x\:::}
1910 \def:Npn \exp_args:Noo {\::o\::o\:::}
1911 \def:Npn \exp_args:Noox {\::o\::o\::x\:::}
1912 \def:Npn \exp_args:Nox {\::o\::x\:::}
1913 \def:Npn \exp_args:Nx {\::x\:::}
1914 \def:Npn \exp_args:Nxo {\::x\::o\:::}
1915 \def:Npn \exp_args:Nxx {\::x\::x\:::}

```

8.7.2 Preventing expansion

```

\exp_not:o
\exp_not:d
\exp_not:f 1916 \def_long_new:Npn\exp_not:o#1{\exp_not:n\exp_after:NN{#1}}
1917 \def_long_new:Npn\exp_not:d#1{
1918   \exp_not:n\exp_after:NN\exp_after:NN\exp_after:NN{#1}
1919 }
1920 \def_long_new:Npn\exp_not:f#1{
1921   \exp_not:n\exp_after:NN{\int_to_roman:w -'0 #1}
1922 }

\exp_not:E Two helper functions, which we can probably live without it.
\exp_not:c
1923 \def_new:Npn\exp_not:E{\exp_after:NN\exp_not:N}
1924 \def_long_new:Npn\exp_not:c#1{\exp_after:NN\exp_not:N\cs:w#1\cs_end:}

```

8.7.3 Single token expansion

Expansion for arguments that are single tokens is done with the functions below. I first thought of using a different module name but then I saw that this wouldn't do since I could then never determine for, say, `\seq_put:no` whether this means single, or general expansion. Therefore I decided to use uppercase 'O' for single expansion.

One of the most important features of these functions is that they are fully expandable and therefore allow to prefix them with `\pref_global:D` for example. This together with the fact that the above concept is much slower in general means that we should convert whenever possible and perhaps remove all remaining occurrences by hand-encoding in the end.

```
\exp_args:No This looks somewhat horrible but it runs well with the other syntax. It is important to
\exp_args:NOo see that these functions really need single tokens as arguments whenever capital letters
\exp_args:NNo are used.
\exp_args:NNO
\exp_args:NOOo 1925 \def_long_new:Npn \exp_args:No #1#2{\exp_after:NN#1\exp_after:NN{#2}}
\exp_args:NNOo 1926 \def_long_new:Npn \exp_args:NOo #1#2#3{\exp_after:NN\exp_args:No \exp_after:NN#1
\exp_args:NNOo 1927 \exp_after:NN#2\exp_after:NN{#3}}
\exp_args:NNOo 1928 \def_long_new:Npn \exp_args:NOOo #1#2#3#4{\exp_after:NN\exp_args:NOo
1929 \exp_after:NN#1\exp_after:NN#2\exp_after:NN#3\exp_after:NN{#4}}
1930 \def_long_new:Npn \exp_args:NNo #1#2#3{\exp_after:NN#1\exp_after:NN#2
1931 \exp_after:NN{#3}}
1932 \def_long_new:Npn \exp_args:NNO #1#2#3 {\exp_after:NN#1
1933 \exp_after:NN#2 #3}
1934 \def_long_new:Npn \exp_args:NNOo #1#2#3#4{\exp_after:NN\exp_args:NNo
1935 \exp_after:NN#1\exp_after:NN#2\exp_after:NN#3\exp_after:NN{#4}}
1936 \def_long_new:Npn \exp_args:NNOo #1#2#3#4{\exp_after:NN#1\exp_after:NN#2
1937 \exp_after:NN#3\exp_after:NN{#4}}
```

```
\exp_args:Nc Here are the functions that turn their argument into csnames but are expandable.
\exp_args:NNc
\exp_args:NOC 1938 \def_long_new:Npn \exp_args:Nc #1#2{\exp_after:NN#1\cs:w#2\cs_end:}
1939 \def_long_new:Npn \exp_args:NNc #1#2#3{\exp_after:NN#1\exp_after:NN#2
\exp_args:Ncc 1940 \cs:w#3\cs_end:}
\exp_args:Nccc 1941 \def_long_new:Npn \exp_args:NOC#1#2#3{\exp_after:NN\exp_args:No\exp_after:NN
1942 #1\exp_after:NN#2\cs:w#3\cs_end:}
1943 \def_long_new:Npn \exp_args:Ncc #1#2#3{\exp_after:NN#1
1944 \cs:w#2\exp_after:NN\cs_end:\cs:w#3\cs_end:}
1945 \def_long_new:Npn \exp_args:Nccc #1#2#3#4{\exp_after:NN#1
1946 \cs:w#2\exp_after:NN\cs_end:\cs:w#3\exp_after:NN
1947 \cs_end:\cs:w #4\cs_end:}
```

```
\exp_args:Nco If we force that the third argument always has braces, we could implement this function
with less tokens and only two arguments.
```

```
1948 \def_long_new:Npn \exp_args:Nco #1#2#3{\exp_after:NN#1\cs:w#2\exp_after:NN
1949 \cs_end:\exp_after:NN{#3}}
```

```
\exp_def_form:nnn This command is a recent addition which was actually added when we wrote the article
for TUGboat (while most of the other code goes way back to 1993).
```



```

1950 \def:Npn\exp_def_form:nnn#1#2#3{
1951   \exp_after:NN
1952   \def:Npn
1953     \cs:w
1954       #1:#3
1955     \exp_after:NN
1956   \cs_end:
1957   \exp_after:NN
1958   {
1959     \cs:w
1960       exp_args:N#3
1961     \exp_after:NN
1962   \cs_end:
1963   \cs:w
1964     #1:#2
1965   \cs_end:
1966   }

```

We also have to test if `exp_args:N#3` is already defined and if not define it via the `\::` commands using the chars in `#3`

```

1967   \cs_if_free:cT
1968     {exp_args:N#3}
1969     {\def:cpx {exp_args:N#3}
1970      {\exp_args_form_x:w #3 :}}
1971   }
1972 }

```

`\exp_args_form_x:w` This command grabs char by char outputting `\::#1` (not expanded further) until we see a `:`. That colon is in fact also turned into `\:::` so that the required structure for `\exp_args...` commands is correctly terminated.

```

1973 \def_new:Npn\exp_args_form_x:w #1 {
1974   \exp_not:c{::#1}
1975   \if_meaning:NN #1 :
1976   \else:
1977     \exp_after:NN\exp_args_form_x:w
1978   \fi:}

```

Show token usage:

```

1979 </initex | package>
1980 <*showmemory>
1981 \showMemUsage
1982 </showmemory>

```

9 Macro Counters

Instead of using counter registers for manipulation of integer values it is sometimes useful to keep such values in macros. For this $\text{\LaTeX}3$ offers the type “num”.

One reason is the limited number of registers inside $\text{T}_{\text{E}}\text{X}$. However, when using $\varepsilon\text{-T}_{\text{E}}\text{X}$ this is no longer an issue. It remains to be seen if there are other compelling reasons to keep this module.

It turns out there might be as with a $\langle num \rangle$ data type, the allocation module can do its bookkeeping without the aid of $\langle int \rangle$ registers.

9.1 Functions

$\backslash\text{num_new:N}$
$\backslash\text{num_new:c}$

 $\backslash\text{num_new:N} \quad \langle num \rangle$

Defines $\langle num \rangle$ to be a new variable of type `num` (initialized to zero). There is no way to define constant counters with these functions.

$\backslash\text{num_incr:N}$
$\backslash\text{num_incr:c}$
$\backslash\text{num_gincr:N}$
$\backslash\text{num_gincr:c}$

 $\backslash\text{num_incr:N} \quad \langle num \rangle$

Increments $\langle num \rangle$ by one. For global variables the global versions should be used.

$\backslash\text{num_decr:N}$
$\backslash\text{num_decr:c}$
$\backslash\text{num_gdecr:N}$
$\backslash\text{num_gdecr:c}$

 $\backslash\text{num_decr:N} \quad \langle num \rangle$

Decrements $\langle num \rangle$ by one. For global variables the global versions should be used.

$\backslash\text{num_zero:N}$
$\backslash\text{num_zero:c}$
$\backslash\text{num_gzero:N}$
$\backslash\text{num_gzero:c}$

 $\backslash\text{num_zero:N} \quad \langle num \rangle$

Resets $\langle num \rangle$ to zero. For global variables the global versions should be used.

$\backslash\text{num_set:Nn}$
$\backslash\text{num_set:cn}$
$\backslash\text{num_gset:Nn}$
$\backslash\text{num_gset:cn}$

 $\backslash\text{num_set:Nn} \quad \langle num \rangle \{ \langle integer \rangle \}$

These functions will set the $\langle num \rangle$ register to the $\langle integer \rangle$ value.

$\backslash\text{num_gset_eq:NN}$
$\backslash\text{num_gset_eq:cN}$
$\backslash\text{num_gset_eq:Nc}$
$\backslash\text{num_gset_eq:cc}$

 $\backslash\text{num_gset_eq:NN} \quad \langle num1 \rangle \langle num2 \rangle$

These functions will set the $\langle num1 \rangle$ register equal to $\langle num2 \rangle$.

<code>\num_add:Nn</code>
<code>\num_add:cn</code>
<code>\num_gadd:Nn</code>
<code>\num_gadd:cn</code>

`\num_add:Nn $\langle num \rangle$ { $\langle integer \rangle$ }`

These functions will add to the $\langle num \rangle$ register the value $\langle integer \rangle$. If the second argument is a $\langle num \rangle$ register too, the surrounding braces can be left out.

<code>\num_use:N</code>
<code>\num_use:c</code>

`\num_use:N $\langle num \rangle$`

This function returns the integer value kept in $\langle num \rangle$ in a way suitable for further processing.

TeXhackers note: Since these $\langle num \rangle$ s are implemented as macros, the function `\num_use:N` is effectively a noop and mainly there for consistency with similar functions in other modules.

<code>\num_eval:n</code>

`\num_eval:n { $\langle integer\text{-}expr \rangle$ }`

Evaluates the integer expression allowing normal mathematical operators like $+-/*$.

<code>\num_compare:nNnTF</code>
<code>\num_compare:cNcTF</code>
<code>\num_compare:nNnT</code>
<code>\num_compare:nNnF</code>

`\num_compare:nNnTF { $\langle num\ expr \rangle$ } $\langle rel \rangle$ { $\langle num\ expr \rangle$ }
{ $\langle true \rangle$ } { $\langle false \rangle$ }`

These functions test two $\langle num \rangle$ expressions against each other. They are both evaluated by `\num_eval:n`.

<code>\num_compare_p:nNn</code>

`\num_compare_p:nNn { $\langle num\ expr \rangle$ } $\langle rel \rangle$ { $\langle num\ expr \rangle$ }`

A predicate version of the above functions.

<code>\num_max_of:nn</code>
<code>\num_min_of:nn</code>

`\num_max_of:nn { $\langle num\ expr \rangle$ } { $\langle num\ expr \rangle$ }`

Return the largest or smallest of two $\langle num \rangle$ expressions.

<code>\num_abs:n</code>

`\num_abs:n { $\langle num\ expr \rangle$ }`

Return the numerical value of a $\langle num \rangle$ expression.

<code>\num_elt_count:n</code>

`\num_elt_count:n { $\langle balanced\ text \rangle$ }`

Discards $\langle balanced\ text \rangle$ and puts a +1 in the input stream. Used to count elements in a token list.

9.2 Formatting a counter value

See the `l3int` module for ways of doing this.

9.3 Variable and constants

`\const_new:Nn` `\const_new:Nn \c_⟨value⟩ { ⟨value⟩ }`

Defines a constant with $\langle value \rangle$. If the constant is negative or very large it requires an $\langle int \rangle$ register.

```
\c_minus_one
\c_zero
\c_one
\c_two
\c_three
\c_four
\c_six
\c_seven
\c_nine
\c_ten
\c_eleven
\c_sixteen
\c_hundred_one
\c_twohundred_fifty_five
\c_twohundred_fifty_six
\c_thousand
\c_ten_thousand
\c_twenty_thousand
```

Set of constants denoting useful values.

T_EXhackers note: Most of these constants have been available under L^AT_EX2 under names like `\tw@`, `\thr@@` etc.

```
\l_tmpa_num
\l_tmpb_num
\l_tmpc_num
\g_tmpa_num
\g_tmpb_num
```

Scratch register for immediate use. They are not used by conditionals or predicate functions.

9.4 Primitive functions

`\num_value:w` `\num_value:w ⟨integer⟩`
`\num_value:w` `\num_value:w ⟨tokens⟩ ⟨optional space⟩`

Expands $\langle tokens \rangle$ until an $\langle integer \rangle$ is formed. One space may be gobbled in the process. Preferably use with `\num_eval:n`.

T_EXhackers note: This is the T_EX primitive `\number`.

`\num_eval:w` `\num_eval:w` *<integer expression>* `\num_eval_end:`

Evaluates *<integer expression>*. The evaluation stops when an unexpandable token of catcode other than 12 is reached or `\num_eval_end:` is read. The latter is gobbled by the scanner mechanism.

T_EXhackers note: This is the ε -T_EX primitive `\numexpr`.

`\if_num:w` `\if_num:w` *<number1>* *<rel>* *<number2>* *<true>* *\else: <false>*
`\fi:`

Compare two numbers. It is recommended to use `\num_eval:n` to correctly evaluate and terminate these numbers. *<rel>* is one of *<*, *=* or *>* with catcode 12.

T_EXhackers note: This is the T_EX primitive `\ifnum`.

`\if_num_odd:w` `\if_num_odd:w` *<number>* *<true>* *\else: <false>* *\fi:*
 Execute *<true>* if *<number>* is odd, *<false>* otherwise.

T_EXhackers note: This is the T_EX primitive `\ifodd`.

`\if_case:w` `\if_case:w` *<number>* *<case0>* *\or: <case1>* *\or: ... \else:*
`\or: <default>` *\fi:*

Chooses case*<number>*. If you wish to use negative numbers as well, you can offset them with `\num_eval:n`.

T_EXhackers note: These are the T_EX primitives `\ifcase` and `\or`.

9.5 The Implementation

We start by ensuring that the required packages are loaded.

```
1983 <package>\ProvidesExplPackage
1984 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
1985 <package&!check>\RequirePackage{l3expan}\par
1986 <package & check>\RequirePackage{l3chk}\par
1987 <*initex | package>
```

```
\num_value:w Here are the remaining primitives for number comparisons and expressions.
\num_eval:w
\num_eval_end: 1988 \let_new:NN \num_value:w \tex_number:D
\num_eval_end: 1989 \let_new:NN \num_eval:w \etex_numexpr:D
\num_eval_end: \if_num:w
\num_eval_end: 1990 \let_new:NN \num_eval_end: \scan_stop:
\num_eval_end: \if_num:w
\num_eval_end: 1991 \let_new:NN \if_num:w \tex_ifnum:D
\num_eval_end: \if_case:w
\num_eval_end: 1992 \let_new:NN \if_num_odd:w \tex_ifodd:D
\num_eval_end: \or:
\num_eval_end: 1993 \let_new:NN \if_case:w \tex_ifcase:D
\num_eval_end: 1994 \let_new:NN \or: \tex_or:D
```

Functions that support L^AT_EX's user accessible counters should be added here, too. But first the internal counters.

```

\num_incr:N   Incrementing and decrementing of integer registers is done with the following functions.
\num_decr:N
1995 \def:Npn \num_incr:N #1{\num_add:Nn#1 1}
\num_gincr:N 1996 \def:Npn \num_decr:N #1{\num_add:Nn#1 \c_minus_one}
\num_gdecr:N 1997 \def:Npn \num_gincr:N #1{\num_gadd:Nn#1 1}
              1998 \def:Npn \num_gdecr:N #1{\num_gadd:Nn#1 \c_minus_one}

\num_incr:c   We also need ...
\num_decr:c
1999 \def_new:Npn \num_incr:c {\exp_args:Nc \num_incr:N}
\num_gincr:c 2000 \def_new:Npn \num_decr:c {\exp_args:Nc \num_decr:N}
\num_gdecr:c 2001 \def_new:Npn \num_gincr:c {\exp_args:Nc \num_gincr:N}
              2002 \def_new:Npn \num_gdecr:c {\exp_args:Nc \num_gdecr:N}

\num_zero:N   We also need ...
\num_zero:c
2003 \def_new:Npn \num_zero:N #1 {\num_set:Nn #1 0}
\num_gzero:N 2004 \def_new:Npn \num_gzero:N #1 {\num_gset:Nn #1 0}
\num_gzero:c 2005 \def_new:Npn \num_zero:c {\exp_args:Nc \num_zero:N}
              2006 \def_new:Npn \num_gzero:c {\exp_args:Nc \num_gzero:N}

\num_new:N    Allocate a new num variable and initialize it with zero.
\num_new:c
2007 \def_new:Npn \num_new:N #1{\tlp_new:Nn #1{0}}
2008 \def_new:Npn \num_new:c {\exp_args:Nc \num_new:N}

\num_eval:n   This function enables us to do all the operations without the aid of an int register.
              2009 \def_new:Npn \num_eval:n #1{\num_eval:w #1\num_eval_end:}

\num_set:Nn   Assigning values to num registers.
\num_set:cn
2010 \def_new:Npn \num_set:Nn #1#2{
\num_gset:Nn 2011   \tlp_set:No #1{ \tex_number:D \num_eval:n {#2} }
\num_gset:cn 2012 }
              2013 \def_new:Npn \num_gset:Nn {\pref_global:D \num_set:Nn}
              2014 \def_new:Npn \num_set:cn {\exp_args:Nc \num_set:Nn }
              2015 \def_new:Npn \num_gset:cn {\exp_args:Nc \num_gset:Nn }

\num_set_eq:NN Setting num registers equal to each other.
\num_set_eq:cN
2016 \let_new:NN \num_set_eq:NN \tlp_set_eq:NN
\num_set_eq:Nc 2017 \def_new:Npn \num_set_eq:cN {\exp_args:Nc \num_set_eq:NN}
\num_set_eq:cc 2018 \def_new:Npn \num_set_eq:Nc {\exp_args:NNc \num_set_eq:NN}
              2019 \def_new:Npn \num_set_eq:cc {\exp_args:Ncc \num_set_eq:NN}

\num_gset_eq:NN Setting num registers equal to each other.
\num_gset_eq:cN
2020 \let_new:NN \num_gset_eq:NN \tlp_gset_eq:NN
\num_gset_eq:Nc 2021 \def_new:Npn \num_gset_eq:cN {\exp_args:Nc \num_gset_eq:NN}
\num_gset_eq:cc 2022 \def_new:Npn \num_gset_eq:Nc {\exp_args:NNc \num_gset_eq:NN}
              2023 \def_new:Npn \num_gset_eq:cc {\exp_args:Ncc \num_gset_eq:NN}

```

```

\num_add:Nn Adding is easily done as the second argument goes through \num_eval:n.
\num_add:cn
\num_gadd:Nn 2024 \def_new:Npn \num_add:Nn #1#2 {\num_set:Nn #1{#1+#2}}
\num_gadd:cn 2025 \def_new:Npn \num_add:cn {\exp_args:Nc\num_add:Nn}
\num_gadd:cn 2026 \def_new:Npn \num_gadd:Nn {\pref_global:D \num_add:Nn}
2027 \def_new:Npn \num_gadd:cn {\exp_args:Nc\num_gadd:Nn}

```

```

\num_use:N Here is how num macros are accessed:
\num_use:c
2028 \let_new:NN\num_use:N \use_arg_i:n
2029 \let_new:NN\num_use:c \cs_use:c

```

```

\num_compare:nNnTF Simple comparison tests.
\num_compare:nNnT
\num_compare:nNnF 2030 \def_test_function_new:npn {num_compare:nNn}#1#2#3{
2031 \if_num:w \num_eval:n {#1}#2\num_eval:n {#3}
2032 }
2033 \def_new:Npn \num_compare:cNcTF { \exp_args:NcNc\num_compare:nNnTF }

```

```

\num_compare_p:nNn A predicate function.
2034 \def_new:Npn \num_compare_p:nNn #1#2#3{
2035 \if_num:w \num_eval:n {#1}#2\num_eval:n {#3}
2036 \c_true
2037 \else:
2038 \c_false
2039 \fi:
2040 }

```

```

\num_max_of:nn Functions for min, max, and absolute value.
\num_min_of:nn
\num_abs:n 2041 \def_new:Npn \num_abs:n#1{
2042 \if_num:w \num_eval:n{#1}<\c_zero \exp_after:NN -\fi: #1
2043 }
2044 \def_new:Npn \num_max_of:nn#1#2{\num_compare:nNnTF {#1}>{#2}{#1}{#2}}
2045 \def_new:Npn \num_min_of:nn#1#2{\num_compare:nNnTF {#1}<{#2}{#1}{#2}}

```

```

\num_elt_count:n Helper function for counting elements in a list.
\num_elt_count_prop:Nn
2046 \def_long_new:Npn \num_elt_count:n #1 { + 1 }
2047 \def_long_new:Npn \num_elt_count_prop:Nn #1#2 { + 1 }

```

```

\l_tmpa_num We provide an number local and two global (num)s, maybe we need more or less.
\l_tmpb_num
\l_tmpc_num 2048 \num_new:N \l_tmpa_num
\g_tmpa_num 2049 \num_new:N \l_tmpb_num
\g_tmpb_num 2050 \num_new:N \l_tmpc_num
2051 \num_new:N \g_tmpa_num
2052 \num_new:N \g_tmpb_num

```

9.5.1 Defining constants

As stated, most constants can be defined as `\tex_chardef:D` or `\tex_mathchardef:D` but that's engine dependent. Omega/Aleph allows `\tex_chardef:D`s up to 65535 which is also the maximum number of registers of all types.

```

\const_new:Nn
\c_max_register_num
\const_new_aux:Nw
2053 % \begin{macrocode}
2054 \engine_if_aleph:TF
2055 {
2056 \let_new:NN \const_new_aux:Nw \tex_chardef:D
2057 \const_new_aux:Nw \c_max_register_num = 65535 \scan_stop:
2058 }
2059 {
2060 \let_new:NN \const_new_aux:Nw \tex_mathchardef:D
2061 \const_new_aux:Nw \c_max_register_num = 32767 \scan_stop:
2062 }
2063 \def_new:Npn \const_new:Nn #1#2 {
2064 \num_compare:nNnTF {#2} > \c_minus_one
2065 {
2066 \num_compare:nNnTF {#2} > \c_max_register_num
2067 {\int_new:N #1 \int_set:Nn #1{#2}}
2068 {\chk_new_cs:N #1 \const_new_aux:Nw #1 = #2 \scan_stop: }
2069 }
2070 {\int_new:N #1 \int_set:Nn #1{#2}}
2071 }

```

`\c_minus_one` And the usual constants, others are still missing. Please, make every constant a real constant at least for the moment. We can easily convert things in the end when we have found what constants are used in critical places and what not.

```

\c_zero
\c_one
\c_two
\c_three 2072 %% \tex_countdef:D \c_minus_one = 10 \scan_stop:
\c_four 2073 %% \c_minus_one = -1 \scan_stop: %% in l3basics
\c_six 2074 %% \tex_chardef:D \c_sixteen = 16 \scan_stop: %% in l3basics
\c_seven 2075 \const_new:Nn \c_zero {0}
\c_nine 2076 \const_new:Nn \c_one {1}
\c_ten 2077 \const_new:Nn \c_two {2}
\c_eleven 2078 \const_new:Nn \c_three {3}
\c_sixteen 2079 \const_new:Nn \c_four {4}
\c_thirty_two 2080 \const_new:Nn \c_six {6}
\c_hundred_one 2081 \const_new:Nn \c_seven {7}
\c_twohundred_fifty_five 2082 \const_new:Nn \c_nine {9}
\c_twohundred_fifty_six 2083 \const_new:Nn \c_ten {10}
\c_thousand 2084 \const_new:Nn \c_eleven {11}
2085 \const_new:Nn \c_thirty_two {32}
\c_ten_thousand
\c_ten_thousand_one
\c_ten_thousand_two
\c_ten_thousand_three 2086 \const_new:Nn \c_hundred_one {101}
\c_ten_thousand_four 2087 \const_new:Nn \c_twohundred_fifty_five {255}
\c_twenty_thousand 2088 \const_new:Nn \c_twohundred_fifty_six {256}
2089 \const_new:Nn \c_thousand {1000}

```

The next one may seem a little odd (obviously!) but is useful when dealing with logical operators.


```

2090 \const_new:Nn \c_ten_thousand      {10000}
2091 \const_new:Nn \c_ten_thousand_one   {10001}
2092 \const_new:Nn \c_ten_thousand_two   {10002}
2093 \const_new:Nn \c_ten_thousand_three {10003}
2094 \const_new:Nn \c_ten_thousand_four  {10004}
2095 \const_new:Nn \c_twenty_thousand    {20000}

2096 </initex | package>

```

10 Sequences

L^AT_EX3 implements a data type called ‘sequences’. These are special token lists that can be accessed via special function on the ‘left’. Appending tokens is possible at both ends. Appended token lists can be accessed only as a union. The token lists that form the individual items of a sequence might contain any tokens except two internal functions that are used to structure sequences (see section internal functions below). It is also possible to map functions on such sequences so that they are executed for every item on the sequence.

All functions that return items from a sequence in some $\langle tlp \rangle$ assume that the $\langle tlp \rangle$ is local. See remarks below if you need a global returned value.

The defined functions are not orthogonal in the sense that every possible variation possible is actually available. If you need a new variant use the expansion functions described in the package `l3expan` to build it.

Adding items to the left of a sequence can currently be done with either something like `\seq_put_left:Nn` or with a “stack” function like `\seq_push:Nn` which has the same effect. Maybe one should therefore remove the “left” functions totally.

10.1 Functions

<code>\seq_new:N</code>
<code>\seq_new:c</code>

`\seq_new:N` $\langle sequence \rangle$

Defines $\langle sequence \rangle$ to be a variable of type sequences.

<code>\seq_clear:N</code>
<code>\seq_clear:c</code>
<code>\seq_gclear:N</code>
<code>\seq_gclear:c</code>

`\seq_clear:N` $\langle sequence \rangle$

These functions locally or globally clear $\langle sequence \rangle$.

<code>\seq_put_left:Nn</code>
<code>\seq_put_left:No</code>
<code>\seq_put_left:Nx</code>
<code>\seq_put_left:cn</code>
<code>\seq_put_right:Nn</code>
<code>\seq_put_right:No</code>
<code>\seq_put_right:Nx</code>

`\seq_put_left:Nn` $\langle sequence \rangle$ $\langle token list \rangle$

Locally appends $\langle token\ list \rangle$ as a single item to the left or right of $\langle sequence \rangle$. $\langle token\ list \rangle$ might get expanded before appending.

<code>\seq_gput_left:Nn</code>
<code>\seq_gput_right:Nn</code>
<code>\seq_gput_right:Nc</code>
<code>\seq_gput_right:No</code>
<code>\seq_gput_right:cn</code>
<code>\seq_gput_right:co</code>
<code>\seq_gput_right:cc</code>

`\seq_gput_left:Nn $\langle sequence \rangle$ $\langle token\ list \rangle$`

Globally appends $\langle token\ list \rangle$ as a single item to the left or right of $\langle sequence \rangle$.

<code>\seq_get:NN</code>
<code>\seq_get:cN</code>

`\seq_get:NN $\langle sequence \rangle$ $\langle tlp \rangle$`

Functions that locally assign the left-most item of $\langle sequence \rangle$ to the token list pointer $\langle tlp \rangle$. Item is not removed from $\langle sequence \rangle$! If you need a global return value you need to code something like this:

```
\seq_get:NN  $\langle sequence \rangle$  \l_tmpa_tlp
\tlp_gset_eq:NN  $\langle global\ tlp \rangle$  \l_tmpa_tlp
```

But if this kind of construction is used often enough a separate function should be provided.

<code>\seq_set_eq:NN</code>

`\seq_set_eq:NN $\langle seq1 \rangle$ $\langle seq2 \rangle$`

Function that locally makes $\langle seq1 \rangle$ identical to $\langle seq2 \rangle$.

<code>\seq_gset_eq:NN</code>
<code>\seq_gset_eq:cN</code>
<code>\seq_gset_eq:Nc</code>
<code>\seq_gset_eq:cc</code>

`\seq_gset_eq:NN $\langle seq1 \rangle$ $\langle seq2 \rangle$`

Function that globally makes $\langle seq1 \rangle$ identical to $\langle seq2 \rangle$.

<code>\seq_gconcat:NNN</code>
<code>\seq_gconcat:ccc</code>

`\seq_gconcat:NNN $\langle seq1 \rangle$ $\langle seq2 \rangle$ $\langle seq3 \rangle$`

Function that concatenates $\langle seq2 \rangle$ and $\langle seq3 \rangle$ and globally assigns the result to $\langle seq1 \rangle$.

<code>\seq_map_variable:NNn</code>
<code>\seq_map_variable:cNn</code>

`\seq_map_variable:NNn $\langle sequence \rangle$ $\langle tlp \rangle$ { $\langle code\ using\ tlp \rangle$ }`

Every element in $\langle sequence \rangle$ is assigned to $\langle tlp \rangle$ and then $\langle code\ using\ tlp \rangle$ is executed. The operation is not expandable which means that it can't be used within write operations etc. However, this function can be nested which the others can't.

<code>\seq_map:NN</code>

`\seq_map:NN $\langle sequences \rangle$ $\langle function \rangle$`

This function applies $\langle function \rangle$ (which must be a function with one argument) to every

item of $\langle sequence \rangle$. $\langle function \rangle$ is not executed within a sub-group so that side effects can be achieved locally. The operation is not expandable which means that it can't be used within write operations etc.

In the current implementation the next functions are more efficient and should be preferred.

$\backslash seq_map_inline:Nn$ $\backslash seq_map_inline:cn$	$\backslash seq_map_inline:Nn \langle sequence \rangle \{ \langle inline\ function \rangle \}$
--	--

Applies $\langle inline\ function \rangle$ (which should be the direct coding for a function with one argument (i.e. use **##1** as the place holder for this argument)) to every item of $\langle sequence \rangle$. $\langle inline\ function \rangle$ is not executed within a sub-group so that side effects can be achieved locally. The operation is not expandable which means that it can't be used within write operations etc.

10.2 Predicates and conditionals

$\backslash seq_if_empty_p:N$	$\backslash seq_if_empty_p:N \langle sequence \rangle$
----------------------------------	---

This predicate returns 'true' if $\langle sequence \rangle$ is 'empty' i.e., doesn't contain any tokens.

$\backslash seq_if_empty:NtF$ $\backslash seq_if_empty:cTF$ $\backslash seq_if_empty:NF$ $\backslash seq_if_empty:cF$	$\backslash seq_if_empty:NtF \langle sequence \rangle \{ \langle true\ code \rangle \} \{ \langle false\ code \rangle \}$
--	---

Set of conditionals that test whether or not a particular $\langle sequence \rangle$ is empty and if so executes either $\langle true\ code \rangle$ or $\langle false\ code \rangle$.

$\backslash seq_if_in:NnTF$ $\backslash seq_if_in:cnTF$ $\backslash seq_if_in:coTF$ $\backslash seq_if_in:cxTF$ $\backslash seq_if_in:NnF$ $\backslash seq_if_in:cnF$	$\backslash seq_if_in:NnTF \langle sequ \rangle \{ \langle item \rangle \} \{ \langle true\ code \rangle \} \{ \langle false\ code \rangle \}$
--	--

Function that tests if $\langle item \rangle$ is in $\langle sequ \rangle$. Depending on the result either $\langle true\ code \rangle$ or $\langle false\ code \rangle$ is executed.

10.3 Internal functions

$\backslash seq_if_empty_err:N$	$\backslash seq_if_empty_err:N \langle sequence \rangle$
------------------------------------	---

Signals an L^AT_EX3 error if $\langle sequence \rangle$ is empty.

$\backslash seq_pop_aux:nnNN$	$\backslash seq_pop_aux:nnNN \langle assign1 \rangle \langle assign2 \rangle \langle sequence \rangle \langle tlp \rangle$
---------------------------------	--

Function that assigns the left-most item of $\langle sequence \rangle$ to $\langle tlp \rangle$ using $\langle assign1 \rangle$ and assigns the tail to $\langle sequence \rangle$ using $\langle assign2 \rangle$. This function could be used to implement a global return function.

<code>\seq_get_aux:w</code>
<code>\seq_pop_aux:w</code>
<code>\seq_put_aux:Nnn</code>
<code>\seq_put_aux:w</code>

Functions used to implement put and get operations. They are not for meant for direct use.

<code>\seq_elt:w</code>
<code>\seq_elt_end:</code>

Functions (usually used as constants) that separates items within a sequence. They might get special meaning during mapping operations and are not supposed to show up as tokens within an item appended to a sequence.

11 Sequence Stacks

Special sequences in L^AT_EX3 are ‘stacks’ with their usual operations of ‘push’, ‘pop’, and ‘top’. They are internally implemented as sequences and share some of the functions (like `\seq_new:N` etc.)

11.1 Functions

<code>\seq_push:Nn</code>
<code>\seq_push:No</code>
<code>\seq_push:cn</code>
<code>\seq_gpush:Nn</code>
<code>\seq_gpush:No</code>
<code>\seq_gpush:cn</code>

`\seq_push:Nn` $\langle stack \rangle$ { $\langle token list \rangle$ }

Locally or globally pushes $\langle token list \rangle$ as a single item onto the $\langle stack \rangle$. $\langle token list \rangle$ might get expanded before the operation.

<code>\seq_pop:NN</code>
<code>\seq_pop:cN</code>
<code>\seq_gpop:NN</code>
<code>\seq_gpop:cN</code>

`\seq_pop:NN` $\langle stack \rangle$ $\langle tlp \rangle$

Functions that assign the top item of $\langle stack \rangle$ to the token list pointer $\langle tlp \rangle$ and removes it from $\langle stack \rangle$!

<code>\seq_top:NN</code>
<code>\seq_top:cN</code>

`\seq_top:NN` $\langle stack \rangle$ $\langle tlp \rangle$

Functions that locally assign the top item of $\langle stack \rangle$ to the token list pointer $\langle tlp \rangle$. Item is not removed from $\langle stack \rangle$!

11.2 Predicates and conditionals

Use `seq` functions.

11.3 Implementation

We start by ensuring that the required packages are loaded.

```

2097 <package>\ProvidesExplPackage
2098 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
2099 <package&!check>\RequirePackage{l3quark}
2100 <package&!check>\RequirePackage{l3tlp}
2101 <package & check>\RequirePackage{l3chk}
2102 <package>\RequirePackage{l3expan}

```

A sequence is a control sequence whose top-level expansion is of the form ‘\seq_elt:w <text₁> \seq_elt_end: ... \seq_elt:w <text_n> ...’. We use explicit delimiters instead of braces around <text> to allow efficient searching for an item in the sequence.

```

\seq_elt:w We allocate the delimiters and make them errors if executed.
\seq_elt_end:
2103 <*initex | package>
2104 \let_new:NN \seq_elt:w \ERROR
2105 \let_new:NN \seq_elt_end: \ERROR

```

\seq_new:N Sequences are implemented using token lists.

```

\seq_new:c
2106 \def_new:Npn \seq_new:N #1{\tlp_new:Nn #1{}}
2107 \def_new:Npn \seq_new:c {\exp_args:Nc \seq_new:N}

```

\seq_clear:N Clearing a sequence is the same as clearing a token list.

```

\seq_clear:c
\seq_gclear:N 2108 \let_new:NN \seq_clear:N \tlp_clear:N
\seq_gclear:c 2109 \let_new:NN \seq_clear:c \tlp_clear:c
\seq_gclear:c 2110 \let_new:NN \seq_gclear:N \tlp_gclear:N
2111 \let_new:NN \seq_gclear:c \tlp_gclear:c

```

\seq_clear_new:N Clearing a sequence is the same as clearing a token list.

```

\seq_clear_new:c
\seq_gclear_new:N 2112 \let_new:NN \seq_clear_new:N \tlp_clear_new:N
\seq_gclear_new:c 2113 \let_new:NN \seq_clear_new:c \tlp_clear_new:c
\seq_gclear_new:c 2114 \let_new:NN \seq_gclear_new:N \tlp_gclear_new:N
2115 \let_new:NN \seq_gclear_new:c \tlp_gclear_new:c

```

\seq_if_empty_p:N A predicate which evaluates to \c_true iff the sequence is empty.

```

2116 \let_new:NN \seq_if_empty_p:N \tlp_if_empty_p:N

```

\seq_if_empty:NTF \seq_if_empty:NTF<seq><true case><false case> will check whether the <seq> is empty and then select one of the other arguments. \seq_if_empty:cTF turns its first argument into a control sequence to get the name of the sequence.

```

\seq_if_empty:NF
\seq_if_empty:cF
2117 \let_new:NN \seq_if_empty:NTF \tlp_if_empty:NTF
2118 \def_new:Npn \seq_if_empty:cTF {\exp_args:Nc\seq_if_empty:NTF}

```

A variant of this, is only to do something if the sequence is *not* empty.

```

2119 \let_new:NN \seq_if_empty:NF \tlp_if_empty:NF
2120 \def_new:Npn \seq_if_empty:cF {\exp_args:Nc\seq_if_empty:NF}

```

`\seq_if_empty_err:N` Signals an error if the sequence is empty.

```
2121 \def_new:Npn \seq_if_empty_err:N #1{\if_meaning:NN#1\c_empty_tlp
```

As I said before, I don't think we need to provide checks for this kind of error, since it is a severe internal macro package error that can not be produced by the user directly. Can it? So the next line of code should be probably removed.

```
2122 \tlp_clear:N \l_testa_tlp % catch prefixes
2123 \err_latex_bug:x{Empty~sequence~'\token_to_string:N#1'}\fi:}
```

`\seq_get:NN` `\seq_get:NN` $\langle sequence \rangle$ $\langle cmd \rangle$ defines $\langle cmd \rangle$ to be the leftmost element of $\langle sequence \rangle$.

`\seq_get:cN`

```
2124 \def_new:Npn \seq_get:NN #1{
2125   \seq_if_empty_err:N #1
2126   \exp_after:NN\seq_get_aux:w #1\q_stop}
2127 \def_new:Npn \seq_get_aux:w \seq_elt:w #1\seq_elt_end:
2128           #2\q_stop #3{\tlp_set:Nn #3{#1}}
2129 \def_new:Npn \seq_get:cN {\exp_args:Nc \seq_get:NN}
```

`\seq_pop_aux:nnNN` `\seq_pop_aux:nnNN` $\langle def_1 \rangle$ $\langle def_2 \rangle$ $\langle sequence \rangle$ $\langle cmd \rangle$ assigns the leftmost element of $\langle sequence \rangle$ to $\langle cmd \rangle$ using $\langle def_2 \rangle$, and assigns the tail of $\langle sequence \rangle$ to $\langle sequence \rangle$ using $\langle def_1 \rangle$.

```
2130 \def_new:Npn \seq_pop_aux:nnNN #1#2#3{
2131   \seq_if_empty_err:N #3
2132   \exp_after:NN\seq_pop_aux:w #3\q_stop #1#2#3}
2133 \def_new:Npn \seq_pop_aux:w \seq_elt:w #1\seq_elt_end:
2134           #2\q_stop #3#4#5#6{#3#5{#2}#4#6{#1}}
```

`\seq_put_aux:Nnn` `\seq_put_aux:Nnn` $\langle sequence \rangle$ $\langle left \rangle$ $\langle right \rangle$ adds the elements specified by $\langle left \rangle$ to the left of $\langle sequence \rangle$, and those specified by $\langle right \rangle$ to the right.

```
2135 \def_new:Npn \seq_put_aux:Nnn #1{
2136   \exp_after:NN\seq_put_aux:w #1\q_stop #1}
2137 \def_new:Npn \seq_put_aux:w #1\q_stop #2#3#4{\tlp_set:Nn #2{#3#1#4}}
```

`\seq_put_left:Nn` Here are the usual operations for adding to the left and right.

`\seq_put_left:No`

`\seq_put_left:Nx` 2138 \def_new:Npn \seq_put_left:Nn #1#2{

`\seq_put_left:cn`

We can't put in a `\use_noop:` instead of `{}` since this argument is passed literally (and we would end up with many `\use_noop:s` inside the sequences.

`\seq_put_right:Nn`

`\seq_put_right:No`

`\seq_put_right:Nx` 2139

```
\seq_put_aux:Nnn #1{\seq_elt:w #2\seq_elt_end:}{}
2140 \def_new:Npn \seq_put_left:cn {\exp_args:Nc\seq_put_left:Nn}
2141 \def_new:Npn \seq_put_left:No {\exp_args:NNo\seq_put_left:Nn}
2142 \def_new:Npn \seq_put_left:Nx {\exp_args:Nnx\seq_put_left:Nn}
2143 \def_new:Npn \seq_put_right:Nn #1#2{
2144   \seq_put_aux:Nnn #1{\seq_elt:w #2\seq_elt_end:}}
2145 \def_new:Npn \seq_put_right:No {\exp_args:NNo\seq_put_right:Nn}
2146 \def_new:Npn \seq_put_right:Nx {\exp_args:Nnx\seq_put_right:Nn}
```

```

\seq_gput_left:Nn   An here the global variants.
\seq_gput_right:Nn
\seq_gput_right:Nc 2147 \def_new:Npn \seq_gput_left:Nn {
\seq_gput_right:No 2148 \*check>
\seq_gput_right:cn 2149 \pref_global_chk:
\seq_gput_right:co 2150 \</check>
\seq_gput_right:cc 2151 \<-check> \pref_global:D
2152 \seq_put_left:Nn}
2153 \def_new:Npn \seq_gput_right:Nn {
2154 \*check>
2155 \pref_global_chk:
2156 \</check>
2157 \<-check> \pref_global:D
2158 \seq_put_right:Nn}
2159 \def_new:Npn \seq_gput_right:No {\exp_args:NNo \seq_gput_right:Nn}
2160 \def_new:Npn \seq_gput_right:Nc {\exp_args:NNc \seq_gput_right:Nn}
2161 \def_new:Npn \seq_gput_right:cn {\exp_args:Nc \seq_gput_right:Nn}
2162 \def_new:Npn \seq_gput_right:co {\exp_args:Nco \seq_gput_right:Nn}
2163 \def_new:Npn \seq_gput_right:cc {\exp_args:Ncc \seq_gput_right:Nn}

\seq_map_variable:NNn Nothing spectacular here. The shuffling of the arguments in \seq_map_variable:NNn
\seq_map_variable:cNn below could also be done with \exp_args:NNnE.
\seq_map_variable_aux:nw
\seq_map_break:w 2164 \def_new:Npn \seq_map_variable_aux:Nnw #1#2\seq_elt:w#3\seq_elt_end:{
2165 \tlp_set:Nn #1{#3}
2166 \quark_if_nil:NT #1 \seq_map_break:w
2167 #2
2168 \seq_map_variable_aux:Nnw #1{#2}
2169 }
2170 \def_new:Npn \seq_map_variable:NNn #1#2#3{
2171 \tlp_set:Nx #2 {\exp_not:n{\seq_map_variable_aux:Nnw #2{#3}}}
2172 \exp_after:NN #2 #1 \seq_elt:w \q_nil\seq_elt_end: \q_stop
2173 }
2174 \def_new:Npn \seq_map_variable:cNn{\exp_args:Nc\seq_map_variable:Nn}
2175 \let_new:NN \seq_map_break:w \use_none_delimit_by_q_stop:w

\seq_map:NN \seq_map:NN <sequence> <cmd> applies <cmd> to each element of <sequence>, from left
to right. Since we don't have braces, this implementation is not very efficient. It might
be better to say that <cmd> must be a function with one argument that is delimited by
\seq_elt_end:.

2176 \def_new:Npn \seq_map:NN #1#2{
2177 \def:Npn \seq_elt:w ##1\seq_elt_end: {#2{##1}}#1
2178 \let:NN \seq_elt:w \ERROR
2179 }

\seq_map_inline:Nn When no braces are used, this version of mapping seems more natural.
\seq_map_inline:cn
2180 \def_new:Npn \seq_map_inline:Nn #1#2{
2181 \def:Npn \seq_elt:w ##1\seq_elt_end: {#2}#1
2182 \let:NN \seq_elt:w \ERROR
2183 }
2184 \def_new:Npn \seq_map_inline:cn{\exp_args:Nc\seq_map_inline:Nn}

```

```

\seq_set_eq:NN We can set one seq equal to another.
\seq_set_eq:Nc
2185 \let_new:NN \seq_set_eq:NN \let:NN
2186 \def_new:Npn \seq_set_eq:Nc {\exp_args:Nnc \seq_set_eq:NN}

\seq_gset_eq:NN An of course globally which seems to be needed far more often.8
\seq_gset_eq:cN
\seq_gset_eq:Nc 2187 \let_new:NN \seq_gset_eq:NN \glet:NN
\seq_gset_eq:cc 2188 \def_new:Npn \seq_gset_eq:cN {\exp_args:Nc \seq_gset_eq:NN}
2189 \def_new:Npn \seq_gset_eq:Nc {\exp_args:Nnc \seq_gset_eq:NN}
2190 \def_new:Npn \seq_gset_eq:cc {\exp_args:Ncc \seq_gset_eq:NN}

\seq_gconcat:NNN \seq_gconcat:NNN <seq 1> <seq 2> <seq 3> will globally assign <seq 1> the concatenation
\seq_gconcat:ccc of <seq 2> and <seq 3>.

2191 \def_new:Npn \seq_gconcat:NNN #1#2#3{
2192 \tlp_gset:Nx #1 {\exp_not:o{#2}\exp_not:o{#3}}
2193 }
2194 \def_new:Npn \seq_gconcat:ccc{\exp_args:Nccc\seq_gconcat:NNN}

\seq_if_in:NnTF \seq_if_in:NnTF <seq> <item> <true case> <false case> will check whether <item> is in <seq>
\seq_if_in:cnTF and then either execute the <true case> or the <false case>. <true case> and <false case>
\seq_if_in:coTF may contain incomplete \if_charcode:w statements.
\seq_if_in:cxTF
\seq_if_in:NnF 2195 \def_new:Npn \seq_if_in:NnTF #1#2{
2196 \def:Npn\tmp:w
\seq_if_in:cnF 2197 ##1\seq_elt:w #2\seq_elt_end: ##2##3\q_stop{

Note that ##2 contains exactly one token which we can compare with \q_no_value.

2198 \if_meaning:NN\q_no_value##2
2199 \exp_after:NN\use_arg_ii:nn
2200 \else:
2201 \exp_after:NN\use_arg_i:nn
2202 \fi:
2203 }
2204 \exp_after:NN
2205 \tmp:w #1\seq_elt:w
2206 #2\seq_elt_end: \q_no_value \q_stop}
2207 \def_new:Npn \seq_if_in:coTF {\exp_args:Nco \seq_if_in:NnTF}
2208 \def_new:Npn \seq_if_in:cnTF {\exp_args:Nc \seq_if_in:NnTF}
2209 \def_new:Npn \seq_if_in:cxTF {\exp_args:Ncx \seq_if_in:NnTF}

2210 \def_new:Npn \seq_if_in:NnF #1#2 { \seq_if_in:NnTF #1{#2}\use_noop: }
2211 \def_new:Npn \seq_if_in:cnF {\exp_args:Nc \seq_if_in:NnF}

```

11.3.1 Stack operations

We build stacks from sequences, but here we put the specific functions together.

⁸To save a bit of space these functions could be made identical to those from the tlp or clist module.


```

\seq_push:Nn Since sequences can be used as stacks, we ought to have both ‘push’ and ‘pop’. In most
\seq_push:No cases they are nothing more then new names for old functions.
\seq_push:cn
\seq_pop:NN 2212 \let_new:NN \seq_push:Nn \seq_put_left:Nn
\seq_pop:cn 2213 \let_new:NN \seq_push:No \seq_put_left:No
2214 \let_new:NN \seq_push:cn \seq_put_left:cn
2215 \def_new:Npn \seq_pop:NN {\seq_pop_aux:nnNN \tlp_set:Nn \tlp_set:Nn}
2216 \def_new:Npn \seq_pop:cn {\exp_args:Nc \seq_pop:NN}

\seq_gpush:Nn I don’t agree with Denys that one needs only local stacks, actually I believe that one
\seq_gpush:No will probably need the functions here more often. In case of \seq_gpop:NN the value is
\seq_gpush:cn nevertheless returned locally.
\seq_gpush:NC
\seq_gpop:NN 2217 \let_new:NN \seq_gpush:Nn \seq_gput_left:Nn
\seq_gpop:cn 2218 \def_new:Npn \seq_gpush:No {\exp_args:NNo \seq_gpush:Nn}
2219 \def_new:Npn \seq_gpush:cn {\exp_args:Nc \seq_gpush:Nn}
2220 \def_new:Npn \seq_gpush:NC {\exp_args:NNC \seq_gpush:Nn}
2221 \def_new:Npn \seq_gpop:NN {\seq_pop_aux:nnNN \tlp_gset:Nn \tlp_set:Nn}
2222 \def_new:Npn \seq_gpop:cn {\exp_args:Nc \seq_gpop:NN}

\seq_top:NN Looking at the top element of the stack without removing it is done with this operation.
\seq_top:cn 2223 \let_new:NN \seq_top:NN \seq_get:NN
2224 \let_new:NN \seq_top:cn \seq_get:cn

```

Show token usage:

```

2225 </initex | package>
2226 < *showmemory>
2227 %\showMemUsage
2228 </showmemory>

```

12 Allocating registers and the like

This module provides the basic mechanism for allocating T_EX’s registers. While designing this we have to take into account the following characteristics:

- \box255 is reserved for use in the output routine, so it should not be allocated otherwise.
- T_EX can load up 256 hyphenation patterns (registers \tex_language:D 0-255),
- T_EX can load no more than 16 math families,
- T_EX supports no more than 16 io-streams for reading (\tex_read:D) and 16 io-streams for writing (\tex_write:D),
- T_EX supports no more than 256 inserts, Omega supports more.
- The other registers (\tex_count:D, \tex_dimen:D, \tex_skip:D, \tex_muskip:D, \tex_box:D, and \tex_toks:D range from 0 to 32768, but registers numbered above 255 are accessed somewhat less efficient.

- Registers could be allocated both globally and locally; the use of registers could also be globally or locally. Here we provide support for globally allocated registers for both global and local use and for locally allocated registers for local use only.

We also need to allow for some bookkeeping: we need to know which register was allocated last and which registers can not be allocated by the standard mechanisms.

12.1 Functions

`\alloc_setup_type:nnn` `\alloc_setup_type:nnn` $\langle type \rangle$ $\langle g_start_num \rangle$ $\langle l_start_num \rangle$

Sets up the storage needed for the administration of registers of type $\langle type \rangle$.

$\langle type \rangle$ should be a token list in braces, it can be one of `int`, `dimen`, `skip`, `muskip`, `box`, `toks`, `ior`, `iow`, `pattern`, or `ins`.

$\langle g_start_num \rangle$ is the number of the first not allocated global register, it will be incremented by 1 when the allocation is done. $\langle l_start_num \rangle$ is the number of the first not allocated local register, it will be decremented by 1 when the allocation is done.

`\alloc_reg:NnNN` `\alloc_reg:NnNN` $\langle g-l \rangle$ $\langle type \rangle$ $\langle alloc_cmd \rangle$ $\langle cs \rangle$

Performs the allocation of a register of type $\langle type \rangle$ to control sequence $\langle cs \rangle$, using the command $\langle alloc_cmd \rangle$. The `g` or `l` indicates whether the allocation should be global or local. This macro is the basic building block for the definition of the $\langle type \rangle_new:N$ commands

12.2 The Implementation

We start by ensuring that the required packages are loaded when this file is loaded as a package on top of L^AT_EX 2_ε.

```
2229 <package>\ProvidesExplPackage
2230 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
2231 <package>\RequirePackage{l3expan}
2232 <package>\RequirePackage{l3num}
2233 <package>\RequirePackage{l3seq}\par
```

```
2234 (*initex | package)
```

`\alloc_setup_type:nnn` For each type of register we need to ‘counters’ that hold the last allocated global or local register. We also need a sequence to store the ‘exceptions’.

```
2235 \def_new:Npn \alloc_setup_type:nnn #1 #2 #3{
2236   \num_new:c {g_ #1 _allocation_num}
2237   \num_new:c {l_ #1 _allocation_num}
2238   \seq_new:c {g_ #1 _allocation_seq}
2239   \num_set:cn {g_ #1 _allocation_num}{#2}
2240   \num_set:cn {l_ #1 _allocation_num}{#3}
2241 }
```

`\alloc_next_g:n` These routines find the next free register. For globally allocated registers we first increment the counter that keeps track of them.

```
2242 \def_new:Npn \alloc_next_g:n #1 {
2243   \num_gincr:c {g_ #1 _allocation_num}
```

Then we need to check whether we have run out of registers.

```
2244   \num_compare:cNcTF {g_ #1 _allocation_num} = {l_ #1 _allocation_num}
2245   {\io_put_term:x{We~ ran~ out~ of~ registers~ of~ type~ g_#1!}}
2246   {
```

We also need to check whether the value of the counter already occurs in the list of already allocated registers.

```
2247       \seq_if_in:cxTF {g_ #1 _allocation_seq}
2248       {\num_use:c{g_ #1 _allocation_num}}
2249       {\io_put_term:x{\num_use:c{g_ #1 _allocation_num}~Already~ allocated!}}
```

If it does, we find the next value.

```
2250           \alloc_next_g:n {#1} }
2251       {\use_noop:}
2252   }
```

By now the `.._allocation_num` counter will contain the number of the register we will assign a control sequence for.

```
2253 }
```

For the locally allocated registers we have a similar function.

```
2254 \def_new:Npn \alloc_next_l:n #1 {
2255   \num_gdecr:c {l_ #1 _allocation_num}
2256   \num_compare:cNcTF {g_ #1 _allocation_num} = {l_ #1 _allocation_num}
2257   {\io_put_term:x{We~ ran~ out~ of~ registers~ of~ type~ l_#1!}}
2258   {
2259       \seq_if_in:cxTF {g_ #1 _allocation_seq}
2260       {\num_use:c{l_ #1 _allocation_num}}
2261       {\io_put_term:x{\num_use:c{l_ #1 _allocation_num}~Already~ allocated!}}
2262       \alloc_next_l:n {#1} }
2263       {\io_put_term:x{\num_use:c{l_ #1 _allocation_num}~Free!}}
2264   }
2265 }
```

`\alloc_reg:NnNN` This internal macro performs the actual allocation. It's first argument is either 'g' for a globally allocated register or 'l' for a locally allocated register. The second argument is the type of register to allocate, the third argument is the command to use and the fourth argument is the control sequence that is to be defined to point to the register.

```
2266 \def_new:Npn \alloc_reg:NnNN #1 #2 #3 #4{
```

It first checks that the control sequence that is to denote the register does not already exist.

```
2267   \chk_new_cs:N #4
```

Next, it decides whether a prefix is needed for the allocation command;

```
2268 \if:w#1g
2269 \exp_after:NN \pref_global:D
2270 \fi:
```

And finally the actual allocation takes place.

```
2271 #3 #4 \num_use:c{#1_ #2 _allocation_num}
2272 %%\cs_record_meaning:N#1
```

All that's left to do is write a message in the log file.

```
2273 \io_put_log:x{
2274 \token_to_string:N#4=#2~register~\num_use:c{#1_ #2 _allocation_num}}
```

Finally, it calls `\alloc_next_<g/l>` to find the next free register number.

```
2275 \cs_use:c{alloc_next_#1:n} {#2}
2276 }
```

```
2277 <*showmemory>
2278 \showMemUsage
2279 </showmemory>
2280 </initex | package>
```

13 Low-level file i/o

\TeX is capable of reading from and writing up to 16 individual streams. These i/o operations are accessible in \LaTeX 3 with functions from the `\io..` modules. In most cases it will be sufficient for the programmer to use the functions provided by the auxiliary file module, but here are the necessary functions for manipulating private streams.

Sometimes it is not known beforehand how much text is going to be written with a single call. As a result some internal \TeX buffer may overflow. To avoid this kind of problem, \LaTeX 3 maintains beside direct write operations like `\iow_expanded:Nn` also so called “long” writes where the output is broken into individual lines on every blank in the text to be written. The resulting files are difficult to read for humans but since they usually serve only as internal storage this poses no problem.

Beside the functions that immediately act (e.g., `\iow_expanded:Nn`, etc.) we also have deferred operations that are saved away until the next page is finished. This allows to expand the `\tokens` at the right time to get correct page numbers etc.

13.1 Functions for output streams

<code>\iow_new:N</code>	<code>\iow_new:N <stream></code>
<code>\iow_new:c</code>	

Defines `<stream>` to be a new identifier denoting an output stream for use in subsequent functions.

T_EXhackers note: `\iow_new:N` corresponds to the plain T_EX `\newwrite` allocation routine.

<code>\iow_open:Nn</code>
<code>\iow_open:cn</code> <code>\iow_open:Nn <stream> { <file name> }</code>

Opens output stream `<stream>` to write to `<file name>`. The output stream is immediately available for use. If the `<stream>` was already used as an output stream to some other file, this file gets closed first.⁹ Also, all output streams still open at the end of the T_EX run will be automatically closed.

<code>\iow_expanded:Nn</code>
<code>\iow_unexpanded:Nn</code> <code>\iow_expanded:Nn <stream> { <tokens> }</code>

This function immediately writes the expansion of `<tokens>` to the output stream `<stream>`. If `<stream>` is not open output goes to the terminal. The variant `\iow_unexpanded:Nn` writes out `<tokens>` without any further expansion (verbatim).

<code>\iow_expanded_log:n</code>
<code>\iow_expanded_term:n</code>
<code>\iow_unexpanded_term:n</code> <code>\iow_expanded_log:n { <tokens> }</code>

These functions write to the transcript or to the terminal respectively. So they are equivalent to `\iow_expanded:Nn` where `<stream>` is the transcript file (`\c_iow_log_stream`) or the terminal (`\c_io_term_stream`).

<code>\iow_long_expanded:Nx</code>
<code>\iow_long_unexpanded:Nn</code> <code>\iow_long_expanded:Nn <stream> { <tokens> }</code>

Like `\iow_expanded:Nn` but splits `<tokens>` at every blank into separate lines.

<code>\iow_unexpanded_if_avail:Nn</code>
<code>\iow_unexpanded_if_avail:cn</code> <code>\iow_unexpanded_if_avail:Nn <stream> { <tokens> }</code>

This special function first checks if the `<stream>` is open of writing. If not it does nothing otherwise it behaves like `\iow_unexpanded:Nn`.

<code>\iow_deferred_expanded:Nn</code>
<code>\iow_deferred_unexpanded:Nn</code> <code>\iow_deferred_expanded:Nn <stream> { <tokens> }</code>

These functions save away `<tokens>` until the next page is ready to be shipped out. Then, in case of `\iow_deferred_expanded:Nn <tokens>` get expanded and afterwards written to `<stream>`. `\iow_deferred_expanded:Nn` also always needs `{}` around the second argument. The use of `\iow_deferred_unexpanded:Nn` is probably seldom necessary.

⁹This is a precaution since on some OS it is possible to open the same file for output more than once which then results in some internal errors at the end of the run.

T_EXhackers note: `\iow_deferred_expanded:Nn` was known as `\write`.

`\iow_newline:` `\iow_newline:`

Function that produces a new line when used within the $\langle token list \rangle$ that gets written some output stream in non-verbatim mode.

13.2 Functions for input streams

`\ior_new:N` `\ior_new:N $\langle stream \rangle$`

This function defines $\langle stream \rangle$ to be a new input stream constant.

T_EXhackers note: This is the new name and new implementation for plain T_EX's `\newread`.

`\ior_open:Nn` `\ior_open:Nn $\langle stream \rangle$ { $\langle file name \rangle$ }`

This function opens $\langle stream \rangle$ as an input stream for the external file $\langle file name \rangle$. If $\langle file name \rangle$ doesn't exist or is an empty file the stream is considered to be fully read, a condition which can be tested with `\ior_eof:N` etc. If $\langle stream \rangle$ was already used to read from some other file this file will be closed first. The input stream is ready for immediate use.

`\ior_close:N` `\ior_close:N $\langle stream \rangle$`

This function closes the read stream $\langle stream \rangle$.

T_EXhackers note: This is a new name for `\closein` but it is considered bad practice to make use of this knowledge :-)

`\ior_eof:N`

`\ior_eof:N` `\ior_eof:N $\langle stream \rangle$ { $\langle true code \rangle$ } { $\langle false code \rangle$ }`

Conditional that tests if some input stream is fully read. The condition is also true if the input stream is not open.

`\if_eof:w` `\if_eof:w $\langle stream \rangle$ $\langle true code \rangle$ \else: $\langle false code \rangle$ \fi:`

T_EXhackers note: This is the primitive `\ifeof` but we allow only a $\langle stream \rangle$ and not a plain number after it.

`\ior_to:NN`

`\ior_gto:NN` `\ior_to:NN $\langle stream \rangle$ $\langle tlp \rangle$`

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input stream $\langle stream \rangle$ and places the result locally or globally into $\langle tlp \rangle$. If $\langle stream \rangle$ is not open input is requested from the terminal.

13.3 Constants

<code>\c_iow_comment_char</code>
<code>\c_iow_lbrace_char</code>
<code>\c_iow_rbrace_char</code>

Constants that can be used to represent comment character, left and right brace in token lists that should be written to a file.

<code>\c_io_term_stream</code>

Input or output stream denoting the terminal. If used as an input stream the user is prompted with the name of the $\langle tlp \rangle$ (that is used in the call $\backslash\text{ior_to:NN}$ or $\backslash\text{ior_gto:NN}$) followed by an equal sign. If you don't want an automatic prompt of this sort "misuse" `\c_iow_log_stream` as an input stream.

<code>\c_iow_log_stream</code>

Output stream that writes only to the transcript file (e.g., the `.log` file on most systems). You may "misuse" this stream as an input stream. In this case it acts as a terminal stream without user prompting.

13.4 Internal functions

<code>\iow_long_expanded_aux:w</code>

Function used to implement immediate writing where a new line is started at every blank.

<code>\tex_read:D</code>
<code>\tex_immediate:D</code>
<code>\tex_closeout:D</code>
<code>\tex_openin:D</code>
<code>\tex_openout:D</code>

These are the functions of the primitive interface to $\text{T}_{\text{E}}\text{X}$.

$\text{T}_{\text{E}}\text{X}$ hackers note: The $\text{T}_{\text{E}}\text{X}$ primitives `\read`, `\immediate`, `\closeout`, `\openin`, and `\openout` are all renamed and should not be used by a programmer since the functionality is covered by the $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}3$ functions above.

13.5 The Implementation

We start by ensuring that the required packages are loaded.

```
2281 <package>\ProvidesExplPackage
2282 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
2283 <package & check>\RequirePackage{l3chk}\par
2284 <package>\RequirePackage{l3toks}\par
2285 <*initex | package>
```

This section is primarily concerned with input and output streams. The naming conventions for i/o streams is `ior` (for read) and `iow` (for write) as module names. e.g. `\c_ior_test_stream` is an input stream variable called 'test'.

13.5.1 Output streams

`\iow_new:N` Allocation of new output streams is done by these functions. As we currently do not
`\iow_new:c` distribute a new allocation module we nick the `\newwrite` function.

```
2286 <*initex>
2287 \alloc_setup_type:nnn {iow} \c_zero \c_sixteen
2288 \def_new:Npn \iow_new:N #1 {\alloc_reg:NnNN g {iow} \tex_chardef:D #1}
2289 </initex>
2290 <package>\let:NN \iow_new:N \newwrite
2291 \def_new:Npn \iow_new:c {\exp_args:Nc \iow_new:N}
```

`\iow_open:Nn` To open streams for reading or writing the following two functions are provided. The
`\iow_open:cn` streams are opened immediately.

From some bad experiences on the mainframe, I learned that it is better to force the close before opening a dataset for writing. We have to check whether this is also necessary in case of `\tex_openin:D`.

```
2292 \def_new:Npn \iow_open:Nn #1#2{\iow_close:N #1
2293   \tex_immediate:D\tex_openout:D#1#2\scan_stop:}
2294 \def_new:Npn \iow_open:cn {\exp_args:Nc \iow_open:Nn}
```

`\iow_close:N` Since we close output streams prior to opening, a separate closing operation is probably not necessary. But here it is, just in case.... Actually you will need this if you intend to write and then read in the same pass from some stream.

```
2295 \def_new:Npn \iow_close:N {\tex_immediate:D\tex_closeout:D}
```

`\c_io_term_stream` Here we allocate two output streams for writing to the transcript file only (`\c_iow_log_stream`)
`\c_iow_log_stream` and to both the terminal and transcript file (`\c_io_term_stream`). The latter can also be used to read from therefore it is called `..io...`

```
2296 \let_new:NN \c_io_term_stream \c_sixteen
2297 \let_new:NN \c_iow_log_stream \c_minus_one
```

Immediate writing

`\iow_expanded:Nn` An abbreviation for an often used operation, which immediately writes its second argument to the output stream.

```
2298 \def_new:Npn \iow_expanded:Nn {\tex_immediate:D\iow_deferred_expanded:Nn}
```

`\iow_unexpanded:Nn` This routine writes the second argument verbatim onto the output stream. If this stream isn't open, the output goes to the terminal. If the first argument is no output stream at all, we get an internal error.

```
2299 \def_new:Npn \iow_unexpanded:Nn #1#2{
2300   \iow_expanded:Nn #1{\exp_not:n{#2}}}
```


`\iow_expanded_log:n` Now we redefine two functions for which we needed a definition very early on. They both write their second argument fully expanded to the output stream.

```
2301 \def:Npn \iow_expanded_log:n {\iow_expanded:Nn \c_iow_log_stream}
2302 \def:Npn \iow_expanded_term:n{\iow_expanded:Nn \c_io_term_stream}
```

The second one isn't exactly equivalent to the old `\typeout` since we need to control expansion in the function we provide for the user.

`\iow_unexpanded_term:n` This function writes its argument verbatim to the the terminal.

```
2303 \def_new:Npn \iow_unexpanded_term:n {\iow_unexpanded:Nn \c_io_term_stream}
```

`\iow_unexpanded_if_avail:Nn` `\iow_unexpanded_if_avail:Nn` *<stream>* *<code>*. This routine writes its second argument unexpanded to the stream given by the first argument, provided that this stream was opened for writing. Note, that `#` characters get doubled within *<code>*.

```
2304 \def_new:Npn \iow_unexpanded_if_avail:Nn #1{
```

In this routine we have to check whether or not the output stream that was requested is defined at all. So we check if the name is still free.

```
2305 \cs_free:NTF #1\use_none:n {\iow_unexpanded:Nn #1}}
```

Note: the next function could be streamlined for speed if we use the faster `\cs_free:cTF`. (space viz time).

```
2306 \def_new:Npn \iow_unexpanded_if_avail:cn {
2307 \exp_args:Nc \iow_unexpanded_if_avail:Nn }
```

`\iow_long_expanded:Nn` `\iow_long_unexpanded:Nn` `\iow_long_expanded_aux:w` Another type of writing onto an output stream is used for potentially long token sequences. We break the output lines at every blank in the second argument. This avoids the problem of buffer overflow when reading back, or badly broken lines on systems with limited file records. The only thing we have to take care of, is the danger of two blanks in succession since these get converted into a `\par` when we read the stuff back. But this can happen only if things like two spaces find their way into the second argument. Usually, multiple spaces are removed by T_EX's scanner.

```
2308 \def_new:Npn \iow_long_expanded_aux:w #1#2#3{
2309 \group_begin:\tex_newlinechar:D'\ #1#2{#3}\group_end:}
2310 \def_new:Npn \iow_long_expanded:Nn {\iow_long_expanded_aux:w
2311 \iow_expanded:Nn}
2312 \def_new:Npn \iow_long_unexpanded:Nn {\iow_long_expanded_aux:w
2313 \iow_unexpanded:Nn}
```

Deferred writing With ϵ -T_EX available deferred writing is easy. The comments below are old.

Deferred writing to output streams is a bit more complicated because there seems to be no nice hack for writing unexpanded. The only relatively sure bet is to use `\token_to_meaning:N` expansion of some token list. That's the way the following functions are implemented.

Another possibility would be to reserve a certain number of scratch token registers that could be used to hold the tokens until after the next `\tex_shipout:D`. But such an approach would probably fail because of the limited number of available token registers that would need to be reserved for this special application.

`\iow_deferred_expanded:Nn` First the easy part, this is the primitive.

```
2314 \let:NN \iow_deferred_expanded:Nn \tex_write:D
```

`\iow_deferred_unexpanded:Nn` Now the harder part:

```
2315 \def_new:Npn \iow_deferred_unexpanded:Nn #1#2{
2316   \iow_deferred_expanded:Nn{\exp_not:n{#2}}
2317 }
2318 %% Old implementation:
2319 %\def_new:Npn \iow_deferred_unexpanded:Nn #1#2{
2320 %  \tlp_set:Nn \l_tmpa_tlp {#2}
2321 %  \tlp_set:Nx \l_tmpb_tlp
2322 %           {\iow_deferred_expanded:Nn #1{\tlp_to_str:N \l_tmpa_tlp}}
2323 %  \l_tmpb_tlp}
```

Long forms of these functions are not possible since the deferred writing will restore the value of `\tex_newlinechar:D` before it will have a chance to act. But on the other hand it is nevertheless possible to make all deferred writes long by setting the `\tex_newlinechar:D` inside the output routine just before the `\tex_shipout:D`. The only disadvantage of this method is the fact that messages to the terminal during this time will also then break at spaces. But we should consider this.

Special characters for writing

`\iow_newline:` Global variable holding the character that forces a new line when something is written to an output stream.

```
2324 \def_new:Npn \iow_newline: {\^~J}
```

`\c_iow_comment_char` We also need to be able to write braces and the comment character. We achieve
`\c_iow_lbrace_char` this by defining global constants to expand into a version of these characters with
`\c_iow_rbrace_char` `\tex_catcode:D = 12`.

```
2325 \tlp_new:Nx \c_iow_comment_char {\cs_to_str:N\%}
```

To avoid another allocation function which is probably only necessary here we use the `\def:Npx` command directly.

```
2326 \tlp_new:Nx \c_iow_lbrace_char{\cs_to_str:N\{ }
2327 \tlp_new:Nx \c_iow_rbrace_char{\cs_to_str:N\}}
```

13.5.2 Input streams

`\ior_new:N` Allocation of new input streams is done by this function. As we currently do not distribute a new allocation module we nick the `\newread` function.

```
2328 <*initex>
2329 \alloc_setup_type:nnn {ior} \c_zero \c_sixteen
2330 \def_new:Npn \ior_new:N #1 {\alloc_reg:NnNN g {ior} \tex_chardef:D #1}
2331 </initex>
2332 <package>\let:NN \ior_new:N \newread
```

`\ior_open:Nn` Processing of input-streams (via `\tex_openin:D` and `closein`) is always ‘immediate’ as far as \TeX is concerned. An extra `\tex_immediate:D` is silently ignored.

```
2333 \let:NN \ior_close:N \tex_closein:D
2334 \def_new:Npn \ior_open:Nn #1#2{\ior_close:N #1\scan_stop:
2335                                \tex_openin:D#1#2\scan_stop:}
```

`\ior_eof:NTF` `\ior_eof:NTF` *<stream>* *<true case>* *<false case>*. To test if some particular input stream is exhausted the following conditional is provided:

```
2336 \def_new:Npn \ior_eof:NTF #1{\if_eof:w#1
2337     \exp_after:NN\use_arg_i:nn \else:
2338     \exp_after:NN\use_arg_ii:nn \fi:}
```

`\ior_eof:NF` `\ior_eof:NF` *<stream>* *<false case>*. Do something if if there is still something to read `\if_eof:w` from this file:

```
2339 \let:NN \if_eof:w \tex_ifeof:D
2340 \def_new:Npn \ior_eof:NF #1{\if_eof:w#1
2341     \exp_after:NN \use_none:nn \fi: \use_arg_i:n}
```

`\ior_to:NN` And here we read from files.

```
\ior_gto:NN
2342 <*check>
2343 \def_new:Npn \ior_to:NN #1#2{\tex_read:D#1to#2
2344     \chk_local_or_pref_global:N #2}
2345 </check>
2346 <-check> \def_new:Npn \ior_to:NN #1{\tex_read:D#1to}
2347 \def_new:Npn \ior_gto:NN {
2348 <*check>
2349     \pref_global_chk:
2350 </check>
2351 <-check> \pref_global:D
2352     \ior_to:NN}
```

Show token usage:

```
2353 </initex | package>
2354 <*showmemory>
2355 \showMemUsage
2356 </showmemory>
```

14 Comma lists

L^AT_EX3 implements a data type called ‘clist (comma-lists)’. These are special token lists that can be accessed via special function on the ‘left’. Appending tokens is possible at both ends. Appended token lists can be accessed only as a union. The token lists that form the individual items of a comma-list might contain any tokens except for commas that are used to structure comma-lists (braces are need if commas are part of the value). It is also possible to map functions on such comma-lists so that they are executed for every item of the comma-list.

All functions that return items from a comma-list in some $\langle tlp \rangle$ assume that the $\langle tlp \rangle$ is local. See remarks below if you need a global returned value.

The defined functions are not orthogonal in the sense that every possible variation possible is actually available. If you need a new variant use the expansion functions described in the package `l3expan` to build it.

Adding items to the left of a comma-list can currently be done with either something like `\clist_put_left:Nn` or with a “stack” function like `\clist_push:Nn` which has the same effect. Maybe one should therefore remove the “left” functions totally.

14.1 Functions

<code>\clist_new:N</code>
<code>\clist_new:c</code>

`\clist_new:N $\langle comma-list \rangle$`

Defines $\langle comma-list \rangle$ to be a variable of type clist.

<code>\clist_clear:N</code>
<code>\clist_clear:c</code>
<code>\clist_gclear:N</code>
<code>\clist_gclear:c</code>

`\clist_clear:N $\langle comma-list \rangle$`

These functions locally or globally clear $\langle comma-list \rangle$.

<code>\clist_put_left:Nn</code>
<code>\clist_put_left:No</code>
<code>\clist_put_left:Nx</code>
<code>\clist_put_left:cn</code>
<code>\clist_put_right:Nn</code>
<code>\clist_put_right:No</code>
<code>\clist_put_right:Nx</code>

`\clist_put_left:Nn $\langle comma-list \rangle$ $\langle token list \rangle$`

Locally appends $\langle token list \rangle$ as a single item to the left or right of $\langle comma-list \rangle$. $\langle token list \rangle$ might get expanded before appending.

<code>\clist_gput_left:Nn</code>
<code>\clist_gput_right:Nn</code>
<code>\clist_gput_right:No</code>
<code>\clist_gput_right:cn</code>
<code>\clist_gput_right:co</code>
<code>\clist_gput_right:cc</code>

`\clist_gput_left:Nn $\langle comma-list \rangle$ $\langle token list \rangle$`

Globally appends $\langle token\ list \rangle$ as a single item to the left or right of $\langle comma-list \rangle$.

<code>\clist_get:NN</code>
<code>\clist_get:cN</code>

`\clist_get:NN $\langle comma-list \rangle$ $\langle tlp \rangle$`

Functions that locally assign the left-most item of $\langle comma-list \rangle$ to the token list pointer $\langle tlp \rangle$. Item is not removed from $\langle comma-list \rangle$! If you need a global return value you need to code something like this:

```
\clist_get:NN  $\langle comma-list \rangle$  \l_tmpa_tlp
\tlp_gset_eq:NN  $\langle global\ tlp \rangle$  \l_tmpa_tlp
```

But if this kind of construction is used often enough a separate function should be provided.

<code>\clist_set_eq:NN</code>

`\clist_set_eq:NN $\langle clist1 \rangle$ $\langle clist2 \rangle$`

Function that locally makes $\langle clist1 \rangle$ identical to $\langle clist2 \rangle$.

<code>\clist_gset_eq:NN</code>
<code>\clist_gset_eq:cN</code>
<code>\clist_gset_eq:Nc</code>
<code>\clist_gset_eq:cc</code>

`\clist_gset_eq:NN $\langle clist1 \rangle$ $\langle clist2 \rangle$`

Function that globally makes $\langle clist1 \rangle$ identical to $\langle clist2 \rangle$.

<code>\clist_concat:NNN</code>
<code>\clist_gconcat:NNN</code>
<code>\clist_gconcat:NNc</code>
<code>\clist_gconcat:ccc</code>

`\clist_gconcat:NNN $\langle clist1 \rangle$ $\langle clist2 \rangle$ $\langle clist3 \rangle$`

Function that concatenates $\langle clist2 \rangle$ and $\langle clist3 \rangle$ and globally assigns the result to $\langle clist1 \rangle$.

<code>\clist_remove_duplicates:N</code>
<code>\clist_gremove_duplicates:N</code>

`\clist_gremove_duplicates:N $\langle clist \rangle$`

Function that removes any duplicate entries in $\langle clist \rangle$.

<code>\clist_use:N</code>
<code>\clist_use:c</code>

`\clist_use:N $\langle clist \rangle$`

Function that inserts the $\langle clist \rangle$ into the processing stream. Mainly useful if one knows what the $\langle clist \rangle$ contains, e.g., for displaying the content of template parameters.

14.2 Mapping functions

We provide three types of mapping functions, each with their own strengths. The `\clist_map_function:NN` is expandable whereas `\clist_map_inline:Nn` type uses `##1`

as a placeholder for the current item in $\langle \textit{clist} \rangle$. Finally we have the `\clist_map_variable:NNn` type which uses a user-defined variable as placeholder. Both the `_inline` and `_variable` versions are nestable.

<code>\clist_map_function:NN</code> <code>\clist_map_function:cN</code> <code>\clist_map_function:nN</code>	<code>\clist_map_function:NN</code> $\langle \textit{comma-list} \rangle$ $\langle \textit{function} \rangle$
---	---

This function applies $\langle \textit{function} \rangle$ (which must be a function with one argument) to every item of $\langle \textit{comma-list} \rangle$. $\langle \textit{function} \rangle$ is not executed within a sub-group so that side effects can be achieved locally. The operation is expandable which means that it can be used within write operations etc.

<code>\clist_map_inline:Nn</code> <code>\clist_map_inline:cn</code> <code>\clist_map_inline:nn</code>	<code>\clist_map_inline:Nn</code> $\langle \textit{comma-list} \rangle$ { $\langle \textit{inline function} \rangle$ }
---	--

Applies $\langle \textit{inline function} \rangle$ (which should be the direct coding for a function with one argument (i.e. use `##1` as the placeholder for this argument)) to every item of $\langle \textit{comma-list} \rangle$. $\langle \textit{inline function} \rangle$ is not executed within a sub-group so that side effects can be achieved locally. The operation is not expandable which means that it can't be used within write operations etc. These functions can be nested.

<code>\clist_map_variable:NNn</code> <code>\clist_map_variable:cNn</code> <code>\clist_map_variable:nNn</code>	<code>\clist_map_variable:NNn</code> $\langle \textit{comma-list} \rangle$ $\langle \textit{temp-var} \rangle$ { $\langle \textit{action} \rangle$ }
--	--

Assigns $\langle \textit{temp-var} \rangle$ to each element in $\langle \textit{clist} \rangle$ and then executes $\langle \textit{action} \rangle$ which should contain $\langle \textit{temp-var} \rangle$. As the operation performs an assignment, it is not expandable.

TeXhackers note: These functions resemble the L^AT_EX 2_ε function `\@for` but does not borrow the somewhat strange syntax.

<code>\clist_map_break:w</code>	<code>\clist_map_break:w</code>
---------------------------------	---------------------------------

For breaking out of a loop. To be used inside TF type functions as in the example below.

```
\def_new:Npn \test_function:n #1 {
  \int_compare:nNnTF {#1}> 3 {\clist_map_break:w}{‘‘#1’’}
}
\clist_map_function:nN {1,2,3,4,5,6,7,8}\test_function:n
```

This would return ‘‘1’’‘‘2’’‘‘3’’.

14.3 Predicates and conditionals

`\clist_if_empty_p:N` `\clist_if_empty_p:N <comma-list>`

This predicate returns ‘true’ if `<comma-list>` is ‘empty’ i.e., doesn’t contain any tokens.

<code>\clist_if_empty:N</code> <code>\clist_if_empty:c</code> <code>\clist_if_empty:N</code> <code>\clist_if_empty:c</code>	<code>\clist_if_empty:N</code> <code><comma-list></code> { <code><true code></code> } { <code><false code></code> }
--	---

Set of conditionals that test whether or not a particular `<comma-list>` is empty and if so executes either `<true code>` or `<false code>`.

<code>\clist_if_eq:N</code>	<code>\clist_if_eq:N</code> <code><comma-list1></code> <code><comma-list2></code> { <code><true code></code> } { <code><false code></code> }
-----------------------------	--

Check if `<comma-list1>` and `<comma-list2>` are equal and execute either `<true code>` or `<false code>` accordingly.

<code>\clist_if_in:N</code> <code>\clist_if_in:c</code> <code>\clist_if_in:N</code> <code>\clist_if_in:c</code>	<code>\clist_if_in:N</code> <code><comma-list></code> { <code><item></code> } { <code><true code></code> } { <code><false code></code> }
--	--

Function that tests if `<item>` is in `<comma-list>`. Depending on the result either `<true code>` or `<false code>` is executed.

14.4 Internal functions

`\clist_if_empty_err:N` `\clist_if_empty_err:N <comma-list>`

Signals an L^AT_EX3 error if `<comma-list>` is empty.

<code>\clist_pop_aux:nn</code>	<code>\clist_pop_aux:nn</code> <code><assign1></code> <code><assign2></code> <code><comma-list></code> <code><tlp></code>
--------------------------------	---

Function that assigns the left-most item of `<comma-list>` to `<tlp>` using `<assign1>` and assigns the tail to `<comma-list>` using `<assign2>`. This function could be used to implement a global return function.

<code>\clist_get_aux:w</code> <code>\clist_pop_aux:w</code> <code>\clist_pop_auxi:w</code> <code>\clist_put_aux:NNnn</code>
--

Functions used to implement put and get operations. They are not for meant for direct use.

<code>\clist_map_function_aux:Nw</code>
<code>\clist_map_inline_aux:Nw</code>
<code>\clist_map_variable_aux:Nnw</code>

Internal helper functions for the $\langle\textit{clist}\rangle$ mapping functions.

<code>\clist_concat_aux:NNNN</code>
<code>\clist_remove_duplicates_aux:NN</code>
<code>\clist_remove_duplicates_aux:n</code>
<code>\l_clist_remove_duplicates_clist</code>

Functions that help concatenate $\langle\textit{clist}\rangle$ s and remove duplicate elements from a $\langle\textit{clist}\rangle$.

14.5 Comma list Stacks

Special comma-lists in L^AT_EX3 are ‘stacks’ with their usual operations of ‘push’, ‘pop’, and ‘top’. They are internally implemented as comma-lists and share some of the functions (like `\clist_new:N` etc.)

<code>\clist_push:Nn</code>
<code>\clist_push:No</code>
<code>\clist_push:cn</code>
<code>\clist_gpush:Nn</code>
<code>\clist_gpush:No</code>
<code>\clist_gpush:cn</code>

`\clist_push:Nn <stack> { <token list> }`

Locally or globally pushes $\langle\textit{token list}\rangle$ as a single item onto the $\langle\textit{stack}\rangle$. $\langle\textit{token list}\rangle$ might get expanded before the operation.

<code>\clist_pop:NN</code>
<code>\clist_pop:cN</code>
<code>\clist_gpop:NN</code>
<code>\clist_gpop:cN</code>

`\clist_pop:NN <stack> <tlp>`

Functions that assign the top item of $\langle\textit{stack}\rangle$ to the token list pointer $\langle\textit{tlp}\rangle$ and removes it from $\langle\textit{stack}\rangle$!

<code>\clist_top:NN</code>
<code>\clist_top:cN</code>

`\clist_top:NN <stack> <tlp>`

Functions that locally assign the top item of $\langle\textit{stack}\rangle$ to the token list pointer $\langle\textit{tlp}\rangle$. Item is not removed from $\langle\textit{stack}\rangle$!

Use `clist` functions.

14.6 The Implementation

We start by ensuring that the required packages are loaded.

```

2357 <package>\ProvidesExplPackage
2358 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
2359 <*package>

```



```

2360 \NeedsTeXFormat{LaTeX2e}
2361 \!check\RequirePackage{l3prg,l3quark}
2362 \check\RequirePackage{l3chk}
2363 \!package
2364 \*initex|package)

```

`\clist_new:N` Comma-Lists are implemented using token lists.

```

\clist_new:c
2365 \def_new:Npn \clist_new:N #1{\tlp_new:Nn #1{}}
2366 \def_new:Npn \clist_new:c {\exp_args:Nc \clist_new:N}

```

`\clist_clear:N` Clearing a comma-list is the same as clearing a token list.

```

\clist_clear:c
\clist_gclear:N 2367 \let_new:NN \clist_clear:N \tlp_clear:N
\clist_gclear:c 2368 \let_new:NN \clist_clear:c \tlp_clear:c
\clist_gclear:c 2369 \let_new:NN \clist_gclear:N \tlp_gclear:N
2370 \let_new:NN \clist_gclear:c \tlp_gclear:c

```

`\clist_set_eq:NN` We can set one *<clist>* equal to another.

```

2371 \let_new:NN \clist_set_eq:NN \let:NN

```

`\clist_set_eq:NN` An of course globally which seems to be needed far more often.

```

\clist_set_eq:cN
\clist_set_eq:Nc 2372 \let_new:NN \clist_gset_eq:NN \glet:NN
\clist_set_eq:cc 2373 \def_new:Npn \clist_gset_eq:cN {\exp_args:Nc \clist_gset_eq:NN}
2374 \def_new:Npn \clist_gset_eq:Nc {\exp_args:NNc \clist_gset_eq:NN}
2375 \def_new:Npn \clist_gset_eq:cc {\exp_args:Ncc \clist_gset_eq:NN}

```

`\clist_if_empty_p:N` A predicate which evaluates to `\c_true` iff the comma-list is empty.

```

2376 \let_new:NN \clist_if_empty_p:N \tlp_if_empty_p:N

```

`\clist_if_empty:NTF` `\clist_if_empty:NTF<clist><true case><false case>` will check whether the *<clist>* is empty and then select one of the other arguments. `\clist_if_empty:cTF` turns its first argument into a control comma-list to get the name of the comma-list.

```

\clist_if_empty:cTF 2377 \def_test_function_new:npn {clist_if_empty:N}#1{\if_meaning:NN#1\c_empty_tlp}
\clist_if_empty:cT 2378 \def_new:Npn \clist_if_empty:cTF {\exp_args:Nc\clist_if_empty:NTF}
\clist_if_empty:cF 2379 \def_new:Npn \clist_if_empty:cT {\exp_args:Nc\clist_if_empty:NT}
2380 \def_new:Npn \clist_if_empty:cF {\exp_args:Nc\clist_if_empty:NF}

```

`\clist_if_empty_err:N` Signals an error if the comma-list is empty.

```

2381 \def_new:Npn \clist_if_empty_err:N #1{
2382   \if_meaning:NN#1\c_empty_tlp
2383   \tlp_clear:N \l_testa_tlp % catch prefixes
2384   \err_latex_bug:x{Empty~comma-list~'\token_to_string:N#1'}
2385   \fi:}

```

`\clist_if_eq:NNTF` As comma lists are token list pointers internally this is just an alias.

```

2386 \let_new:NN \clist_if_eq:NNTF \tlp_if_eq:NNTF

```

\clist_get:NN \clist_get:NN $\langle comma-list \rangle$ $\langle cmd \rangle$ defines $\langle cmd \rangle$ to be the left-most element of $\langle comma-list \rangle$.

```
\clist_get:cN
2387 \def_new:Npn \clist_get:NN #1{
2388   \clist_if_empty_err:N #1
2389   \exp_after:NN\clist_get_aux:w #1,\q_stop}

2390 \def_new:Npn \clist_get_aux:w #1,#2\q_stop #3{\tlp_set:Nn #3{#1}}

2391 \def_new:Npn \clist_get:cN {\exp_args:Nc \clist_get:NN}
```

\clist_pop_aux:nnNN \clist_pop_aux:nnNN $\langle def_1 \rangle$ $\langle def_2 \rangle$ $\langle comma-list \rangle$ $\langle cmd \rangle$ assigns the left-most element of $\langle comma-list \rangle$ to $\langle cmd \rangle$ using $\langle def_2 \rangle$, and assigns the tail of $\langle comma-list \rangle$ to $\langle comma-list \rangle$ using $\langle def_1 \rangle$.

```
2392 \def_new:Npn \clist_pop_aux:nnNN #1#2#3{
2393   \clist_if_empty_err:N #3
2394   \exp_after:NN\clist_pop_aux:w #3,\q_nil\q_stop #1#2#3}

2395 \def_new:Npn \clist_pop_aux:w #1,#2\q_stop #3#4#5#6{
2396   #4#6{#1}
2397   #3#5{#2}}
```

If there was only one element in the original clist, it now contains only \q_nil.

```
2398   \quark_if_nil:NTF #5
2399   { #3#5{} }
2400   { \clist_pop_auxi:w #2 #3#5 }
2401 }

2402 \def_new:Npn\clist_pop_auxi:w #1,\q_nil #2#3 {#2#3{#1}}
```

\clist_put_aux:NNnnNn The generic put function.

```
2403 \def_new:Npn \clist_put_aux:NNnnNn #1#2#3#4#5#6{
```

When adding we have to distinguish between an empty $\langle clist \rangle$ and one that contains at least one item (otherwise we accumulate commas).

```
2404   \clist_if_empty:NTF#5 {#1 #5{#6}}
```

MH says: Perhaps we should make sure that empty arguments don't get on the stack as that is probably a mistake. That's what I've implemented here. Since \tlist_if_empty:nF is expandable prefixes are still allowed.

```
2405   { \tlist_if_empty:nF {#6}{ #2 #5{#3#6#4} } }
2406 }
```

\clist_put_left:Nn The operations for adding to the left.

```
\clist_put_left:No
\clist_put_left:Nx 2407 \def_new:Npn \clist_put_left:Nn {
\clist_put_left:cN 2408   \clist_put_aux:NNnnNn \tlp_set:Nn \tlp_put_left:Nn {} ,
2409 }
2410 \def_new:Npn \clist_put_left:cN {\exp_args:Nc \clist_put_left:Nn}
2411 \def_new:Npn \clist_put_left:No {\exp_args:NNo\clist_put_left:Nn}
2412 \def_new:Npn \clist_put_left:Nx {\exp_args:Nnx\clist_put_left:Nn}
```

\clist_gput_left:Nn Global versions.

```
2413 \def_new:Npn \clist_gput_left:Nn {
2414   \clist_put_aux:NNnnNn \tlp_gset:Nn \tlp_gput_left:Nn {},
2415 }
```

\clist_put_right:Nn Adding something to the right side is almost the same.

```
\clist_put_right:cn
\clist_put_right:No 2416 \def_new:Npn \clist_put_right:Nn {
2417   \clist_put_aux:NNnnNn \tlp_set:Nn \tlp_put_right:Nn , {}
\clist_put_right:Nx 2418 }
2419 \def_new:Npn \clist_put_right:cn {\exp_args:Nc \clist_put_right:Nn}
2420 \def_new:Npn \clist_put_right:No {\exp_args:Nno \clist_put_right:Nn}
2421 \def_new:Npn \clist_put_right:Nx {\exp_args:Nnx \clist_put_right:Nn}
```

\clist_gput_right:Nn An here the global variants.

```
\clist_gput_right:No
\clist_gput_right:cn 2422 \def_new:Npn \clist_gput_right:Nn {
2423   \clist_put_aux:NNnnNn \tlp_gset:Nn \tlp_gput_right:Nn , {}
\clist_gput_right:co 2424 }
\clist_gput_right:cc 2425 \def_new:Npn \clist_gput_right:No {\exp_args:Nno \clist_gput_right:Nn}
\clist_gput_right:NC 2426 \def_new:Npn \clist_gput_right:cn {\exp_args:Nc \clist_gput_right:Nn}
2427 \def_new:Npn \clist_gput_right:co {\exp_args:Nco \clist_gput_right:Nn}
2428 \def_new:Npn \clist_gput_right:cc {\exp_args:Ncc \clist_gput_right:Nn}
2429 \def_new:Npn \clist_gput_right:NC {\exp_args:NNC \clist_gput_right:Nn}
```

\clist_map_function:nN \clist_map_function:NN *<comma-list>* *<cmd>* applies *<cmd>* to each element of *<comma-list>*,
\clist_map_function:cN from left to right.

```
\clist_map_function:NN
2430 \def_new:Npn \clist_map_function:NN #1#2{
2431   \clist_if_empty:NF #1
2432   {
2433     \exp_after:NN \clist_map_function_aux:Nw
2434     \exp_after:NN #2 #1 , \q_recursion_tail , \q_recursion_stop
2435   }
2436 }
2437 \def_new:Npn \clist_map_function:cN{\exp_args:Nc \clist_map_function:NN}
2438 \def_new:Npn \clist_map_function:nN #1#2{
2439   \tlist_if_blank:nF {#1}
2440   { \clist_map_function_aux:Nw #2 #1 , \q_recursion_tail , \q_recursion_stop }
2441 }
```

\clist_map_function_aux:Nw The general loop. Tests if we hit the first stop marker and exits if we did. If we didn't,
place the function #1 in front of the element #2, which is surrounded by braces.

```
2442 \def_long_new:Npn \clist_map_function_aux:Nw #1#2,{
2443   \quark_if_recursion_tail_stop:n{#2}
2444   #1{#2}
2445   \clist_map_function_aux:Nw #1
2446 }
```

`\clist_map_break:w` The break statement is easy. Same as in other modules, gobble everything up to the special recursion stop marker.

```
2447 \let_new:NN \clist_map_break:w \use_none_delimit_by_q_recursion_stop:w
```

`\clist_map_inline:Nn` The inline type is faster but not expandable. In order to make it nestable, we use a counter to keep track of the nesting level so that all of the functions called have distinct names. A simpler approach would of course be to use grouping and thus the save stack but then you lose the ability to do things locally.

A funny little thing occurred in one document: The command setting up the first call of `\clist_map_inline:Nn` was used in a tabular cell and the inline code used `\\` so the loop broke as soon as this happened. Lesson to be learned from this: If you wish to have group like structure but not using the groupings of `TeX`, then do every operation globally.

```
2448 \int_new:N \g_clist_inline_level_int
2449 \def_long_new:Npn \clist_map_inline:Nn #1#2{
2450   \clist_if_empty:NF #1
2451   {
2452     \int_gincr:N \g_clist_inline_level_int
2453     \gdef_long:cpn {clist_map_inline_ \int_use:N \g_clist_inline_level_int :n}
2454     ##1{#2}
```

Recall that the E in `\exp_args:NcE` means ‘single token expanded once and no braces added’. It is a lot more efficient to carry over the special function rather than constructing the same csname over and over again, so we just do it once. We reuse `\clist_map_function_aux:Nw` for the actual loop.

```
2455   \exp_args:NcE \clist_map_function_aux:Nw
2456   {clist_map_inline_ \int_use:N \g_clist_inline_level_int :n}
2457   #1 , \q_recursion_tail , \q_recursion_stop
2458   \int_gdecr:N \g_clist_inline_level_int
2459   }
2460 }
2461 \def_new:Npn \clist_map_inline:cn{\exp_args:Nc\clist_map_inline:Nn}
2462 \def_long_new:Npn \clist_map_inline:nn #1#2{
2463   \tlist_if_empty:NF {#1}
2464   {
2465     \int_gincr:N \g_clist_inline_level_int
2466     \gdef_long:cpn {clist_map_inline_ \int_use:N \g_clist_inline_level_int :n}
2467     ##1{#2}
2468     \exp_args:Nc \clist_map_function_aux:Nw
2469     {clist_map_inline_ \int_use:N \g_clist_inline_level_int :n}
2470     #1 , \q_recursion_tail , \q_recursion_stop
2471     \int_gdecr:N \g_clist_inline_level_int
2472   }
2473 }
```

`\clist_map_variable:nNn` `\clist_map:NNn` `<comma-list>` `<temp>` `<action>` assigns `<temp>` to each element and executes `<action>`.

`\clist_map_variable:cNn`

```
2474 \def_new:Npn \clist_map_variable:nNn #1#2#3{
2475   \tlist_if_empty:NF{#1}
```

```

2476 {
2477     \clist_map_variable_aux:Nnw #2 {#3} #1
2478     , \q_recursion_tail , \q_recursion_stop
2479 }
2480 }
2481 \def_new:Npn \clist_map_variable:NNn {\exp_args:No \clist_map_variable:nNn}
2482 \def_new:Npn \clist_map_variable:cNn {\exp_args:Nc \clist_map_variable:NNn}

```

\clist_map_variable_aux:Nnw The general loop. Assign the temp variable #1 to the current item #3 and then check if that's the stop marker. If it is, break the loop. If not, execute the action #2 and continue.

```

2483 \def_new:Npn \clist_map_variable_aux:Nnw #1#2#3,{
2484     \def:Npn #1{#3}
2485     \quark_if_recursion_tail_stop:N #1
2486     #2 \clist_map_variable_aux:Nnw #1{#2}
2487 }

```

```

\clist_concat_aux:NNNN \clist_gconcat:NNN <clist 1> <clist 2> <clist 3> will globally assign <clist 1> the con-
\clist_concat:NNN catenation of <clist 2> and <clist 3>.
\clist_gconcat:NNN
\clist_gconcat:NNc 2488 \def_new:Npn \clist_concat_aux:NNNN #1#2#3#4{
\clist_gconcat:ccc 2489     \toks_set:No \l_tmpa_toks {#3}
2490     \toks_set:No \l_tmpb_toks {#4}
2491     #1 #2 {
2492         \toks_use:N \l_tmpa_toks

```

Again the situation is a bit more complicated because of the use of commas between items, so if either list is empty we have to avoid adding a comma.

```

2493     \toks_if_empty:NF \l_tmpa_toks {\toks_if_empty:NF \l_tmpb_toks ,}
2494     \toks_use:N \l_tmpb_toks
2495 }
2496 }
2497 \def_new:Npn \clist_concat:NNN {\clist_concat_aux:NNNN \tlp_set:Nx}
2498 \def_new:Npn \clist_gconcat:NNN {\clist_concat_aux:NNNN \tlp_gset:Nx}

```

And the usual versions.

```

2499 \def_new:Npn \clist_gconcat:NNc{\exp_args:Nnnc\clist_gconcat:NNN}
2500 \def_new:Npn \clist_gconcat:ccc{\exp_args:Nccc\clist_gconcat:NNN}

```

list_remove_duplicates_aux:NN Removing duplicate entries in a <clist> is fairly straight forward. We use a temporary variable and then go through the list from left to right. For each element check if the element is already present in the list.

```

\clist_remove_duplicates:N
\clist_gremove_duplicates:N
2501 \def:Npn \clist_remove_duplicates_aux:NN #1#2 {
2502     \clist_clear:N \l_clist_remove_duplicates_clist
2503     \clist_map_function:NN #2 \clist_remove_duplicates_aux:n
2504     #1 #2 \l_clist_remove_duplicates_clist
2505 }
2506 \def:Npn \clist_remove_duplicates_aux:n #1 {
2507     \clist_if_in:NnTF \l_clist_remove_duplicates_clist {#1} {}
2508     {\clist_put_right:Nn \l_clist_remove_duplicates_clist {#1}}
2509 }

```

The high level functions are just for telling if it should be a local or global setting.

```

2510 \def_new:Npn \clist_remove_duplicates:N {
2511   \clist_remove_duplicates_aux:NN \clist_set_eq:NN
2512 }
2513 \def_new:Npn \clist_gremove_duplicates:N {
2514   \clist_remove_duplicates_aux:NN \clist_gset_eq:NN
2515 }

```

`\clist_remove_duplicates_clist`

```

2516 \clist_new:N \l_clist_remove_duplicates_clist

```

`\clist_use:N` Using a $\langle \textit{clist} \rangle$ is just executing it but...

`\clist_use:c`

```

2517 \def_new:Npn \clist_use:N #1 {
2518   \if_meaning:NN #1 \scan_stop:

```

... if $\langle \textit{clist} \rangle$ equals `\scan_stop:` it is probably stemming from a `\cs:w ... \cs_end:` that was created by mistake somewhere.

```

2519   \err_latex_bug:x {Comma~list~ '\token_to_string:N #1'~
2520                   has~ an~ erroneous~ structure!}
2521   \else:
2522     \exp_after:NN #1
2523   \fi:
2524 }
2525 \def_new:Npn \clist_use:c {\exp_args:Nc \clist_use:N}

```

`\clist_if_in:NnTF` `\clist_if_in:NnTF` $\langle \textit{clist} \rangle \langle \textit{item} \rangle \langle \textit{true case} \rangle \langle \textit{false case} \rangle$ will check whether $\langle \textit{item} \rangle$ is in $\langle \textit{clist} \rangle$ and then either execute the $\langle \textit{true case} \rangle$ or the $\langle \textit{false case} \rangle$. $\langle \textit{true case} \rangle$ and $\langle \textit{false case} \rangle$ may contain incomplete `\if_charcode:w` statements.

`\clist_if_in:cnTF`

```

\clist_if_in:coTF
2526 \def_new:Npn \clist_if_in:NnTF #1#2{
2527   \def:Npn \tmp:w ##1 ,#2, ##2##3\q_stop{
2528     \if_meaning:NN\q_no_value##2
2529     \exp_after:NN\use_arg_ii:nn
2530   \else:
2531     \exp_after:NN\use_arg_i:nn
2532   \fi:
2533 }
2534 \exp_after:NN \tmp:w
2535 \exp_after:NN , #1, #2, \q_no_value \q_stop
2536 }
2537 \def_new:Npn \clist_if_in:NoTF {\exp_args:NNo \clist_if_in:NnTF}
2538 \def_new:Npn \clist_if_in:coTF {\exp_args:Nco \clist_if_in:NnTF}
2539 \def_new:Npn \clist_if_in:cnTF {\exp_args:Nc \clist_if_in:NnTF}

```

14.6.1 Stack operations

We build stacks from comma-lists, but here we put the specific functions together.

`\clist_push:Nn` Since comma-lists can be used as stacks, we ought to have both ‘push’ and ‘pop’. In most
`\clist_push:No` cases they are nothing more then new names for old functions.
`\clist_push:cn`
`\clist_pop:NN` 2540 `\let_new:NN \clist_push:Nn \clist_put_left:Nn`
`\clist_pop:NN` 2541 `\let_new:NN \clist_push:No \clist_put_left:No`
`\clist_pop:cn` 2542 `\let_new:NN \clist_push:cn \clist_put_left:cn`
2543 `\def_new:Npn \clist_pop:NN {\clist_pop_aux:nnNN \tlp_set:Nn \tlp_set:Nn}`
2544 `\def_new:Npn \clist_pop:cn {\exp_args:Nc \clist_pop:NN}`

`\clist_gpush:Nn` I don’t agree with Denys that one needs only local stacks, actually I believe that one will
`\clist_gpush:No` probably need the functions here more often. In case of `\clist_gpop:NN` the value is
`\clist_gpush:cn` nevertheless returned locally.
`\clist_gpop:NN`
`\clist_gpop:cn` 2545 `\let_new:NN \clist_gpush:Nn \clist_gput_left:Nn`
2546 `\def_new:Npn \clist_gpush:No {\exp_args:NNo \clist_gpush:Nn}`
2547 `\def_new:Npn \clist_gpush:cn {\exp_args:Nc \clist_gpush:Nn}`
2548 `\def_new:Npn \clist_gpop:NN {\clist_pop_aux:nnNN \tlp_gset:Nn \tlp_set:Nn}`
2549 `\def_new:Npn \clist_gpop:cn {\exp_args:Nc \clist_gpop:NN}`

`\clist_top:NN` Looking at the top element of the stack without removing it is done with this operation.
`\clist_top:cn` 2550 `\let_new:NN \clist_top:NN \clist_get:NN`
2551 `\let_new:NN \clist_top:cn \clist_get:cn`

Show token usage:

```

2552 </initex | package>
2553 < *showmemory>
2554 %\showMemUsage
2555 </showmemory>

```

15 Property lists

\LaTeX 3 implements a data structure which allows to store information associated with individual tokens.

15.1 Functions

<code>\prop_new:N</code>	<code>\prop_new:N <plist></code>
<code>\prop_new:c</code>	

Defines `<plist>` to be a variable of type p-list.

<code>\prop_clear:N</code>	<code>\prop_clear:N <plist></code>
<code>\prop_gclear:N</code>	

These functions locally or globally clear `<plist>`.

<code>\prop_put:Nnn</code>
<code>\prop_put:ccn</code>
<code>\prop_gput:Nnn</code>
<code>\prop_gput:Nno</code>
<code>\prop_gput:Noo</code>
<code>\prop_gput:Ncn</code>
<code>\prop_gput:Ooo</code>
<code>\prop_gput:Nox</code>
<code>\prop_gput:cnn</code>
<code>\prop_gput:ccn</code>
<code>\prop_gput:cco</code>
<code>\prop_gput:ccx</code>

`\prop_put:Nnn <plist> {<key>} {<token list>}`

Locally or globally associates *<token list>* with *<key>* in the p-list *<plist>*. If *<key>* has already a meaning within *<plist>* this value is overwritten.

<code>\prop_gput_if_new:Nnn</code>

`\prop_gput_if_new:Nnn <plist> {<key>} {<token list>}`

Globally associates *<token list>* with *<key>* in the p-list *<plist>* but only if *<key>* has so far no meaning within *<plist>*. overwritten.

<code>\prop_get:NnN</code>
<code>\prop_get:cnN</code>
<code>\prop_gget:NnN</code>
<code>\prop_gget:NcN</code>
<code>\prop_gget:cnN</code>

`\prop_get:NnN <plist> {<key>} <tlp>`

If *<info>* is the information associated with *<key>* in the p-list *<plist>* then the token list pointer *<tlp>* gets *<info>* assigned. Otherwise its value is the special quark `\q_no_value`. The assignment is done either locally or globally.

<code>\prop_set_eq:NN</code>
<code>\prop_set_eq:cc</code>
<code>\prop_gset_eq:NN</code>
<code>\prop_gset_eq:cc</code>

`\prop_set_eq:NN <plist 1> <plist 2>`

A fast assignment of *<plist>*s.

<code>\prop_get_gdel:NnN</code>

`\prop_get_gdel:NnN <plist> {<key>} <tlp>`

Like `\prop_get:NnN` but additionally removes *<key>* (and its *<info>*) from *<plist>*.

<code>\prop_del:Nn</code>

<code>\prop_gdel:Nn</code>

`\prop_del:Nn <plist> {<key>}`

Locally or globally deletes *<key>* and its *<info>* from *<plist>* if found. Otherwise does nothing.

<code>\prop_map_function:NN</code>
<code>\prop_map_function:cN</code>
<code>\prop_map_function:Nc</code>
<code>\prop_map_function:cc</code>

`\prop_map_function:NN <plist> <function>`

Maps $\langle function \rangle$ which should be a function with two arguments ($\langle key \rangle$ and $\langle info \rangle$) over every $\langle key \rangle$ $\langle info \rangle$ pair of $\langle plist \rangle$. Expandable.

<code>\prop_map_inline:Nn</code>	<code>\prop_map_inline:Nn $\langle plist \rangle$ { $\langle inline function \rangle$ }</code>
<code>\prop_map_inline:cn</code>	

Just like `\prop_map_function:NN` but with the function of two arguments supplied as inline code. Within $\langle inline function \rangle$ refer to the arguments via #1 ($\langle key \rangle$) and #2 ($\langle info \rangle$). Nestable.

<code>\prop_map_break:w</code>	<code>\prop_map_break:w</code>
--------------------------------	--------------------------------

For breaking out of a loop. To be used inside TF type functions.

15.2 Predicates and conditionals

<code>\prop_if_empty:NTF</code>	<code>\prop_if_empty:NTF $\langle plist \rangle$ {$\langle true code \rangle$}{$\langle false code \rangle$}</code>
---------------------------------	--

Set of conditionals that test whether or not a particular $\langle plist \rangle$ is empty.

<code>\prop_if_eq:NNF</code>	<code>\prop_if_eq:NNF $\langle plist1 \rangle$ $\langle plist2 \rangle$ {$\langle false code \rangle$}</code>
<code>\prop_if_eq:ccF</code>	

Execute $\langle false code \rangle$ if $\langle plist1 \rangle$ doesn't hold the same token list as $\langle plist2 \rangle$.

<code>\prop_if_in:NnTF</code>	<code>\prop_if_in:NnTF $\langle plist \rangle${$\langle key \rangle$}{$\langle true code \rangle$}{$\langle false code \rangle$}</code>
<code>\prop_if_in:NoTF</code>	
<code>\prop_if_in:ccTF</code>	

Tests if $\langle key \rangle$ is used in $\langle plist \rangle$ and then either executes $\langle true code \rangle$ or $\langle false code \rangle$.

15.3 Internal functions

<code>\prop_put_aux:w</code>	<code>\prop_put_if_new_aux:w</code>	Internal functions implementing the put operations.

<code>\prop_get_aux:w</code>	<code>\prop_get_del_aux:w</code>	<code>\prop_del_aux:w</code>	Internal functions implementing the get and delete operations.
<code>\prop_get_del_aux:w</code>			
<code>\prop_del_aux:w</code>			

<code>\prop_if_in_aux:w</code>	Internal function implementing the key test operation.
--------------------------------	--

<code>\prop_map_function_aux:NNn</code>	<code>\prop_map_inline_aux:Nn</code>	Internal functions implementing the map operations.

<code>\g_prop_inline_level_num</code>	Fake integer used in internal name for function used inside <code>\prop_map_inline:NN</code> .
---------------------------------------	--

`\prop_split_aux:Nnn` `\prop_split_aux:Nnn` $\langle plist \rangle$ $\langle key \rangle$ $\langle cmd \rangle$

Internal function that invokes $\langle cmd \rangle$ with 3 arguments: 1st is the beginning of $\langle plist \rangle$ before $\langle key \rangle$, 2nd is the value associated with $\langle key \rangle$, 3rd is the rest of $\langle plist \rangle$ after $\langle key \rangle$. If there is no key $\langle key \rangle$ in $\langle plist \rangle$, then the 2 arg is `\q_no_value` and the 3rd arg is empty; otherwise the 3rd argument has the two extra tokens $\langle prop \rangle$ `\q_no_value` at the end.

This function is used to implement various get operations.

15.4 The Implementation

We start by ensuring that the required packages are loaded.

```
2556 <package>\ProvidesExplPackage
2557 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
2558 <package>\RequirePackage{l3toks}\par
2559 <package>\RequirePackage{l3quark}\par
2560 *initex | package
```

A property list is a token register whose contents is of the form ‘ $\langle q \text{ } prop \rangle key_1 \langle q \text{ } prop \rangle \langle val_1 \rangle \dots \langle q \text{ } prop \rangle key_n \langle q \text{ } prop \rangle \langle val_n \rangle$ ’. The properties have to be single token, the values might be arbitrary token lists they get surrounded by braces.

`\q_prop`

```
2561 \quark_new:N\q_prop
```

To get values from property-lists, token lists should be passed to the appropriate functions.

`\prop_new:N` Property lists are implemented as token lists.

```
\prop_new:c
2562 \def_new:Npn \prop_new:N #1{\toks_new:N #1}
2563 \def_new:Npn \prop_new:c {\exp_args:Nc \prop_new:N}
```

`\prop_clear:N` The same goes for clearing a property list, either locally or globally.

```
\prop_clear:c
2564 \let_new:NN \prop_clear:N \toks_clear:N
\prop_gclear:N
2565 \def_new:Npn \prop_clear:c {\exp_args:Nc \prop_clear:N}
\prop_gclear:c
2566 \let_new:NN \prop_gclear:N \toks_gclear:N
2567 \def_new:Npn \prop_gclear:c {\exp_args:Nc \prop_gclear:N}
```

`\prop_split_aux:Nnn` `\prop_split_aux:Nnn` $\langle plist \rangle$ $\langle prop \rangle$ $\langle cmd \rangle$ invokes $\langle cmd \rangle$ with 3 arguments: 1st is the beginning of $\langle plist \rangle$ before $\langle prop \rangle$, 2nd is the value associated with $\langle prop \rangle$, 3rd is the rest of $\langle plist \rangle$ after $\langle prop \rangle$. If there is no property $\langle prop \rangle$ in $\langle plist \rangle$, then the 2nd argument will be `\q_no_value` and the 3rd argument is empty; otherwise the 3rd argument has the extra tokens `\q_prop` $\langle prop \rangle$ `\q_prop` `\q_no_value` at the end.

```
2568 \def_long_new:Npn \prop_split_aux:Nnn #1#2#3{
2569   \def:Npn \tmp:w ##1\q_prop#2\q_prop##2##3\q_stop {#3{##1}{##2}{##3}}
2570   \exp_after:NN\tmp:w \toks_use:N#1\q_prop#2\q_prop\q_no_value \q_stop
2571 }
```

```

\prop_get:NnN \prop_get:NNN <plist><prop><tlp> defines <tlp> to be the value associated with <prop> in
\prop_get:cnN <plist>, \q_no_value if not found.
\prop_get_aux:w
2572 \def_long_new:NNn \prop_get:NnN 2{
2573   \prop_split_aux:Nnn #1{#2}\prop_get_aux:w}
2574 \def_long_new:NNn \prop_get_aux:w 4{\tlp_set:Nx#4{\exp_not:n{#2}}}
2575 \def_new:Npn \prop_get:cnN { \exp_args:Nc \prop_get:NnN }

\prop_gget:NnN The global version of the previous function.
\prop_gget:NcN
\prop_gget:cnN 2576 \def_long_new:NNn \prop_gget:NnN 2{
\prop_gget_aux:w 2577   \prop_split_aux:Nnn #1{#2}\prop_gget_aux:w}
2578 \def_new:Npn \prop_gget:NcN {\exp_args:NNc \prop_gget:NnN}
2579 \def_new:Npn \prop_gget:cnN {\exp_args:Nc \prop_gget:NnN}
2580 \def_long_new:NNn \prop_gget_aux:w 4{\tlp_gset:Nx#4{\exp_not:n{#2}}}

\prop_get_gdel:NNN \prop_get_gdel:NNN is the same as \prop_get:NNN but the <property key> and its value
\prop_get_del_aux:w are afterwards globally removed from <property list>. One probably also needs the local
variants or only the local one, or... We decide this later.

2581 \def_long_new:NNn \prop_get_gdel:NnN 3{
2582   \prop_split_aux:Nnn #1{#2}{\prop_get_del_aux:w #3{\toks_gset:Nn #1}{#2}}}
2583 \def_long_new:NNn \prop_get_del_aux:w 6{
2584   \tlp_set:Nx #1{\exp_not:n{#5}}
2585   \quark_if_no_value:NF #1 {
2586     \def:Npn \tmp:w ##1\q_prop#3\q_prop\q_no_value {#2{#4##1}}
2587     \tmp:w #6}
2588 }

\prop_put:Nnn \prop_put:Nnn <plist><prop><val> adds/changes the value associated with <prop> in
\prop_put:ccn <plist> to <val>.
\prop_gput:Nnn
\prop_gput:Nno 2589 \def_long_new:NNn \prop_put:Nnn 2{
\prop_gput:Nnx 2590   \prop_split_aux:Nnn #1{#2}{
\prop_gput:Nox 2591     \prop_clear:N #1
\prop_gput:Noo 2592     \prop_put_aux:w {\toks_put_right:Nn #1}{#2}}
2593 }
\prop_gput:Ncn 2594 \def_new:Npn \prop_put:ccn {\exp_args:Ncc \prop_put:Nnn }
\prop_gput:Noo 2595
\prop_gput:cNn 2596 \def_long_new:NNn \prop_gput:Nnn 2{
\prop_gput:ccn 2597   \prop_split_aux:Nnn #1{#2}{
\prop_gput:cco 2598     \prop_gclear:N #1
\prop_gput:ccx 2599     \prop_put_aux:w {\toks_gput_right:Nn #1}{#2}}
2600 }
\prop_put_aux:w
2601
2602 \def_long_new:NNn \prop_put_aux:w 6{
2603   #1{\q_prop#2\q_prop{#6}#3}
2604   \tlist_if_empty:nF{#5}
2605   {
2606     \def:Npn \tmp:w ##1\q_prop#2\q_prop\q_no_value {#1{##1}}
2607     \tmp:w #5
2608   }
2609 }
2610 \def_new:Npn \prop_gput:Nno {\exp_args:NNno \prop_gput:Nnn}

```

```

2611 \def_new:Npn \prop_gput:Nnx {\exp_args:NNnx \prop_gput:Nnn}
2612 \def_new:Npn \prop_gput:Nox {\exp_args:NNox \prop_gput:Nnn}
2613 \def_new:Npn \prop_gput:Noo {\exp_args:NNoo \prop_gput:Nnn}
2614 \def_new:Npn \prop_gput:Ncn {\exp_args:NNc \prop_gput:Nnn}
2615 \def_new:Npn \prop_gput:Ooo {\exp_args:NOoo \prop_gput:Nnn}
2616 \def_new:Npn \prop_gput:cnn {\exp_args:Nc \prop_gput:Nnn}
2617 \def_new:Npn \prop_gput:ccn {\exp_args:Ncc \prop_gput:Nnn}
2618 \def_new:Npn \prop_gput:cco {\exp_args:Ncco \prop_gput:Nnn}
2619 \def_new:Npn \prop_gput:ccx {\exp_args:Nccx \prop_gput:Nnn}

```

`\prop_del:Nn \prop_del:NN` $\langle plist \rangle \langle prop \rangle$ deletes the entry for $\langle prop \rangle$ in $\langle plist \rangle$, if any.

```

\prop_gdel:Nn
\prop_del_aux:w 2620 \def_long_new:NNn \prop_del:Nn 2{
2621   \prop_split_aux:Nnn #1{#2}{\prop_del_aux:w {\toks_set:Nn #1}{#2}}}
2622 \def_long_new:NNn \prop_gdel:Nn 2{
2623   \prop_split_aux:Nnn #1{#2}{\prop_del_aux:w {\toks_gset:Nn #1}{#2}}}
2624 \def_long_new:NNn \prop_del_aux:w 5{
2625   \def:Npn \tmp:w {#4}
2626   \quark_if_no_value:NF \tmp:w
2627   {\def:Npn \tmp:w ##1\q_prop#2\q_prop\q_no_value {#1{#3##1}}}
2628   \tmp:w #5}}

```

`\prop_if_in:NnTF \prop_if_in:NNTF` $\langle property \text{'list'} \rangle \langle property \text{'key'} \rangle \langle true \text{'case'} \rangle \langle false \text{'case'} \rangle$ will check whether $\langle property \text{'key'} \rangle$ is on the $\langle property \text{'list'} \rangle$ and then select either the true or false case.

```

\prop_if_in:ccTF
\prop_if_in_aux:w 2629 \def_new:NNn \prop_if_in:NnTF 2{
2630   \prop_split_aux:Nnn #1{#2}\prop_if_in_aux:w}
2631 \def_new:NNn \prop_if_in_aux:w 3{\quark_if_no_value:nFT {#2}}
2632
2633 \def_new:Npn \prop_if_in:NoTF {\exp_args:NNo \prop_if_in:NnTF}
2634 \def_new:Npn \prop_if_in:ccTF {\exp_args:Ncc \prop_if_in:NnTF}

```

`\prop_gput_if_new:Nnn \prop_gput_if_new:NNn` $\langle property \text{'list'} \rangle \langle property \text{'key'} \rangle \langle property \text{'value'} \rangle$ is equivalent to

```

\prop_if_in:NNTF \langle property \rangle \langle property \text{'key'} \rangle
  {}%
  {\prop_gput:Nnn
    \langle property \text{'list'} \rangle
    \langle property \text{'key'} \rangle
    \langle property \text{'value'} \rangle}

```

Here we go (listening to Porgy & Bess in a recording with Ella F. and Louis A. which makes writing macros sometimes difficult; I find myself humming instead of working):

```

2635 \def_long_new:NNn \prop_gput_if_new:Nnn 2{
2636   \prop_split_aux:Nnn #1{#2}{\prop_put_if_new_aux:w #1{#2}}}
2637 \def_long_new:NNn \prop_put_if_new_aux:w 6{
2638   \tlist_if_empty:nT {#5}{#1{\q_prop#2\q_prop{#6}{#3}}}

```

`\prop_set_eq:NN` This makes two $\langle plist \rangle$ s have the same contents.

```

\prop_set_eq:Nc
\prop_set_eq:cN 2639 \let_new:NN \prop_set_eq:NN \toks_set_eq:NN
\prop_set_eq:cc
\prop_gset_eq:NN
\prop_gset_eq:Nc
\prop_gset_eq:cN
\prop_gset_eq:cc

```

```

2640 \let_new:NN \prop_set_eq:Nc \toks_set_eq:Nc
2641 \let_new:NN \prop_set_eq:cN \toks_set_eq:cN
2642 \let_new:NN \prop_set_eq:cc \toks_set_eq:cc
2643 \let_new:NN \prop_gset_eq:NN \toks_gset_eq:NN
2644 \let_new:NN \prop_gset_eq:Nc \toks_gset_eq:Nc
2645 \let_new:NN \prop_gset_eq:cN \toks_gset_eq:cN
2646 \let_new:NN \prop_gset_eq:cc \toks_gset_eq:cc

```

\prop_if_empty_p:N This conditional takes a *<plist>* as its argument and evaluates either the true or the false case, depending on whether or not *<plist>* contains any properties.

```

\prop_if_empty_p:c
\prop_if_empty:NNTF
\prop_if_empty:NT 2647 \let_new:NN \prop_if_empty_p:N \toks_if_empty_p:N
\prop_if_empty:NF 2648 \let_new:NN \prop_if_empty_p:c \toks_if_empty_p:c
2649 \let_new:NN \prop_if_empty:NNTF \toks_if_empty:NNTF
\prop_if_empty:cTF 2650 \let_new:NN \prop_if_empty:NT \toks_if_empty:NT
\prop_if_empty:cT 2651 \let_new:NN \prop_if_empty:NF \toks_if_empty:NF
\prop_if_empty:cF 2652 \let_new:NN \prop_if_empty:cTF \toks_if_empty:cTF
2653 \let_new:NN \prop_if_empty:cT \toks_if_empty:cTF
2654 \let_new:NN \prop_if_empty:cF \toks_if_empty:cF

```

\prop_if_eq:NNTF This function test whether the property lists that are in its first two arguments are equal; if they are not #3 is executed.

```

\prop_if_eq:NNTF
\prop_if_eq:NNT 2655 \def_new:NNn \prop_if_eq:NNTF 2 {
\prop_if_eq:NcTF 2656 \tlist_if_eq:xxTF{\toks_use:N #1}{\toks_use:N #2}
\prop_if_eq:NcT 2657 }
\prop_if_eq:NcF 2658 \def_new:NNn \prop_if_eq:NNT 2 {
\prop_if_eq:cNTF 2659 \tlist_if_eq:xxT{\toks_use:N #1}{\toks_use:N #2}
\prop_if_eq:cNT 2660 }
\prop_if_eq:cNF 2661 \def_new:NNn \prop_if_eq:NNTF 2 {
\prop_if_eq:ccTF 2662 \tlist_if_eq:xxF{\toks_use:N #1}{\toks_use:N #2}
\prop_if_eq:ccT 2663 }
\prop_if_eq:ccF 2664 \def_new:Npn \prop_if_eq:NcTF {\exp_args:Nnc \prop_if_eq:NNTF}
2665 \def_new:Npn \prop_if_eq:NcT {\exp_args:Nnc \prop_if_eq:NNT}
2666 \def_new:Npn \prop_if_eq:NcF {\exp_args:Nnc \prop_if_eq:NNTF}
2667 \def_new:Npn \prop_if_eq:cNTF {\exp_args:Nc \prop_if_eq:NNTF}
2668 \def_new:Npn \prop_if_eq:cNT {\exp_args:Nc \prop_if_eq:NNT}
2669 \def_new:Npn \prop_if_eq:cNF {\exp_args:Nc \prop_if_eq:NNTF}
2670 \def_new:Npn \prop_if_eq:ccTF {\exp_args:Ncc \prop_if_eq:NNTF}
2671 \def_new:Npn \prop_if_eq:ccT {\exp_args:Ncc \prop_if_eq:NNT}
2672 \def_new:Npn \prop_if_eq:ccF {\exp_args:Ncc \prop_if_eq:NNTF}

```

\prop_map_function:NN Maps a function on every entry in the property list. The function must take 2 arguments: a key and a value.

```

\prop_map_function:cN
\prop_map_function:Nc
\prop_map_function:cc 2673 \def_new:Npn \prop_map_function:NN #1#2{
\prop_map_function_aux:w 2674 \exp_after:NN \prop_map_function_aux:w
2675 \exp_after:NN #2 \toks_use:N #1 \q_prop{\q_prop \q_no_value \q_stop
2676 }
2677 \def_new:Npn \prop_map_function_aux:w #1\q_prop#2\q_prop#3{
2678 \if:w \tlist_if_empty_p:n{#2}
2679 \exp_after:NN \prop_map_break:w
2680 \fi:

```

```

2681  #1{#2}{#3}
2682  \prop_map_function_aux:w #1
2683 }
2684 % problem with the above impl, is that an empty key stops the mapping but all
2685 % other functions in the module allow the use of empty keys (as one value)
2686
2687 \def:Npn \prop_map_function:NN #1#2{
2688   \exp_after:NN \prop_map_function_aux:w
2689   \exp_after:NN #2 \toks_use:N #1 \q_prop \q_no_value \q_prop \q_no_value
2690 }
2691 \def:Npn \prop_map_function_aux:w #1\q_prop#2\q_prop#3{
2692   \quark_if_no_value:nF{#2}
2693   {
2694     #1{#2}{#3}
2695     \prop_map_function_aux:w #1
2696   }
2697 }
2698 % problem with the above impl is that \quark_if_no_value:nF is fairly slow and
2699 % if \quark_if_no_value:NF is used instead we have to do an assignment thus
2700 % making the mapping not expandable (is that important?)
2701
2702 \def:Npn \prop_map_function:NN #1#2{
2703   \exp_after:NN \prop_map_function_aux:w
2704   \exp_after:NN #2 \toks_use:N #1 \q_prop \q_nil \q_prop \q_no_value \q_stop
2705 }
2706 \def:Npn \prop_map_function_aux:w #1\q_prop#2\q_prop#3{
2707   \if_meaning:NN \q_nil #2
2708   \exp_after:NN \prop_map_break:w
2709   \fi:
2710   #1{#2}{#3}
2711   \prop_map_function_aux:w #1
2712 }
2713
2714 % (potential) problem with the above impl is that it will returns true is #2
2715 % contains more than just \q_nil thus executing whatever follows. Claim: this
2716 % can't happen :- ) so we should be ok
2717
2718 \def_new:Npn \prop_map_function:cN {\exp_args:Nc \prop_map_function:NN }
2719 \def_new:Npn \prop_map_function:Nc {\exp_args:NNc \prop_map_function:NN }
2720 \def_new:Npn \prop_map_function:cc {\exp_args:Ncc \prop_map_function:NN }

```

\prop_map_inline:Nn The inline functions are straight forward. It takes longer to test if the list is empty than
\prop_map_inline:cn to run it on an empty list so we don't waste time doing that.

```

\g_prop_inline_level_num
2721 \num_new:N \g_prop_inline_level_num
2722 \def_new:Npn \prop_map_inline:Nn #1#2 {
2723   \num_gincr:N \g_prop_inline_level_num
2724   \gdef_long:cpn {prop_map_inline_ \num_use:N \g_prop_inline_level_num :n}
2725   ##1##2{#2}
2726   \prop_map_function:Nc #1
2727   {prop_map_inline_ \num_use:N \g_prop_inline_level_num :n}
2728   \num_gdecr:N \g_prop_inline_level_num
2729 }
2730 \def_new:Npn \prop_map_inline:cN { \exp_args:Nc \prop_map_inline:NN }

```

`\prop_map_break:w` The break statement.

2731 `\let_new:NN \prop_map_break:w \use_none_delimit_by_q_stop:w`

Finally a bunch of compatibility commands with the old syntax: they will vanish soon!

```
2732 \def:Npn \prop_put:NNn {\typeout{Warning:~name~
2733         changed~ to~ \string\prop_put:Nnn}\prop_put:Nnn}
2734 \def:Npn \prop_gput:NNn {\typeout{Warning:~name~
2735         changed~ to~ \string\prop_gput:Nnn }\prop_gput:Nnn }
2736 \def:Npn \prop_gput:NNo {\typeout{Warning:~name~
2737         changed~ to~ \string\prop_gput:Nno }\prop_gput:Nno }
2738 \def:Npn \prop_gput:cNn {\typeout{Warning:~name~
2739         changed~ to~ \string\prop_gput:cnn }\prop_gput:cnn }
2740 \def:Npn \prop_gput_if_new:NNn {\typeout{Warning:~name~
2741         changed~ to~ \string\prop_gput_if_new:Nnn }\prop_gput_if_new:Nnn }
2742 \def:Npn \prop_get:NNN {\typeout{Warning:~name~
2743         changed~ to~ \string\prop_get:NnN }\prop_get:NnN }
2744 \def:Npn \prop_get:cNN {\typeout{Warning:~name~
2745         changed~ to~ \string\prop_get:cnN }\prop_get:cnN }
2746 \def:Npn \prop_gget:NNN {\typeout{Warning:~name~
2747         changed~ to~ \string\prop_gget:NnN }\prop_gget:NnN }
2748 \def:Npn \prop_gget:cNN {\typeout{Warning:~name~
2749         changed~ to~ \string\prop_gget:cnN }\prop_gget:cnN }
2750 \def:Npn \prop_get_gdel:NNN {\typeout{Warning:~name~
2751         changed~ to~ \string\prop_get_gdel:NnN }\prop_get_gdel:NnN }
2752 \def:Npn \prop_del:NN {\typeout{Warning:~name~
2753         changed~ to~ \string\prop_del:Nn }\prop_del:Nn }
2754 \def:Npn \prop_gdel:NN {\typeout{Warning:~name~
2755         changed~ to~ \string\prop_gdel:Nn }\prop_gdel:Nn }
2756 \def:Npn \prop_if_in:NNTF {\typeout{Warning:~name~
2757         changed~ to~ \string\prop_if_in:NnTF }\prop_if_in:NnTF }
```

Show token usage:

```
2758 </initex | package>
2759 <*showmemory>
2760 %\showMemUsage
2761 </showmemory>

2762 <*unused>
2763 \file_input_stop:

2764 </unused>
```

16 Integers

L^AT_EX3 maintains two type of integer registers for internal use. One (associated with the name `num`) for low level uses in the allocation mechanism using macros only and `int`: the one described here.

The `int` type uses the built-in counter registers of T_EX and is therefore relatively fast compared to the `num` type and should be preferred in all cases as there is little chance we should ever run out of registers when being based on at least ε -T_EX.

16.1 Functions

<code>\int_new:N</code> <code>\int_new:c</code> <code>\int_new_l:N</code>	<code>\int_new:N <int></code>
---	--

Globally defines $\langle int \rangle$ to be a new variable of type `int` although you can still choose if it should be a `\l_` or `\g_` type. There is no way to define constant counters with these functions. The function `\int_new_l:N` defines $\langle int \rangle$ locally only.

T_EXhackers note: `\int_new:N` is the equivalent to plain T_EX's `\newcount`. However, the internal register allocation is done differently.

<code>\int_incr:N</code> <code>\int_incr:c</code> <code>\int_gincr:N</code> <code>\int_gincr:c</code>	<code>\int_incr:N <int></code>
--	---

Increments $\langle int \rangle$ by one. For global variables the global versions should be used.

<code>\int_decr:N</code> <code>\int_decr:c</code> <code>\int_gdecr:N</code> <code>\int_gdecr:c</code>	<code>\int_decr:N <int></code>
--	---

Decrements $\langle int \rangle$ by one. For global variables the global versions should be used.

<code>\int_set:Nn</code> <code>\int_set:cn</code> <code>\int_gset:Nn</code> <code>\int_gset:cn</code>	<code>\int_set:Nn <int> { <integer expr> }</code>
--	--

These functions will set the $\langle int \rangle$ register to the $\langle integer\ expr \rangle$ value. This value can contain simple calc-like expressions as provided by ε -T_EX.

<code>\int_zero:N</code> <code>\int_zero:c</code> <code>\int_gzero:N</code> <code>\int_gzero:c</code>	<code>\int_zero:N <int></code>
--	---

These functions sets the $\langle int \rangle$ register to zero either locally or globally.

<code>\int_add:Nn</code> <code>\int_add:cn</code> <code>\int_gadd:Nn</code> <code>\int_gadd:cn</code>	<code>\int_add:Nn <int> { <integer expr> }</code>
--	--

These functions will add to the $\langle int \rangle$ register the value $\langle integer\ expr \rangle$. If the second argument is a $\langle int \rangle$ register too, the surrounding braces can be left out.

<code>\int_sub:Nn</code>
<code>\int_sub:cn</code>
<code>\int_gsub:Nn</code>
<code>\int_gsub:cn</code>

`\int_gsub:Nn <int> { <integer expr> }`

These functions will subtract from the $\langle int \rangle$ register the value $\langle integer\ expr \rangle$. If the second argument is a $\langle int \rangle$ register too, the surrounding braces can be left out.

<code>\int_use:N</code>
<code>\int_use:c</code>

`\int_use:N <int>`

This function returns the integer value kept in $\langle int \rangle$ in a way suitable for further processing.

T_EXhackers note: The function `\int_use:N` could be implemented directly as the T_EX primitive `\tex_the:D` which is also responsible to produce the values for other internal quantities. We have chosen to use individual functions for counters, dimensions etc. to allow checks and to make the code more self-explaining.

16.2 Formatting a counter value

<code>\int_to_arabic:n</code>
<code>\int_to_alph:n</code>
<code>\int_to_Alph:n</code>
<code>\int_to_roman:n</code>
<code>\int_to_Roman:n</code>
<code>\int_to_symbol:n</code>

`\int_to_alph:n { <integer> }`
`\int_to_alph:n <int>`

If some $\langle integer \rangle$ or the the current value of a $\langle int \rangle$ should be displayed or typeset in a special ways (e.g., as uppercase roman numerals) these function can be used. We need braces if the argument is a simple $\langle integer \rangle$, they can be omitted in case of a $\langle int \rangle$. By default the letters produced by `\int_to_roman:n` and `\int_to_Roman:n` have catcode 11.

All functions are fully expandable and will therefore produce the correct output when used inside of deferred writes, etc. In case the number in an `alph` or `Alph` function is greater than the default base number (26) it follows a simple conversion rule so that 27 is turned into `aa`, 50 into `ax` and so on and so forth. These two functions can be modified quite easily to take a different base number and conversion rule so that other languages can be supported.

T_EXhackers note: These are more or less the internal L^AT_EX2 functions `\@arabic`, `\@alph`, `\@Alph`, `\@roman`, `\@Roman`, and `\@fnsymbol` except that `\int_to_symbol:n` is also allowed outside math mode.

16.2.1 Internal functions

<code>\int_to_roman:w</code>

`\int_to_roman:w <integer> <space> or <non-expandable token>`

Converts $\langle integer \rangle$ to it lowercase roman representation. Note that it produces a string of letters with catcode 12.

T_EXhackers note: This is the T_EX primitive `\romannumeral` renamed.

<code>\int_roman_lcuc_mapping:Nnn</code>	<code>\int_roman_lcuc_mapping:Nnn <roman_char> {<licr>}</code>
<code>\int_to_roman_lcuc:NN</code>	<code>{<LICR>}</code> <code>\int_to_roman_lcuc:NN <roman_char> <char></code>

`\int_roman_lcuc_mapping:Nnn` specifies how the roman numeral `<roman_char>` (i, v, x, l, c, d, or m) should be interpreted when converting the number. `<licr>` is the lower case and `<LICR>` is the uppercase mapping. `\int_to_roman_lcuc:NN` is a recursive function converting the roman numerals.

<code>\int_convert_number_with_rule:nnN</code>	
<code>\int_alph_default_conversion_rule:n</code>	
<code>\int_Alph_default_conversion_rule:n</code>	
<code>\int_symbol_math_conversion_rule:n</code>	<code>\int_convert_number_with_rule:nnN {<int1>} {<int2>}</code>
<code>\int_symbol_text_conversion_rule:n</code>	<code><function></code> <code>\int_alph_default_conversion_rule:n {<int>}</code>

`\int_convert_number_with_rule:nnN` converts `<int1>` into letters, symbols, whatever as defined by `<function>`. `<int2>` denotes the base number for the conversion.

16.3 Variable and constants

<code>\c_max_int</code>	Constant that denote the maximum value which can be stored in an <code><int></code> register.
-------------------------	---

<code>\l_tmpa_int</code>	
<code>\l_tmpb_int</code>	
<code>\l_tmpc_int</code>	
<code>\g_tmpa_int</code>	
<code>\g_tmpb_int</code>	Scratch register for immediate use. They are not used by conditionals or predicate functions.

16.4 Testing and evaluating integer expressions

<code>\int_eval:n</code>	<code>\int_eval:n {<int expr>}</code>
<code>\int_div_truncate:nn</code>	<code>\int_div_truncate:n {<int expr>} {<int expr>}</code>
<code>\int_div_round:nn</code>	<code>\int_div_truncate:nn {<int expr>} {<int expr>}</code>
<code>\int_mod:nn</code>	<code>\int_mod:nn {<int expr>} {<int expr>}</code>

Evaluates the value of a integer expression so that `\int_eval:n {3*5/4}` puts 4 back into the input stream. Note that the results of divisions are rounded by the primitive operations. If you want the result of a division to be truncated use `\int_div_truncate:nn`. `\int_div_round:nn` is added for completeness. `\int_mod:nn` returns the remainder of a division. All of these functions are expandable.

T_EXhackers note: `\int_eval:n` is the ϵ -T_EX primitive `\numexpr` turned into a function taking an argument.

<code>\int_compare:nNnTF</code>	<code>\int_compare:nNnTF {\langle int expr \rangle} \langle rel \rangle {\langle int expr \rangle}</code>
<code>\int_compare:nNnT</code>	
<code>\int_compare:nNnF</code>	

These functions test two integer expressions against each other. They are both evaluated by `\int_eval:n`. Note that if both expressions are normal integer variables as in

`\int_compare:nNnTF \l_temp_int < \c_zero {negative}{non-negative}`

you can safely omit the braces.

T_EXhackers note: This is the T_EX primitive `\ifnum` turned into a function.

<code>\int_compare_p:nNn</code>	<code>\int_compare_p:nNn {\langle int expr \rangle} \langle rel \rangle {\langle int expr \rangle}</code>
---------------------------------	--

A predicate version of the above mentioned functions.

<code>\int_max_of:nn</code>	<code>\int_max_of:nn {\langle int expr \rangle} {\langle int expr \rangle}</code>
<code>\int_min_of:nn</code>	

Return the largest or smallest of two integer expressions.

<code>\int_abs:n</code>	<code>\int_abs:n {\langle int expr \rangle}</code>
-------------------------	---

Return the numerical value of an integer expression.

<code>\int_if_odd:nTF</code>	<code>\int_if_odd:nTF {\langle int expr \rangle} {\langle true \rangle} {\langle false \rangle}</code>
<code>\int_if_odd_p:n</code>	

These functions test if an integer expression is even or odd. We also define a predicate version of it.

T_EXhackers note: This is the T_EX primitive `\ifodd` turned into a function.

<code>\int_whiledo:nNnT</code>	<code>\int_whiledo:nNnT \langle int expr \rangle \langle rel \rangle \langle int expr \rangle {\langle true \rangle}</code>
<code>\int_whiledo:nNnF</code>	
<code>\int_dowhile:nNnT</code>	
<code>\int_dowhile:nNnF</code>	

`\int_whiledo:nNnT` tests the integer expressions and if true performs the body T until the test fails. `\int_dowhile:nNnT` is similar but executes the body first and then performs the check, thus ensuring that the body is executed at least once. The F versions are similar but continue the loop as long as the test is false. They could be omitted as it is just a matter of switching the arguments in the test.

16.5 Conversion

`\int_convert_from_base_ten:nn` `\int_convert_from_base_ten:nn {<number>}{<base>}`

Converts the base 10 number `<number>` into its equivalent representation written in base `<base>`. Expandable.

`\int_convert_to_base_ten:nn` `\int_convert_to_base_ten:nn {<number>}{<base>}`

Converts the base `<base>` number `<number>` into its equivalent representation written in base 10. `<number>` can consist of digits and ascii letters. Expandable.

16.6 The Implementation

We start by ensuring that the required packages are loaded.

```
2765 <package>\ProvidesExplPackage
2766 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
2767 <package&!check>\RequirePackage{l3num}
2768 <package & check>\RequirePackage{l3chk}
2769 <*initex | package>
```

```
\int_to_roman:w A new name for the primitives.
\int_to_number:w
\int_advance:w 2770 \let_new:NN \int_to_roman:w \tex_romannumeral:D
2771 \let_new:NN \int_to_number:w \tex_number:D
2772 \let_new:NN \int_advance:w \tex_advance:D
```

Functions that support L^AT_EX's user accessible counters should be added here, too. But first the internal counters.

```
\int_incr:N Incrementing and decrementing of integer registers is done with the following functions.
\int_decr:N
\int_gincr:N 2773 \def_new:Npn \int_incr:N #1{\int_advance:w#1\c_one
\int_gdecr:N 2774 <*check>
2775 \chk_local_or_pref_global:N #1
\int_incr:c 2776 </check>
\int_decr:c 2777 }
\int_gincr:c 2778 \def_new:Npn \int_decr:N #1{\int_advance:w#1\c_minus_one
\int_gdecr:c 2779 <*check>
2780 \chk_local_or_pref_global:N #1
2781 </check>
2782 }
2783 \def_new:Npn \int_gincr:N {
```

We make sure that a local variable is not updated globally by changing the internal test (i.e. `\chk_local_or_pref_global:N`) before making the assignment. This is done by `\pref_global_chk:` which also issues the necessary `\pref_global:D`. This is

not very efficient, but this code will be only included for debugging purposes. Using `\pref_global:D` in front of the local function is better in the production versions.

```

2784 <*check>
2785   \pref_global_chk:
2786 </check>
2787 <-check> \pref_global:D
2788   \int_incr:N}
2789 \def_new:Npn \int_gdecr:N {
2790 <*check>
2791   \pref_global_chk:
2792 </check>
2793 <-check> \pref_global:D
2794   \int_decr:N}

```

With the `\int_add:Nn` functions we can shorten the above code. If this makes it too slow ...

```

2795 \def:Npn \int_incr:N #1{\int_add:Nn#1\c_one}
2796 \def:Npn \int_decr:N #1{\int_add:Nn#1\c_minus_one}
2797 \def:Npn \int_gincr:N #1{\int_gadd:Nn#1\c_one}
2798 \def:Npn \int_gdecr:N #1{\int_gadd:Nn#1\c_minus_one}
2799 \def:Npn \int_incr:c {\exp_args:Nc\int_incr:N}
2800 \def:Npn \int_decr:c {\exp_args:Nc\int_decr:N}
2801 \def:Npn \int_gincr:c {\exp_args:Nc\int_gincr:N}
2802 \def:Npn \int_gdecr:c {\exp_args:Nc\int_gdecr:N}

```

`\int_new:N` Allocation of a new internal counter is already done above. Here we define the next likely variant.

```

\int_new:c
2803 <*initex>
2804 \alloc_setup_type:nnn {int} \c_eleven \c_max_register_num
2805 \def_new:Npn \int_new:N #1 {\alloc_reg:NnNN g {int} \tex_countdef:D#1}
2806 \def_new:Npn \int_new_l:N #1 {\alloc_reg:NnNN l {int} \tex_countdef:D#1}
2807 </initex>
2808 <package>\let:NN \int_new:N \newcount% allocation better nick the LaTeX one...
2809 \def_new:Npn \int_new:c {\exp_args:Nc \int_new:N}

```

`\int_set:Nn` Setting counters is again something that I would like to make uniform at the moment to
`\int_set:cn` get a better overview.

```

\int_gset:Nn
\int_gset:cn
2810 \def_new:Npn \int_set:Nn #1#2{#1 \int_eval:w #2\int_eval_end:
2811 <*check>
2812 \chk_local_or_pref_global:N #1
2813 </check>
2814 }
2815 \def_new:Npn \int_gset:Nn {
2816 <*check>
2817   \pref_global_chk:
2818 </check>
2819 <-check> \pref_global:D
2820   \int_set:Nn }
2821 \def_new:Npn \int_set:cn {\exp_args:Nc \int_set:Nn }
2822 \def_new:Npn \int_gset:cn {\exp_args:Nc \int_gset:Nn }

```

```

\int_zero:N Functions that reset an  $\langle int \rangle$  register to zero.
\int_zero:c
\int_gzero:N 2823 \def_new:Npn \int_zero:N #1 {\#1=\c_zero}
\int_gzero:c 2824 \def_new:Npn \int_zero:c #1 {\exp_args:Nc \int_zero:N}
\int_gzero:c 2825 \def_new:Npn \int_gzero:N #1 {\pref_global:D #1=\c_zero}
2826 \def_new:Npn \int_gzero:c {\exp_args:Nc \int_gzero:N}

\int_add:Nn Adding and subtracting to and from a counter ... We should think of using these func-
\int_add:cn tions
\int_gadd:Nn
\int_gadd:cn 2827 \def_new:Npn \int_add:Nn #1#2{
\int_sub:Nn We need to say by in case the first argument is a register accessed by its number, e.g.,
\int_sub:cn \count23. Not that it should ever happen but...
\int_gsub:Nn
\int_gsub:cn 2828 \int_advance:w #1 by \int_eval:w #2\int_eval_end:
2829 \*check
2830 \chk_local_or_pref_global:N #1
2831 \*check
2832 }
2833 \def_new:Npn \int_add:cn{\exp_args:Nc \int_add:Nn}
2834 \def_new:Npn \int_sub:Nn #1#2{
2835 \int_advance:w #1-\int_eval:w #2\int_eval_end:
2836 \*check
2837 \chk_local_or_pref_global:N #1
2838 \*check
2839 }
2840 \def_new:Npn \int_gadd:Nn {
2841 \*check
2842 \pref_global_chk:
2843 \*check
2844 \*check \pref_global:D
2845 \int_add:Nn }
2846 \def_new:Npn \int_gsub:Nn {
2847 \*check
2848 \pref_global_chk:
2849 \*check
2850 \*check \pref_global:D
2851 \int_sub:Nn }
2852 \def_new:Npn \int_gadd:cn{\exp_args:Nc \int_gadd:Nn}
2853 \def_new:Npn \int_sub:cn{\exp_args:Nc \int_sub:Nn}
2854 \def_new:Npn \int_gsub:cn{\exp_args:Nc \int_gsub:Nn}

\int_use:N Here is how counters are accessed:
\int_use:c
2855 \let_new:NN \int_use:N \tex_the:D
2856 \def_new:Npn \int_use:c #1{\int_use:N \cs:w#1\cs_end:}

```

```

\int_to_arabic:n Nothing exciting here.

```

```

2857 \def_new:Npn \int_to_arabic:n #1{\int_to_number:w \int_eval:n{#1}}

```

```

\int_roman_lcuc_mapping:Nnn Using TEX's built-in feature for producing roman numerals has some surprising features.
One is the the characters resulting from \int_to_roman:w have category code 12 so they

```

may fail in certain comparison tests. Therefore we use a mapping from the character \TeX produces to the character we actually want which will give us letters with category code 11.

```
2858 \def_new:Npn \int_roman_lcuc_mapping:Nnn #1#2#3{
2859   \def:cpn {int_to_lc_roman_#1:}{#2}
2860   \def:cpn {int_to_uc_roman_#1:}{#3}
2861 }
```

Here are the default mappings. I haven't found any examples of say Turkish doing the mapping $i \rightarrow \text{\i}$ $I \rightarrow \text{\I}$ but at least there is a possibility for it if needed. Note: I have now asked a Turkish person and he tells me they do the $i \rightarrow \text{\i}$ $I \rightarrow \text{\I}$ mapping.

```
2862 \int_roman_lcuc_mapping:Nnn i i I
2863 \int_roman_lcuc_mapping:Nnn v v V
2864 \int_roman_lcuc_mapping:Nnn x x X
2865 \int_roman_lcuc_mapping:Nnn l l L
2866 \int_roman_lcuc_mapping:Nnn c c C
2867 \int_roman_lcuc_mapping:Nnn d d D
2868 \int_roman_lcuc_mapping:Nnn m m M
```

For the delimiter we cheat and let it gobble its arguments instead.

```
2869 \int_roman_lcuc_mapping:Nnn Q \use_none:nn \use_none:nn
```

\int_to_roman:n The commands for producing the lower and upper case roman numerals run a loop on
 \int_to_Roman:n one character at a time and also carries some information for upper or lower case with
 $\text{\int_to_roman_lcuc:NN}$ it. We put it through \int_eval:n first which is safer and more flexible.

```
2870 \def_new:Npn \int_to_roman:n #1 {
2871   \exp_after:NN \int_to_roman_lcuc:NN \exp_after:NN l
2872   \int_to_roman:w \int_eval:n {#1} Q
2873 }
2874 \def_new:Npn \int_to_Roman:n #1 {
2875   \exp_after:NN \int_to_roman_lcuc:NN \exp_after:NN u
2876   \int_to_roman:w \int_eval:n {#1} Q
2877 }
2878 \def_new:Npn \int_to_roman_lcuc:NN #1#2{
2879   \cs_use:c {int_to_#1c_roman_#2:}
2880   \int_to_roman_lcuc:NN #1
2881 }
```

$\text{_convert_number_with_rule:nnN}$ This is our major workhorse for conversions. $\#1$ is the number we want converted, $\#2$ is the base number, and $\#3$ is the function converting the number. This function expects to receive a non-negative integer and as such is ideal for something using \if_case:w internally.

The basic example is this: We want to convert the number 50 ($\#1$) into an alphabetic equivalent ax . For the English language our list contains 26 elements so this is our argument $\#2$ while the function $\#3$ just turns 1 into a , 2 into b , etc. Hence our goal is to turn 50 into the sequence $\#3\{1\}\#1\{24\}$ so what we do is to first divide 50 by 26 and truncating the result returning 1. Then before we execute this we call the function again but this time on the result of the remainder of the division. This goes on until the

remainder is less than or equal to the base number where we just call the function #3 directly on the number.

We do a little pre-expansion of the arguments below as they otherwise have a tendency to grow quite large.

```
2882 \def:Npn \int_convert_number_with_rule:nnN #1#2#3{
2883   \int_compare:nNnTF {#1}>{#2}
2884   {
2885     \exp_args:No \int_convert_number_with_rule:nnN
2886     { \int_use:N\int_div_truncate:nn {#1-1}{#2} }{#2}
2887     #3
```

Note that we have to nudge our modulus function so it won't return 0 as that wouldn't work with \if_case:w when that expects a positive number to produce a letter.

```
2888     \exp_args:No #3 { \int_use:N\int_eval:n{1+\int_mod:nn {#1-1}{#2}} }
2889   }
2890   { \exp_args:No #3{ \int_use:N\int_eval:n{#1} } }
2891 }
```

As can be seen it is even simpler to convert to number systems that contain 0, since then we don't have to add or subtract 1 here and there.

lph_default_conversion_rule:n Now we just set up a default conversion rule. Ideally every language should have one
lph_default_conversion_rule:n such rule, as say in Danish there are 29 letters in the alphabet.

```
2892 \def_new:Npn \int_alph_default_conversion_rule:n #1{
2893   \if_case:w #1
2894     \or: a\or: b\or: c\or: d\or: e\or: f
2895     \or: g\or: h\or: i\or: j\or: k\or: l
2896     \or: m\or: n\or: o\or: p\or: q\or: r
2897     \or: s\or: t\or: u\or: v\or: w\or: x
2898     \or: y\or: z
2899   \fi:
2900 }
2901 \def_new:Npn \int_Alph_default_conversion_rule:n #1{
2902   \if_case:w #1
2903     \or: A\or: B\or: C\or: D\or: E\or: F
2904     \or: G\or: H\or: I\or: J\or: K\or: L
2905     \or: M\or: N\or: O\or: P\or: Q\or: R
2906     \or: S\or: T\or: U\or: V\or: W\or: X
2907     \or: Y\or: Z
2908   \fi:
2909 }
```

\int_to_alph:n The actual functions are just instances of the generic function. The second argument of
\int_to_Alph:n \int_convert_number_with_rule:nnN should of course match the number of \or:s in the conversion rule.

```
2910 \def_new:Npn \int_to_alph:n #1{
2911   \int_convert_number_with_rule:nnN {#1}{26}
2912   \int_alph_default_conversion_rule:n
2913 }
```



```

2914 \def_new:Npn \int_to_Alph:n #1{
2915   \int_convert_number_with_rule:nnN {#1}{26}
2916   \int_Alph_default_conversion_rule:n
2917 }

```

`\int_to_symbol:n` Turning a number into a symbol is also easy enough.

```

2918 \def_new:Npn \int_to_symbol:n #1{
2919   \mode_if_math:TF
2920   {
2921     \int_convert_number_with_rule:nnN {#1}{9}
2922     \int_symbol_math_conversion_rule:n
2923   }
2924   {
2925     \int_convert_number_with_rule:nnN {#1}{9}
2926     \int_symbol_text_conversion_rule:n
2927   }
2928 }

```

`symbol_math_conversion_rule:n` Nothing spectacular here.

```

symbol_text_conversion_rule:n
2929 \def_new:Npn \int_symbol_math_conversion_rule:n #1 {
2930   \if_case:w #1
2931   \or: *
2932   \or: \dagger
2933   \or: \ddagger
2934   \or: \mathsection
2935   \or: \mathparagraph
2936   \or: \l
2937   \or: **
2938   \or: \dagger\dagger
2939   \or: \ddagger\ddagger
2940   \fi:
2941 }
2942 \def_new:Npn \int_symbol_text_conversion_rule:n #1 {
2943   \if_case:w #1
2944   \or: \textasteriskcentered
2945   \or: \textdagger
2946   \or: \textdaggerdbl
2947   \or: \textsection
2948   \or: \textparagraph
2949   \or: \textbardbl
2950   \or: \textasteriskcentered\textasteriskcentered
2951   \or: \textdagger\textdagger
2952   \or: \textdaggerdbl\textdaggerdbl
2953   \fi:
2954 }

```

`\l_tmpa_int` We provide four local and two global scratch counters, maybe we need more or less.

```

\l_tmpb_int
\l_tmpc_int 2955 \int_new:N \l_tmpa_int
\g_tmpa_int 2956 \int_new:N \l_tmpb_int
\g_tmpb_int 2957 \int_new:N \l_tmpc_int
\l_loop_int 2958 \int_new:N \g_tmpa_int

```

```

2959 \int_new:N \g_tmpb_int
2960 \int_new:N \l_loop_int % a variable for use in loops (whilenum etc)

```

```

\int_eval:n Evaluating a calc expression using normal operators. Many of these are exactly the same
\int_eval:w as the ones in the num module so we just use them.
\int_eval_end:
2961 \let_new:NN \int_eval:n \num_eval:n
2962 \let_new:NN \int_eval:w \num_eval:w
2963 \let_new:NN \int_eval_end: \num_eval_end:

```

\c_max_int The largest number allowed is $2^{31} - 1$

```

2964 \const_new:Nn \c_max_int {2147483647}

```

```

\int_pre_eval_one_arg:Nn These might be handy when handing down values to other functions. All they do is
\int_pre_eval_two_args:Nnn evaluate the number in advance.

```

```

2965 \def:Npn \int_pre_eval_one_arg:Nnn #1#2{\exp_args:No#1{\int_eval:w#2}}
2966 \def:Npn \int_pre_eval_two_args:Nnn #1#2#3{
2967   \exp_args:Noo#1{\int_use:N\int_eval:w#2}{\int_use:N\int_eval:w#3}
2968 }

```

```

\int_div_truncate:nn As \num_eval:w rounds the result of a division we also provide a version that truncates
\int_div_round:nn the result.
\int_mod:nn
\int_div_truncate_raw:nn 2969 \def_new:Npn \int_div_truncate:nn {
\int_div_round_raw:nn 2970   \int_pre_eval_two_args:Nnn\int_div_truncate_raw:nn
\int_mod_raw:nn 2971 }

```

Initial version didn't work correctly with eTeX's implementation.

```

2972 %\def_new:Npn \int_div_truncate_raw:nn #1#2 {
2973 %  \int_eval:n{ (2*#1 - #2) / (2* #2) }
2974 %}

```

New version by Heiko:

```

2975 \def_new:Npn \int_div_truncate_raw:nn #1#2 {
2976   \int_eval:w
2977   \if_num:w \int_eval:w#1 = \c_zero
2978     0
2979   \else:
2980     (#1
2981     \if_num:w \int_eval:w #1 < \c_zero
2982       \if_num:w \int_eval:w#2 < \c_zero
2983         -( #2 +
2984       \else:
2985         +( #2 -
2986     \fi:
2987   \else:
2988     \if_num:w \int_eval:w #2 < \c_zero
2989       +( #2 +
2990     \else:
2991       -( #2 -

```

```

2992         \fi:
2993         \fi:
2994         1)/2)
2995     \fi:
2996     /(#2)
2997 \int_eval_end:
2998 }

```

For the sake of completeness:

```

2999 \def_new:Npn \int_div_round:nn {
3000     \int_pre_eval_two_args:Nnn\int_div_round_raw:nn
3001 }
3002 \def_new:Npn \int_div_round_raw:nn #1#2 {\int_eval:n{#1/#2}}

```

Finally there's the modulus operation.

```

3003 \def_new:Npn \int_mod:nn {\int_pre_eval_two_args:Nnn\int_mod_raw:nn}
3004 \def_new:Npn \int_mod_raw:nn #1#2 {
3005     \int_eval:n{ #1 - \int_div_truncate_raw:nn {#1}{#2} * #2 }
3006 }

```

```

\int_compare:nNnTF Simple comparison tests.
\int_compare:nNnT
\int_compare:nNnF 3007 \let_new:NN \int_compare:nNnTF \num_compare:nNnTF
3008 \let_new:NN \int_compare:nNnT \num_compare:nNnT
3009 \let_new:NN \int_compare:nNnF \num_compare:nNnF

```

```

\int_max_of:nn Simple comparison tests.
\int_min_of:nn
\int_abs:n 3010 \let_new:NN \int_max_of:nn \num_max_of:nn
3011 \let_new:NN \int_min_of:nn \num_min_of:nn
3012 \let_new:NN \int_abs:nn \num_abs:nn

```

```

\int_compare_p:nNn A predicate function.

3013 \let_new:NN \int_compare_p:nNn \num_compare_p:nNn

```

```

\int_if_odd_p:n A predicate function.
\int_if_odd:nTF
\int_if_odd:nT 3014 \def_new:Npn \int_if_odd_p:n #1 {
3015     \if_num_odd:w \int_eval:n{#1}
\int_if_odd:nF 3016     \c_true
3017     \else:
3018     \c_false
3019     \fi:
3020 }
3021 \def_test_function_new:npn {int_if_odd:n}#1{\if_num_odd:w \int_eval:n{#1}}

```

```

\int_whiledo:nNnT These are quite easy given the above functions. The while versions test first and then
\int_whiledo:nNnF execute the body. The dowhile does it the other way round. The have to be defined as
\int_dowhile:nNnT "long" since the T argument might contain \par tokens.
\int_dowhile:nNnF
3022 \def_long_new:Npn \int_whiledo:nNnT #1#2#3#4{

```

```

3023 \int_compare:nNnT {#1}#2{#3}{#4} \int_whiledo:nNnT {#1}#2{#3}{#4}}
3024 }
3025 \def_long_new:Npn \int_whiledo:nNnF #1#2#3#4{
3026 \int_compare:nNnF {#1}#2{#3}{#4} \int_whiledo:nNnF {#1}#2{#3}{#4}}
3027 }
3028 \def_long_new:Npn \int_dowhile:nNnT #1#2#3#4{
3029 #4 \int_compare:nNnT {#1}#2{#3}{\int_dowhile:nNnT {#1}#2{#3}{#4}}
3030 }
3031 \def_long_new:Npn \int_dowhile:nNnF #1#2#3#4{
3032 #4 \int_compare:nNnF {#1}#2{#3}{\int_dowhile:nNnF {#1}#2{#3}{#4}}
3033 }

```

16.6.1 Scanning and conversion

Conversion between different numbering schemes requires meticulous work. A number can be preceded by any number of + and/or -. We define a generic function which will return the sign and/or the remainder.

```

\int_get_sign_and_digits:n A number may be preceded by any number of +s and -s. Start out by assuming we have
\int_get_sign:n a positive number.
\int_get_digits:n
get_sign_and_digits_aux:nNNN 3034 \def_new:Npn \int_get_sign_and_digits:n #1{
get_sign_and_digits_aux:oNNN 3035 \int_get_sign_and_digits_aux:nNNN {#1} \c_true \c_true \c_true
3036 }
3037 \def_new:Npn \int_get_sign:n #1{
3038 \int_get_sign_and_digits_aux:nNNN {#1} \c_true \c_true \c_false
3039 }
3040 \def_new:Npn \int_get_digits:n #1{
3041 \int_get_sign_and_digits_aux:nNNN {#1} \c_true \c_false \c_true
3042 }

```

Now check the first character in the string. Only a - can change if a number is positive or negative, hence we reverse the boolean governing this. Then gobble the - and start over.

```

3043 \def_new:Npn \int_get_sign_and_digits_aux:nNNN #1#2#3#4{
3044 \tlist_if_head_eq_charcode:fNTF {#1} -
3045 {
3046 \bool_if:NTF #2
3047 { \int_get_sign_and_digits_aux:oNNN {\use_none:n #1} \c_false #3#4 }
3048 { \int_get_sign_and_digits_aux:oNNN {\use_none:n #1} \c_true #3#4 }
3049 }

```

The other cases are much simpler since we either just have to gobble the + or exit immediately and insert the correct sign.

```

3050 {
3051 \tlist_if_head_eq_charcode:fNTF {#1} +
3052 { \int_get_sign_and_digits_aux:oNNN {\use_none:n #1} #2#3#4}
3053 {

```

The boolean #3 is for printing the sign while #4 is for printing the digits.

```

3054     \bool_double_if:NNnnnn #3#4
3055     { \bool_if:NF #2 - #1 }
3056     { \bool_if:NF #2 -      }
3057     { #1 } {   }
3058   }
3059 }
3060 }
3061 \def_new:Npn \int_get_sign_and_digits_aux:oNNN{
3062   \exp_args:No\int_get_sign_and_digits_aux:nNNN
3063 }

```

`\int_convert_from_base_ten:nn` #1 is the base 10 number to be converted to base #2. We split off the sign first, print if if
`convert_from_base_ten_aux:nnn` there and then convert only the number. Since this is supposedly a base 10 number we
`convert_from_base_ten_aux:non` can let T_EX do the reading of + and -.
`convert_from_base_ten_aux:fon`

```

3064 \def:Npn \int_convert_from_base_ten:nn#1#2{
3065   \num_compare:nNnTF {#1}<\c_zero
3066   {
3067     - \int_convert_from_base_ten_aux:non {}
3068     { \int_use:N \int_eval:n {-#1} }
3069   }
3070   {
3071     \int_convert_from_base_ten_aux:non {}
3072     { \int_use:N \int_eval:n {#1} }
3073   }
3074   {#2}
3075 }

```

The algorithm runs like this:

1. If the number $\langle num \rangle$ is greater than $\langle base \rangle$, calculate modulus of $\langle num \rangle$ and $\langle base \rangle$ and carry that over for next round. The remainder is calculated as a truncated division of $\langle num \rangle$ and $\langle base \rangle$. Start over with these new values.
2. If $\langle num \rangle$ is less than or equal to $\langle base \rangle$ convert it to the correct symbol, print the previously calculated digits and exit.

#1 is the carried over result, #2 the remainder and #3 the base number.

```

3076 \def_new:Npn \int_convert_from_base_ten_aux:nnn#1#2#3{
3077   \num_compare:nNnTF {#2}<{#3}
3078   { \int_convert_number_to_letter:n{#2} #1 }
3079   {
3080     \int_convert_from_base_ten_aux:fon
3081     {
3082       \int_convert_number_to_letter:n {\int_use:N\int_mod_raw:nn {#2}{#3}}
3083       #1
3084     }
3085     {\int_use:N \int_div_truncate_raw:nn{#2}{#3}}
3086     {#3}
3087   }
3088 }
3089 \def:Npn \int_convert_from_base_ten_aux:non{
3090   \exp_args:Nno\int_convert_from_base_ten_aux:nnn

```

```

3091 }
3092 \def:Npn \int_convert_from_base_ten_aux:fon{
3093   \exp_args:Nfo\int_convert_from_base_ten_aux:nnn
3094 }

```

`\int_convert_number_to_letter:n` Turning a number for a different base into a letter or digit.

```

3095 \def:Npn \int_convert_number_to_letter:n #1{ \if_case:w \int_eval:w
3096   #1-10\scan_stop: \exp_after:NN A \or: \exp_after:NN B \or:
3097   \exp_after:NN C \or: \exp_after:NN D \or: \exp_after:NN E \or:
3098   \exp_after:NN F \or: \exp_after:NN G \or: \exp_after:NN H \or:
3099   \exp_after:NN I \or: \exp_after:NN J \or: \exp_after:NN K \or:
3100   \exp_after:NN L \or: \exp_after:NN M \or: \exp_after:NN N \or:
3101   \exp_after:NN O \or: \exp_after:NN P \or: \exp_after:NN Q \or:
3102   \exp_after:NN R \or: \exp_after:NN S \or: \exp_after:NN T \or:
3103   \exp_after:NN U \or: \exp_after:NN V \or: \exp_after:NN W \or:
3104   \exp_after:NN X \or: \exp_after:NN Y \or: \exp_after:NN Z \else:
3105   \use_arg_i_after_fi:nw{ #1 }\fi: }

```

`\int_convert_to_base_ten:nn` #1 is the number, #2 is its base. First we get the sign, then use only the digits/letters from it and pass that onto a new function.

```

3106 \def:Npn \int_convert_to_base_ten:nn #1#2 {
3107   \int_use:N\int_eval:n{
3108     \int_get_sign:n{#1}
3109     \exp_args:Nf\int_convert_to_base_ten_aux:nn {\int_get_digits:n{#1}}{#2}
3110   }
3111 }

```

This is an intermediate function to get things started.

```

3112 \def_new:Npn \int_convert_to_base_ten_aux:nn #1#2{
3113   \int_convert_to_base_ten_auxi:nnN {0}{#2} #1 \q_nil
3114 }

```

Here we check each letter/digit and calculate the next number. #1 is the previously calculated result (to be multiplied by the base), #2 is the base and #3 is the next letter/digit to be added.

```

3115 \def_new:Npn \int_convert_to_base_ten_auxi:nnN#1#2#3{
3116   \quark_if_nil:NTF #3
3117   {#1}
3118   {\exp_args:No\int_convert_to_base_ten_auxi:nnN
3119     {\int_use:N \int_eval:n{ #1*#2+\int_convert_letter_to_number:N #3} }
3120     {#2}
3121   }
3122 }

```

This is for turning a letter or digit into a number. This function also takes care of handling lowercase and uppercase letters. Hence a is turned into 11 and so is A.

```

3123 \def:Npn \int_convert_letter_to_number:N #1{
3124   \int_compare:nNnTF{ '#1 }<{58}{#1}
3125   {

```

```

3126 \int_eval:n{ '#1 -
3127 \if:w\int_compare_p:nNn{ '#1}<{91}
3128 55
3129 \else:
3130 87
3131 \fi:
3132 }
3133 }
3134 }

```

Show token usage:

```

3135 </initex | package>
3136 <*showmemory>
3137 \showMemUsage
3138 </showmemory>

```

17 Length registers

L^AT_EX3 knows about two types of length registers for internal use: rubber lengths (**skips**) and rigid lengths (**dims**).

17.1 Skip registers

17.1.1 Functions

<code>\skip_new:N</code> <code>\skip_new:c</code> <code>\skip_new_l:N</code>	<code>\skip_new:N</code> $\langle skip \rangle$
--	---

Defines $\langle skip \rangle$ to be a new variable of type **skip**.

T_EXhackers note: `\skip_new:N` is the equivalent to plain T_EX's `\newskip`. However, the internal register allocation is done differently.

<code>\skip_zero:N</code> <code>\skip_zero:c</code> <code>\skip_gzero:N</code> <code>\skip_gzero:c</code>	<code>\skip_zero:N</code> $\langle skip \rangle$
--	--

Locally or globally reset $\langle skip \rangle$ to zero. For global variables the global versions should be used.

<code>\skip_set:Nn</code> <code>\skip_set:cn</code> <code>\skip_gset:Nn</code> <code>\skip_gset:cn</code>	<code>\skip_set:Nn</code> $\langle skip \rangle$ { $\langle skip \text{ value} \rangle$ }
--	---

These functions will set the $\langle skip \rangle$ register to the $\langle length \rangle$ value.

<code>\skip_add:Nn</code>
<code>\skip_add:cn</code>
<code>\skip_gadd:Nn</code>
<code>\skip_gadd:cn</code>

`\skip_add:Nn <skip> { <length> }`

These functions will add to the `<skip>` register the value `<length>`. If the second argument is a `<skip>` register too, the surrounding braces can be left out.

<code>\skip_sub:Nn</code>
<code>\skip_gsub:Nn</code>

`\skip_gsub:Nn <skip> { <length> }`

These functions will subtract from the `<skip>` register the value `<length>`. If the second argument is a `<skip>` register too, the surrounding braces can be left out.

<code>\skip_use:N</code>
<code>\skip_use:c</code>

`\skip_use:N <skip>`

This function returns the length value kept in `<skip>` in a way suitable for further processing.

T_EXhackers note: The function `\skip_use:N` could be implemented directly as the T_EX primitive `\tex_the:D` which is also responsible to produce the values for other internal quantities. We have chosen to use individual functions for counters, dimensions etc. to allow checks and to make the code more self-explanatory.

<code>\skip_horizontal:N</code>
<code>\skip_horizontal:c</code>
<code>\skip_horizontal:n</code>
<code>\skip_vertical:N</code>
<code>\skip_vertical:c</code>
<code>\skip_vertical:n</code>

`\skip_horizontal:N <skip>`
`\skip_horizontal:n { <length> }`

The `hor` functions insert `<skip>` or `<length>` with the T_EX primitive `\hskip`. The `vertical` variants do the same with `\vskip`. The `n` versions evaluate `<length>` with `\skip_eval:n`.

<code>\skip_infinite_glue:nTF</code>

`\skip_infinite_glue:nTF {<skip>} {<true>} {<false>}`

Checks if `<skip>` contains infinite stretch or shrink components and executes either `<true>` or `<false>`. Also works on input like `3pt plus .5in`.

<code>\skip_split_finite_else_action:nnNN</code>
--

`\skip_split_finite_else_action:nnNN {<skip>} {<action>}`
`<dimen1> <dimen2>`

Checks if `<skip>` contains finite glue. If it does then it assigns `<dimen1>` the stretch component and `<dimen2>` the shrink component. If it contains infinite glue set `<dimen1>` and `<dimen2>` to zero and execute #2 which is usually an error or warning message of some sort.

<code>\skip_eval:n</code>

`\skip_eval:n {<skip expr>}`

Evaluates the value of `<skip expr>` so that `\skip_eval:n {5pt plus 3fil + 3pt minus 1fil}` puts `8.0pt plus 3.0fil minus 1.0fil` back into the input stream. Expandable.

T_EXhackers note: This is the ε -T_EX primitive `\glueexpr` turned into a function taking an argument.

17.1.2 Formatting a skip register value

17.1.3 Variable and constants

<code>\c_max_skip</code>	Constant that denotes the maximum value which can be stored in a $\langle skip \rangle$ register.
--------------------------	---

<code>\c_zero_skip</code>	Set of constants denoting useful values.
---------------------------	--

<code>\l_tmpa_skip</code> <code>\l_tmpb_skip</code> <code>\l_tmpc_skip</code> <code>\g_tmpa_skip</code> <code>\g_tmpb_skip</code>	Scratch register for immediate use.
---	-------------------------------------

17.2 Dim registers

17.2.1 Functions

<code>\dim_new:N</code> <code>\dim_new:c</code> <code>\dim_new_l:N</code>	<code>\dim_new:N</code> $\langle dim \rangle$
---	---

Defines $\langle dim \rangle$ to be a new variable of type `dim`.

T_EXhackers note: `\dim_new:N` is the equivalent to plain T_EX's `\newdimen`. However, the internal register allocation is done differently.

<code>\dim_zero:N</code> <code>\dim_zero:c</code> <code>\dim_gzero:N</code> <code>\dim_gzero:c</code>	<code>\dim_zero:N</code> $\langle dim \rangle$
--	--

Locally or globally reset $\langle dim \rangle$ to zero. For global variables the global versions should be used.

<code>\dim_set:Nn</code> <code>\dim_set:cn</code> <code>\dim_gset:Nn</code> <code>\dim_gset:cn</code>	<code>\dim_set:Nn</code> $\langle dim \rangle$ { $\langle dim value \rangle$ }
--	--

These functions will set the $\langle dim \rangle$ register to the $\langle dim value \rangle$ value.

<code>\dim_add:Nn</code>
<code>\dim_add:cn</code>
<code>\dim_gadd:Nn</code>
<code>\dim_gadd:cn</code>

`\dim_add:Nn <dim> { <length> }`

These functions will add to the $\langle dim \rangle$ register the value $\langle length \rangle$. If the second argument is a $\langle dim \rangle$ register too, the surrounding braces can be left out.

<code>\dim_sub:Nn</code>
<code>\dim_gsub:Nn</code>

`\dim_gsub:Nn <dim> { <length> }`

These functions will subtract from the $\langle dim \rangle$ register the value $\langle length \rangle$. If the second argument is a $\langle dim \rangle$ register too, the surrounding braces can be left out.

<code>\dim_use:N</code>
<code>\dim_use:c</code>

`\dim_use:N <dim>`

This function returns the length value kept in $\langle dim \rangle$ in a way suitable for further processing.

T_EXhackers note: The function `\dim_use:N` could be implemented directly as the T_EX primitive `\tex_the:D` which is also responsible to produce the values for other internal quantities. We have chosen to use individual functions for counters, dimensions etc. to allow checks and to make the code more self-explanatory.

<code>\dim_eval:n</code>

`\dim_eval:n {<dim expr>}`

Evaluates the value of a dimension expression so that `\dim_eval:n {5pt+3pt}` puts 8pt back into the input stream. Expandable.

T_EXhackers note: This is the ε -T_EX primitive `\dimexpr` turned into a function taking an argument.

<code>\if_dim:w</code>

`\if_dim:w <dimen1> <rel> <dimen2> <true> \else: <false> \fi:`

Compare two dimensions. It is recommended to use `\dim_eval:n` to correctly evaluate and terminate these numbers. $\langle rel \rangle$ is one of $<$, $=$ or $>$ with catcode 12.

T_EXhackers note: This is the T_EX primitive `\ifdim`.

<code>\dim_compare:nNnTF</code>
<code>\dim_compare:nNnT</code>
<code>\dim_compare:nNnF</code>

`\dim_compare:nNnTF {<dim expr>} <rel> {<dim expr>} {<true>} {<false>}`

These functions test two dimension expressions against each other. They are both evaluated by `\dim_eval:n`. Note that if both expressions are normal dimension variables as in

```
\dim_compare:nNnTF \l_temp_dim < \c_zero_skip {negative}{non-negative}
```

you can safely omit the braces.

T_EXhackers note: This is the T_EX primitive `\ifdim` turned into a function.

<code>\dim_compare_p:nNn</code>	<code>\dim_compare_p:nNn</code>	<code>{\dim expr}</code>	<code>\rel</code>	<code>{\dim expr}</code>
---------------------------------	---------------------------------	--------------------------	-------------------	--------------------------

Predicate version of the above functions.

<code>\dim_while:nNnT</code>	<code>\dim_while:nNnT</code>	<code>\dim expr</code>	<code>\rel</code>	<code>\dim expr</code>	<code>\true</code>
<code>\dim_while:nNnF</code>					
<code>\dim_dowhile:nNnT</code>					
<code>\dim_dowhile:nNnF</code>					

`\dim_while:nNnT` tests the dimension expressions and if true performs the body T until the test fails. `\dim_dowhile:nNnT` is similar but executes the body first and then performs the check, thus ensuring that the body is executed at least once. The F versions are similar but continue the loop as long as the test is false.

17.2.2 Variable and constants

<code>\c_max_dim</code>	Constant that denotes the maximum value which can be stored in a $\langle dim \rangle$ register.
-------------------------	--

<code>\c_zero_dim</code>	Set of constants denoting useful values.
--------------------------	--

<code>\l_tmpa_dim</code>	Scratch register for immediate use.
<code>\l_tmppb_dim</code>	
<code>\l_tmppc_dim</code>	
<code>\l_tmppd_dim</code>	
<code>\g_tmpa_dim</code>	
<code>\g_tmppb_dim</code>	
<code>\g_tmppc_dim</code>	
<code>\g_tmppd_dim</code>	

17.3 Muskips

<code>\muskip_new:N</code>	<code>\muskip_new:N</code>	$\langle muskip \rangle$
<code>\muskip_new_l:N</code>		

Defines $\langle muskip \rangle$ to be a new variable of type `muskip`.

T_EXhackers note: `\muskip_new:N` is the equivalent to plain T_EX's `\newmuskip`. However, the internal register allocation is done differently.

<code>\muskip_set:Nn</code>	<code>\muskip_set:Nn</code>	$\langle muskip \rangle$	<code>{ \langle muskip value \rangle }</code>
<code>\muskip_gset:Nn</code>			

These functions will set the $\langle muskip \rangle$ register to the $\langle length \rangle$ value.

`\muskip_add:Nn`

`\muskip_gadd:Nn` `\muskip_add:Nn` $\langle muskip \rangle$ { $\langle length \rangle$ }

These functions will add to the $\langle muskip \rangle$ register the value $\langle length \rangle$. If the second argument is a $\langle muskip \rangle$ register too, the surrounding braces can be left out.

`\muskip_sub:Nn`

`\muskip_gsub:Nn` `\muskip_gsub:Nn` $\langle muskip \rangle$ { $\langle length \rangle$ }

These functions will subtract from the $\langle muskip \rangle$ register the value $\langle length \rangle$. If the second argument is a $\langle muskip \rangle$ register too, the surrounding braces can be left out.

17.4 The Implementation

We start by ensuring that the required packages are loaded.

```
3139 <package>\ProvidesExplPackage
3140 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
3141 <package&!check>\RequirePackage{13int}
3142 <package&!check>\RequirePackage{13prg}
3143 <package & check>\RequirePackage{13chk}
3144 <*initex | package>
```

17.4.1 Skip registers

```
\skip_new:N Allocation of a new internal registers.
\skip_new:c
\skip_new_l:N 3145 <*initex>
3146 \alloc_setup_type:nnn {skip} \c_zero \c_max_register_num
3147 \def_new:Npn \skip_new:N #1 {\alloc_reg:NnNN g {skip} \tex_skipdef:D #1 }
3148 \def_new:Npn \skip_new_l:N #1 {\alloc_reg:NnNN l {skip} \tex_skipdef:D #1 }
3149 </initex>
3150 <package>\let:NN \skip_new:N \newskip
3151 \def_new:Npn \skip_new:c {\exp_args:Nc \skip_new:N}
```

`\skip_set:Nn` Setting skips is again something that I would like to make uniform at the moment to get
`\skip_set:cn` a better overview.

```
\skip_gset:Nn
\skip_gset:cn 3152 \def_new:Npn \skip_set:Nn #1#2{#1\skip_eval:n{#2}
3153 <*check>
3154 \chk_local_or_pref_global:N #1
3155 </check>
3156 }
3157 \def_new:Npn \skip_gset:Nn {
3158 <*check>
3159 \pref_global_chk:
3160 </check>
3161 <-check> \pref_global:D
3162 \skip_set:Nn }
3163 \def_new:Npn \skip_set:cn {\exp_args:Nc \skip_set:Nn }
3164 \def_new:Npn \skip_gset:cn {\exp_args:Nc \skip_gset:Nn }
```

```

\skip_zero:N Reset the register to zero.
\skip_gzero:N
\skip_zero:c 3165 \def_new:Npn \skip_zero:N #1{#1\c_zero_skip \scan_stop:
\skip_gzero:c 3166 \*check>
3167 \chk_local_or_pref_global:N #1
3168 \</check>
3169 }
3170 \def_new:Npn \skip_gzero:N {

```

We make sure that a local variable is not updated globally by changing the internal test (i.e. `\chk_local_or_pref_global:N`) before making the assignment. This is done by `\pref_global_chk:` which also issues the necessary `\pref_global:D`. This is not very efficient, but this code will be only included for debugging purposes. Using `\pref_global:D` in front of the local function is better in the production versions.

```

3171 \*check>
3172 \pref_global_chk:
3173 \</check>
3174 \<-check> \pref_global:D
3175 \skip_zero:N}
3176 \def_new:Npn \skip_zero:c {\exp_args:Nc \skip_zero:N}
3177 \def_new:Npn \skip_gzero:c {\exp_args:Nc \skip_gzero:N}

```

```

\skip_add:Nn Adding and subtracting to and from \skip's
\skip_add:cn
\skip_gadd:Nn 3178 \def_new:Npn \skip_add:Nn #1#2{
\skip_sub:Nn We need to say by in case the first argument is a register accessed by its number, e.g.,
\skip_gsub:Nn \skip23.

```

```

3179 \tex_advance:D#1 by \skip_eval:n{#2}
3180 \*check>
3181 \chk_local_or_pref_global:N #1
3182 \</check>
3183 }
3184 \def_new:Npn\skip_add:cn{\exp_args:Nc\skip_add:Nn}
3185 \def_new:Npn \skip_sub:Nn #1#2{
3186 \tex_advance:D#1-\skip_eval:n{#2}
3187 \*check>
3188 \chk_local_or_pref_global:N #1
3189 \</check>
3190 }
3191 \def_new:Npn \skip_gadd:Nn {
3192 \*check>
3193 \pref_global_chk:
3194 \</check>
3195 \<-check> \pref_global:D
3196 \skip_add:Nn }
3197 \def_new:Npn \skip_gsub:Nn {
3198 \*check>
3199 \pref_global_chk:
3200 \</check>
3201 \<-check> \pref_global:D
3202 \skip_sub:Nn }

```

```

\skip_horizontal:N Inserting skips.
\skip_horizontal:c
\skip_horizontal:n 3203 \let_new:NN \skip_horizontal:N \tex_hskip:D
\skip_vertical:N 3204 \def_new:Npn \skip_horizontal:c {\exp_args:Nc\skip_horizontal:N}
\skip_vertical:c 3205 \def_new:Npn \skip_horizontal:n #1{\skip_horizontal:N \skip_eval:n{#1}}
\skip_vertical:c 3206 \let_new:NN \skip_vertical:N \tex_vskip:D
\skip_vertical:n 3207 \def_new:Npn \skip_vertital:c {\exp_args:Nc\skip_vertical:N}
3208 \def_new:Npn \skip_vertical:n #1{\skip_vertical:N \skip_eval:n{#1}}

```

```

\skip_use:N Here is how skip registers are accessed:
\skip_use:c
3209 \let_new:NN \skip_use:N \tex_the:D
3210 \def_new:Npn \skip_use:c #1{\exp_args:Nc\skip_use:N}

```

`\skip_eval:n` Evaluating a calc expression.

```

3211 \def_new:Npn \skip_eval:n #1 {\etex_glueexpr:D #1 \scan_stop:}

```

`\l_tmpa_skip` We provide three local and two global scratch registers, maybe we need more or less.

```

\l_tmpb_skip
\l_tmpc_skip 3212 %%\chk_new_cs:N \l_tmpa_skip
\g_tmpa_skip 3213 %%\tex_skipdef:D\l_tmpa_skip 255 %currently taken up by \skip@
\g_tmpb_skip 3214 \skip_new:N \l_tmpa_skip
3215 \skip_new:N \l_tmpb_skip
3216 \skip_new:N \l_tmpc_skip
3217 \skip_new:N \g_tmpa_skip
3218 \skip_new:N \g_tmpb_skip

```

```

\c_zero_skip
\c_max_skip
3219 <!\package>
3220 \skip_new:N \c_zero_skip
3221 \skip_set:Nn \c_zero_skip {0pt}
3222 \skip_new:N \c_max_skip
3223 \skip_set:Nn \c_max_skip {16383.99999pt}
3224 </!\package>
3225 <!\initex>
3226 \let:NN \c_zero_skip \z@
3227 \let:NN \c_max_skip \maxdimen
3228 </!\initex>

```

`\skip_infinite_glue:nTF` With ε -TeX we all of a sudden get access to a lot information we should otherwise consider ourselves lucky to get. One is the stretch and shrink components of a skip register and the order of those components. `\skip_infinite_glue:nTF` tests it directly by looking at the stretch and shrink order. If either of the predicate functions return `<true>` `\prg_logic_or_p:nn` will return `<true>` and the logic test will take the true branch.

```

3229 \def_new:Npn \skip_infinite_glue:nTF #1{
3230   \predicate:nTF {
3231     \int_compare_p:nNn {\etex_gluestretchorder:D #1 } > \c_zero ||
3232     \int_compare_p:nNn {\etex_glueshrinkorder:D #1 } > \c_zero
3233   }
3234 }

```

`split_finite_else_action:nnNN` This macro is useful when performing error checking in certain circumstances. If the `<skip>` register holds finite glue it sets `#3` and `#4` to the stretch and shrink component resp. If it holds infinite glue set `#3` and `#4` to zero and issue the special action `#2` which is probably an error message. Assignments are global.

```

3235 \def_new:Npn \skip_split_finite_else_action:nnNN #1#2#3#4{
3236   \skip_infinite_glue:nTF {#1}
3237   {
3238     #3 = \c_zero_skip
3239     #4 = \c_zero_skip
3240     #2
3241   }
3242   {
3243     #3 = \etex_gluestretch:D #1 \scan_stop:
3244     #4 = \etex_glueshrink:D #1 \scan_stop:
3245   }
3246 }
```

17.4.2 Dimen registers

```

\dim_new:N Allocating <dim> registers...
\dim_new:c
\dim_new_l:N 3247 (*initex)
3248 \alloc_setup_type:nnn {dimen} \c_zero \c_max_register_num
3249 \def_new:Npn \dim_new:N #1 {\alloc_reg:NnNN g {dimen} \tex_dimendef:D #1 }
3250 \def_new:Npn \dim_new_l:N #1 {\alloc_reg:NnNN l {dimen} \tex_dimendef:D #1 }
3251 
```

`/initex`

```

3252 \package\let:NN \dim_new:N \newdimen
3253 \def_new:Npn \dim_new:c {\exp_args:Nc \dim_new:N}

\dim_set:Nn We add \dim_eval:n in order to allow simple arithmetic and a space just for those using
\dim_gset:Nn \dimen1 or alike. See OR!
\dim_set:cn
\dim_set:Nc 3254 \def_new:Npn \dim_set:Nn #1#2{#1~ \dim_eval:n{#2}}
3255 \def_new:Npn \dim_gset:Nn {\pref_global:D \dim_set:Nn }
\dim_gset:cn 3256 \def_new:Npn \dim_set:cn {\exp_args:Nc \dim_set:Nn }
\dim_gset:Nc 3257 \def_new:Npn \dim_set:Nc {\exp_args:NNc \dim_set:Nn }
\dim_gset:cc 3258 \def_new:Npn \dim_gset:cn {\exp_args:Nc \dim_gset:Nn }
3259 \def_new:Npn \dim_gset:Nc {\exp_args:NNc \dim_gset:Nn }
3260 \def_new:Npn \dim_gset:cc {\exp_args:Ncc \dim_gset:Nn }

\dim_zero:N Resetting.
\dim_gzero:N
\dim_zero:c 3261 \def_new:Npn \dim_zero:N #1{#1\c_zero_skip}
3262 \def_new:Npn \dim_gzero:N {\pref_global:D \dim_zero:N}
\dim_gzero:N 3263 \def_new:Npn \dim_zero:c {\exp_args:Nc \dim_zero:N}
3264 \def_new:Npn \dim_gzero:c {\exp_args:Nc \dim_gzero:N}

\dim_add:Nn Addition.
\dim_add:cn
\dim_add:Nc 3265 \def_new:Npn \dim_add:Nn #1#2{
\dim_gadd:Nn
\dim_gadd:cn
```

We need to say by in case the first argument is a register accessed by its number, e.g., `\dimen23`.

```
3266 \tex_advance:D#1 by \dim_eval:n{#2}\scan_stop:
3267 }
3268 \def_new:Npn\dim_add:cn{\exp_args:Nc\dim_add:Nn}
3269 \def_new:Npn\dim_add:Nc{\exp_args:NNc\dim_add:Nn}
3270 \def_new:Npn \dim_gadd:Nn { \pref_global:D \dim_add:Nn }
3271 \def_new:Npn\dim_gadd:cn{\exp_args:Nc\dim_gadd:Nn}
```

`\dim_sub:Nn` Subtracting.

```
\dim_sub:cn
\dim_sub:Nc 3272 \def_new:Npn \dim_sub:Nn #1#2{\tex_advance:D#1-#2\scan_stop:}
\dim_gsub:Nn 3273 \def_new:Npn\dim_sub:cn{\exp_args:Nc\dim_sub:Nn}
\dim_gsub:cn 3274 \def_new:Npn\dim_sub:Nc{\exp_args:NNc\dim_sub:Nn}
3275 \def_new:Npn \dim_gsub:Nn { \pref_global:D \dim_sub:Nn }
3276 \def_new:Npn\dim_gsub:cn{\exp_args:Nc\dim_gsub:Nn}
```

`\dim_use:N` Accessing a $\langle dim \rangle$.

```
\dim_use:c
3277 \let_new:NN \dim_use:N \tex_the:D
3278 \def_new:Npn \dim_use:c {\exp_args:Nc\dim_use:N}
```

`\l_tmpa_dim` Some scratch registers.

```
\l_tmpb_dim
\l_tmpc_dim 3279 \dim_new:N \l_tmpa_dim
\l_tmpd_dim 3280 \dim_new:N \l_tmpb_dim
\g_tmpa_dim 3281 \dim_new:N \l_tmpc_dim
\g_tmpb_dim 3282 \dim_new:N \l_tmpd_dim
3283 \dim_new:N \g_tmpa_dim
3284 \dim_new:N \g_tmpb_dim
```

`\c_zero_dim` Just aliases.

```
\c_max_dim
3285 \let_new:NN \c_zero_dim \c_zero_skip
3286 \let_new:NN \c_max_dim \c_max_skip
```

`\dim_eval:n` Evaluating a calc expression.

```
3287 \def_new:Npn \dim_eval:n #1 {\etex_dimexpr:D #1 \scan_stop:}
```

`\if_dim:w` The comparison primitive.

```
3288 \let_new:NN \if_dim:w \tex_ifdim:D
```

`\dim_compare:nNnTF` Check the expression and choose branch.

```
\dim_compare:nNnT
\dim_compare:nNnF 3289 \def_new:Npn \dim_compare:nNnTF #1#2#3{
3290 \if_dim:w \dim_eval:n {#1} #2 \dim_eval:n {#3}
3291 \exp_after:NN \use_arg_i:nn
3292 \else:
3293 \exp_after:NN \use_arg_ii:nn
3294 \fi:
```



```

3295 }
3296 \def_new:Npn \dim_compare:nNnT #1#2#3{
3297   \if_dim:w \dim_eval:n {#1} #2 \dim_eval:n {#3}
3298     \exp_after:NN \use_arg_ii:nn
3299   \fi:
3300   \use_none:n
3301 }
3302 \def_new:Npn \dim_compare:nNnF #1#2#3{
3303   \if_dim:w \dim_eval:n {#1} #2 \dim_eval:n {#3}
3304     \exp_after:NN \use_none:n
3305   \else:
3306     \exp_after:NN \use_arg_i:n
3307   \fi:
3308 }

```

`\dim_compare_p:nNn` A predicate function.

```

3309 \def_new:Npn \dim_compare_p:nNn #1#2#3{
3310   \if_dim:w \dim_eval:n {#1} #2 \dim_eval:n {#3}
3311     \c_true
3312   \else:
3313     \c_false
3314   \fi:
3315 }

```

`\dim_while:nNnT` while and do-while functions for dimensions. Same as for the int type only the names
`\dim_while:nNnF` have changed.

```

\dim_dowhile:nNnT
\dim_dowhile:nNnF
3316 \def_new:Npn \dim_while:nNnT #1#2#3#4{
3317   \dim_compare:nNnT {#1}#2{#3}-{#4} \dim_while:nNnT {#1}#2{#3}-{#4}}
3318 }
3319 \def_new:Npn \dim_while:nNnF #1#2#3#4{
3320   \dim_compare:nNnF {#1}#2{#3}-{#4} \dim_while:nNnF {#1}#2{#3}-{#4}}
3321 }
3322 \def_new:Npn \dim_dowhile:nNnT #1#2#3#4{
3323   #4 \dim_compare:nNnT {#1}#2{#3}{\dim_dowhile:nNnT {#1}#2{#3}-{#4}}
3324 }
3325 \def_new:Npn \dim_dowhile:nNnF #1#2#3#4{
3326   #4 \dim_compare:nNnF {#1}#2{#3}{\dim_dowhile:nNnF {#1}#2{#3}-{#4}}
3327 }

```

17.4.3 Muskips

`\muskip_new:N` And then we add muskips.

```

\muskip_new_l:N
3328 (*initex)
3329 \alloc_setup_type:nnn {muskip} \c_zero \c_max_register_num
3330 \def_new:Npn \muskip_new:N #1{\alloc_reg:NnNN g {muskip} \tex_muskipdef:D #1}
3331 \def_new:Npn \muskip_new_l:N #1{\alloc_reg:NnNN l {muskip} \tex_muskipdef:D #1}
3332 \end{initex}
3333 \package\let_new:NN \muskip_new:N \newmuskip % nicked from LaTeX

```

```

\muskip_set:Nn Simple functions for muskips.
\muskip_gset:Nn
\muskip_add:Nn 3334 \def_new:Npn \muskip_set:Nn#1#2{#1\etex_muexpr:D#2\scan_stop:}
\muskip_gadd:Nn 3335 \def_new:Npn \muskip_gset:Nn{\pref_global:D\muskip_set:Nn}
\muskip_sub:Nn 3336 \def_new:Npn \muskip_add:Nn#1#2{\tex_advance:D#1\etex_muexpr:D#2\scan_stop:}
\muskip_gsub:Nn 3337 \def_new:Npn \muskip_gadd:Nn{\pref_global:D\muskip_add:Nn}
3338 \def_new:Npn \muskip_sub:Nn#1#2{\tex_advance:D#1-\etex_muexpr:D#2\scan_stop:}
3339 \def_new:Npn \muskip_gsub:Nn{\pref_global:D\muskip_sub:Nn}

3340 </initex | package>

```

18 Token Registers

There is a second form beside token list pointers in which L^AT_EX3 stores token lists, namely the internal T_EX token registers. Functions dealing with these registers got the prefix `\toks_`. Unlike token list pointers we have an accessing function as one can see below.

The main difference between $\langle toks \rangle$ (token registers) and $\langle tlp \rangle$ (token list pointers) is their behavior regarding expansion. While $\langle tlp \rangle$'s expand fully (i.e., until only unexpandable tokens are left) inside an argument that is subject to expansion (i.e., denote by \mathbf{x}) $\langle toks \rangle$'s expand always only up to one level, i.e., passing their contents without further expansion.

18.1 Functions

<pre> \toks_new:N \toks_new:c \toks_new_l:N </pre>	<pre> \toks_new:N <toks> </pre>
--	---------------------------------------

Defines $\langle toks \rangle$ to be a new token list register.

T_EXhackers note: This is the L^AT_EX3 allocation for what was called `\newtoks` in plain T_EX.

<pre> \toks_set:Nn \toks_set:No \toks_set:Nd \toks_set:Nf \toks_set:Nx \toks_set:cn \toks_set:co \toks_set:cf \toks_set:cx \toks_gset:Nn \toks_gset:No \toks_gset:Nx </pre>	<pre> \toks_set:Nn <toks> {\token list} </pre>
---	--

Defines $\langle toks \rangle$ to hold the token list $\langle token list \rangle$. Global variants of this command assign the value globally the other variants expand the $\langle token list \rangle$ up to a certain level before

the assignment or interpret the $\langle token\ list \rangle$ as a character list and form a control sequence out of it.

T_EXhackers note: `\toks_set:Nn` could have been specified in plain T_EX by $\langle toks \rangle = \{ \langle token\ list \rangle \}$ but all other functions have no counterpart in plain T_EX. Additionally the functions above will check for correct local and global assignments, something that isn't available in plain T_EX.

<code>\toks_gset_eq:NN</code>	<code>\toks_gset_eq:NN $\langle toks1 \rangle$ $\langle toks2 \rangle$</code>
-------------------------------	---

The $\langle toks1 \rangle$ globally set to the value of $\langle toks2 \rangle$. Don't try to use `\toks_gset:Nn` for this purpose if the second argument is also a token register.

<code>\toks_clear:N</code>	<code>\toks_gclear:N</code>	<code>\toks_clear:N $\langle toks \rangle$</code>
----------------------------	-----------------------------	--

The $\langle toks \rangle$ is locally or globally cleared.

<code>\toks_put_left:Nn</code> <code>\toks_gput_left:Nn</code> <code>\toks_put_right:Nn</code> <code>\toks_put_right:No</code> <code>\toks_put_right:Nx</code> <code>\toks_gput_right:Nn</code> <code>\toks_gput_right:No</code> <code>\toks_gput_right:Nx</code>	<code>\toks_put_left:Nn $\langle toks \rangle$ {$\langle token\ list \rangle$}</code>
--	---

These functions will append $\langle token\ list \rangle$ to the left or right of $\langle toks \rangle$. Assignment is done either locally or globally. If possible append to the right since this operation is faster.

<code>\toks_use:N</code>	<code>\toks_use:N $\langle toks \rangle$</code>
--------------------------	--

Accesses the contents of $\langle toks \rangle$. Contrary to token list pointers $\langle toks \rangle$ can't be access simply by calling them directly.

T_EXhackers note: Something like `\the $\langle toks \rangle$` .

<code>\toks_use_clear:N</code>	<code>\toks_gclear:N</code>	<code>\toks_use_clear:N $\langle toks \rangle$</code>
--------------------------------	-----------------------------	--

Accesses the contents of $\langle toks \rangle$ and clears (locally or globally) it afterwards. Actually the clearing operation is done in a way that does not prohibit the access of the following tokens in the input stream with functions stored in the token register. In other words this function is not exactly the same as calling `\toks_use:N $\langle toks \rangle$ \toks_clear:N $\langle toks \rangle$` in sequence.

18.2 Predicates and conditionals

<code>\toks_if_empty_p:N</code>
<code>\toks_if_empty:NTF</code>
<code>\toks_if_empty:NT</code>
<code>\toks_if_empty:NF</code>
<code>\toks_if_empty_p:c</code>
<code>\toks_if_empty:cTF</code>
<code>\toks_if_empty:cT</code>
<code>\toks_if_empty:cF</code>

`\toks_if_empty:NTF <toks> {{true code}}{{false code}}`

Tests if `<toks>` is empty.

18.3 Variable and constants

<code>\c_empty_toks</code>	Constant that is always empty.
----------------------------	--------------------------------

<code>\l_tmpa_toks</code>
<code>\l_tmpb_toks</code>
<code>\g_tmpa_toks</code>
<code>\g_tmpb_toks</code>

Scratch register for immediate use. They are not used by conditionals or predicate functions.

18.3.1 Internal functions

<code>\toks_put_left_aux:w</code>	Used by <code>\toks_put_left:Nn</code> and its variants.
-----------------------------------	--

<code>\tex_toksdef:D</code>	Primitive function for defining a <code><cs></code> to correspond to a token register should not be used by a programmer.
-----------------------------	---

TeXhackers note: This function was named `\toksdef`.

18.4 The Implementation

We start by ensuring that the required packages are loaded. We check for `l3expan` since this a basic package that is essential for use of any higher-level package.

```

3341 <package>\ProvidesExplPackage
3342 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
3343 <package & check>\RequirePackage{l3chk}\par
3344 <package>\RequirePackage{l3expan}\par
3345 <*initex | package>

```

`\toks_new:N` Allocates a new token register. This function is already defined above.

```
\toks_new_l:N
\toks_new:c 3346 \*initex>
3347 \alloc_setup_type:nnn {toks} \c_zero \c_max_register_num
3348 \def_new:Npn \toks_new:N #1{\alloc_reg:NnNN g {toks} \tex_toksdef:D #1}
3349 \def_new:Npn \toks_new_l:N #1{\alloc_reg:NnNN l {toks} \tex_toksdef:D #1}
3350 \*initex>
3351 \package\let:NN \toks_new:N \newtoks % nick from LaTeX for the moment
3352 \def_new:Npn \toks_new:c {\exp_args:Nc\toks_new:N}
```

`\toks_clear:N` These functions clear a token register, either locally or globally.

```
\toks_gclear:N
3353 \def_new:Npn \toks_clear:N #1{#1\c_empty_toks
3354 \*check>
3355 \chk_local_or_pref_global:N #1
3356 \*check>
3357 }
3358 \def_new:Npn \toks_gclear:N {
3359 \*check>
3360 \pref_global_chk:
3361 \*check>
3362 \-check> \pref_global:D
3363 \toks_clear:N}
```

`\toks_use:N` This function just returns the contents of a token register.

```
\toks_use:c
3364 \let_new:NN \toks_use:N \the_internal:D
3365 \def_new:Npn \toks_use:c {\exp_args:Nc\toks_use:N}
```

`\toks_use_clear:N` These functions clear a token register (locally or globally) after returning the contents.

`\toks_use_gclear:N` They make sure that clearing the register does not interfere with following tokens. In other words, the contents of the register might operate on what follows in the input stream. A direct implementation will save one `\exp_after:NN` but for the sake of checking we do it this way now.

```
3366 \def_new:Npn \toks_use_clear:N#1{
3367 \exp_after:NN
3368 \toks_clear:N
3369 \exp_after:NN
3370 #1
3371 \toks_use:N#1}
3372 \def_new:Npn \toks_use_gclear:N{
3373 \*check>
3374 \pref_global_chk:
3375 \*check>
3376 \-check> \pref_global:D
3377 \toks_use_clear:N}
```

`\toks_put_left:Nn` `\toks_put_left:Nn <toks><stuff>` adds the tokens of *stuff* on the ‘left-side’ of the token register *<toks>*. `\toks_put_left:No` does the same, but expands the tokens once. We need to look out for brace stripping so we add a token, which is then later removed.

```
\toks_gput_left:Nn
\toks_gput_left:Nx
\toks_put_left_aux:w 3378 \def_new:Npn \toks_put_left:Nn #1{
```

```

3379 \exp_after:NN\toks_put_left_aux:w\exp_after:NN\q_mark
3380 \toks_use:N #1\q_stop #1}
3381 \def_new:Npn \toks_put_left:No {\exp_args:NNo \toks_put_left:Nn}
3382 \def_new:Npn \toks_gput_left:Nn {
3383   \check
3384   \pref_global_chk:
3385 }/check
3386 \check \pref_global:D
3387 \toks_put_left:Nn}
3388 \def_new:Npn \toks_gput_left:Nx {\exp_args:NNx \toks_gput_left:Nn}

```

A helper function for `\toks_put_left:Nn`. Its arguments are subsequently the tokens of `\stuffs`, the token register `\toks` and the current contents of `\toks`. We make sure to remove the token we inserted earlier.

```

3389 \def_long_new:Npn \toks_put_left_aux:w #1\q_stop #2#3{
3390   #2\exp_after:NN{\use_arg_i:nn{#3}#1}
3391 }/check
3392 \chk_local_or_pref_global:N #2
3393 }/check
3394 }

```

`\toks_put_right:Nn` These macros add a list of tokens to the right of a token register.

```

\toks_gput_right:Nn
\toks_put_right:No 3395 \def_long_new:Npn \toks_put_right:Nn #1#2{#1\exp_after:NN{\toks_use:N #1#2}
\toks_put_right:No 3396 }/check
\toks_put_right:Nd 3397 \chk_local_or_pref_global:N #1
\toks_put_right:Nf 3398 }/check
\toks_put_right:Nx 3399 }
\toks_gput_right:No 3400 \def_new:Npn \toks_gput_right:Nn {
\toks_gput_right:Nx 3401 }/check
3402 \pref_global_chk:
3403 }/check
3404 \check \pref_global:D
3405 \toks_put_right:Nn}

```

`\toks_gput_right:Nx` expands its (second) argument.

```

3406 \check \def_new:Npn \toks_put_right:No {\exp_args:NNo \toks_put_right:Nn }
3407 \check \def_long_new:Npn \toks_put_right:No#1#2{#1\exp_after:NN\exp_after:NN
3408 \check \exp_after:NN{\exp_after:NN\toks_use:N\exp_after:NN #1#2}}
3409 \check \def_new:Npn \toks_put_right:Nd {\exp_args:NNd \toks_put_right:Nn }
3410 \check \def_long_new:Npn \toks_put_right:Nd#1#2{
3411 \check \exp_after:NN\toks_put_right:No\exp_after:NN#1\exp_after:NN{#2}}

```

We implement `\toks_put_right:Nf` by hand because I think I might use it in the `l3keyval` module in which case it is going to be used a lot.

```

3412 \check \def_new:Npn \toks_put_right:Nf {\exp_args:NNf \toks_put_right:Nn }
3413 \check \def_long_new:Npn \toks_put_right:Nf #1#2{
3414 \check #1\exp_after:NN\exp_after:NN\exp_after:NN{
3415 \check \exp_after:NN\toks_use:N\exp_after:NN #1\int_to_roman:w -'0#2}}
3416 \def_new:Npn \toks_put_right:Nx {\exp_args:NNx \toks_put_right:Nn }
3417 \def_new:Npn \toks_gput_right:No {\exp_args:NNo\toks_gput_right:Nn}
3418 \def_new:Npn \toks_gput_right:Nx {\exp_args:NNx\toks_gput_right:Nn}

```

```

\toks_set:Nn \toks_set:Nn<toks><stuff> stores <stuff> without expansion in <toks>. \toks_set:No
\toks_set:No and \toks_set:Nx expand <stuff> once and fully.
\toks_set:Nd
\toks_set:Nf 3419 <*check>
\toks_set:Np 3420 \def_new:Npn \toks_set:Nn #1{\chk_local:N #1#1}
\toks_set:Nx 3421 </check>
\toks_set:cn
\toks_set:co If we don't check if <toks> is a local register then the \toks_set:Nn function has nothing
\toks_set:cf to do.
\toks_set:cx

```

```

3422 <-check> \let_new:NN \toks_set:Nn\use_noop:
3423 <-check> \def_long_new:Npn \toks_set:No#1#2{#1\exp_after:NN{#2}}
3424 <-check> \def_long_new:Npn \toks_set:Nd#1#2{
3425 <-check> #1\exp_after:NN\exp_after:NN\exp_after:NN{#2}}
3426 <check> \def_new:Npn \toks_set:No {\exp_args:NNo \toks_set:Nn}
3427 <check> \def_new:Npn \toks_set:Nd {\exp_args:NNd \toks_set:Nn}
3428 \def_new:Npn \toks_set:Nx {\exp_args:NNx \toks_set:Nn}

```

We implement \toks_set:Nf by hand when not checking because this is going to be used *extensively* in keyval processing!

```

3429 <check> \def_new:Npn \toks_set:Nf {\exp_args:NNf \toks_set:Nn}
3430 <-check> \def_long_new:Npn \toks_set:Nf #1#2{
3431 <-check> #1\exp_after:NN{\int_to_roman:w -'0#2}}
3432 \def_new:Npn \toks_set:cf {\exp_args:Nc\toks_set:Nf}
3433 \def_new:Npn \toks_set:cn {\exp_args:Nc\toks_set:Nn}
3434 \def_new:Npn \toks_set:co {\exp_args:Nc\toks_set:No}
3435 \def_new:Npn \toks_set:cx {\exp_args:Nc\toks_set:Nx}

```

\toks_gset:Nn These functions are the global variants of the above.

```

\toks_gset:No
\toks_gset:Nx 3436 <*check>
\toks_gset:cn 3437 \def_new:Npn \toks_gset:Nn #1{\chk_global:N #1\pref_global:D#1}
\toks_gset:co 3438 </check>
\toks_gset:cx 3439 <-check> \let_new:NN \toks_gset:Nn\pref_global:D
\toks_gset:cn 3440 \def_new:Npn \toks_gset:No {\exp_args:NNo \toks_gset:Nn}
\toks_gset:co 3441 \def_new:Npn \toks_gset:Nx {\exp_args:NNx \toks_gset:Nn}
\toks_gset:cx 3442 \def_new:Npn \toks_gset:cn {\exp_args:Nc \toks_gset:Nn}
\toks_gset:co 3443 \def_new:Npn \toks_gset:co {\exp_args:Nc \toks_gset:No}
\toks_gset:cx 3444 \def_new:Npn \toks_gset:cx {\exp_args:Nc \toks_gset:Nx}

```

\toks_set_eq:NN \toks_set_eq:NN<toks1><toks2> copies the contents of <toks2> in <toks1>.

```

\toks_set_eq:Nc
\toks_set_eq:cn 3445 <*check>
\toks_set_eq:cc 3446 \def_new:Npn \toks_set_eq:NN#1#2{
\toks_set_eq:cc 3447 \chk_local:N#1
\toks_gset_eq:NN 3448 \chk_var_or_const:N#2
\toks_gset_eq:Nc 3449 #1#2}
\toks_gset_eq:cn 3450 \def_new:Npn \toks_gset_eq:NN#1#2{
\toks_gset_eq:cc 3451 \chk_global:N#1
\toks_gset_eq:cc 3452 \chk_var_or_const:N#2
\toks_gset_eq:cc 3453 \pref_global:D#1#2}
\toks_gset_eq:cc 3454 </check>
\toks_gset_eq:cc 3455 <-check> \let_new:NN \toks_set_eq:NN \use_noop:

```

```

3456 <-check> \let_new:NN \toks_gset_eq:NN \pref_global:D
3457 \def_new:Npn \toks_set_eq:Nc {\exp_args:NNc\toks_set_eq:NN}
3458 \def_new:Npn \toks_set_eq:cN {\exp_args:Nc\toks_set_eq:NN}
3459 \def_new:Npn \toks_set_eq:cc {\exp_args:Ncc\toks_set_eq:NN}
3460 \def_new:Npn \toks_gset_eq:Nc {\exp_args:NNc\toks_gset_eq:NN}
3461 \def_new:Npn \toks_gset_eq:cN {\exp_args:Nc\toks_gset_eq:NN}
3462 \def_new:Npn \toks_gset_eq:cc {\exp_args:Ncc\toks_gset_eq:NN}

```

\toks_if_empty_p:N \toks_if_empty:NTF<toks><true code><false code> tests if a token register is empty and
\toks_if_empty_p:c executes either <true code> or <false code>. This test had the advantage of being expand-
\toks_if_empty:NTF able. Otherwise one has to do an x type expansion in order to prevent problems with
\toks_if_empty:cTF parameter tokens.

```

\toks_if_empty:NT
\toks_if_empty:cT 3463 \def_new:Npn\toks_if_empty_p:N#1{
\toks_if_empty:NF 3464 \if:w \tlist_if_empty_p:o{\toks_use:N #1}
\toks_if_empty:cF 3465 \c_true
3466 \else:
3467 \c_false
3468 \fi:
3469 }
3470 \def_test_function_new:npn{toks_if_empty:N}#1{\if:w \toks_if_empty_p:N #1}
3471 \def_new:Npn\toks_if_empty:cTF{\exp_args:Nc\toks_if_empty:NTF}
3472 \def_new:Npn\toks_if_empty:cT{\exp_args:Nc\toks_if_empty:NT}
3473 \def_new:Npn\toks_if_empty:cF{\exp_args:Nc\toks_if_empty:NF}

```

\toks_if_eq:NNTF This function test whether two token registers contain the same.

```

\toks_if_eq:NNT
\toks_if_eq:NNTF 3474 \def_new:NNn \toks_if_eq:NNTF 2 {
\toks_if_eq:NNTF 3475 \tlist_if_eq:xxTF{\toks_use:N #1}{\toks_use:N #2}
\toks_if_eq:NcTF 3476 }
\toks_if_eq:NcT 3477 \def_new:NNn \toks_if_eq:NNT 2 {
\toks_if_eq:NcF 3478 \tlist_if_eq:xxT{\toks_use:N #1}{\toks_use:N #2}
\toks_if_eq:cNTF 3479 }
\toks_if_eq:cNT 3480 \def_new:NNn \toks_if_eq:NNTF 2 {
\toks_if_eq:cNF 3481 \tlist_if_eq:xxF{\toks_use:N #1}{\toks_use:N #2}
\toks_if_eq:ccTF 3482 }
\toks_if_eq:ccT 3483 \def_new:Npn \toks_if_eq:NcTF {\exp_args:NNc \toks_if_eq:NNTF}
\toks_if_eq:ccF 3484 \def_new:Npn \toks_if_eq:NcT {\exp_args:NNc \toks_if_eq:NNT}
\toks_if_eq:p:NN 3485 \def_new:Npn \toks_if_eq:NcF {\exp_args:NNc \toks_if_eq:NNTF}
\toks_if_eq:p:cN 3486 \def_new:Npn \toks_if_eq:cNTF {\exp_args:Nc \toks_if_eq:NNTF}
\toks_if_eq:p:cN 3487 \def_new:Npn \toks_if_eq:cNT {\exp_args:Nc \toks_if_eq:NNT}
\toks_if_eq:p:Nc 3488 \def_new:Npn \toks_if_eq:cNF {\exp_args:Nc \toks_if_eq:NNTF}
\toks_if_eq:p:cc 3489 \def_new:Npn \toks_if_eq:ccTF {\exp_args:Ncc \toks_if_eq:NNTF}
3490 \def_new:Npn \toks_if_eq:ccT {\exp_args:Ncc \toks_if_eq:NNT}
3491 \def_new:Npn \toks_if_eq:ccF {\exp_args:Ncc \toks_if_eq:NNTF}
3492 \def_new:NNn \toks_if_eq_p:NN 2 {
3493 \tlist_if_eq_p:xx {\toks_use:N #1} {\toks_use:N #2}
3494 }
3495 \def_new:Npn \toks_if_eq_p:cN {\exp_args:Nc \toks_if_eq_p:NN}
3496 \def_new:Npn \toks_if_eq_p:Nc {\exp_args:NNc \toks_if_eq_p:NN}
3497 \def_new:Npn \toks_if_eq_p:cc {\exp_args:Ncc \toks_if_eq_p:NN}

```

\l_tmpa_toks Some scratch register ...

```

\l_tmpb_toks
\l_tmpc_toks
\g_tmpa_toks
\g_tmpb_toks
\g_tmpc_toks

```



```

3498 \tex_toksdef:D \l_tmpa_toks = 255
3499 \initex\seq_put_right:Nn \g_toks_allocation_seq {255}
3500 \toks_new:N \l_tmpb_toks
3501 \toks_new:N \l_tmpc_toks
3502 \toks_new:N \g_tmpa_toks
3503 \toks_new:N \g_tmpb_toks
3504 \toks_new:N \g_tmpc_toks

```

`\c_empty_toks` And here is a constant, which is a (permanently) empty token register.

```

3505 \toks_new:N \c_empty_toks

```

`\toks_remove_extra_brace_group:N` Small function for removing an extra brace group if present. Hmm, not really needed
`\toks_remove_extra_brace_group_aux:NNw` anymore.

```

3506 \def_new:Npn \toks_remove_extra_brace_group:N #1{
3507   \exp_after:NN \toks_remove_extra_brace_group_aux:NNw
3508   \exp_after:NN \toks_set:Nn \exp_after:NN #1
3509   \toks_use:N#1\q_nil
3510 }
3511 \def_long_new:Npn\toks_remove_extra_brace_group_aux:NNw #1#2#3\q_nil{#1#2{#3}}

```

Show token usage:

```

3512 \</initex | package>
3513 \<*showmemory>
3514 \showMemUsage
3515 \</showmemory>

```

19 Communicating with the user

Sometimes it is necessary to pass information back to the user about what is going on. The information can be just that, information, or it can be a warning that something might not happen to his expectation. It could also be that something has gone awry and that processing can't reliably continue without some help from the user. In such a case an error is signalled. When things are really bad, processing may have to stop as there is no way to enter additional commands that put things right again. In such a case we have a fatal error and the L^AT_EX run will be aborted.

19.1 Displaying the information

First of all we need a couple of fairly low level functions that deal with the job of passing the information to the user.

Real information is usually only written to the log file, while warnings are displayed on the screen as well.

<code>\err_info:nn</code>	\err_info:nn { <i><message></i> } { <i><continuation></i> }
<code>\err_warn:nn</code>	

The *<message>* will be written to the log file. When it contains the command

`\err_newline:` a line break will occur and the new line will start with the *<continuation>*. The function `\err_warn:nn` writes the message to the terminal as well. When an erroneous situation is encountered, a message is displayed and the user is given the opportunity to enter some additional code in an attempt to put things right. He may first ask for some help, in which case some extra text will be displayed to him.

`\err_interrupt:NNw` `\err_interrupt:NNw <err id> <label> <more args>`

This function signals a user error by searching the error file denoted by *<err id>* for an error message associated with *<label>*, i.e., specified by a corresponding `\err_interrupt_new:NNNnnn` command. Depending on the number of arguments specified as *<argno>* when the error message was defined, further arguments are read. Then the error message is displayed as explained in `\err_interrupt_new:NNNnnn`.

Finally, when something really serious occurs, L^AT_EX will tell the user about it and abort the run.

`\err_fatal:nn` `\err_fatal:nn { <message> } {<continuation>}`

Just displays the *<message>* and then aborts the L^AT_EX run.

`\err_newline:` `\err_newline:`

Is used to break an informational, warning or error message up into multiple lines. May be defined in such a way that the new line starts with a standard *<continuation>*. A normal line break in such messages can be achieved with `\iow_newline:` from the l3iow module.

19.2 Storing the information

The informational and warning messages are usually short and can be stored as part of a macro; but error messages need to be more verbose. Therefore error messages are stored in external files which are read and searched for the correct error message at the time of the error. In this way it is possible to write extensive help texts without cluttering T_EX's main memory.

19.2.1 Dealing with the error file

`\err_file_new:Nn` `\err_file_new:Nn <err id> {<err file name>}`

Opens a new error file to write errors to. *<err id>* is a unique identifier for the external *<err file name>*. By convention *<err id>* is declared as a constant (i.e., starts with `\c_`) and ends with `_tlp`. If this command is issued while some other error file is open we get an internal error message.

`\err_file_close:N` `\err_file_close:N <err id>`

Closes the currently open error file and checks that it matches *<err id>*, i.e., that everything is alright in the code.

19.2.2 Declaring an error message in the error file

```

\err_interrupt_new:NNNnnn <err id> <label> <argno>
{ <short msg> }
{ <long msg> }
\err_interrupt_new:NNNnnn { <recovery code> }

```

This function declares a new error message which can be addressed via `\err_interrupt:NNw`. The pair $(\langle err id \rangle, \langle label \rangle)$ has to be unique where $\langle label \rangle$ can be some otherwise arbitrary token (usually the function name in which the error routine is called). Actually, the pair $(\langle err id \rangle, \text{expansion of } \langle label \rangle)$ has to be unique since for reasons of speed, tests are carried out using `\if_meaning:NN`.

$\langle argno \rangle$ specifies the number of extra arguments that will be supplied to the error routine when `\err_interrupt:NNw` is called. These arguments can be used within $\langle short msg \rangle$, $\langle long msg \rangle$, and/or $\langle recovery code \rangle$ to provide further information to the user. They are denoted with #1, #2, etc. within these arguments.

The $\langle short msg \rangle$ is displayed directly on the terminal if the error occurs, $\langle long msg \rangle$ is displayed when the user types `h` in response to the error prompt of `TEX`, and $\langle recovery code \rangle$ is executed afterwards. This means that $\langle recovery code \rangle$ is inserted after any deletions or insertions given by the user. All three arguments are expanded while they are written to the error file, therefore one has to prevent expansion of tokens with `\token_to_string:N` that should be expanded when the error is triggered.

19.3 Internal functions

`\err_display_aux:w` This function is constructed on the fly while reading the error file. It grabs following arguments from the code (if any) and then displays the error message and inserts the $\langle recovery code \rangle$.

`\err_interrupt_new_aux:w` Helper function used to write the error message info onto the error file.

```

\err_msgline_aux:NNNnnn <argno> <label> {<short>}
\err_msgline_aux:NNNnnn {<long msg>}{<recovery code>}

```

Function written in front of every error message on the error file. It will be executed when the error file is read back in comparing $\langle label \rangle$ to `\l_err_label_token`. If they are the same, `\err_display_aux:w` will be defined and the reading process will stop.

`\err_message:x` `\err_message:x {<error message>}`

Function that directly triggers `TEX`'s error handler. It should not be used directly.

T_EXhackers note: This is the L^AT_EX3 name for the `\errormessage` primitive.

19.4 Kernel specific functions

For a number of the functions described above specific variants are provided that are used in the kernel of L^AT_EX3.

<code>\err_kernel_info:n</code>
<code>\err_kernel_warn:n</code>
<code>\err_kernel_fatal:n</code>

`\err_kernel_info:n {⟨message⟩}`

<code>\err_kernel_interrupt:Nw</code>
<code>\err_kernel_interrupt_new:NNnnn</code>

Abbreviations for writing and accessing kernel error messages that go to the error file `\c_kernel_err_tlp`.

<code>\err_latex_bug:x</code>

`\err_latex_bug:x {⟨error message⟩}`

Creates an internal error message. This is intended to be used in places that should not be reached in normal operation. Something is wrong with the code.

19.5 Variables and constants

<code>\c_iow_err_stream</code>

Output stream used to access the error files during their generation.

<code>\c_kernel_err_tlp</code>

Identifier denoting the kernel error file. (Its contents is the name of the external file.)

<code>\g_err_curr_fname</code>

Global variable containing the name of the currently open error file. Empty when no such file is open for writing.

<code>\tex_errorcontextlines:D</code>

Variable determining the amount of macro expansion contents shown to the user when an error is triggered. L^AT_EX3 sets this to -1 since to the average user this contents is of no interest.

T_EXhackers note: This is the L^AT_EX3 name for the T_EX3 primitive `\errorcontextlines`.

<code>\g_err_help_toks</code>

Token register that holds the message that will be shown if the user types `h` in response to an error message that was produced by `\err_message:x`.

T_EXhackers note: This is the L^AT_EX3 name for the T_EX primitive `\errhelp`.

<code>\l_err_label_token</code>

Variable holding the `⟨label⟩` to look up in an error file.

19.6 The implementation

```

3516 <package>\ProvidesExplPackage
3517 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
3518 <package>\RequirePackage{l3basics}
3519 <package>\RequirePackage{l3tlp}
3520 <package>\RequirePackage{l3expan}
3521 <package>\RequirePackage{l3num}
3522 <package>\RequirePackage{l3io}
3523 <package>\RequirePackage{l3int}
3524 <package>\RequirePackage{l3toks}
3525 <package>\RequirePackage{l3token}
3526 <*initex | package>

```

19.6.1 Code to be moved to other modules

`\g_file_curr_name_tlp` This variable is used to store the name of the file currently being processed. It should be part of the code that defines the higher level I/O commands.

```

3527 \tlp_new:Nn \g_file_curr_name_tlp {no~file}

```

`\err_message:x` The L^AT_EX3 name for a T_EX primitive. This should perhaps move to `l3names.dtx`.

```

3528 \let_new:NN \err_message:x \tex_errmessage:D

```

`\text_put_sp:` We need these functions for certain error and warning messages right away. They put
`\text_put_four_sp:` one and four spaces into the message stream.

```

3529 \def_new:Npn \text_put_sp: {~}
3530 \def_new:Npn \text_put_four_sp: {\text_put_sp: \text_put_sp:
3531                                \text_put_sp: \text_put_sp: }

```

`\cmd_arg_list_build` This macro takes a digit as its argument and creates a string of # characters and argument numbers, such as `##1##2##3`. This list can then later be used in defining a new macro. To do this it locally uses a count register and a token register.

```

3532 \def:Npn\cmd_arg_list_build#1{

```

First we need to make sure that the token register we will be using for temporary storage is empty.

```

3533 \toks_clear:N\l_tmpb_toks

```

Then we can store the argument in a count register that will be decremented until it's value is zero. Beacuse of the use of the result of this macro, the argument needs to be between 1 and 9; this could be tested, but such a test is not (yet) added.

```

3534 \int_set:Nn \l_tmpa_int {#1}
3535 \int_while:nNnT \l_tmpa_int > \c_zero {

```

In the loop we first add the value of our counter to the contents of the token register;

```

3536 \toks_put_left:No \l_tmpb_toks {\the_internal:D\l_tmpa_int}

```

and precede it with two hash marks.

```
3537 \toks_put_left:Nn \l_tmpb_toks {##}
```

Now the count register is decremented and another iteration will follow so long as zero isn't reached.

```
3538 \int_decr:N\l_tmpa_int
3539 }
```

Finally the contents of the token register needs to be copied as the expansion of a local variable.

```
3540 \def:Npx\l_cmd_arg_list{\the_internal:D\l_tmpb_toks}
3541 }
```

`\cmd_declare:Nnn` This macro is a first replacement for L^AT_EX 2_ε's `\newcommand`. It takes the name of a new macro as its first argument, the number of arguments for the new macro is taken as the second argument.

```
3542 \def:Npn\cmd_declare:Nnn#1[#2]{
3543 \cmd_arg_list_build{#2}
3544 \exp_args:NNO\def:Npn#1\l_cmd_arg_list
3545 }
```

`\io_show_file_lineno:` A function to add the number of the line and the name of the file to a message as an indication of where the message was triggered.

```
3546 \def_new:Npn \io_show_file_lineno:{
3547 on~line~\the_internal:D\tex_inputlineno:D\text_put_sp:~of~
3548 file~\g_file_curr_name_tlp}
```

19.6.2 Variables and constants

`\g_err_help_toks` A token register to store the help text for an error message in.

```
3549 \let:NwN \g_err_help_toks \tex_errhelp:D
```

`\l_err_label_token` This will hold the current error label.

```
3550 \def_new:Npn \l_err_label_token {}
```

`\tex_errorcontextlines:D` Since we are producing our own error and help messages we can turn off the nasty stack information coming from T_EX's stomach.

```
3551 \int_set:Nn\tex_errorcontextlines:D\c_minus_one
```

19.6.3 Displaying the information

Here we define the fairly low level commands needed to communicate with the user.

`\err_info:nn` Write a message to the log file (`\err_info:nn`) or to both the log file and the terminal
`\err_warn:nn` (`\err_warn:nn`).

```
3552 \def_new:Npn \err_info:nn #1#2{
```

Make sure that the *continuation* is part of `\err_newline:.`

```
3553 \def:Npn\err_newline:{\iow_newline:#2}
```

Then write the message.

```
3554 \io_put_log:x {#1~\io_show_file_lineno:}}
```

```
3555 \def_new:Npn \err_warn:nn #1#2{
```

```
3556 \def:Npn\err_newline:{\iow_newline:#2}
```

```
3557 \io_put_term:x {#1~\io_show_file_lineno:}}
```

`\err_info_noline:nn` These variants of the above two functions don't add the linenumber to the message.

`\err_warn_noline:nn`

```
3558 \def_new:Npn \err_info_noline:nn #1#2{
```

```
3559 \def:Npn\err_newline:{\iow_newline:#2}
```

```
3560 \io_put_log:x {#1}}
```

```
3561 \def_new:Npn \err_warn_noline:nn #1#2{
```

```
3562 \def:Npn\err_newline:{\iow_newline:#2}
```

```
3563 \io_put_term:x {#1}}
```

`\err_interrupt:NNw` `\err_interrupt:NNw` is the function that is called when some error occurs in the code. It takes at least two arguments, the `<errfile>` which is a token list that holds the name of the file where the error message should be fetched from, and the label to identify the error message in the error file. However, it may have additional arguments that are picked up by the error handler extracted from the error file. This is specified in the third argument to `\err_interrupt_new:NNNnnn`.

```
3564 \def_new:Npn \err_interrupt:NNw #1#2{\let:NwN \l_err_label_token #2
```

```
3565 \group_begin:
```

For some reason we get some `\pars` into the file if we use the current definition of `\iow_long_unexpanded:N` to write the messages. This is probably a consequence of using token registers to prohibit the expansion of code.

```
3566 \let:NwN \par\use_noop:
```

We want to ensure that we are not in programmer's mode (no spaces) but we want to switch on internal naming conventions.

```
3567 \CodeStop
```

```
3568 \NamesStart:
```

We better clear all short references that are active, otherwise we may get surprising results.

```
3569 %\clearshortrefmaps
```

```
3570 \tex_input:D #1~\err_display_aux:w}
```

`\err_fatal:nn` Write a message to the log file and to the terminal.
`\err_fatal_noline:nn`

```
3571 \def_new:Npn \err_fatal:nn #1#2{
    Make sure that the continuation is part of \err_newline.
3572   \def:Npn\err_newline:{\iow_newline:#2}
    Then write the message.
3573   \io_put_term:x {#1~\io_show_file_lineno:}
    Finally abort the LATEX run.
3574   \tex_end:D
3575 }
```

A variant that doesn't include the line number where the error occurred.

```
3576 \def_new:Npn \err_fatal_noline:nn #1#2{
3577   \def:Npn\err_newline:{\iow_newline:#2}
3578   \io_put_term:x {#1}
3579   \tex_end:D
3580 }
```

`\err_newline:` `\err_newline:` is used to introduce a new line in an error message. I would like to use `\\` but this would mean redefinition which should be avoided to make the error message the last action before control is given to the user (otherwise something like `\group_end:` would interfere with insertions/deletions by the user). `\err_newline:` will be redefined by the various functions displaying messages to include the correct *continuation*.

```
3581 \def_new:Npn \err_newline: {^^J}
```

19.6.4 Dealing with the error file

This section contains code that combines Michaels original thoughts on the the subject with Denys' further ideas.

`\c_iow_err_stream` Error messages are logged using the output stream `\c_iow_err_stream`.

```
3582 \iow_new:N \c_iow_err_stream
```

`\g_err_curr_fname` A nick name for the currently open error file. It is empty if no error file is currently open.

```
3583 \tlp_new:Nn \g_err_curr_fname{}
```

`\err_file_new:Nn` This function defines a new error file. The first argument is a token list which should hold the name of the error file, the second argument is the name of the error file. The token list should be a constant defined by this function.

```
3584 \def_new:Npn \err_file_new:Nn #1#2{
3585   \tlp_if_empty:NF\g_err_curr_fname
3586   {\err_latex_bug:x{Unclosed~error~file~'\g_err_curr_fname'}}}
3587   \iow_open:Nn \c_iow_err_stream {#2}
3588   \err_kernel_info:n{Errorfile~'\g_err_curr_fname'~opened~for~output}
3589   \tlp_gset:Nn \g_err_curr_fname{#2}
3590   \tlp_new:Nn #1{#2}}
```


`\err_file_close:N` This function closes the current error file.

```
3591 \def_new:Npn \err_file_close:N#1{
```

Before we close the stream, we write out a final error handler that catches mismatch within error message labels and their calls. Actually this should be integrated into `\err_file_new:Nn`, too.

```
3592 \tlp_if_eq:NNF#1\g_err_curr_fname
3593   {\err_latex_bug:x{You~closed~the~wrong~error~file~‘#1’.~
3594     Open~is~‘\g_err_curr_fname’.}}
3595 \iow_long_unexpanded:Nn \c_iow_err_stream {\err_latex_bug:x{Didn’t~find~the~
3596   correct~error~message~to~show.\iow_newline:
3597   Was~searching~for~a~function~
3598   with~the~following~meaning:\iow_newline:
3599   \token_to_string:N\token_to_meaning:N
3600   \token_to_string:N\l_err_label_token}}
```

The `\group_end:` here matches the one from `\err_interrupt:NNw` that is used to hide changes to `\par` etc.

```
3601   \group_end:}
3602 \iow_close:N \c_iow_err_stream
3603 \err_kernel_info:n{Errorfile~‘\g_err_curr_fname’~closed}
3604 \tlp_gset_eq:NN\g_err_curr_fname\c_empty_tlp
3605 }
```

19.6.5 Declaring an error message in the error file

`\err_interrupt_new:NNNnnn` This function declares a new error message. `\err_interrupt_new:NNNnnn` *<errfile>* *<errlabel>* *<argno>* *<errmsg>* *<helpmsg>* *<code>*. That error message is fetched from the error file *<errfile>*. The label to search for is *<errlabel>*, the error handler has *<argno>* number of arguments (actually *<argno>* + 3 since the *<errmsg>*, *<helpmsg>* and *<code>* are also arguments), and *<errmsg>* is the message to display, *<helpmsg>* is the message that is displayed when the user enters `h`, while *<code>* is extra code to perform when the error occurs. *<code>* is perhaps not necessary, we will see.

We have to check that the label associated with the error message is unique. This means that its replacement text (labels are simply arbitrary functions) is different from the replacement text of any other label in the same error set.

```
3606 \def_new:Npn \err_interrupt_new:NNNnnn #1{
```

Both *<errmsg>* and *<code>* might contain hashmarks denoting arguments to the error handler.

```
3607   \group_begin: \char_set_catcode:nn{‘\#’}{12}
```

We also have to check that output goes to the correct error file.

```
3608   \if_meaning:NN#1\g_err_curr_fname
3609   \else:
3610     \err_latex_bug:x{Error~text~goes~to~wrong~err~file:~
3611       ‘\g_err_curr_fname’~is~open~but~you~requested~
```

```

3612         '#1'}
3613     \fi:
3614     \err_interrupt_new_aux:w}
3615 \def_long_new:Npn \err_interrupt_new_aux:w #1#2#3#4#5{
3616     \iow_long_unexpanded:Nn \c_iow_err_stream
3617         {\err_msgline_aux:NNnnn #1#2{#3}{#4}{#5}\use_noop:}
3618     \group_end:}

```

`\err_msgline_aux:NNnnn` This function is executed when an error file is read back by `\err_interrupt:NNw`. It compares its first argument against `\l_err_label_token` using `\if_meaning:NN` and if this fails the function does nothing; otherwise it defines `\err_display_aux:w` in a way that it will pick up the arguments (if any) from the code and generates a suitable error message.

```

3619 \def_new:Npn \err_msgline_aux:NNnnn #1#2#3#4#5{
3620     \if_meaning:NN#1\l_err_label_token

```

At the moment we simply use the old L^AT_EX error code and `\renewcommand` to generate the error handler. After displaying the error message we insert error code this can be manipulated by the user with the deletion/insertion facility of T_EX's error mechanism.

The `\group_end:` at the very beginning matches the `\group_begin:` when the file starts.

```

3621     \cmd_declare:Nnn\err_display_aux:w [#2]{
3622         \group_end:
3623         \toks_gset:Nx\g_err_help_toks{#4}
3624         \io_put_term:x{LaTeX~error~\io_show_file_lineno:.\iow_newline:
3625             \text_put_sp:\text_put_four_sp: \text_put_sp:
3626             See~LaTeX~manual~for~explanation.\iow_newline:
3627             \text_put_sp:\text_put_four_sp: \text_put_sp:
3628             Type~\text_put_sp: H~<return>~\text_put_sp: for~
3629             immediate~help.}
3630         \err_message:x{#3}
3631         #5}
3632     \tex_endinput:D
3633     \fi:}

```

`\err_display_aux:w` We should make sure that this function is definable.

```

3634 \def_new:Npn \err_display_aux:w {}

```

19.6.6 Kernel specific functions

`\err_kernel_interrupt:Nw` `\err_kernel_interrupt:Nw` is just the abbreviation to read from the standard system error file.

```

3635 \def_new:Npn \err_kernel_interrupt:Nw {\err_interrupt:NNw \c_kernel_err_tlp}

```

`\err_kernel_interrupt_new:NNnnn` To ease the coding in case of system messages that should all go to one and the same error file (if!) we also have the following function.

```

3636 \def_new:Npn \err_kernel_interrupt_new:NNnnn {
3637     \err_interrupt_new:NNNnnn \c_kernel_err_tlp}

```

```

\err_kernel_info:n These variants are specific for the LATEX kernel.
\err_kernel_warn:n
\err_kernel_fatal:n 3638 \def_new:Npn \err_kernel_info:n #1 {
\err_kernel_info_noline:n 3639 \err_info:nn {LaTeX~Info:~#1}
\err_kernel_warn_noline:n 3640 {\text_put_four_sp:\text_put_four_sp:\text_put_four_sp:}
\err_kernel_fatal_noline:n 3641 }
3642 \def_new:Npn \err_kernel_warn:n #1 {
3643 \err_warn:nn {LaTeX~Warning:~#1}
3644 {\text_put_sp:\text_put_sp:\text_put_sp:
3645 \text_put_four_sp:\text_put_four_sp:\text_put_four_sp:}
3646 }
3647 \def_new:Npn \err_kernel_fatal:n #1 {
3648 \err_fatal:nn {LaTeX~Fatal:~#1}
3649 {\text_put_sp:
3650 \text_put_four_sp:\text_put_four_sp:\text_put_four_sp:}
3651 }
3652 \def_new:Npn \err_kernel_info_noline:n #1 {
3653 \err_info_noline:nn {LaTeX~Info:~#1}
3654 {\text_put_four_sp:\text_put_four_sp:\text_put_four_sp:}
3655 }
3656 \def_new:Npn \err_kernel_warn_noline:n #1 {
3657 \err_warn_noline:nn {LaTeX~Warning:~#1}
3658 {\text_put_sp:\text_put_sp:\text_put_sp:
3659 \text_put_four_sp:\text_put_four_sp:\text_put_four_sp:}
3660 }
3661 \def_new:Npn \err_kernel_fatal_noline:n #1 {
3662 \err_fatal_noline:nn {LaTeX~Fatal:~#1}
3663 {\text_put_sp:
3664 \text_put_four_sp:\text_put_four_sp:\text_put_four_sp:}
3665 }
3666 </initex | package>

```

At a later stage variants may be provided for what in L^AT_EX 2_ε used to be called document classes and packages.

`\c_kernel_err_tlp` Most error messages will go to the system error file; it's name is stored in `\c_kernel_err_tlp`.

```

3667 <initex>\err_file_new:Nn \c_kernel_err_tlp {ltxkernel.err}
3668 <package>\err_file_new:Nn \c_kernel_err_tlp {l3in2e.err}

```

The code below is a temporary implementation of a few of L^AT_EX 209 error messages with the new syntax. They are only included in the package as the L^AT_EX 3 kernel will certainly have it's own error message definitions that differ from L^AT_EX 2_ε's way of signalling errors. These primarily serve as an example on how to use this concept of dealing with errors.

First we declare a couple of helper macros that contain texts that are used frequently throughout L^AT_EX.

```

3669 <*package>
3670 \def:Npn\err_help_ignored: {
3671 Your~command~was~ignored.\iow_newline:
3672 Type \text_put_sp: I~<command>~<return>
3673 \text_put_sp: to~replace~it~with~another~command,\iow_newline:
3674 or~\text_put_sp: <return> \text_put_sp: to~continue~without~it.}

```

```

3675
3676 \def:Npn\err_help_textlost: {
3677   You've~lost~some~text.\text_put_sp: \err_help_return_or_X:}
3678
3679 \def:Npn\err_help_return_or_X: {
3680   Try~typing\text_put_sp: <return>
3681   \text_put_sp: to~proceed.\iow_newline:
3682   If~that~doesn't~work,~type
3683   \text_put_sp: X~<return>\text_put_sp: to~quit.}
3684
3685 \def:Npn\err_help_trouble: {
3686   You're~ in~ trouble~ here.
3687   \text_put_sp:\err_help_return_or_X:}

```

Below are the definitions of the complete messages

```

3688
3689 \err_kernel_interrupt_new:NNnnn\cs_free_p:N{1}
3690   {Command~name~'\tex_string:D#1'~already~used}
3691   {You~tried~to~define~a~command~which~already~has~
3692     a~meaning.\iow_newline:
3693     If~you~really~want~to~redefine~it~try~
3694     \token_to_string:N\cmd_declare:Nnn\text_put_sp:
3695     next~time.\iow_newline:
3696     For~this~run~I~will~ignore~your~definition.}
3697   {}
3698
3699 \err_kernel_interrupt_new:NNnnn\newline{0}
3700   {There's~no~line~here~to~end}
3701   {You~tried~to~end~a~line~at~a~place~where~I~thought~
3702     we~were~already~between~paragraphs.}
3703   {}
3704
3705 \err_kernel_interrupt_new:NNnnn\newcnt{0}
3706   {No~such~counter}
3707   {The~counter~name~mentioned~in~the~operation~is~not~
3708     known~to~me.\iow_newline:
3709     Check~the~spelling.}
3710   {}
3711
3712 \err_kernel_interrupt_new:NNnnn\nodocument{0}
3713   {Missing~\token_to_string:N\begin{document}}
3714   {\err_help_trouble:}
3715   {}
3716
3717 \err_kernel_interrupt_new:NNnnn\badmath{0}
3718   {Bad~math~environment~delimiter}
3719   {\err_help_ignored:}
3720   {}
3721
3722 \err_kernel_interrupt_new:NNnnn\toodeep{0}
3723   {Too~deeply~nested}
3724   {\err_help_trouble:}
3725   {}
3726

```

```

3727 \err_kernel_interrupt_new:NNnnn\badpoptabs{0}
3728     {\token_to_string:N\pushtabs \text_put_sp:
3729     and~\token_to_string:N\poptabs
3730     \text_put_sp: don't~match}
3731     {\err_help_trouble:}
3732     {}
3733
3734 \err_kernel_interrupt_new:NNnnn\badtab{0}
3735     {Undefined~tab~position}
3736     {\err_help_trouble:}
3737     {}
3738
3739 \err_kernel_interrupt_new:NNnnn\preamerr{}
3740     {\if_case:w #1~Illegal~character~or:
3741     Missing~@-exp~or: Missing~p-arg~fi:\text_put_sp:
3742     in~array~arg}
3743     {\err_help_trouble:}
3744     {}
3745
3746 \err_kernel_interrupt_new:NNnnn\badlinearg{}
3747     {Bad~\token_to_string:N\line
3748     \text_put_sp: or~\token_to_string:N\vector
3749     \text_put_sp: argument}
3750     {\err_help_textlost:}
3751     {}
3752
3753 \err_kernel_interrupt_new:NNnnn\parmoderr{0}
3754     {Not~in~outer~par~mode}
3755     {\err_help_textlost:}
3756     {}
3757
3758 \err_kernel_interrupt_new:NNnnn\fltovf{0}
3759     {Too~many~unprocessed~floats}
3760     {\err_help_textlost:}
3761     {}
3762
3763 \err_kernel_interrupt_new:NNnnn\badcrerr{0}
3764     {Bad~use~of~\token_to_string:N\}
3765     {\err_help_return_or_X:}
3766     {}
3767
3768 \err_kernel_interrupt_new:NNnnn\noitemerr{0}
3769     {Something's~wrong--perhaps~a~missing~
3770     \token_to_string:N\item}
3771     {\err_help_return_or_X:}
3772     {}
3773
3774 \err_kernel_interrupt_new:NNnnn\notprerr{0}
3775     {Can~be~used~only~in~preamble}
3776     {\err_help_ignored:}
3777     {}
3778
3779 \err_file_close:N\c_kernel_err_tlp
3780 <package>

```

Show token usage:

```
3781 \showmemory>
3782 \showMemUsage
3783 \showmemory>
```

20 Boxes

There are three kinds of box operations: horizontal mode denoted with prefix `\hbox_`, vertical mode with prefix `\vbox_`, and the generic operations working in both modes with prefix `\box_`.

20.1 Generic functions

<code>\box_new:N</code> <code>\box_new:c</code> <code>\box_new_l:N</code>	<code>\box_new:N</code> $\langle box \rangle$
---	---

Defines $\langle box \rangle$ to be a new variable of type `box`.

T_EXhackers note: `\box_new:N` is the equivalent of plain T_EX's `\newbox`. However, the internal register allocation is done differently.

<code>\if_hbox:N</code> <code>\if_vbox:N</code> <code>\if_box_empty:N</code>	<code>\if_hbox:N</code> $\langle box \rangle$ $\langle true\ code \rangle$ $\langle false\ code \rangle$ \fi : <code>\if_box_empty:N</code> $\langle box \rangle$ $\langle true\ code \rangle$ $\langle false\ code \rangle$ \fi :
--	---

`\if_hbox:N` and `\if_vbox:N` check if $\langle box \rangle$ is an horizontal or vertical box resp. `\if_box_empty:N` tests if $\langle box \rangle$ is empty (void) and executes `code` according to the test outcome.

T_EXhackers note: These are the T_EX primitives `\ifhbox`, `\ifvbox` and `\ifvoid`.

<code>\box_if_empty_p:N</code> <code>\box_if_empty_p:c</code> <code>\box_if_empty:NTF</code> <code>\box_if_empty:cTF</code> <code>\box_if_empty:NT</code> <code>\box_if_empty:cT</code> <code>\box_if_empty:NF</code> <code>\box_if_empty:cF</code>	<code>\box_if_empty:NTF</code> $\langle box \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
--	---

Tests if $\langle box \rangle$ is empty (void) and executes `code` according to the test outcome.

T_EXhackers note: `\box_if_empty:NTF` is the L^AT_EX3 function name for `\ifvoid`.

<code>\box_set_eq:NN</code>
<code>\box_set_eq:cN</code>
<code>\box_set_eq:Nc</code>
<code>\box_set_eq:cc</code>

`\box_set_eq:NN <box1> <box2>`

Sets $\langle box1 \rangle$ equal to $\langle box2 \rangle$. Note that this eradicates the contents of $\langle box2 \rangle$ afterwards.

<code>\box_gset_eq:NN</code>
<code>\box_gset_eq:cN</code>
<code>\box_gset_eq:Nc</code>
<code>\box_gset_eq:cc</code>

`\box_gset_eq:NN <box1> <box2>`

Globally sets $\langle box1 \rangle$ equal to $\langle box2 \rangle$.

<code>\box_set_to_last:N</code>
<code>\box_set_to_last:c</code>
<code>\box_gset_to_last:N</code>
<code>\box_gset_to_last:c</code>

`\box_set_to_last:N <box>`

Sets $\langle box \rangle$ equal to the previous box `\R_last_box` and removes `\R_last_box` from the current list (unless in outer vertical or math mode).

<code>\box_move_right:nn</code>
<code>\box_move_left:nn</code>
<code>\box_move_up:nn</code>
<code>\box_move_down:nn</code>

`\box_move_left:nn {<dimen>} {<box function>}`

Moves $\langle box \text{ function} \rangle$ $\langle dimen \rangle$ in the direction specified. $\langle box \text{ function} \rangle$ is either an operation on a box such as `\box_use:N` or a “raw” box specification like `\vbox:n{xyz}`.

<code>\box_clear:N</code>
<code>\box_clear:c</code>
<code>\box_gclear:N</code>
<code>\box_gclear:c</code>

`\box_clear:N <box>`

Clears $\langle box \rangle$ by setting it to the constant `\c_void_box`. `\box_gclear:N` does it globally.

<code>\box_use:N</code>
<code>\box_use:c</code>
<code>\box_use_clear:N</code>
<code>\box_use_clear:c</code>

`\box_use:N <box>`
`\box_use_clear:N <box>`

`\box_use:N` puts a copy of $\langle box \rangle$ on the current list while `\box_use_clear:N` puts the box on the current list and then eradicates the contents of it.

T_EXhackers note: `\box_use:N` and `\box_use_clear:N` are the T_EX primitives `\copy` and `\box` with new (descriptive) names.

<code>\box_ht:N</code>
<code>\box_ht:c</code>
<code>\box_dp:N</code>
<code>\box_dp:c</code>
<code>\box_wd:N</code>
<code>\box_wd:c</code>

`\box_ht:N <box>`

Returns the height, depth, and width of $\langle box \rangle$ for use in dimension settings.

T_EXhackers note: These are the T_EX primitives `\ht`, `\dp` and `\wd`.

<code>\box_show:N</code>
<code>\box_show:c</code>

`\box_show:N <box>`

Writes the contents of $\langle box \rangle$ to the log file.

T_EXhackers note: This is the T_EX primitive `\showbox`.

<code>\c_empty_box</code>
<code>\l_tmpa_box</code>
<code>\l_tmpb_box</code>

`\c_empty_box` is the constantly empty box. The others are scratch boxes.

<code>\R_last_box</code>

`\R_last_box` is a read-only box register. You can set other boxes to this box, which will then be removed from the current list.

20.2 Horizontal mode

<code>\hbox:n</code>

`\hbox:n {<contents>}`

Places a `hbox` of natural size.

<code>\hbox_set:Nn</code>
<code>\hbox_set:cn</code>
<code>\hbox_gset:Nn</code>
<code>\hbox_gset:cn</code>

`\hbox_set:Nn <box> {<contents>}`

Sets $\langle box \rangle$ to be a vertical mode box containing $\langle contents \rangle$. It has its natural size. `\hbox_gset:Nn` does it globally.

<code>\hbox_set_to_wd:Nnn</code>
<code>\hbox_set_to_wd:cnn</code>
<code>\hbox_gset_to_wd:Nnn</code>
<code>\hbox_gset_to_wd:cnn</code>

`\hbox_set_to_wd:Nnn <box> {<dimen>} {<contents>}`

Sets $\langle box \rangle$ to contain $\langle contents \rangle$ and have width $\langle dimen \rangle$. `\hbox_gset_to_wd:Nn` does it globally.

<code>\hbox_to_wd:nn</code>	<code>\hbox_to_wd:nn {<dimen>} <contents></code>
-----------------------------	--

Places a $\langle box \rangle$ of width $\langle dimen \rangle$ containing $\langle contents \rangle$.

<code>\hbox_set_inline_begin:N</code>	
<code>\hbox_set_inline_begin:c</code>	
<code>\hbox_set_inline_end:</code>	
<code>\hbox_gset_inline_begin:N</code>	
<code>\hbox_gset_inline_begin:c</code>	<code>\hbox_set_inline_begin:N <box> <contents></code>
<code>\hbox_gset_inline_end:</code>	<code>\hbox_set_inline_end:</code>

Sets $\langle box \rangle$ to contain $\langle contents \rangle$. This type is useful for use in environment definitions.

<code>\hbox_unpack:N</code>	
<code>\hbox_unpack_clear:N</code>	<code>\hbox_unpack:N <box></code>

`\hbox_unpack:N` unpacks the contents of the $\langle box \rangle$ register and `\hbox_unpack_clear:N` also clears the $\langle box \rangle$ after unpacking it.

T_EXhackers note: These are the T_EX primitives `\unhcopy` and `\unhbox`.

20.3 Vertical mode

<code>\vbox_set:Nn</code>	
<code>\vbox_set:cn</code>	
<code>\vbox_gset:Nn</code>	
<code>\vbox_gset:cn</code>	<code>\vbox_set:Nn <box> {<contents>}</code>

Sets $\langle box \rangle$ to be a vertical mode box containing $\langle contents \rangle$. It has it's natural size. `\vbox_gset:Nn` does it globally.

<code>\vbox_set_to_ht:Nnn</code>	
<code>\vbox_set_to_ht:cnn</code>	
<code>\vbox_gset_to_ht:Nnn</code>	
<code>\vbox_gset_to_ht:cnn</code>	
<code>\vbox_gset_to_ht:ccn</code>	<code>\vbox_set_to_ht:Nnn <box> {<dimen>} {<contents>}</code>

Sets $\langle box \rangle$ to contain $\langle contents \rangle$ and have total height $\langle dimen \rangle$. `\vbox_gset_to_ht:Nn` does it globally.

<code>\vbox_set_inline_begin:N</code>	
<code>\vbox_set_inline_end:</code>	
<code>\vbox_gset_inline_begin:N</code>	<code>\vbox_set_inline_begin:N <box> <contents></code>
<code>\vbox_gset_inline_end:</code>	<code>\vbox_set_inline_end:</code>

Sets $\langle box \rangle$ to contain $\langle contents \rangle$. This type is useful for use in environment definitions.

<code>\vbox_set_split_to_ht:NNn</code>	<code>\vbox_set_split_to_ht:NNn <box1> <box2> {<dimen>}</code>
--	--

Sets $\langle box1 \rangle$ to contain the top $\langle dimen \rangle$ part of $\langle box2 \rangle$.

T_EXhackers note: This is the T_EX primitive `\vsplit`.

<code>\vbox:n</code>	<code>\vbox:n {<contents>}</code>
----------------------	---

Places a `vbox` of natural size with baseline equal to the baseline of the last line in the box.

<code>\vbox_to_ht:nn</code>	<code>\vbox_to_ht:nn {<dimen>} <contents></code>
<code>\vbox_to_zero:n</code>	<code>\vbox_to_zero:n <contents></code>

Places a $\langle box \rangle$ of size $\langle dimen \rangle$ containing $\langle contents \rangle$.

<code>\vbox_unpack:N</code>	<code>\vbox_unpack:N <box></code>
<code>\vbox_unpack_clear:N</code>	

`\vbox_unpack:N` unpacks the contents of the $\langle box \rangle$ register and `\vbox_unpack_clear:N` also clears the $\langle box \rangle$ after unpacking it.

T_EXhackers note: These are the T_EX primitives `\unvcopy` and `\unvbox`.

20.4 The Implementation

Announce and ensure that the required packages are loaded.

```

3784 <package>\ProvidesExplPackage
3785 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
3786 <package&!check>\RequirePackage{l3prg,l3token}\par
3787 <package & check>\RequirePackage{l3chk}\par
3788 (*initex | package)

```

The code in this module is very straight forward so I'm not going to comment it very extensively.

20.4.1 Generic boxes

```

\box_new:N Defining a new <box> register.
\box_new_l:N
\box_new:c 3789 (*initex)
           3790 \alloc_setup_type:nnn {box} \c_zero \c_max_register_num

```

Now, remember that `\box255` has a special role in T_EX, it shouldn't be allocated...

```

3791 \seq_put_right:Nn \g_box_allocation_seq {255}
3792 \def_new:Npn \box_new:N #1 {\alloc_reg:NnNN g {box} \tex_mathchardef:D #1}
3793 \def_new:Npn \box_new_l:N #1 {\alloc_reg:NnNN l {box} \tex_mathchardef:D #1}
3794 </initex>

```

When we run on top of L^AT_EX, we just use its allocation mechanism.

```
3795 \package\let_new:NN \box_new:N \newbox
3796 \def_new:Npn \box_new:c {\exp_args:Nc \box_new:N}
```

\if_hbox:N The primitives for testing if a $\langle box \rangle$ is empty/void or which type of box it is.

\if_vbox:N

```
\if_box_empty:N 3797 \let_new:NN \if_hbox:N \tex_ifhbox:D
3798 \let_new:NN \if_vbox:N \tex_ifvbox:D
3799 \let_new:NN \if_box_empty:N \tex_ifvoid:D
```

\box_if_empty_p:N Testing if a $\langle box \rangle$ is empty/void.

\box_if_empty_p:c

```
\box_if_empty:N 3800 \def_new:Npn \box_if_empty_p:N #1{
\box_if_empty:N 3801 \if_box_empty:N #1 \c_true \else: \c_false \fi:}
\box_if_empty:cTF 3802 \def_new:Npn \box_if_empty_p:c {\exp_args:Nc \box_if_empty_p:N}
\box_if_empty:NT 3803 \def_test_function_new:npn {\box_if_empty:N}#1{\if_box_empty:N #1}
\box_if_empty:cT 3804 \def_new:Npn \box_if_empty:cTF {\exp_args:Nc \box_if_empty:N}#1{\if_box_empty:N #1}
\box_if_empty:NF 3805 \def_new:Npn \box_if_empty:cT {\exp_args:Nc \box_if_empty:N}#1{\if_box_empty:N #1}
\box_if_empty:cF 3806 \def_new:Npn \box_if_empty:cF {\exp_args:Nc \box_if_empty:N}#1{\if_box_empty:N #1}
```

\box_set_eq:NN Assigning the contents of a box to be another box. This clears the second box globally
\box_set_eq:cN (that's how T_EX does it).

\box_set_eq:Nc

```
\box_set_eq:cc 3807 \def_new:Npn \box_set_eq:NN #1#2 {\tex_setbox:D #1 \tex_box:D #2}
3808 \def_new:Npn \box_set_eq:cN {\exp_args:Nc \box_set_eq:NN}
3809 \def_new:Npn \box_set_eq:Nc {\exp_args:NNc \box_set_eq:NN}
3810 \def_new:Npn \box_set_eq:cc {\exp_args:Ncc \box_set_eq:NN}
```

\box_gset_eq:NN Global version of the above.

\box_gset_eq:cN

```
\box_gset_eq:Nc 3811 \def_new:Npn \box_gset_eq:NN {\pref_global:D\box_set_eq:NN}
\box_gset_eq:cc 3812 \def_new:Npn \box_gset_eq:cN {\exp_args:Nc \box_gset_eq:NN}
3813 \def_new:Npn \box_gset_eq:Nc {\exp_args:NNc \box_gset_eq:NN}
3814 \def_new:Npn \box_gset_eq:cc {\exp_args:Ncc \box_gset_eq:NN}
```

\R_last_box A different name for this read-only primitive.

```
3815 \let_new:NN \R_last_box \tex_lastbox:D
```

\box_set_to_last:N Set a box to the previous box.

\box_set_to_last:c

```
\box_gset_to_last:N 3816 \def_new:Npn \box_set_to_last:N #1{\tex_setbox:D#1\R_last_box}
\box_gset_to_last:c 3817 \def_new:Npn \box_set_to_last:c {\exp_args:Nc \box_set_to_last:N}
3818 \def_new:Npn \box_gset_to_last:N {\pref_global:D \box_set_to_last:N}
3819 \def_new:Npn \box_gset_to_last:c {\exp_args:Nc \box_gset_to_last:N}
```

\box_move_left:nn Move box material in different directions.

\box_move_right:nn

```
\box_move_up:nn 3820 \def_long_new:Npn \box_move_left:nn #1#2{\tex_movelleft:D\dim_eval:n{#1}{#2}}
\box_move_down:nn 3821 \def_long_new:Npn \box_move_right:nn #1#2{\tex_moveright:D\dim_eval:n{#1}{#2}}
3822 \def_long_new:Npn \box_move_up:nn #1#2{\tex_raise:D\dim_eval:n{#1}{#2}}
3823 \def_long_new:Npn \box_move_down:nn #1#2{\tex_lower:D\dim_eval:n{#1}{#2}}
```

```

\box_clear:N    Clear a  $\langle box \rangle$  register.
\box_clear:c
\box_gclear:N  3824 \def_new:Npn \box_clear:N #1{\box_set_eq:NN #1 \c_empty_box }
\box_gclear:c  3825 \def_new:Npn \box_clear:c {\exp_args:Nc \box_clear:N }
                3826 \def_new:Npn \box_gclear:N {\pref_global:D\box_clear:N}
                3827 \def_new:Npn \box_gclear:c {\exp_args:Nc \box_gclear:c }

\box_ht:N      Accessing the height, depth, and width of a  $\langle box \rangle$  register.
\box_ht:c
\box_dp:N      3828 \let_new:NN \box_ht:N \tex_ht:D
\box_dp:c      3829 \def_new:Npn \box_ht:c {\exp_args:Nc \box_ht:N}
\box_wd:n      3830 \let_new:NN \box_dp:N \tex_dp:D
\box_wd:c      3831 \def_new:Npn \box_dp:c {\exp_args:Nc \box_dp:N}
                3832 \let_new:NN \box_wd:N \tex_wd:D
                3833 \def_new:Npn \box_wd:c {\exp_args:Nc \box_wd:N}

\box_use_clear:N Using a  $\langle box \rangle$ . This is just TEX primitives with meaningful names.
\box_use_clear:c
\box_use:N      3834 \let_new:NN \box_use_clear:N \tex_box:D
\box_use:c      3835 \def_new:Npn \box_use_clear:c {\exp_args:Nc \box_use_clear:N}
                3836 \let_new:NN \box_use:N \tex_copy:D
                3837 \def_new:Npn \box_use:c {\exp_args:Nc \box_use:N}

\box_show:N     Show the contents of a box and write it into the log file.
\box_show:c
                3838 \let:NN \box_show:N \tex_showbox:D
                3839 \def_new:Npn \box_show:c {\exp_args:Nc \box_show:N}

\c_empty_box    We allocate some  $\langle box \rangle$  registers here (and borrow a few from LATEX).
\l_tmpa_box
\l_tmpb_box     3840 \package\let:NN \c_empty_box \voidb@x
                3841 \package\let_new:NN \l_tmpa_box \@temptboxa
                3842 \initex\box_new:N \c_empty_box
                3843 \initex\box_new:N \l_tmpa_box
                3844 \box_new:N \l_tmpb_box

```

20.4.2 Vertical boxes

```

\ vbox:n        Put a vertical box directly into the input stream.

                3845 \def_new:Npn \vbox:n {\tex_vbox:D \scan_stop:}

\vbox_set:Nn     Storing material in a vertical box with a natural height.
\vbox_set:cn
\vbox_gset:Nn    3846 \def_long_new:Npn \vbox_set:Nn #1#2 {\tex_setbox:D #1 \tex_vbox:D {#2}}
\vbox_gset:cn    3847 \def_new:Npn \vbox_set:cn {\exp_args:Nc \vbox_set:Nn}
                3848 \def_new:Npn \vbox_gset:Nn {\pref_global:D \vbox_set:Nn}
                3849 \def_new:Npn \vbox_gset:cn {\exp_args:Nc \vbox_gset:Nn}

\vbox_set_to_ht:Nnn Storing material in a vertical box with a specified height.
\vbox_set_to_ht:cn
\vbox_gset_to_ht:Nnn 3850 \def_long_new:Npn \vbox_set_to_ht:Nnn #1#2#3 {
\vbox_gset_to_ht:cn
\vbox_gset_to_ht:ccn

```

```

3851 \tex_setbox:D #1 \tex_vbox:D to #2 {#3}}
3852 \def_new:Npn \vbox_set_to_ht:cnn{\exp_args:Nc \vbox_set_to_ht:Nnn }
3853 \def_new:Npn \vbox_gset_to_ht:Nnn {\pref_global:D \vbox_set_to_ht:Nnn }
3854 \def_new:Npn \vbox_gset_to_ht:cnn{\exp_args:Nc \vbox_gset_to_ht:Nnn }
3855 \def_new:Npn \vbox_gset_to_ht:ccn {\exp_args:Ncc \vbox_gset_to_ht:Nnn}

```

`\vbox_set_inline_begin:N` Storing material in a vertical box. This type is useful in environment definitions.

```

\vbox_set_inline_end:
\vbox_gset_inline_begin:N 3856 \def_new:Npn \vbox_set_inline_begin:N #1 {
3857 \tex_setbox:D #1 \tex_vbox:D \c_group_begin_token }
\vbox_set_inline_end: 3858 \let_new:NN \vbox_set_inline_end: \c_group_end_token
3859 \def_new:Npn \vbox_gset_inline_begin:N {
3860 \pref_global:D \vbox_set_inline_begin:N }
3861 \let_new:NN \vbox_gset_inline_end: \c_group_end_token

```

`\vbox_to_ht:nn` Put a vertical box directly into the input stream.

```

\vbox_to_zero:n 3862 \def_long_new:Npn \vbox_to_ht:nn #1#2{\tex_vbox:D to \dim_eval:n{#1}{#2}}
3863 \def_long_new:Npn \vbox_to_zero:n #1 {\tex_vbox:D to \c_zero_dim {#1}}

```

`\vbox_set_split_to_ht:NNn` Splitting a vertical box in two.

```

3864 \def_new:Npn \vbox_set_split_to_ht:NNn #1#2#3{
3865 \tex_setbox:D #1 \tex_vsplit:D #2 to #3
3866 }

```

`\vbox_unpack:N` Unpacking a box and if requested also clear it.

```

\vbox_unpack:c 3867 \let_new:NN \vbox_unpack:N \tex_unvcopy:D
\vbox_unpack_clear:N 3868 \def_new:Npn \vbox_unpack:c {\exp_args:Nc \vbox_unpack:N}
\vbox_unpack_clear:c 3869 \let_new:NN \vbox_unpack_clear:N \tex_unvbox:D
3870 \def_new:Npn \vbox_unpack_clear:c {\exp_args:Nc \vbox_unpack_clear:N}

```

20.4.3 Horizontal boxes

`\hbox:n` Put a horizontal box directly into the input stream.

```

3871 \def_new:Npn \hbox:n {\tex_hbox:D \scan_stop:}

```

`\hbox_set:Nn` Assigning the contents of a box to be another box. This clears the second box globally
`\hbox_set:cn` (that's how $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ does it).

```

\vbox_gset:Nn 3872 \def_long_new:Npn \hbox_set:Nn #1#2 {\tex_setbox:D #1 \tex_hbox:D {#2}}
\vbox_gset:cn 3873 \def_new:Npn \hbox_set:cn {\exp_args:Nc \hbox_set:Nn}
3874 \def_new:Npn \hbox_gset:Nn {\pref_global:D \hbox_set:Nn}
3875 \def_new:Npn \hbox_gset:cn {\exp_args:Nc \hbox_gset:Nn}

```

`\hbox_set_to_wd:Nnn` Storing material in a horizontal box with a specified width.

```

\vbox_set_to_wd:cnn 3876 \def_long_new:Npn \hbox_set_to_wd:Nnn #1#2#3 {
\vbox_gset_to_wd:Nnn 3877 \tex_setbox:D #1 \tex_hbox:D to \dim_eval:n{#2} {#3}}
\vbox_gset_to_wd:cnn 3878 \def_new:Npn \hbox_set_to_wd:cnn{\exp_args:Nc \hbox_set_to_wd:Nnn }
3879 \def_new:Npn \hbox_gset_to_wd:Nnn {\pref_global:D \hbox_set_to_wd:Nnn }
3880 \def_new:Npn \hbox_gset_to_wd:cnn{\exp_args:Nc \hbox_gset_to_wd:Nnn }

```

```

\hbox_set_inline_begin:N Storing material in a horizontal box. This type is useful in environment definitions.
\hbox_set_inline_begin:c
\hbox_set_inline_end: 3881 \def_new:Npn \hbox_set_inline_begin:N #1 {
\hbox_gset_inline_begin:N 3882 \tex_setbox:D #1 \tex_hbox:D \c_group_begin_token }
\hbox_gset_inline_begin:c 3883 \def:Npn \hbox_set_inline_begin:c {\exp_args:Nc
\hbox_gset_inline_begin:c 3884 \hbox_set_inline_begin:N}
\hbox_set_inline_end: 3885 \let_new:NN \hbox_set_inline_end: \c_group_end_token
3886 \def_new:Npn \hbox_gset_inline_begin:N {
3887 \pref_global:D \hbox_set_inline_begin:N }
3888 \def:Npn \hbox_gset_inline_begin:c {\exp_args:Nc
3889 \hbox_gset_inline_begin:N }
3890 \let_new:NN \hbox_gset_inline_end: \c_group_end_token

\hbox_to_wd:nn Put a horizontal box directly into the input stream.
\hbox_to_zero:n
3891 \def_long_new:Npn \hbox_to_wd:nn #1#2 {\tex_hbox:D to #1 {#2}}
3892 \def_long_new:Npn \hbox_to_zero:n #1 {\tex_hbox:D to \c_zero_skip {#1}}

\hbox_unpack:N Unpacking a box and if requested also clear it.
\hbox_unpack:c
\hbox_unpack_clear:N 3893 \let_new:NN \hbox_unpack:N \tex_unhcopy:D
\hbox_unpack_clear:c 3894 \def_new:Npn \hbox_unpack:c {\exp_args:Nc \hbox_unpack:N}
\hbox_unpack_clear:c 3895 \let_new:NN \hbox_unpack_clear:N \tex_unhbox:D
3896 \def_new:Npn \hbox_unpack_clear:c {\exp_args:Nc \hbox_unpack_clear:N}

3897 </initex | package>
3898 <*showmemory>
3899 \showMemUsage
3900 </showmemory>

```

21 Control sequence functions extended ...

\cs_gen_sym:N	\cs_gen_sym:N <tlp>
\cs_ggen_sym:N	

These functions will generate a new control sequence name for use as a pointer, e.g. some tree structure like the LDB. The new unique name is returned locally in <tlp> for further use. The names are generated using the roman numeral representation of some special counters together with a prefix of \l* (local) or \g*(global).

\cs_record_name:N	\cs_record_name:N <cs>
-------------------	------------------------

Takes the <cs> and saves it in a special places for pre-compiling purposes on a file later on. All control sequences that are recorded with this function will be dumped by \cs_dump:. This function is internally automatically used to record all symbols generated by \cs_gen_sym:N and \cs_ggen_sym:N.

\cs_load_dump:n	\cs_load_dump:n { <file name> }
-----------------	---------------------------------

Loads and executes the file <file name> if found. Then scans further ignoring everything until finding \cs_dump: where normal execution continues. If <file name> is not found,

the name is saved and normal execution of all following code is done until `\cs_dump:` is scanned. Then all symbols marked for dumping are dumped into $\langle file\ name \rangle$.

`\cs_dump:` Dumps the symbols recorded by `\cs_record_name:N` in the file given by the argument in `\cs_load_dump:n`. Dumping means that for every $\langle cs \rangle$ recorded by `\cs_record_name:N` a line

`\def:Npn $\langle cs \rangle$ { $\langle current\ meaning\ of\ cs \rangle$ }`

is written to this file. This means that when loading the file the definitions of all these $\langle cs \rangle$'s are directly available.

21.1 Internal variables

`\g_gen_sym_num`
`\g_ggen_sym_num` Holds the number of the last generated symbol by `\cs_gen_sym:N` or `\cs_ggen_sym:N`.

`\g_cs_dump_seq` Sequence in which the symbols to be dumped are stored.

`\c_cs_dump_stream` Output stream used for writing out the definitions of the recorded $\langle tlp \rangle$.

21.2 The Implementation

We start by ensuring that the required packages are loaded.

```
3901 <package>\ProvidesExplPackage
3902 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
3903 <package>\RequirePackage{l3num}
3904 <package>\RequirePackage{l3io}
3905 <package>\RequirePackage{l3seq}
3906 <package>\RequirePackage{l3int}
```

It might speed up the processing of documents when certain parts of the document style file are ‘precompiled’ and stored in a separate file.

`\c_cs_dump_stream` We need to allocate an output stream in order to be able to write the precompiled code out. Stream number for the dump.

```
3907 <*initex | package>
3908 <*precompile>
3909 \iow_new:N\c_cs_dump_stream
```

`\g_cs_dump_name_tlp` This $\langle tlp \rangle$ is used to store the name of the file.

```
3910 \tlp_new:Nn\g_cs_dump_name_tlp{}
```

`\g_cs_dump_seq` While processing the documentstyle we build up a list of control sequence names to be dumped. For this purpose we use the `\g_cs_dump_seq` sequence.

```
3911 \seq_new:N\g_cs_dump_seq
```

`\cs_record_name:N` These functions mark a control sequence for dumping into a precompiled style.

`\cs_record_name:c` When the trace ‘module’ is included in the code we also write information about the control sequence into a `.dmp` file.

```
3912 \def_new:Npn\cs_record_name:N#1{
3913   (*trace)
3914   \seq_gput_left:Nn
3915     \g_cs_trace_seq#1
3916   (/trace)
3917   \seq_gput_left:Nn
3918     \g_cs_dump_seq#1}
3919 \def_new:Npn\cs_record_name:c{\exp_args:Nc\cs_record_name:N}
```

As you can see `\csdump` When a document style calls `\cs_dump:` it triggers this code to write all the precompilation information out to a file.

Before dumping, we write a message to the terminal informing the ‘user’ of this fact.

```
3920 \def_new:Npn\cs_dump:{
3921   \iow_expanded_term:n{Precompiling~style~into~(\g_cs_dump_name_tlp)}
3922   \iow_open:Nn\c_cs_dump_stream{\g_cs_dump_name_tlp}
```

The first thing we write on a ‘dump’ file is a command that allows us to use `*` in control sequences. We also need to be able to write to (and read from) the file internal control sequences, containing `_` and `..`.

```
3923 \iow_expanded:Nn\c_cs_dump_stream
3924 {\group_begin:
3925   \tex_catcode:D'\token_to_string:N*=11\scan_stop:
3926   \token_to_string:N\CodeStart
3927 }
3928 \seq_map_inline:Nn
3929   \g_cs_dump_seq
3930   {\tex_message:D{.}}
3931 \iow_expanded:Nn\c_cs_dump_stream
```

We use a direct `\gdef:Npn` to disable any type of local/global check on the pointers.

```
3932   {\exp_not:n{\gdef:Npn ##1}
3933     {\tlp_to_str:N##1}}
3934 }
```

We also need to remember the current values of the `\g_gen_sym_num` and `\g_ggen_sym_num` counters to allow further updates after a database was dumped.

```
3935 \iow_expanded:Nn \c_cs_dump_stream {\exp_not:n{\num_gset:Nn
3936   \g_gen_sym_num}
3937   {\num_use:N\g_gen_sym_num}^^J
3938 \exp_not:n{\num_gset:Nn \g_ggen_sym_num}
```



```

3939           {\num_use:N\g_ggen_sym_num}}
3940 \iow_expanded:Nn
3941 \c_cs_dump_stream
3942 {\group_end:}
3943 \iow_close:N\c_cs_dump_stream
3944 \tex_message:D{~finished}
3945 }
3946 </precompile>

```

`\cs_load_dump:n` A function to read a precompiled file into memory and skip until a `\cs_dump:` command is found. If no such file is found, processing continues and a subsequent `\cs_dump:` command will then create the dump file.

```

3947 \def_new:Npn\cs_load_dump:n#1{
3948 \file_not_found:nTF{#1.cmp}
3949 (*precompile)
3950 {\tlp_gset:Nn\g_cs_dump_name_tlp{#1.cmp}}
3951 </precompile>
3952 <-precompile> {\tex_errmessage:D{Cannot~ dump~ with~ this~ format}}
3953 {\input{#1.cmp}
3954 \let:NN\cs_dump:\fi:
3955 \if_false:}}

```

`\g_gen_sym_num` Two counters to make up new local or global *short* names in pointer structures like the
`\g_ggen_sym_num` LDB. We use a fake counters since operations with them are seldom.

```

3956 \num_new:N\g_gen_sym_num \num_gset:Nn\g_gen_sym_num{0}
3957 \num_new:N\g_ggen_sym_num \num_gset:Nn\g_ggen_sym_num{0}

```

`\cs_gen_sym:N` We need to be able to generate control sequences on the fly. They will exist of a prefix,
`\cs_ggen_sym:N` either `l*` or `g*`, followed by the value of the counter `\g_gen_sym_num` (`\g_ggen_sym_num`) in roman numeral representation. The generated control sequence is locally stored in the token that was passed in `#1`.

```

3958 \def_new:Npn\cs_gen_sym:N#1{
3959 \num_gincr:N\g_gen_sym_num
3960 \tlp_set:Nc#1{l*\tex_romannumeral:D\num_use:N\g_gen_sym_num}
3961 (*precompile)
3962 \exp_after:NN\cs_record_name:N#1
3963 </precompile>

```

We still want to define the initial value for the new symbol globally to make sure that during compilation something is written to the output file.

```

3964 \exp_after:NN\tlp_clear_new:N#1}

```

The global variant

```

3965 \def_new:Npn\cs_ggen_sym:N#1{
3966 \num_gincr:N\g_ggen_sym_num
3967 \tlp_set:Nc#1{g*\tex_romannumeral:D\num_use:N\g_ggen_sym_num}
3968 (*precompile)
3969 \exp_after:NN\cs_record_name:N#1
3970 </precompile>
3971 \exp_after:NN\tlp_clear_new:N#1}

```

`\g_cs_trace_seq` A sequence which holds the control sequence names that are to be dumped. They are stored together with their meaning.

ATTENTION: as we currently don't distribute allocation routines for primitive registers this code will have no effect!

```
3972 (*trace)
3973 \seq_new:N\g_cs_trace_seq
```

`\g_register_trace_seq` Sequence holding the register names to be dumped with their corresponding values.

ATTENTION: as we currently don't distribute allocation routines for primitive registers this code will have no effect!

```
3974 \seq_new:N\g_register_trace_seq
```

`\cs_record_meaning:N` Function marking a control sequence for dumping with meaning.

```
3975 \def:Npn\cs_record_meaning:N#1{
3976 \seq_gput_left:Nn
3977   \g_cs_trace_seq#1}
```

`\register_record_name:N` Function marking a register for dumping with value.

```
3978 \def:Npn\register_record_name:N#1{
3979 \seq_gput_left:Nn
3980   \g_register_trace_seq#1}
```

`\dumpLaTeXstate` The function `\dumpLaTeXstate` is used to write control sequences and registers, together with their meaning or value in the `.dmp` file. We write informational messages to the terminal during the dump.

ATTENTION: as we currently don't distribute allocation routines for primitive registers this part of the code will dump nothing unless `\register_record_name:N` is explicitly used.

```
3981 \def_new:Npn\dumpLaTeXstate#1{
3982   \iow_expanded_term:n{Dumping~commands~into~(#{1}.dmp)}
3983   \iow_open:Nn\c_cs_dump_stream{#{1}.dmp}
3984   \seq_map_inline:Nn
3985     \g_cs_trace_seq
3986     {\tex_message:D{.}
3987      \iow_expanded:Nn\c_cs_dump_stream
3988        {\token_to_string:N##1~
3989         \token_to_meaning:N##1}
3990      }
3991   \tex_message:D{~registers}
3992   \seq_map_inline:Nn
3993     \g_register_trace_seq
3994     {\tex_message:D{.}
3995      \iow_expanded:Nn\c_cs_dump_stream
3996        {\token_to_string:N##1
3997         \the_internal:D##1}
3998      }
```

```

3999 \tex_message:D{~finished}
4000 }
4001 </trace>
4002 </initex | package>

```

Show token usage:

```

4003 <*showmemory>
4004 \showMemUsage
4005 </showmemory>

```

22 Quarks

A special type of constants in L^AT_EX3 are ‘quarks’. These are control sequences that expand to themselves and should therefore NEVER be executed directly in the code. This would result in an endless loop!

They are meant to be used as delimiter is weird functions (for example as the stop token (i.e., `\q_stop`). They also permit the following ingenious trick: when you pick up a token in a temporary, and you want to know whether you have picked up a particular quark, all you have to do is compare the temporary to the quark using `\if_meaning:NN`. A set of special quark testing functions is set up below. All the quark testing functions are expandable although the ones testing only single tokens are much faster.

By convention all constants of type quark start out with `\q_`.

The documentation needs some updating.

22.1 Functions

<code>\quark_new:N</code>	<code>\quark_new:N</code> <i><quark></i>
---------------------------	--

Defines *<quark>* to be a new constant of type quark.

<code>\quark_if_no_value_p:n</code>	
<code>\quark_if_no_value:nTF</code>	
<code>\quark_if_no_value:nF</code>	
<code>\quark_if_no_value:nT</code>	
<code>\quark_if_no_value_p:N</code>	
<code>\quark_if_no_value:nTF</code>	<code>\quark_if_no_value:nTF {<token list>}</code>
<code>\quark_if_no_value:NTF</code>	<code>{<true code>}{<false code>}</code>
<code>\quark_if_no_value:NT</code>	<code>\quark_if_no_value:NTF <tlp></code>
<code>\quark_if_no_value:NF</code>	<code>{<true code>}{<false code>}</code>

This tests whether or not *<token list>* contains only the quark `\q_no_value`.

If *<token list>* to be tested is stored in a token list pointer use `\quark_if_no_value:NTF`, or `\quark_if_no_value:nF` or check the value directly with `\if_meaning:NN`. All those cases are faster then `\quark_if_no_value:nTF` so should be preferred.¹⁰

¹⁰Clarify semantic of the “n” case ... i think it is not implement according to what we originally intended /FMi

T_EXhackers note: But be aware of the fact that `\if_meaning:NN` can result in an overflow of T_EX’s parameter stack since it leaves the corresponding `\fi:` on the input until the whole replacement text is processed. It is therefore better in recursions to use `\quark_if_no_value:NTF` as it will remove the conditional prior to processing the T or F case and so allows tail-recursion.

<code>\quark_if_nil_p:N</code>	
<code>\quark_if_nil:NTF</code>	
<code>\quark_if_nil:NT</code>	<code>\quark_if_nil:NTF <token></code>
<code>\quark_if_nil:NF</code>	<code>{<true code>}{<false code>}</code>

This tests whether or not `<token>` is equal to the quark `\q_nil`.

This is a useful test for recursive loops which typically has `\q_nil` as an end marker.

<code>\quark_if_nil_p:n</code>	
<code>\quark_if_nil:nTF</code>	
<code>\quark_if_nil:nT</code>	
<code>\quark_if_nil:nF</code>	
<code>\quark_if_nil_p:o</code>	
<code>\quark_if_nil:oTF</code>	
<code>\quark_if_nil:oT</code>	<code>\quark_if_nil:nTF {<tokens>}</code>
<code>\quark_if_nil:oF</code>	<code>{<true code>}{<false code>}</code>

This tests whether or not `<tokens>` is equal to the quark `\q_nil`.

This is a useful test for recursive loops which typically has `\q_nil` as an end marker.

22.2 Recursion

This module provides a uniform interface to intercepting and terminating loops as when one is doing tail recursion. The building blocks follow below.

<code>\q_recursion_tail</code>	This quark is appended to the data structure in question and appears as a real element there. This means it gets any list separators around it.
--------------------------------	---

<code>\q_recursion_stop</code>	This quark is added <i>after</i> the data structure. Its purpose is to make it possible to terminate the recursion at any point easily.
--------------------------------	---

<code>\quark_if_recursion_tail_stop:N</code>	
<code>\quark_if_recursion_tail_stop:n</code>	<code>\quark_if_recursion_tail_stop:n {<list element>}</code>
<code>\quark_if_recursion_tail_stop:o</code>	<code>\quark_if_recursion_tail_stop:N <list element></code>

This tests whether or not `<list element>` is equal to `\q_recursion_tail` and then exits, i.e., it gobbles the remainder of the list up to and including `\q_recursion_stop` which *must* be present.

If `<list element>` is not under your complete control it is advisable to use the n. If you wish to use the N form you *must* ensure it is really a single token such as if you have

`\tlp_set:Nn \l_tmpa_tlp { <list element> }`

<code>\quark_if_recursion_tail_stop_do:Nn</code>	<code>\quark_if_recursion_tail_stop_do:nn</code>
<code>\quark_if_recursion_tail_stop_do:nn</code>	<code>{<list element>} { <post action> }</code>
<code>\quark_if_recursion_tail_stop_do:nn</code>	<code>\quark_if_recursion_tail_stop_do:Nn</code>
<code>\quark_if_recursion_tail_stop_do:nn</code>	<code><list element> { <post action> }</code>

Same as `\quark_if_recursion_tail_stop:N` except here the second argument is executed after the recursion has been terminated.

22.3 Constants

`\q_no_value` The canonical ‘missing value quark’ that is returned by certain functions to denote that a requested value is not found in the data structure.

`\q_stop` This constant is used as a a marker in parameter text. This allows a scanning function to find the end of some input string.

`\q_nil` This constant represent the nil pointer in pointer structures.

22.4 The Implementation

We start by ensuring that the required packages are loaded. We check for `l3expan` since this a basic package that is essential for use of any higher-level package.

```

4006 <package>\ProvidesExplPackage
4007 <package> {<filename>}{<filedate>}{<fileversion>}{<filedescription>}
4008 <package>\RequirePackage{l3expan}\par
4009 <*initex | package>

```

`\quark_new:N` Allocate a new quark.

```

4010 \def_new:Npn \quark_new:N #1{\tlp_new:Nn #1{#1}}

```

`\q_stop` `\q_stop` is often used as a marker in parameter text, `\q_no_value` is the canonical missing value, and `\q_nil` represents a nil pointer in some data structures.

```

\q_nil
4011 \quark_new:N \q_stop
4012 \quark_new:N \q_no_value
4013 \quark_new:N \q_nil

```

`\q_error` We need two additional quarks. `\q_error` delimits the end of the computation for purposes of error recovery. `\q_mark` is used in parameter text when we need a scanning boundary that is distinct from `\q_stop`.

```

4014 \quark_new:N\q_error
4015 \quark_new:N\q_mark

```

`\q_recursion_tail` Quarks for ending recursions. Only ever used there! `\q_recursion_tail` is appended to whatever list structure we are doing recursion on, meaning it is added as a proper list item with whatever list separator is in use. `\q_recursion_stop` is placed directly after the list.

```
4016 \quark_new:N\q_recursion_tail
4017 \quark_new:N\q_recursion_stop
```

`\quark_if_recursion_tail_stop:n` When doing recursions it is easy to spend a lot of time testing if we found the end marker.
`\quark_if_recursion_tail_stop:N` To avoid this, we use a recursion end marker every time we do this kind of task. Also, if
`\quark_if_recursion_tail_stop:o` the recursion end marker is found, we wrap things up and finish.

```
4018 \def_long_new:Npn \quark_if_recursion_tail_stop:n #1 {
4019   \exp_after:NN\if_meaning:NN
4020     \quark_if_recursion_tail_aux:w #1?q_nil\q_recursion_tail\q_recursion_tail
4021     \exp_after:NN \use_none_delimit_by_q_recursion_stop:w
4022   \fi:
4023 }
4024 \def_long_new:Npn \quark_if_recursion_tail_stop:N #1 {
4025   \if_meaning:NN#1\q_recursion_tail
4026   \exp_after:NN \use_none_delimit_by_q_recursion_stop:w
4027   \fi:
4028 }
4029 \def_new:Npn \quark_if_recursion_tail_stop:o{
4030   \exp_args:No\quark_if_recursion_tail_stop:n
4031 }
```

```
\if_recursion_tail_stop_do:nn
\if_recursion_tail_stop_do:Nn
\if_recursion_tail_stop_do:on 4032 \def_long_new:Npn \quark_if_recursion_tail_stop_do:nn #1#2 {
4033   \exp_after:NN\if_meaning:NN
4034     \quark_if_recursion_tail_aux:w #1?q_nil\q_recursion_tail\q_recursion_tail
4035     \exp_after:NN \use_arg_i_delimit_by_q_recursion_stop:nw
4036   \else:
4037     \exp_after:NN\use_none:n
4038   \fi:
4039   {#2}
4040 }
4041 \def_long_new:Npn \quark_if_recursion_tail_stop_do:Nn #1#2 {
4042   \if_meaning:NN #1\q_recursion_tail
4043   \exp_after:NN \use_arg_i_delimit_by_q_recursion_stop:nw
4044   \else:
4045     \exp_after:NN\use_none:n
4046   \fi:
4047   {#2}
4048 }
4049 \def_new:Npn \quark_if_recursion_tail_stop_do:on{
4050   \exp_args:No\quark_if_recursion_tail_stop_do:nn
4051 }
```

`\quark_if_recursion_tail_aux:w` Helper macros for picking up the first token of a list to see if it is `\q_recursion_tail`
`\delimit_by_q_recursion_stop:w` and to stop the recursion.
`\delimit_by_q_recursion_stop:nw`

```

4052 \def_long_new:Npn \quark_if_recursion_tail_aux:w
4053   #1#2\q_nil\q_recursion_tail{#1}
4054 \def_long_new:Npn\use_none_delimit_by_q_recursion_stop:w
4055   #1\q_recursion_stop {}
4056 \def_long_new:Npn\use_arg_i_delimit_by_q_recursion_stop:nw
4057   #1#2\q_recursion_stop {#1}

```

\quark_if_no_value_p:N Here we test if we found a special quark as the first argument.

```

\quark_if_no_value:NFTF
\quark_if_no_value:NFTF 4058 \def_long_test_function_new:npn {quark_if_no_value:N} #1 {

```

\quark_if_no_value:NFTF We better start with \q_no_value as the first argument since the whole thing may otherwise loop if #1 is wrongly given a string like aabc instead of a single token.¹¹

```

\quark_if_no_value:NFTF 4059   \if_meaning:NN\q_no_value#1}
\quark_if_no_value:NFTF 4060 \def_long_new:Npn \quark_if_no_value_p:N #1{
\quark_if_no_value:NFTF 4061   \if_meaning:NN \q_no_value #1 \c_true
4062   \else: \c_false \fi:
4063 }

```

We also provide an n type. If run under a sufficiently new pdf ϵ -TeX, it uses a built-in primitive for string comparisons, otherwise it uses the slower \str_if_eq_var_p:nf function. In the latter case it would be faster to use a temporary token list pointer but it would render the function non-expandable. Using the pdf ϵ -TeX primitive is the preferred approach. Note that we have to add a manual space token in the first part of the comparison, otherwise it is gobbled by \str_if_eq_var_p:nf. The reason for using this function instead of \str_if_eq_p:nn is that a sequence like $_ \backslash \text{q_no_value}$ will test equal to \q_no_value using the latter test function and unfortunately this example turned up in one application.

```

4064 \cs_if_really_free:cTF{pdf_strcmp:D}{
4065   \def_long_new:Npn \quark_if_no_value_p:n #1{
4066     \if:w \exp_args:No \str_if_eq_var_p:nf
4067     {package}      {\token_to_string:N\q_no_value\space}
4068     {initex}       {\token_to_string:N\q_no_value\text_put_sp:}
4069     {\tlist_to_str:n{#1}}
4070     \c_true
4071   \else:
4072     \c_false
4073   \fi:
4074 }
4075 }
4076 {
4077   \def_long_new:Npn \quark_if_no_value_p:n #1{
4078     \if_num:w
4079     \pdf_strcmp:D {\exp_not:N \q_no_value}{\exp_not:n{#1}}=\c_zero
4080     \c_true \else: \c_false \fi:
4081   }
4082 }
4083 \def_long_test_function_new:npn {quark_if_no_value:n} #1 {
4084   \if:w \quark_if_no_value_p:n{#1}}

```

¹¹It may still loop in special circumstances however!

We also define a version where the true and false code is ordered differently.

```

4085 \def_long:Npn \quark_if_no_value:nFT #1{
4086   \if:w \quark_if_no_value_p:n{#1}
4087     \exp_after:NN\use_arg_ii:nn
4088   \else:
4089     \exp_after:NN\use_arg_i:nn
4090   \fi:
4091 }

```

\quark_if_nil_p:N A function to check for the presence of \q_nil.

```

\quark_if_nil:NTF
\quark_if_nil:NT 4092 \def_long_new:Npn \quark_if_nil_p:N #1{
\quark_if_nil:NF 4093   \if_meaning:NN \q_nil #1 \c_true
4094   \else: \c_false \fi:
4095 }
4096 \def_long_test_function_new:npn {quark_if_nil:N}#1{
4097   \if_meaning:NN\q_nil#1}

```

\quark_if_nil_p:n A function to check for the presence of \q_nil.

```

\quark_if_nil:nTF
\quark_if_nil:nT 4098 \cs_if_really_free:cTF{pdf_strcmp:D}{
\quark_if_nil:nF 4099   \def_long_new:Npn \quark_if_nil_p:n #1{
4100     \if:w \exp_args:No \str_if_eq_var_p:nf
\quark_if_nil_p:o 4101 {package}      {\token_to_string:N\q_nil\space}
\quark_if_nil:oTF 4102 {initex}      {\token_to_string:N\q_nil\text_put_sp:}
\quark_if_nil:oT 4103   {\tlist_to_str:n{#1}}
\quark_if_nil:oF 4104   \c_true
4105   \else:
4106     \c_false
4107   \fi:
4108 }
4109 }
4110 {
4111   \def_long_new:Npn \quark_if_nil_p:n #1{
4112     \if_num:w
4113     \pdf_strcmp:D {\exp_not:N \q_nil}{\exp_not:n{#1}}=\c_zero
4114     \c_true \else: \c_false \fi:
4115   }
4116 }
4117 \def_long_test_function_new:npn {quark_if_nil:n} #1 {
4118   \if:w \quark_if_nil_p:n{#1}}
4119 \def_new:Npn \quark_if_nil_p:o{\exp_args:No\quark_if_nil_p:n}
4120 \def_new:Npn \quark_if_nil:oTF{\exp_args:No\quark_if_nil:nTF}
4121 \def_new:Npn \quark_if_nil:oT {\exp_args:No\quark_if_nil:nT}
4122 \def_new:Npn \quark_if_nil:oF {\exp_args:No\quark_if_nil:nF}

```

Show token usage:

```

4123 </initex | package>
4124 < *showmemory>
4125 \showMemUsage
4126 </showmemory>

```


23 Control structures

23.1 Choosing modes

<pre>\mode_if_vertical_p: \mode_if_vertical:TF \mode_if_vertical:T \mode_if_vertical:F</pre>	<pre>\mode_if_vertical:TF {\true code} {\false code}</pre>
--	--

Determines if T_EX is in vertical mode or not and executes either *⟨true code⟩* or *⟨false code⟩* accordingly.

<pre>\mode_if_horizontal_p: \mode_if_horizontal:TF \mode_if_horizontal:T \mode_if_horizontal:F</pre>	<pre>\mode_if_horizontal:TF {\true code} {\false code}</pre>
--	--

Determines if T_EX is in horizontal mode or not and executes either *⟨true code⟩* or *⟨false code⟩* accordingly.

<pre>\mode_if_inner_p: \mode_if_inner:TF \mode_if_inner:T \mode_if_inner:F</pre>	<pre>\mode_if_inner:TF {\true code} {\false code}</pre>
--	---

Determines if T_EX is in inner mode or not and executes either *⟨true code⟩* or *⟨false code⟩* accordingly.

<pre>\mode_if_math:TF \mode_if_math:T \mode_if_math:F</pre>	<pre>\mode_if_math:TF {\true code} {\false code}</pre>
---	--

Determines if T_EX is in math mode or not and executes either *⟨true code⟩* or *⟨false code⟩* accordingly.

T_EXhackers note: This version will choose the right branch even at the beginning of an alignment cell.

23.1.1 Alignment safe grouping and scanning

<pre>\scan_align_safe_stop:</pre>	<pre>\scan_align_safe_stop:</pre>
-----------------------------------	-----------------------------------

This function gets T_EX on the right track inside an alignment cell but without destroying any kerning.

<code>\group_align_safe_begin:</code> <code>\group_align_safe_end:</code>	<code>\group_align_safe_begin: <...> \group_align_safe_end:</code>
--	--

Encloses `<...>` inside a group but is safe inside an alignment cell. See the implementation of `\peek_token_generic:NNTF` for an application.

23.2 Producing n copies

There are often several different requirements for producing multiple copies of something. Sometimes one might want to produce a number of identical copies of a sequence of tokens whereas at other times the goal is to simulate a for loop as known from most real programming languages.

<code>\prg_replicate:nn</code>	<code>\prg_replicate:nn { <number> } { <arg> }</code>
--------------------------------	---

Creates `<number>` copies of `<arg>`. Expandable.

<code>\prg_stepwise_function:nnnN</code>	<code>\prg_stepwise_function:nnnN {<start>} {<step>} {<end>} <function></code>
--	--

This function performs `<action>` once for each step starting at `<start>` and ending once `<end>` is passed. `<function>` is placed directly in front of a brace group holding the current number so it should usually be a function taking one argument. The `\prg_stepwise_function:nnnN` function is expandable.

<code>\prg_stepwise_inline:nnnn</code>	<code>\prg_stepwise_inline:nnnn {<start>} {<step>} {<end>} {<action>}</code>
--	--

Same as `\prg_stepwise_function:nnnN` except here `<action>` is performed each time with `##1` as a placeholder for the number currently being tested. This function is not expandable and it is nestable.

<code>\prg_stepwise_variable:nnnNn</code>	<code>\prg_stepwise_variable:nnnn {<start>} {<step>} {<end>} <temp-var> {<action>}</code>
---	---

Same as `\prg_stepwise_inline:nnnn` except here the current value is stored in `<temp-var>` and the programmer can use it in `<action>`. This function is not expandable.

23.3 Conditionals and logical operations

L^AT_EX3 has two primary forms of conditional flow processing. The one type deals with the truth value of a test directly as in `\cs_free:NNTF` where you test if a control sequence was undefined and then execute either the `<true>` or `<false>` part depending on the result and after exiting the underlying `\if...\fi:` structure. The second type has to do with predicate functions like `\cs_free_p:N` which return either `\c_true` or `\c_false` to be used in testing with `\if:w`.

This section describes a boolean data type which is closely connected to both parts as sometimes you want to execute some code depending on the value of a switch (e.g., draft/final) and other times you perhaps want to use it as a predicate function in an `\if:w` test. Parsing `\iffalse` and `\iftrue` tokens can be quite tricky at times so the easiest is to simply let a boolean either be `\c_true` or `\c_false`. This also means we get the logical operations And, Or, and Not which can then be used on both the boolean type and predicate functions. All functions by the name `\predicate` are expandable and expect the input to also be fully expandable. More generic constructs do not contain `predicate` in their names.

23.3.1 The boolean data type

<code>\bool_new:N</code> <code>\bool_new:c</code>	<code>\bool_new:N <bool></code>
--	---------------------------------------

Define a new boolean variable. The initial value is `<false>`. A boolean is actually just either `\c_true` or `\c_false`.

<code>\bool_set_true:N</code> <code>\bool_set_true:c</code> <code>\bool_set_false:N</code> <code>\bool_set_false:c</code> <code>\bool_gset_true:N</code> <code>\bool_gset_true:c</code> <code>\bool_gset_false:N</code> <code>\bool_gset_false:c</code>	<code>\bool_gset_false:N <bool></code>
--	--

Set `<bool>` either true or false. We can also do this globally.

<code>\bool_set_eq:NN</code> <code>\bool_set_eq:Nc</code> <code>\bool_set_eq:cN</code> <code>\bool_set_eq:cc</code> <code>\bool_gset_eq:NN</code> <code>\bool_gset_eq:Nc</code> <code>\bool_gset_eq:cN</code> <code>\bool_gset_eq:cc</code>	<code>\bool_set_eq:NN <bool1> <bool2></code>
--	--

Set `<bool1>` equal to the value of `<bool2>`.

<code>\bool_if:NTF</code> <code>\bool_if:NT</code> <code>\bool_if:NF</code> <code>\bool_if_p:N</code>	<code>\bool_if:NTF <bool> {<true>} {<false>}</code> <code>\bool_if_p:N <bool></code>
--	---

Test the truth value of the boolean and execute the `<true>` or `<false>` code. `\bool_if_p:N` is a predicate function for use in `\if:w` tests.

\bool_whiledo:NT	
\bool_whiledo:NF	
\bool_dowhile:NT	\bool_whiledo:NT $\langle bool \rangle$ $\{\langle true \rangle\}$
\bool_dowhile:NF	\bool_whiledo:NF $\langle bool \rangle$ $\{\langle false \rangle\}$

The T versions execute the $\langle true \rangle$ code as long as the boolean is true and the F versions execute the $\langle false \rangle$ code as long as the boolean is false. The **whiledo** functions execute the body after testing the boolean and the **dowhile** functions executes the body first and then tests the boolean.

\l_tmpa_bool
\g_tmpa_bool

Reserved booleans.

23.3.2 Logical operations

Somewhat related to the subject of conditional flow processing is logical operators as these deal with $\langle true \rangle$ and $\langle false \rangle$ statements which is precisely what the predicate functions return.

\predicate_p:n	
\predicate:nTF	
\predicate:nT	\predicate:nTF $\{\langle list\ of\ predicates \rangle\}$ $\{\langle true \rangle\}$
\predicate:nF	$\{\langle false \rangle\}$

The functions evaluate the truth value of $\langle list\ of\ predicates \rangle$ where each predicate is separated by `&&` or `||` denoting logical And and Or functions. Minimal evaluation is carried out so that whenever a truth value cannot be changed anymore, the remaining tests are not carried out. Hence

```

\predicate_p:n{
  \int_compare_p:nNn 1=1 &&
  \predicate_p:n {
    \int_compare_p:nNn 2=3 ||
    \int_compare_p:nNn 4=4 ||
    \int_compare_p:nNn 1=\error % is skipped
  } &&
  \int_compare_p:nNn 2=2
}

```

returns $\langle true \rangle$.

\predicate_not_p:n	\predicate_not_p:n $\{\langle list\ of\ predicates \rangle\}$
--------------------	---

\predicate_not_p:n reverses the truth value of its argument. Thus

```
\prg_if_predicate_not_p:n {\prg_if_predicate_not_p:n {\c_true}}
```

ultimately returns $\langle true \rangle$.

23.3.3 Case switches

```

\prg_case_int:nnn {\langle integer expr \rangle} {
  {\langle integer expr_1 \rangle}{\langle code_1 \rangle}{\langle integer expr_2 \rangle}{\langle code_2 \rangle}
  ...{\langle integer expr_n \rangle}{\langle code_n \rangle}
\prg_case_int:nnn } {\langle else case \rangle}

```

This function evaluates the first $\langle integer\ expr \rangle$ and then compares it to the values found in the list. Thus the expression

```

\prg_case:nnn{2*5}{
  {5}{Small} {4+6}{Medium} {-2*10}{Negative}
}{Other}

```

evaluates first the term to look for and then tries to find this value in the list of values. If the value is found, the code on its right is executed after removing the remainder of the list. If the value is not found, the $\langle else\ case \rangle$ is executed. The example above will return “Medium”.

The function is expandable and is written in such a way that **f** style expansion can take place cleanly, i.e., no tokens from within the function are left over.

```

\prg_case_int:nnn {\langle dim expr \rangle} {
  {\langle dim expr_1 \rangle}{\langle code_1 \rangle}{\langle dim expr_2 \rangle}{\langle code_2 \rangle}
  ...{\langle dim expr_n \rangle}{\langle code_n \rangle}
\prg_case_dim:nnn } {\langle else case \rangle}

```

This function works just like `\prg_case_int:nnn` except it works for $\langle dim \rangle$ registers.

```

\prg_case_str:nnn {\langle string \rangle} {
  {\langle string_1 \rangle}{\langle code_1 \rangle}{\langle string_2 \rangle}{\langle code_2 \rangle}
  ...{\langle string_n \rangle}{\langle code_n \rangle}
\prg_case_str:nnn } {\langle else case \rangle}

```

This function works just like `\prg_case_int:nnn` except it compares strings. Each string is evaluated fully using **x** style expansion.

The function is expandable¹² and is written in such a way that **f** style expansion can take place cleanly, i.e., no tokens from within the function are left over.

23.3.4 Generic loops

```

\prg_whiledo:nT
\prg_whiledo:nF
\prg_dowhile:nT \prg_whiledo:nT {\langle test \rangle} {\langle true \rangle}
\prg_dowhile:nF \prg_whiledo:nF {\langle test \rangle} {\langle false \rangle}

```

The **T** versions execute the $\langle true \rangle$ code as long as $\langle test \rangle$ is true and the **F** versions execute the $\langle false \rangle$ code as long as $\langle test \rangle$ is false. The **whiledo** functions execute the body after testing the boolean and the **dowhile** functions executes the body first and then tests the boolean. For the **T** versions, $\langle test \rangle$ should end with a function executing only the $\langle true \rangle$ code for some test such as `\tlp_if_eq:NNT`. Similarly the **F** types should end with `\tlp_if_eq:NNF`.

¹²Provided you use pdfTeX v1.30 or later

23.4 Sorting

<code>\prg_quicksort:n</code>	<code>{ \{<element 1>\} \{<element 2>\}</code> <code>... \{<element n>\} }</code>
-------------------------------	--

Performs a Quicksort on the token list. The comparisons are performed by the function `\prg_quicksort_compare:nnTF` which is up to the programmer to define. When the sorting process is over, all elements are given as argument to the function `\prg_quicksort_function:n` which the programmer also controls.

<code>\prg_quicksort_function:n</code> <code>\prg_quicksort_compare:nnTF</code>	<code>\prg_quicksort_function:n \{<element>\}</code> <code>\prg_quicksort_compare:nnTF \{<element 1>\} \{<element 2>\}</code>
--	--

The two functions the programmer must define before calling `\prg_quicksort:n`. As an example we could define

```
\def:NNn\prg_quicksort_function:n 1{\#1}
\def:NNn\prg_quicksort_compare:nnTF 2{\num_compare:nNnTF{\#1}>{\#2}}
```

Then the function call

```
\prg_quicksort:n {876234520}
```

would return `{0}{2}{2}{3}{4}{5}{6}{7}{8}`. An alternative example where one sorts a list of words, `\prg_quicksort_compare:nnTF` could be defined as

```
\def:NNn\prg_quicksort_compare:nnTF 2{
  \num_compare:nNnTF{\tlist_compare:nn{\#1}{\#2}}>\c_zero }
```

23.5 The Implementation

We start by ensuring that the required packages are loaded.

```
4127 <*package>
4128 \ProvidesExplPackage
4129   {\filename}{\filedate}{\fileversion}{\filedescription}
4130 \RequirePackage{l3quark}
4131 \RequirePackage{l3toks}
4132 \RequirePackage{l3int}
4133 </package>
4134 <*initex | package>
```

23.5.1 Choosing modes

`\mode_if_vertical_p:` For testing vertical mode.

```
\mode_if_vertical:TF
\mode_if_vertical:T 4135 \def_new:Npn \mode_if_vertical_p: {
\mode_if_vertical:F 4136   \if_mode_vertical: \c_true \else: \c_false\fi:}
4137 \def_test_function_new:npn{mode_if_vertical:}{\if_mode_vertical:}
```

```

\mode_if_horizontal_p: For testing horizontal mode.
\mode_if_horizontal:TF
\mode_if_horizontal:T 4138 \def_new:Npn \mode_if_horizontal_p: {
\mode_if_horizontal:F 4139   \if_mode_horizontal: \c_true \else: \c_false\fi:}
4140 \def_test_function_new:npn{mode_if_horizontal:}{\if_mode_horizontal:}

\mode_if_inner_p: For testing inner mode.
\mode_if_inner:TF
\mode_if_inner:T 4141 \def_new:Npn \mode_if_inner_p: {
\mode_if_inner:F 4142   \if_mode_inner: \c_true \else: \c_false\fi:}
4143 \def_test_function_new:npn{mode_if_inner:}{\if_mode_inner:}

\mode_if_math:TF For testing math mode. Uses the kern-save \scan_align_safe_stop:.
\mode_if_math:T
\mode_if_math:F 4144 \def_test_function_new:npn{mode_if_math:} {
4145   \scan_align_safe_stop: \if_mode_math: }

```

Alignment safe grouping and scanning

`\group_align_safe_begin:` T_EX’s alignment structures present many problems. As Knuth says himself in *T_EX: The Program*: “It’s sort of a miracle whenever `\halign` or `\valign` work, [...]” One problem relates to commands that internally issues a `\cr` but also peek ahead for the next character for use in, say, an optional argument. If the next token happens to be a `&` with category code 4 we will get some sort of weird error message because the underlying `\tex_futurelet:D` will store the token at the end of the alignment template. This could be a `&4` giving a message like `! Misplaced \cr.` or even worse: it could be the `\endtemplate` token causing even more trouble! To solve this we have to open a special group so that T_EX still thinks it’s on safe ground but at the same time we don’t want to introduce any brace group that may find its way to the output. The following functions help with this by using code documented only in Appendix D of *The T_EXbook*...

```

4146 \def_new:Npn \group_align_safe_begin: {
4147   \if_false:{\fi:\if_num:w'=\c_zero\fi:}
4148 \def_new:Npn \group_align_safe_end:   {\if_num:w'=\c_zero}\fi:}

```

`\scan_align_safe_stop:` When T_EX is in the beginning of an align cell (right after the `\cr`) it is in a somewhat strange mode as it is looking ahead to find an `\tex_omit:D` or `\tex_noalign:D` and hasn’t looked at the preamble yet. Thus an `\tex_ifmmode:D` test will always fail unless we insert `\scan_stop:` to stop T_EX’s scanning ahead. On the other hand we don’t want to insert a `\scan_stop:` every time as that will destroy kerning between letters¹³ Unfortunately there is no way to detect if we’re in the beginning of an alignment cell as they have different characteristics depending on column number etc. However we *can* detect if we’re in an alignment cell by checking the current group type and we can also check if the previous node was a character or ligature. What is done here is that `\scan_stop:` is only inserted iff a) we’re in the outer part of an alignment cell and b) the last node *wasn’t* a char node or a ligature node.

```

4149 \def_new:Npn \scan_align_safe_stop: {
4150   \num_compare:nNnT \etex_currentgroup\type:D = \c_six

```

¹³Unless we enforce an extra pass with an appropriate value of `\pretolerance`.

```

4151 {
4152   \num_compare:nNnF \etex_lastnodetype:D = \c_zero
4153   {
4154     \num_compare:nNnF \etex_lastnodetype:D = \c_seven
4155     \scan_stop:
4156   }
4157 }
4158 }

```

23.5.2 Making n copies

`\prg_replicate:nn` This function uses a cascading csname technique by David Kastrup (who else :-)

`\prg_replicate_aux:N` The idea is to make the input 25 result in first adding five, and then 20 copies of the

`\prg_replicate_first_aux:N` code to be replicated. The technique uses cascading csnames which means that we start building several csnames so we end up with a list of functions to be called in reverse order. This is important here (and other places) because it means that we can for instance make the function that inserts five copies of something to also hand down ten to the next function in line. This is exactly what happens here: in the example with 25 then the next function is the one that inserts two copies but it sees the ten copies handed down by the previous function. In order to avoid the last function to insert say, 100 copies of the original argument just to gobble them again we define separate functions to be inserted first. Finally we must ensure that the cascade comes to a peaceful end so we make it so that the original csname `\TeX` is creating is simply `\use_noop:` expanding to nothing.

This function has one flaw though: Since it constantly passes down ten copies of its previous argument it will severely affect the main memory once you start demanding hundreds of thousands of copies. Now I don't think this is a real limitation for any ordinary use. An alternative approach is to create a string of `m`'s with `\int_to_roman:w` which can be done with just four macros but that method has its own problems since it can exhaust the string pool. Also, it is considerably slower than what we use here so the few extra csnames are well spent I would say.

```

4159 \def_new:Npn \prg_replicate:nn #1{
4160   \cs:w use_noop:
4161   \exp_after:NN\prg_replicate_first_aux:N
4162   \int_use:N \int_eval:n{#1} \cs_end:
4163   \cs_end:
4164 }
4165 \def_new:Npn \prg_replicate_aux:N#1{
4166   \cs:w prg_replicate_#1:n\prg_replicate_aux:N
4167 }
4168 \def_new:Npn \prg_replicate_first_aux:N#1{
4169   \cs:w prg_replicate_first_#1:n\prg_replicate_aux:N
4170 }

```

Then comes all the functions that do the hard work of inserting all the copies.

```

4171 \def_new:Npn \prg_replicate_ :n #1{}% no, this is not a typo!
4172 \def_long_new:cpn {prg_replicate_0:n}#1{\cs_end:{#1#1#1#1#1#1#1#1#1#1}}
4173 \def_long_new:cpn {prg_replicate_1:n}#1{\cs_end:{#1#1#1#1#1#1#1#1#1#1}#1}

```



```

4174 \def_long_new:cpn {prg_replicate_2:n}#1{\cs_end:{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}
4175 \def_long_new:cpn {prg_replicate_3:n}#1{
4176   \cs_end:{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}
4177 \def_long_new:cpn {prg_replicate_4:n}#1{
4178   \cs_end:{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}
4179 \def_long_new:cpn {prg_replicate_5:n}#1{
4180   \cs_end:{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}
4181 \def_long_new:cpn {prg_replicate_6:n}#1{
4182   \cs_end:{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}
4183 \def_long_new:cpn {prg_replicate_7:n}#1{
4184   \cs_end:{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}
4185 \def_long_new:cpn {prg_replicate_8:n}#1{
4186   \cs_end:{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}
4187 \def_long_new:cpn {prg_replicate_9:n}#1{
4188   \cs_end:{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}

```

Users shouldn't ask for something to be replicated once or even not at all but...

```

4189 \def_long_new:cpn {prg_replicate_first_0:n}#1{\cs_end: }
4190 \def_long_new:cpn {prg_replicate_first_1:n}#1{\cs_end: #1}
4191 \def_long_new:cpn {prg_replicate_first_2:n}#1{\cs_end: #1#1}
4192 \def_long_new:cpn {prg_replicate_first_3:n}#1{\cs_end: #1#1#1}
4193 \def_long_new:cpn {prg_replicate_first_4:n}#1{\cs_end: #1#1#1#1}
4194 \def_long_new:cpn {prg_replicate_first_5:n}#1{\cs_end: #1#1#1#1#1}
4195 \def_long_new:cpn {prg_replicate_first_6:n}#1{\cs_end: #1#1#1#1#1#1}
4196 \def_long_new:cpn {prg_replicate_first_7:n}#1{\cs_end: #1#1#1#1#1#1#1}
4197 \def_long_new:cpn {prg_replicate_first_8:n}#1{\cs_end: #1#1#1#1#1#1#1#1}
4198 \def_long_new:cpn {prg_replicate_first_9:n}#1{\cs_end: #1#1#1#1#1#1#1#1#1}

```

\prg_stepwise_function:nnnN A stepwise function. Firstly we check the direction of the steps #2 since that will depend on which test we should use. If the step is positive we use a greater than test, otherwise g_stepwise_function_incr:nnnN a less than test. If the test comes out true exit, otherwise perform #4, add the step to g_stepwise_function_decr:nnnN #1 and try again with this new value of #1.

```

4199 \def_long_new:NNn \prg_stepwise_function:nnnN 2{
4200   \num_compare:nNnTF{#2}<\c_zero
4201     {\exp_args:No \prg_stepwise_function_decr:nnnN }
4202     {\exp_args:No \prg_stepwise_function_incr:nnnN }
4203     {\int_use:N\int_eval:n{#1}}{#2}
4204 }
4205 \def_long_new:NNn \prg_stepwise_function_incr:nnnN 4{
4206   \num_compare:nNnF {#1}>{#3}
4207   {
4208     #4{#1}
4209     \exp_args:No \prg_stepwise_function_incr:nnnN
4210     {\int_use:N\int_eval:n{#1 + #2}}
4211     {#2}{#3}{#4}
4212   }
4213 }
4214 \def_long_new:NNn \prg_stepwise_function_decr:nnnN 4{
4215   \num_compare:nNnF {#1}<{#3}
4216   {
4217     #4{#1}
4218     \exp_args:No \prg_stepwise_function_decr:nnnN

```

```

4219     {\int_use:N\int_eval:n{#1 + #2}}
4220     {#2}{#3}{#4}
4221   }
4222 }

```

\g_prg_inline_level_int This function uses the same approach as for instance \clist_map_inline:Nn to allow arbitrary nesting. First construct the special function and then call an auxiliary one which just carries the newly constructed csname. Must make assignments global when we maintain our own stack.

```

4223 \int_new:N\g_prg_inline_level_int
4224 \def_long_new:NNn\prg_stepwise_inline:nnnn 4{
4225   \int_gincr:N \g_prg_inline_level_int
4226   \gdef:cpn{\prg_stepwise_inline\int_use:N\g_prg_inline_level_int :n}##1{#4}
4227   \num_compare:nNnTF {#2}<\c_zero
4228   {\exp_args:Nco \prg_stepwise_inline_decr:Nnnn }
4229   {\exp_args:Nco \prg_stepwise_inline_incr:Nnnn }
4230   {\prg_stepwise_inline\int_use:N\g_prg_inline_level_int :n}
4231   {\int_use:N\int_eval:n{#1}} {#2} {#3}
4232   \int_gdecr:N \g_prg_inline_level_int
4233 }
4234 \def_long_new:NNn \prg_stepwise_inline_incr:Nnnn 4{
4235   \num_compare:nNnF {#2}>{#4}
4236   {
4237     #1{#2}
4238     \exp_args:NNo \prg_stepwise_inline_incr:Nnnn #1
4239     {\int_use:N\int_eval:n{#2 + #3}} {#3}{#4}
4240   }
4241 }
4242 \def_long_new:NNn \prg_stepwise_inline_decr:Nnnn 4{
4243   \num_compare:nNnF {#2}<{#4}
4244   {
4245     #1{#2}
4246     \exp_args:NNo \prg_stepwise_inline_decr:Nnnn #1
4247     {\int_use:N\int_eval:n{#2 + #3}} {#3}{#4}
4248   }
4249 }

```

\prg_stepwise_variable:nnnNn Almost the same as above. Just store the value in #4 and execute #5.

```

\prg_stepwise_variable_decr:nnnNn
\prg_stepwise_variable_incr:nnnNn
4250 \def_long_new:NNn \prg_stepwise_variable:nnnNn 2 {
4251   \num_compare:nNnTF {#2}<\c_zero
4252   {\exp_args:NNo\prg_stepwise_variable_decr:nnnNn}
4253   {\exp_args:NNo\prg_stepwise_variable_incr:nnnNn}
4254   {\int_use:N\int_eval:n{#1}}{#2}
4255 }
4256 \def_long_new:NNn \prg_stepwise_variable_incr:nnnNn 5 {
4257   \num_compare:nNnF {#1}>{#3}
4258   {
4259     \def:Npn #4{#1} #5
4260     \exp_args:NNo \prg_stepwise_variable_incr:nnnNn
4261     {\int_use:N\int_eval:n{#1 + #2}}{#2}{#3}{#4}{#5}
4262   }
4263 }

```

```

4264 \def_long_new:NNn \prg_stepwise_variable_decr:nnnNn 5 {
4265   \num_compare:nNnF {#1}<{#3}
4266   {
4267     \def:Npn #4{#1} #5
4268     \exp_args:No \prg_stepwise_variable_decr:nnnNn
4269     {\int_use:N\int_eval:n{#1 + #2}}{#2}{#3}#4{#5}
4270   }
4271 }

```

23.5.3 Booleans

For normal booleans we set them to either `\c_true` or `\c_false` and then use `\if:w` to choose the right branch. The functions return either the TF, T, or F case *after* ending the `\if:w`. We only define the N versions here as the c versions can easily be constructed with the expansion module.

```

\bool_new:N Defining and setting a boolean is easy.
\bool_new:c
\bool_set_true:N 4272 \def_new:Npn \bool_new:N #1 { \let_new:NN #1 \c_false }
\bool_set_true:c 4273 \def_new:Npn \bool_new:c #1 { \let_new:cN {#1} \c_false }
\bool_set_false:N 4274 \def_new:Npn \bool_set_true:N #1 { \let:NN #1 \c_true }
\bool_set_false:c 4275 \def_new:Npn \bool_set_true:c #1 { \let:cN {#1} \c_true }
\bool_set_false:N 4276 \def_new:Npn \bool_set_false:N #1 { \let:NN #1 \c_false }
\bool_set_false:c 4277 \def_new:Npn \bool_set_false:c #1 { \let:cN {#1} \c_false }
\bool_gset_true:N 4278 \def_new:Npn \bool_gset_true:N #1 { \glet:NN #1 \c_true }
\bool_gset_true:c 4279 \def_new:Npn \bool_gset_true:c #1 { \glet:cN {#1} \c_true }
\bool_gset_false:N 4280 \def_new:Npn \bool_gset_false:N #1 { \glet:NN #1 \c_false }
\bool_gset_false:c 4281 \def_new:Npn \bool_gset_false:c #1 { \glet:cN {#1} \c_false }

\bool_set_eq:NN Setting a boolean to another is also pretty easy.
\bool_set_eq:Nc
\bool_set_eq:cN 4282 \let_new:NN \bool_set_eq:NN \let:NN
\bool_set_eq:cc 4283 \let_new:NN \bool_set_eq:Nc \let:Nc
\bool_gset_eq:NN 4284 \let_new:NN \bool_set_eq:cN \let:cN
\bool_gset_eq:Nc 4285 \let_new:NN \bool_gset_eq:NN \glet:NN
\bool_gset_eq:cN 4286 \let_new:NN \bool_gset_eq:Nc \glet:Nc
\bool_gset_eq:cc 4287 \let_new:NN \bool_gset_eq:cN \glet:cN
4288 \let_new:NN \bool_gset_eq:cc \glet:cc
4289 \let_new:NN \bool_gset_eq:cc \glet:cc

\l_tmpa_bool A few booleans just if you need them.
\g_tmpa_bool
4290 \bool_new:N \l_tmpa_bool
4291 \bool_new:N \g_tmpa_bool

\bool_if:NTF Straight forward here.
\bool_if:NT
\bool_if:NF 4292 \def_test_function_new:npn{bool_if:N}#1{\if:w #1}
\bool_if:cTF 4293 \def_new:Npn \bool_if:cTF{\exp_args:Nc\bool_if:NTF}
\bool_if:cT 4294 \def_new:Npn \bool_if:cT{\exp_args:Nc\bool_if:NT}
\bool_if:cF 4295 \def_new:Npn \bool_if:cF{\exp_args:Nc\bool_if:NF}

```

\bool_if_p:N We also make a predicate function for the bool data type but since we use \c_true and
 \bool_if_p:c \c_false it's rather simple... Not that there's anything wrong in simplicity – on the
 contrary!

```
4296 \def_new:Npn \bool_if_p:N #1 { #1 }
4297 \let_new:NN \bool_if_p:c \cs_use:c
```

\bool_whiledo:NT A while loop where the boolean is tested before executing the statement. The NT version
 \bool_whiledo:cT executes the T part as long as the boolean is true while the NF version executes the F
 \bool_whiledo:NF part as long as the boolean is false.

```
\bool_whiledo:cF
4298 \def_long_new:Npn \bool_whiledo:NT #1 #2 {
4299   \bool_if:NT #1 {#2 \bool_whiledo:NT #1 {#2}}
4300 }
4301 \def_new:Npn \bool_whiledo:cT{\exp_args:Nc\bool_whiledo:NT}
4302 \def_long_new:Npn \bool_whiledo:NF #1 #2 {
4303   \bool_if:NF #1 {#2 \bool_whiledo:NF #1 {#2}}
4304 }
4305 \def_new:Npn \bool_whiledo:cF{\exp_args:Nc\bool_whiledo:NF}
```

\bool_dowhile:NT A do-while loop where the body is performed at least once and the boolean is tested
 \bool_dowhile:cT after executing the body. Otherwise identical to the above functions.

```
\bool_dowhile:NF
4306 \def_long_new:Npn \bool_dowhile:NT #1 #2 {
4307   #2 \bool_if:NT #1 {\bool_dowhile:NT #1 {#2}}
4308 }
4309 \def_new:Npn \bool_dowhile:cT{\exp_args:Nc\bool_dowhile:NT}
4310 \def_long_new:Npn \bool_dowhile:NF #1 #2 {
4311   #2 \bool_if:NF #1 {\bool_dowhile:NF #1 {#2}}
4312 }
4313 \def_new:Npn \bool_dowhiledo:cF{\exp_args:Nc\bool_dowhile:cF}
```

\bool_double_if:NNnnnn Execute #3 iff TT, #4 iff TF, #5 iff FT and #6 iff FF. The name isn't that great but I'll
 \bool_double_if:cNnnnnn have to think about that. Ideally it should be something with TF since only one of the
 \bool_double_if:Ncnnnn cases is executed but we haven't got any naming scheme for this kind of thing so for now
 \bool_double_if:ccnnnn I'll just stick to simple nnnn.

```
4314 \def_new:Npn \bool_double_if:NNnnnn#1#2{
4315   \if_case:w \num_eval:w #1\scan_stop:
4316     \if_case:w \num_eval:w #2\scan_stop:
4317       \exp_after:NN\exp_after:NN\exp_after:NN \use_arg_i:nnnn
4318     \else:
4319       \exp_after:NN\exp_after:NN\exp_after:NN \use_arg_ii:nnnn
4320     \fi:
4321   \else:
4322     \if_case:w \num_eval:w #2\scan_stop:
4323       \exp_after:NN\exp_after:NN\exp_after:NN \use_arg_iii:nnnn
4324     \else:
4325       \exp_after:NN\exp_after:NN\exp_after:NN \use_arg_iv:nnnn
4326     \fi:
4327   \fi:
4328 }
4329 \def_new:Npn \bool_double_if:cNnnnn{\exp_args:Nc\bool_double_if:NNnnnn}
4330 \def_new:Npn \bool_double_if:Ncnnnn{\exp_args:NNc\bool_double_if:NNnnnn}
4331 \def_new:Npn \bool_double_if:ccnnnn{\exp_args:Ncc\bool_double_if:NNnnnn}
```

23.5.4 Generic testing

`\prg_whiledo:nT` We provide these four generic while loops. `#1` is a test function and for the T functions it should be a test function ending with just the true case. Similar for the F types.

```
\prg_whiledo:nF
\prg_dowhile:nT
\prg_dowhile:nF
4332 \def_long_new:Npn \prg_whiledo:nT #1#2{
4333   #1 {#2 \prg_whiledo:nT {#1}{#2}}
4334 }
4335 \def_long_new:Npn \prg_whiledo:nF #1#2{
4336   #1 {#2 \prg_whiledo:nF {#1}{#2}}
4337 }
4338 \def_long_new:Npn \prg_dowhile:nT #1#2{
4339   #2 #1 {\prg_dowhile:nT {#1}{#2}}
4340 }
4341 \def_long_new:Npn \prg_dowhile:nF #1#2{
4342   #2 #1 {\prg_dowhile:nF {#1}{#2}}
4343 }
```

`\predicate_p:n` Evaluating the truth value of a list of predicates is done using an input syntax somewhat similar to the one found in other programming languages. The function evaluates predicates from left to right, expanding them to 00 and 01 resp., which leads to six different situations of tokens in the input stream:

```
\predicate_auxi:NN
\predicate_auxii:NNN
\predicate_88_0:w
\predicate_88_1:w
\predicate_II_0:w
\predicate_II_1:w
\predicate_02_0:w
\predicate_02_1:w
00&& Current truth value is true, logical And seen, continue to see if next is also true.
01&& Current truth value is false, logical And seen, break the scanning and return <false>.
00|| Current truth value is true, logical Or seen, break the scanning and return <true>.
01|| Current truth value is false, logical Or seen, continue to see if a later predicate is true.
0002 Current truth value is true, end marker seen, return <true>.
0102 Current truth value is false, end marker seen, return <false>.
```

To accomplish this we pre-expand the predicate list using `f` type expansion which leads to 00 or 01, possibly with a sequence of unfinished `\else:` `\c_false` `\fi:` or similar after it, which we remove using the same trick. We also carry over the truth value of the evaluated predicate. The expansion stops when it sees the end marker or `&&` or `||` (assuming these are not active characters at the programming level).

```
4344 \def_long_new:Npn \predicate_p:n #1{
4345   \group_align_safe_begin:
4346   \exp_after:NN \predicate_auxi:NN
4347   \int_to_roman:w-'\q #1 02\scan_stop:
4348 }
4349 \def_long_test_function_new:npn {predicate:n}#1{
4350   \group_align_safe_begin:
4351   \if:w \exp_after:NN \predicate_auxi:NN
4352   \int_to_roman:w-'\q #1 02\scan_stop:
4353 }
4354 \def_new:Npn \predicate_auxi:NN 0 #1{
4355   \exp_after:NN \predicate_auxii:NNN \exp_after:NN #1
4356   \int_to_roman:w-'\q
4357 }
```

After removing trailing conditionals we call a macro for the case we are in (see list above).

```

4358 \def_new:Npn \predicate_auxii:NNN #1#2#3{
4359   \cs_use:c{predicate_#2#3_#1:w} }
4360 \def_new:cpn{predicate_&&_0:w}{
4361   \exp_after:NN \predicate_auxi:NN\int_to_roman:w-'\q
4362 }
4363 \def_long_new:cpn{predicate_&&_1:w} #1 02\scan_stop:{
4364   \group_align_safe_end: 01}
4365 \def_long_new:cpn{predicate_||_0:w} #1 02\scan_stop:{
4366   \group_align_safe_end: 00}
4367 \def_new:cpn{predicate_||_1:w}{
4368   \exp_after:NN \predicate_auxi:NN\int_to_roman:w-'\q
4369 }
4370 \def_new:cpn{predicate_02_0:w}\scan_stop:{ \group_align_safe_end: 00 }
4371 \def_new:cpn{predicate_02_1:w}\scan_stop:{ \group_align_safe_end: 01 }

```

`\predicate_not_p:n` The not variant just reverses the outcome of `\predicate_p:n`.

```

4372 \def_long_new:Npn \predicate_not_p:n #1{
4373   \if:w \predicate_p:n{#1} \c_false \else: \c_true \fi:
4374 }

```

23.5.5 Case switch

`\prg_case_int:nnn` This case switch is in reality quite simple. It takes three arguments:
`\prg_case_int_aux:nnn`

1. An integer expression you wish to find.
2. A list of pairs of $\{ \langle integer\ expr \rangle \} \{ \langle code \rangle \}$. The list can be as long as is desired and $\langle integer\ expr \rangle$ can be negative.
3. The code to be executed if the value wasn't found.

We don't need the else case here yet, so leave it dangling in the input stream.

```

4375 \def_long:Npn \prg_case_int:nnn #1 #2 {

```

We will be parsing on `#1` for each step so we might as well evaluate it first in case it is complicated.

```

4376   \exp_args:No \prg_case_int_aux:nnn {\num_value:w \int_eval:n{#1}} #2

```

The `?` below is just so there are enough arguments when we reach the end. And it made you look. ;-)

```

4377   \q_recursion_tail ? \q_recursion_stop
4378 }
4379 \def_long_new:Npn \prg_case_int_aux:nnn #1#2#3{

```

If we reach the end, return the else case. We just remove braces.

```

4380   \quark_if_recursion_tail_stop_do:nn{#2}{\use_arg_i:n}

```

Otherwise we compare (which evaluates #2 for us)

```
4381 \num_compare:nNnTF{#1}={#2}
```

If true, we want to remove the remainder of the list, the else case and then execute the code specified. Why not use #3\use_none:n? Because if we are doing f style expansion, we will get leftovers. If the test was false, we try the next pair, carrying the #1.

```
4382 { \use_arg_i_delimit_by_q_recursion_stop:nw {\use_arg_i:nn{#3}} }
4383 { \prg_case_int_aux:nnn {#1}}
4384 }
```

\prg_case_dim:nnn Same as \prg_case_dim:nnn except it is for $\langle dim \rangle$ registers.

```
\prg_case_dim_aux:nnn
4385 \def_long:Npn \prg_case_dim:nnn #1 #2 {
4386 \exp_args:No \prg_case_dim_aux:nnn {\dim_use:N \dim_eval:n{#1}} #2
4387 \q_recursion_tail ? \q_recursion_stop
4388 }
4389 \def_long_new:Npn \prg_case_dim_aux:nnn #1#2#3{
4390 \quark_if_recursion_tail_stop_do:nn{#2}{\use_arg_i:n}
4391 \dim_compare:nNnTF{#1}={#2}
4392 { \use_arg_i_delimit_by_q_recursion_stop:nw {\use_arg_i:nn{#3}} }
4393 { \prg_case_dim_aux:nnn {#1}}
4394 }
```

\prg_case_str:nnn Same as \prg_case_dim:nnn except it is for strings.

```
\prg_case_str_aux:nnn
4395 \def_long:Npn \prg_case_str:nnn #1 #2 {
4396 \prg_case_str_aux:nnn {#1} #2
4397 \q_recursion_tail ? \q_recursion_stop
4398 }
4399 \def_long_new:Npn \prg_case_str_aux:nnn #1#2#3{
4400 \quark_if_recursion_tail_stop_do:nn{#2}{\use_arg_i:n}
4401 \tlist_if_eq:xxTF{#1}{#2}
4402 { \use_arg_i_delimit_by_q_recursion_stop:nw {\use_arg_i:nn{#3}} }
4403 { \prg_case_str_aux:nnn {#1}}
4404 }
```

23.5.6 Sorting

\prg_define_quicksort:nnn #1 is the name, #2 and #3 are the tokens enclosing the argument. For the somewhat strange $\langle clist \rangle$ type which doesn't enclose the items but uses a separator we define it by hand afterwards. When doing the first pass, the algorithm wraps all elements in braces and then uses a generic quicksort which works on token lists.

As an example

```
\prg_define_quicksort:nnn{seq}{\seq_elt:w}{\seq_elt_end:w}
```

defines the user function \seq_quicksort:n and furthermore expects to use the two functions \seq_quicksort_compare:nnTF which compares the items and \seq_quicksort_function:n which is placed before each sorted item. It is up to the programmer to define these functions when needed. For the seq type a sequence is a token list pointer, so one additionally has to define

```
\def:Npn \seq_quicksort:N{\exp_args:No\seq_quicksort:n}
```

For details on the implementation see “Sorting in T_EX’s Mouth” by Bernd Raichle. Firstly we define the function for parsing the inital list and then the braced list afterwards.

```
4405 \def_new:NNn \prg_define_quicksort:nnn 3 {
4406   \def_long:cNx{#1_quicksort:n}1{
4407     \exp_not:c{#1_quicksort_start_partition:w} ##1
4408     \exp_not:n{#2\q_nil#3\q_stop}
4409   }
4410   \def_long:cNx{#1_quicksort_braced:n}1{
4411     \exp_not:c{#1_quicksort_start_partition_braced:n} ##1
4412     \exp_not:N\q_nil\exp_not:N\q_stop
4413   }
4414   \def_long:cpx {#1_quicksort_start_partition:w} #2 ##1 #3{
4415     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
4416     \exp_not:c{#1_quicksort_do_partition_i:nnnw} {##1}{\}{}
4417   }
4418   \def_long:cNx {#1_quicksort_start_partition_braced:n} 1 {
4419     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
4420     \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn} {##1}{\}{}
4421   }
```

Now for doing the partitions.

```
4422   \def_long:cpx {#1_quicksort_do_partition_i:nnnw} ##1##2##3 #2 ##4 #3 {
4423     \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
4424     {
4425       \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
4426       \exp_not:c{#1_quicksort_partition_greater_ii:nnnn}
4427       \exp_not:c{#1_quicksort_partition_less_ii:nnnn}
4428     }
4429     {##1}{##2}{##3}{##4}
4430   }
4431   \def_long:cNx {#1_quicksort_do_partition_i_braced:nnnn} 4 {
4432     \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
4433     {
4434       \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
4435       \exp_not:c{#1_quicksort_partition_greater_ii_braced:nnnn}
4436       \exp_not:c{#1_quicksort_partition_less_ii_braced:nnnn}
4437     }
4438     {##1}{##2}{##3}{##4}
4439   }
4440   \def_long:cpx {#1_quicksort_do_partition_ii:nnnw} ##1##2##3 #2 ##4 #3 {
4441     \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
4442     {
4443       \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
4444       \exp_not:c{#1_quicksort_partition_less_i:nnnn}
4445       \exp_not:c{#1_quicksort_partition_greater_i:nnnn}
4446     }
4447     {##1}{##2}{##3}{##4}
4448   }
4449   \def_long:cNx {#1_quicksort_do_partition_ii_braced:nnnn} 4 {
4450     \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
```



```

4451 {
4452   \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
4453   \exp_not:c{#1_quicksort_partition_less_i:nnnn}{##2}{##3}
4454   \exp_not:c{#1_quicksort_partition_greater_i:nnnn}{##2}{##3}
4455 }
4456 {##1}{##2}{##3}{##4}
4457 }

```

This part of the code handles the two branches in each sorting. Again we will also have to do it braced.

```

4458 \def_long:cNx {#1_quicksort_partition_less_i:nnnn} 4{
4459   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##3}
4460 \def_long:cNx {#1_quicksort_partition_less_ii:nnnn} 4{
4461   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}
4462 \def_long:cNx {#1_quicksort_partition_greater_i:nnnn} 4{
4463   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##3}
4464 \def_long:cNx {#1_quicksort_partition_greater_ii:nnnn} 4{
4465   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}
4466 \def_long:cNx {#1_quicksort_partition_less_i:nnnn} 4{
4467   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##3}{##4}
4468 \def_long:cNx {#1_quicksort_partition_less_ii:nnnn} 4{
4469   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}
4470 \def_long:cNx {#1_quicksort_partition_greater_i:nnnn} 4{
4471   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##3}{##4}
4472 \def_long:cNx {#1_quicksort_partition_greater_ii:nnnn} 4{
4473   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}

```

Finally, the big kahuna! This is where the sub-lists are sorted.

```

4474 \def_long:cpx {#1_do_quicksort_braced:nnnnw} ##1##2##3##4\q_stop {
4475   \exp_not:c{#1_quicksort_braced:n}{##2}
4476   \exp_not:c{#1_quicksort_function:n}{##1}
4477   \exp_not:c{#1_quicksort_braced:n}{##3}
4478 }
4479 }

```

`\prg_quicksort:n` A simple version. Sorts a list of tokens, uses the function `\prg_quicksort_compare:nnTF` to compare items, and places the function `\prg_quicksort_function:n` in front of each of them.

```

4480 \prg_define_quicksort:nnn {prg}{}{ }

```

```

\prg_quicksort_function:n
\prg_quicksort_compare:nnTF
4481 \let:NN \prg_quicksort_function:n \ERROR
4482 \let:NN \prg_quicksort_compare:nnTF \ERROR

```

That's it (for now).

```

4483 </initex | package>
4484 < *showmemory>
4485 \showMemUsage
4486 </showmemory>

```

24 A token of my appreciation...

This module deals with tokens. Now this is perhaps not the most precise description so let's try with a better description: When programming in \TeX , it is often desirable to know just what a certain token is: is it a control sequence or something else. Similarly one often needs to know if a control sequence is expandable or not, a macro or a primitive, how many arguments it takes etc. Another thing of great importance (especially when it comes to document commands) is looking ahead in the token stream to see if a certain character is present and maybe even remove it or disregard other tokens while scanning. This module provides functions for both and as such will have two primary function categories: $\backslash\text{token}$ for anything that deals with tokens and $\backslash\text{peek}$ for looking ahead in the token stream.

Most of the time we will be using the term 'token' but most of the time the function we're describing can equally well be used on a control sequence as such one is one token as well.

We shall refer to list of tokens as **tlists** and such lists represented by a single control sequence is a 'token list pointer' **tlp**. Functions for these two types are found in the **l3tlp** module.

24.1 Character tokens

Setting category codes of characters.

$\backslash\text{char_set_catcode:nn}$	$\backslash\text{char_set_catcode:nn } \{ \langle \text{char} \rangle \} \{ \langle \text{number} \rangle \}$
$\backslash\text{char_set_catcode:w}$	$\backslash\text{char_set_catcode:w } \langle \text{char} \rangle = \langle \text{number} \rangle$
$\backslash\text{char_value_catcode:n}$	$\backslash\text{char_value_catcode:n } \{ \langle \text{char} \rangle \}$
$\backslash\text{char_value_catcode:w}$	$\backslash\text{char_value_catcode:w } \langle \text{char} \rangle = \langle \text{number} \rangle$
$\backslash\text{char_show_value_catcode:n}$	$\backslash\text{char_show_value_catcode:n } \{ \langle \text{char} \rangle \}$
$\backslash\text{char_show_value_catcode:w}$	$\backslash\text{char_show_value_catcode:w } \langle \text{char} \rangle = \langle \text{number} \rangle$

$\backslash\text{char_set_catcode:nn}$ sets the category code of a character, $\backslash\text{char_value_catcode:n}$ returns its value for use in integer tests and $\backslash\text{char_show_value_catcode:n}$ prints the value on the terminal and in the log file. The **:w** should be avoided unless you have to fiddle with the catcode of **{** or **}**.

\TeX hackers note: $\backslash\text{char_set_catcode:w}$ is the \TeX primitive $\backslash\text{catcode}$ renamed.

$\backslash\text{char_set_lccode:nn}$	$\backslash\text{char_set_lccode:nn } \{ \langle \text{char} \rangle \} \{ \langle \text{number} \rangle \}$
$\backslash\text{char_set_lccode:w}$	$\backslash\text{char_set_lccode:w } \langle \text{char} \rangle = \langle \text{number} \rangle$
$\backslash\text{char_value_lccode:n}$	$\backslash\text{char_value_lccode:n } \{ \langle \text{char} \rangle \}$
$\backslash\text{char_value_lccode:w}$	$\backslash\text{char_value_lccode:w } \langle \text{char} \rangle = \langle \text{number} \rangle$
$\backslash\text{char_show_value_lccode:n}$	$\backslash\text{char_show_value_lccode:n } \{ \langle \text{char} \rangle \}$
$\backslash\text{char_show_value_lccode:w}$	$\backslash\text{char_show_value_lccode:w } \langle \text{char} \rangle = \langle \text{number} \rangle$

Set the lower case representation of $\langle \text{char} \rangle$ for when $\langle \text{char} \rangle$ is being converted in

`\tlist_to_lowercase:n`. As above, the `:w` form is only for people who really, really know what they are doing.

T_EXhackers note: `\char_set_lccode:w` is the T_EX primitive `\lccode` renamed.

<code>\char_set_uccode:nn</code>	
<code>\char_set_uccode:w</code>	
<code>\char_value_uccode:n</code>	<code>\char_set_uccode:nn {<char>} {<number>}</code>
<code>\char_value_uccode:w</code>	<code>\char_set_uccode:w <char> = <number></code>
<code>\char_show_value_uccode:n</code>	<code>\char_value_uccode:n {<char>}</code>
<code>\char_show_value_uccode:w</code>	<code>\char_show_value_uccode:n {<char>}</code>

Set the uppercase representation of `<char>` for when `<char>` is being converted in `\tlist_to_uppercase:n`. As above, the `:w` form is only for people who really, really know what they are doing.

T_EXhackers note: `\char_set_uccode:w` is the T_EX primitive `\uccode` renamed.

<code>\char_set_sfcodes:nn</code>	
<code>\char_set_sfcodes:w</code>	
<code>\char_value_sfcodes:n</code>	<code>\char_set_sfcodes:nn {<char>} {<number>}</code>
<code>\char_value_sfcodes:w</code>	<code>\char_set_sfcodes:w <char> = <number></code>
<code>\char_show_value_sfcodes:n</code>	<code>\char_value_sfcodes:n {<char>}</code>
<code>\char_show_value_sfcodes:w</code>	<code>\char_show_value_sfcodes:n {<char>}</code>

Set the space factor for `<char>`.

T_EXhackers note: `\char_set_sfcodes:w` is the T_EX primitive `\sfcodes` renamed.

<code>\char_set_mathcode:nn</code>	
<code>\char_set_mathcode:w</code>	
<code>\char_gset_mathcode:nn</code>	
<code>\char_gset_mathcode:w</code>	
<code>\char_value_mathcode:n</code>	<code>\char_set_mathcode:nn {<char>} {<number>}</code>
<code>\char_value_mathcode:w</code>	<code>\char_set_mathcode:w <char> = <number></code>
<code>\char_show_value_mathcode:n</code>	<code>\char_value_mathcode:n {<char>}</code>
<code>\char_show_value_mathcode:w</code>	<code>\char_show_value_mathcode:n {<char>}</code>

Set the math code for `<char>`.

T_EXhackers note: `\char_set_mathcode:w` is the T_EX primitive `\mathcode` renamed.

24.2 Generic tokens

<code>\token_new:Nn</code>

`\token_new:Nn <token 1> {<token 2>}`

Defines $\langle token\ 1 \rangle$ to globally be a snapshot of $\langle token\ 2 \rangle$. This will be an implicit representation of $\langle token\ 2 \rangle$.

<code>\c_group_begin_token</code>
<code>\c_group_end_token</code>
<code>\c_math_shift_token</code>
<code>\c_alignment_tab_token</code>
<code>\c_parameter_token</code>
<code>\c_math_superscript_token</code>
<code>\c_math_subscript_token</code>
<code>\c_space_token</code>
<code>\c_letter_token</code>
<code>\c_other_char_token</code>
<code>\c_active_char_token</code>

Some useful constants. They have category codes 1, 2, 3, 4, 6, 7, 8, 10, 11, 12, and 13 respectively. They are all implicit tokens.

<code>\token_if_group_begin_p:N</code>
<code>\token_if_group_begin:NTF</code>
<code>\token_if_group_begin:NT</code>
<code>\token_if_group_begin:NF</code>

`\token_if_group_begin:NTF <token> {<true>} {<false>}`

Check if $\langle token \rangle$ is a begin group token.

<code>\token_if_group_end_p:N</code>
<code>\token_if_group_end:NTF</code>
<code>\token_if_group_end:NT</code>
<code>\token_if_group_end:NF</code>

`\token_if_group_end:NTF <token> {<true>} {<false>}`

Check if $\langle token \rangle$ is an end group token.

<code>\token_if_math_shift_p:N</code>
<code>\token_if_math_shift:NTF</code>
<code>\token_if_math_shift:NT</code>
<code>\token_if_math_shift:NF</code>

`\token_if_math_shift:NTF <token> {<true>} {<false>}`

Check if $\langle token \rangle$ is a math shift token.

<code>\token_if_aligment_tab_p:N</code>
<code>\token_if_aligment_tab:NTF</code>
<code>\token_if_aligment_tab:NT</code>
<code>\token_if_aligment_tab:NF</code>

`\token_if_aligment_tab:NTF <token> {<true>} {<false>}`

Check if $\langle token \rangle$ is an aligment tab token.

<code>\token_if_parameter_p:N</code> <code>\token_if_parameter:NTF</code> <code>\token_if_parameter:NT</code> <code>\token_if_parameter:NF</code>	<code>\token_if_parameter:NTF <token> {\true} {\false}</code>
--	---

Check if $\langle token \rangle$ is a parameter token.

<code>\token_if_math_superscript_p:N</code> <code>\token_if_math_superscript:NTF</code> <code>\token_if_math_superscript:NT</code> <code>\token_if_math_superscript:NF</code>	<code>\token_if_math_superscript:NTF <token> {\true} {\false}</code>
--	--

Check if $\langle token \rangle$ is a math superscript token.

<code>\token_if_math_subscript_p:N</code> <code>\token_if_math_subscript:NTF</code> <code>\token_if_math_subscript:NT</code> <code>\token_if_math_subscript:NF</code>	<code>\token_if_math_subscript:NTF <token> {\true} {\false}</code>
--	--

Check if $\langle token \rangle$ is a math subscript token.

<code>\token_if_space_p:N</code> <code>\token_if_space:NTF</code> <code>\token_if_space:NT</code> <code>\token_if_space:NF</code>	<code>\token_if_space:NTF <token> {\true} {\false}</code>
--	---

Check if $\langle token \rangle$ is a space token.

<code>\token_if_letter_p:N</code> <code>\token_if_letter:NTF</code> <code>\token_if_letter:NT</code> <code>\token_if_letter:NF</code>	<code>\token_if_letter:NTF <token> {\true} {\false}</code>
--	--

Check if $\langle token \rangle$ is a letter token.

<code>\token_if_other_char_p:N</code> <code>\token_if_other_char:NTF</code> <code>\token_if_other_char:NT</code> <code>\token_if_other_char:NF</code>	<code>\token_if_other_char:NTF <token> {\true} {\false}</code>
--	--

Check if $\langle token \rangle$ is an other char token.

<code>\token_if_active_char_p:N</code> <code>\token_if_active_char:NTF</code> <code>\token_if_active_char:NT</code> <code>\token_if_active_char:NF</code>	<code>\token_if_active_char:NTF <token> {\true} {\false}</code>
--	---

Check if $\langle token \rangle$ is an active char token.

<code>\token_if_eq_meaning_p:NN</code>	
<code>\token_if_eq_meaning:NNTF</code>	
<code>\token_if_eq_meaning:NNT</code>	<code>\token_if_eq_meaning:NNTF <token1></code>
<code>\token_if_eq_meaning:NNF</code>	<code><token2>{\<true>}{\<false>}</code>

Check if the meaning of two tokens are identical.

<code>\token_if_eq_catcode_p:NN</code>	
<code>\token_if_eq_catcode:NNTF</code>	
<code>\token_if_eq_catcode:NNT</code>	<code>\token_if_eq_catcode:NNTF <token1></code>
<code>\token_if_eq_catcode:NNF</code>	<code><token2>{\<true>}{\<false>}</code>

Check if the category codes of two tokens are equal. If both tokens are control sequences the test will be true.

<code>\token_if_eq_charcode_p:NN</code>	
<code>\token_if_eq_catcode:NNTF</code>	
<code>\token_if_eq_catcode:NNT</code>	<code>\token_if_eq_catcode:NNTF <token1></code>
<code>\token_if_eq_catcode:NNF</code>	<code><token2>{\<true>}{\<false>}</code>

Check if the character codes of two tokens are equal. If both tokens are control sequences the test will be true.

<code>\token_if_macro_p:N</code>	
<code>\token_if_macro:NNTF</code>	
<code>\token_if_macro:NT</code>	
<code>\token_if_macro:NF</code>	<code>\token_if_macro:NNTF <token> {\<true>} {\<false>}</code>

Check if `<token>` is a macro.

<code>\token_if_cs_p:N</code>	
<code>\token_if_cs:NNTF</code>	
<code>\token_if_cs:NT</code>	
<code>\token_if_cs:NF</code>	<code>\token_if_cs:NNTF <token> {\<true>} {\<false>}</code>

Check if `<token>` is a control sequence or not. This can be useful for situations where the next token in the input stream is being looked at and you want to determine what should be done to it.

<code>\token_if_expandable_p:N</code>	
<code>\token_if_expandable:NNTF</code>	
<code>\token_if_expandable:NT</code>	
<code>\token_if_expandable:NF</code>	<code>\token_if_expandable:NNTF <token> {\<true>} {\<false>}</code>

Check if `<token>` is expandable or not. Note that `<token>` can very well be an active character.

The next set of functions here are for picking apart control sequences. Sometimes it is useful to know if a control sequence has arguments and if so, how many. Similarly its status with respect to `\long` or `\protected` is good to have. Finally it can be very useful

to know if a control sequence is of a certain type: Is this $\langle toks \rangle$ register we’re trying to to something with really a $\langle toks \rangle$ register at all?

$\backslash token_if_long_macro_p:N$ $\backslash token_if_long_macro:NTF$ $\backslash token_if_long_macro:NT$ $\backslash token_if_long_macro:NF$	$\backslash token_if_long_macro:NTF \langle token \rangle \{ \langle true \rangle \} \{ \langle false \rangle \}$
---	--

Check if $\langle token \rangle$ is a “long” macro.

$\backslash token_if_protected_macro_p:N$ $\backslash token_if_protected_macro:NTF$ $\backslash token_if_protected_macro:NT$ $\backslash token_if_protected_macro:NF$	$\backslash token_if_long_macro:NTF \langle token \rangle \{ \langle true \rangle \} \{ \langle false \rangle \}$
---	--

Check if $\langle token \rangle$ is a “protected” macro. This test does *not* return $\langle true \rangle$ if the macro is also “long”, see below.

$\backslash token_if_protected_long_macro_p:N$ $\backslash token_if_protected_long_macro:NTF$ $\backslash token_if_protected_long_macro:NT$ $\backslash token_if_protected_long_macro:NF$	$\backslash token_if_protected_long_macro:NTF \langle token \rangle \{ \langle true \rangle \} \{ \langle false \rangle \}$
---	---

Check if $\langle token \rangle$ is a “protected long” macro.

$\backslash token_if_chardef_p:N$ $\backslash token_if_chardef:NTF$ $\backslash token_if_chardef:NT$ $\backslash token_if_chardef:NF$	$\backslash token_if_chardef:NTF \langle token \rangle \{ \langle true \rangle \} \{ \langle false \rangle \}$
---	--

Check if $\langle token \rangle$ is defined to be a chardef.

$\backslash token_if_mathchardef_p:N$ $\backslash token_if_mathchardef:NTF$ $\backslash token_if_mathchardef:NT$ $\backslash token_if_mathchardef:NF$	$\backslash token_if_mathchardef:NTF \langle token \rangle \{ \langle true \rangle \} \{ \langle false \rangle \}$
---	--

Check if $\langle token \rangle$ is defined to be a mathchardef.

$\backslash token_if_int_register_p:N$ $\backslash token_if_int_register:NTF$ $\backslash token_if_int_register:NT$ $\backslash token_if_int_register:NF$	$\backslash token_if_int_register:NTF \langle token \rangle \{ \langle true \rangle \} \{ \langle false \rangle \}$
---	--

Check if $\langle token \rangle$ is defined to be an integer register.

\token_if_dim_register_p:N \token_if_dim_register:NTF \token_if_dim_register:NT \token_if_dim_register:NF	\token_if_dim_register:NTF $\langle token \rangle$ $\{\langle true \rangle\}$ $\{\langle false \rangle\}$
--	---

Check if $\langle token \rangle$ is defined to be a dimension register.

\token_if_skip_register_p:N \token_if_skip_register:NTF \token_if_skip_register:NT \token_ifskip_register:NF	\token_if_skip_register:NTF $\langle token \rangle$ $\{\langle true \rangle\}$ $\{\langle false \rangle\}$
---	--

Check if $\langle token \rangle$ is defined to be a skip register.

\token_if_toks_register_p:N \token_if_toks_register:NTF \token_if_toks_register:NT \token_if_toks_register:NF	\token_if_toks_register:NTF $\langle token \rangle$ $\{\langle true \rangle\}$ $\{\langle false \rangle\}$
--	--

Check if $\langle token \rangle$ is defined to be a toks register.

\token_get_prefix_spec:N \token_get_arg_spec:N \token_get_replacement_spec:N	\token_get_arg_spec:N $\langle token \rangle$
--	---

If token is a macro with definition `\def_long:Npn\next #1#2{x'#1--#2'y}`, the `prefix` function will return the string `\long`, the `arg` function returns the string `#1#2` and the `replacement` function returns the string `x'#1--#2'y`. If $\langle token \rangle$ isn't a macro, these functions return the `\scan_stop:` token.

24.2.1 Useless code: because we can!

\token_if_primitive_p:N \token_if_primitive:NTF \token_if_primitive:NT \token_if_primitive:NF	\token_if_primitive:NTF $\langle token \rangle$ $\{\langle true \rangle\}$ $\{\langle false \rangle\}$
--	--

Check if $\langle token \rangle$ is a primitive. Probably not a very useful function.

24.3 Peeking ahead at the next token

\l_peek_token \g_peek_token \l_peek_search_token
--

Some useful variables. Initially they are set to ?.

<code>\peek_after:NN</code> <code>\peek_gafter:NN</code>	<code>\peek_after:NN <function><token></code>
---	---

Assign $\langle token \rangle$ to `\l_peek_token` and then run $\langle function \rangle$ which should perform some sort of test on this token. Leaves $\langle token \rangle$ in the input stream. `\peek_gafter:NN` does this globally to the token `\g_peek_token`.

TeXhackers note: This is the primitive `\futurelet` turned into a function.

<code>\peek_meaning:NTF</code> <code>\peek_meaning_ignore_spaces:NTF</code> <code>\peek_meaning_remove:NTF</code> <code>\peek_meaning_remove_ignore_spaces:NTF</code>	<code>\peek_meaning:NTF <token> {\langle true \rangle}{\langle false \rangle}</code>
--	--

`\peek_meaning:NTF` checks (by using `\if_meaning:NN`) if $\langle token \rangle$ equals the next token in the input stream and executes either $\langle true\ code \rangle$ or $\langle false\ code \rangle$ accordingly. `\peek_meaning_remove:NTF` does the same but additionally removes the token if found. The `ignore_spaces` versions skips blank spaces before making the decision.

<code>\peek_charcode:NTF</code> <code>\peek_charcode_ignore_spaces:NTF</code> <code>\peek_charcode_remove:NTF</code> <code>\peek_charcode_remove_ignore_spaces:NTF</code>	<code>\peek_charcode:NTF <token> {\langle true \rangle}{\langle false \rangle}</code>
--	---

Same as for the `\peek_meaning:NTF` functions above but these use `\if_charcode:w` to compare the tokens.

<code>\peek_catcode:NTF</code> <code>\peek_catcode_ignore_spaces:NTF</code> <code>\peek_catcode_remove:NTF</code> <code>\peek_catcode_remove_ignore_spaces:NTF</code>	<code>\peek_catcode:NTF <token> {\langle true \rangle}{\langle false \rangle}</code>
--	--

Same as for the `\peek_meaning:NTF` functions above but these use `\if_catcode:w` to compare the tokens.

<code>\peek_token_generic:NNTF</code> <code>\peek_token_remove_generic:NNTF</code>	<code>\peek_token_generic:NNTF <token><function></code> <code>{\langle true \rangle}{\langle false \rangle}</code>
---	---

`\peek_token_generic:NNTF` looks ahead and checks if the next token in the input stream is equal to $\langle token \rangle$. It uses $\langle function \rangle$ to make that decision. `\peek_token_remove_generic:NNTF` does the same thing but additionally removes $\langle token \rangle$ from the input stream if it is found. This also works if $\langle token \rangle$ is either `\c_group_begin_token` or `\c_group_end_token`.

<code>\peek_execute_branches_meaning:</code> <code>\peek_execute_branches_charcode:</code> <code>\peek_execute_branches_catcode:</code>	<code>\peek_execute_branches_meaning:</code>
---	--

These functions compare the token we are searching for with the token found (after optional ignoring of specific tokens). They come in the usual three versions when T_EX is comparing tokens: meaning, character code, and category code.

24.3.1 Internal functions

<code>\l_peek_true_tlp</code> <code>\l_peek_false_tlp</code>

These token list pointers are used internally when choosing either the true or false branches of a test.

<code>\peek_tmp:w</code>

Scratch function used to gobble tokens from the input stream.

<code>\l_peek_true_aux_tlp</code> <code>\l_peek_true_remove_next_tlp</code>
--

These token list pointers are used internally when choosing either the true or false branches of a test.

<code>\peek_ignore_spaces_execute_branches:</code> <code>\peek_ignore_spaces_aux:</code>

Functions used to ignore space tokens in the input stream.

24.4 Implementation

First a few required packages to get this going.

```

4487 <package>\ProvidesExplPackage
4488 <package> {\filename}{\filedate}{\fileversion}{\filedescription}
4489 <package>\RequirePackage{l3prg}
4490 <package>\RequirePackage{l3int}
4491 <{*initex | package>

```

24.4.1 Character tokens

```

\char_set_catcode:w
\char_set_catcode:nn
\char_value_catcode:w 4492 \let_new:NN \char_set_catcode:w \tex_catcode:D
\char_value_catcode:n 4493 \def_new:Npn \char_set_catcode:nn #1#2{
\char_show_value_catcode:w 4494 \char_set_catcode:w #1 = \int_eval:n{#2}
4495 }
\char_show_value_catcode:n 4496 \def_new:Npn \char_value_catcode:w {\int_use:N\tex_catcode:D}
4497 \def_new:Npn \char_value_catcode:n #1{\char_value_catcode:w \int_eval:n{#1}}
4498 \def_new:Npn \char_show_value_catcode:w {\tex_showthe:D\tex_catcode:D}
4499 \def_new:Npn \char_show_value_catcode:n #1{
4500 \char_show_value_catcode:w \int_eval:n{#1}}

```

```

\char_set_mathcode:w Math codes.
\char_set_mathcode:nn
\char_gset_mathcode:w 4501 \let_new:NN \char_set_mathcode:w \tex_mathcode:D
\char_gset_mathcode:nn 4502 \def_new:Npn \char_set_mathcode:nn #1#2{
\char_value_mathcode:w 4503 \char_set_mathcode:w #1 = \int_eval:n{#2}
\char_value_mathcode:n 4504 }
\char_value_mathcode:n 4505 \def_protected_new:Npn \char_gset_mathcode:w {\pref_global:D\tex_mathcode:D}
\char_show_value_mathcode:w 4506 \def_new:Npn \char_gset_mathcode:nn #1#2{
\char_show_value_mathcode:n 4507 \char_gset_mathcode:w #1 = \int_eval:n{#2}
4508 }
4509 \def_new:Npn \char_value_mathcode:w {\int_use:N\tex_mathcode:D}
4510 \def_new:Npn \char_value_mathcode:n #1{\char_value_mathcode:w \int_eval:n{#1}}
4511 \def_new:Npn \char_show_value_mathcode:w {\tex_showthe:D\tex_mathcode:D}
4512 \def_new:Npn \char_show_value_mathcode:n #1{
4513 \char_show_value_mathcode:w \int_eval:n{#1}}

\char_set_lccode:w
\char_set_lccode:nn
\char_value_lccode:w 4514 \let_new:NN \char_set_lccode:w \tex_lccode:D
\char_value_lccode:n 4515 \def_new:Npn \char_set_lccode:nn #1#2{
\char_value_lccode:n 4516 \char_set_lccode:w #1 = \int_eval:n{#2}
\char_show_value_lccode:w 4517 }
\char_show_value_lccode:n 4518 \def_new:Npn \char_value_lccode:w {\int_use:N\tex_lccode:D}
4519 \def_new:Npn \char_value_lccode:n #1{\char_value_lccode:w \int_eval:n{#1}}
4520 \def_new:Npn \char_show_value_lccode:w {\tex_showthe:D\tex_lccode:D}
4521 \def_new:Npn \char_show_value_lccode:n #1{
4522 \char_show_value_lccode:w \int_eval:n{#1}}

\char_set_uccode:w
\char_set_uccode:nn
\char_value_uccode:w 4523 \let_new:NN \char_set_uccode:w \tex_uccode:D
\char_value_uccode:n 4524 \def_new:Npn \char_set_uccode:nn #1#2{
\char_value_uccode:n 4525 \char_set_uccode:w #1 = \int_eval:n{#2}
\char_show_value_uccode:w 4526 }
\char_show_value_uccode:n 4527 \def_new:Npn \char_value_uccode:w {\int_use:N\tex_uccode:D}
4528 \def_new:Npn \char_value_uccode:n #1{\char_value_uccode:w \int_eval:n{#1}}
4529 \def_new:Npn \char_show_value_uccode:w {\tex_showthe:D\tex_uccode:D}
4530 \def_new:Npn \char_show_value_uccode:n #1{
4531 \char_show_value_uccode:w \int_eval:n{#1}}

\char_set_sfcode:w
\char_set_sfcode:nn
\char_value_sfcode:w 4532 \let_new:NN \char_set_sfcode:w \tex_sfcode:D
\char_value_sfcode:n 4533 \def_new:Npn \char_set_sfcode:nn #1#2{
\char_value_sfcode:n 4534 \char_set_sfcode:w #1 = \int_eval:n{#2}
\char_show_value_sfcode:w 4535 }
\char_show_value_sfcode:n 4536 \def_new:Npn \char_value_sfcode:w {\int_use:N\tex_uccode:D}
4537 \def_new:Npn \char_value_sfcode:n #1{\char_value_sfcode:w \int_eval:n{#1}}
4538 \def_new:Npn \char_show_value_sfcode:w {\tex_showthe:D\tex_sfcode:D}
4539 \def_new:Npn \char_show_value_sfcode:n #1{
4540 \char_show_value_sfcode:w \int_eval:n{#1}}

```

24.4.2 Generic tokens

`\token_new:Nn` Creates a new token.

```
4541 \def_new:Npn \token_new:Nn #1#2{\glet_new:NN #1#2}
```

`\c_group_begin_token` We define these useful tokens. We have to do it by hand with the brace tokens for obvious reasons.

```
\c_group_end_token
\c_math_shift_token
\c_alignment_tab_token 4542 \let_new:NN \c_group_begin_token {
\c_parameter_token      4543 \let_new:NN \c_group_end_token }
4544 \group_begin:
\c_math_superscript_token 4545 \char_set_catcode:nn{'\*}{3}
\c_math_subscript_token 4546 \token_new:Nn \c_math_shift_token {*}
\c_space_token          4547 \char_set_catcode:nn{'\*}{4}
\c_letter_token         4548 \token_new:Nn \c_alignment_tab_token {*}
\c_other_char_token     4549 \token_new:Nn \c_parameter_token {#}
\c_active_char_token    4550 \token_new:Nn \c_math_superscript_token {~}
                        4551 \char_set_catcode:nn{'\*}{8}
                        4552 \token_new:Nn \c_math_subscript_token {*}
                        4553 \token_new:Nn \c_space_token {~}
                        4554 \token_new:Nn \c_letter_token {a}
                        4555 \token_new:Nn \c_other_char_token {1}
                        4556 \char_set_catcode:nn{'\*}{13}
                        4557 \token_new:Nn \c_active_char_token {*}
                        4558 \group_end:
```

`\token_if_group_begin_p:N` Check if token is a begin group token. We use the constant `\c_group_begin_token` for this.

```
\token_if_group_begin:NTF
\token_if_group_begin:NT 4559 \def_new:Npn \token_if_group_begin_p:N #1{
\token_if_group_begin:NF 4560 \if_catcode:w \exp_not:N #1\c_group_begin_token
                        4561 \c_true
                        4562 \else:
                        4563 \c_false
                        4564 \fi:
                        4565 }
                        4566 \def_test_function_new:npn {token_if_group_begin:N} #1{
                        4567 \if:w\token_if_group_begin_p:N #1}
```

`\token_if_group_end_p:N` Check if token is a end group token. We use the constant `\c_group_end_token` for this.

```
\token_if_group_end:NTF
\token_if_group_end:NT 4568 \def_new:Npn \token_if_group_end_p:N #1{
\token_if_group_end:NF 4569 \if_catcode:w \exp_not:N #1\c_group_end_token
                        4570 \c_true
                        4571 \else:
                        4572 \c_false
                        4573 \fi:
                        4574 }
                        4575 \def_test_function_new:npn {token_if_group_end:N} #1{
                        4576 \if:w\token_if_group_end_p:N #1}
```

```

\token_if_math_shift_p:N Check if token is a math shift token. We use the constant \c_math_shift_token for
\token_if_math_shift:NTF this.
\token_if_math_shift:NT
\token_if_math_shift:NF 4577 \def_new:Npn \token_if_math_shift_p:N #1{
4578   \if_catcode:w \exp_not:N #1\c_math_shift_token
4579     \c_true
4580   \else:
4581     \c_false
4582   \fi:
4583 }
4584 \def_test_function_new:npn {token_if_math_shift:N} #1{
4585   \if:w\token_if_math_shift_p:N#1}

\token_if_alignment_tab_p:N Check if token is an alignment tab token. We use the constant \c_alignment_tab_token
\token_if_alignment_tab:NTF for this.
\token_if_alignment_tab:NT
\token_if_alignment_tab:NF 4586 \def_new:Npn \token_if_alignment_tab_p:N #1{
4587   \if_catcode:w \exp_not:N #1\c_alignment_tab_token
4588     \c_true
4589   \else:
4590     \c_false
4591   \fi:
4592 }
4593 \def_test_function_new:npn {token_if_alignment_tab:N} #1{
4594   \if:w\token_if_alignment_tab_p:N#1}

\token_if_parameter_p:N Check if token is a parameter token. We use the constant \c_parameter_token for this.
\token_if_parameter:NTF We have to trick TEX a bit to avoid an error message.
\token_if_parameter:NT
\token_if_parameter:NF 4595 \def_new:Npn \token_if_parameter_p:N #1{
4596   \exp_after:NN\if_catcode:w \cs:w c_parameter_token\cs_end:\exp_not:N #1
4597     \c_true
4598   \else:
4599     \c_false
4600   \fi:
4601 }
4602 \def_test_function_new:npn {token_if_parameter:N} #1{
4603   \if:w\token_if_parameter_p:N#1}

token_if_math_superscript_p:N Check if token is a math superscript token. We use the constant \c_math_superscript_token
token_if_math_superscript:NTF for this.
\token_if_math_superscript:NT
\token_if_math_superscript:NF 4604 \def_new:Npn \token_if_math_superscript_p:N #1{
4605   \if_catcode:w \exp_not:N #1\c_math_superscript_token
4606     \c_true
4607   \else:
4608     \c_false
4609   \fi:
4610 }
4611 \def_test_function_new:npn {token_if_math_superscript:N} #1{
4612   \if:w\token_if_math_superscript_p:N #1}

\token_if_math_subscript_p:N Check if token is a math subscript token. We use the constant \c_math_subscript_token
\token_if_math_subscript:NTF for this.
\token_if_math_subscript:NT
\token_if_math_subscript:NF

```

```

4613 \def_new:Npn \token_if_math_subscript_p:N #1{
4614   \if_catcode:w \exp_not:N #1\c_math_subscript_token
4615     \c_true
4616   \else:
4617     \c_false
4618   \fi:
4619 }
4620 \def_test_function_new:npn {token_if_math_subscript:N} #1{
4621   \if:w\token_if_math_subscript_p:N #1}

```

\token_if_space_p:N Check if token is a space token. We use the constant \c_space_token for this.

```

\token_if_space:NTF
\token_if_space:NT 4622 \def_new:Npn \token_if_space_p:N #1{
\token_if_space:NF 4623   \if_catcode:w \exp_not:N #1\c_space_token
4624     \c_true
4625   \else:
4626     \c_false
4627   \fi:
4628 }
4629 \def_test_function_new:npn {token_if_space:N} #1{
4630   \if:w\token_if_space_p:N #1}

```

\token_if_letter_p:N Check if token is a letter token. We use the constant \c_letter_token for this.

```

\token_if_letter:NTF
\token_if_letter:NT 4631 \def_new:Npn \token_if_letter_p:N #1{
\token_if_letter:NF 4632   \if_catcode:w \exp_not:N #1\c_letter_token
4633     \c_true
4634   \else:
4635     \c_false
4636   \fi:
4637 }
4638 \def_test_function_new:npn {token_if_letter:N} #1{
4639   \if:w\token_if_letter_p:N #1}

```

\token_if_other_char_p:N Check if token is an other char token. We use the constant \c_other_char_token for this.

```

\token_if_other_char:NTF
\token_if_other_char:NT
\token_if_other_char:NF 4640 \def_new:Npn \token_if_other_char_p:N #1{
4641   \if_catcode:w \exp_not:N #1\c_other_char_token
4642     \c_true
4643   \else:
4644     \c_false
4645   \fi:
4646 }
4647 \def_test_function_new:npn {token_if_other_char:N} #1{
4648   \if:w\token_if_other_char_p:N #1}

```

\token_if_active_char_p:N Check if token is an active char token. We use the constant \c_active_char_token for this.

```

\token_if_active_char:NTF
\token_if_active_char:NT
\token_if_active_char:NF 4649 \def_new:Npn \token_if_active_char_p:N #1{
4650   \if_catcode:w \exp_not:N #1\c_active_char_token
4651     \c_true

```

```

4652     \else:
4653         \c_false
4654     \fi:
4655 }
4656 \def_test_function_new:npn {token_if_active_char:N} #1{
4657     \if:w\token_if_active_char_p:N #1}

```

\token_if_eq_meaning_p:NN Check if the tokens #1 and #2 have same meaning.

```

\token_if_eq_meaning:NNTF
\token_if_eq_meaning:NNT 4658 \def_new:Npn \token_if_eq_meaning_p:NN #1#2 {
\token_if_eq_meaning:NNT 4659     \if_meaning:NN #1 #2
\token_if_eq_meaning:NNT 4660         \c_true
4661     \else:
4662         \c_false
4663     \fi:
4664 }
4665 \def_test_function_new:npn {token_if_eq_meaning:NN}#1#2{
4666     \if_meaning:NN #1 #2}

```

\token_if_eq_catcode_p:NN Check if the tokens #1 and #2 have same category code.

```

\token_if_eq_catcode:NNTF
\token_if_eq_catcode:NNT 4667 \def_new:Npn \token_if_eq_catcode_p:NN #1#2 {
\token_if_eq_catcode:NNT 4668     \if_catcode:w \exp_not:N #1 \exp_not:N #2
\token_if_eq_catcode:NNT 4669         \c_true
4670     \else:
4671         \c_false
4672     \fi:
4673 }
4674 \def_test_function_new:npn {token_if_eq_catcode:NN}#1#2{
4675     \if:w\token_if_eq_catcode_p:NN#1#2}

```

\token_if_eq_charcode_p:NN Check if the tokens #1 and #2 have same character code.

```

\token_if_eq_charcode:NNTF
\token_if_eq_charcode:NNT 4676 \def_new:Npn \token_if_charcode_eq_p:NN #1#2 {
\token_if_eq_charcode:NNT 4677     \if_charcode:w \exp_not:N #1 \exp_not:N #2
\token_if_eq_charcode:NNT 4678         \c_true
4679     \else:
4680         \c_false
4681     \fi:
4682 }
4683 \def_test_function_new:npn {token_if_eq_charcode:NN}#1#2{
4684     \if:w\token_if_eq_charcode_p:NN#1#2}

```

\token_if_macro_p:N When a token is a macro, \token_to_meaning:N will always output something like
\token_if_macro_p_aux:w \long macro:#1->#1 so we simply check to see if the meaning contains ->. Argument
\token_if_macro:NNTF #2 in the code below will be empty if the string -> isn't present, proof that the token was
\token_if_macro:NT not a macro (which is why we reverse the emptiness test). However this function will fail
\token_if_macro:NF on its own auxiliary function (and a few other private functions as well) but that should
certainly never be a problem!

```

4685 \def_new:Npn \token_if_macro_p:N #1 {
4686     \exp_after:NN \token_if_macro_p_aux:w \token_to_meaning:N #1 -> \q_nil
4687 }

```

```

4688 \def_new:Npn \token_if_macro_p_aux:w #1 -> #2 \q_nil{
4689   \if:w \tlist_if_empty_p:n{#2} \c_false \else: \c_true \fi:
4690 }
4691 \def_test_function_new:npn {token_if_macro:N} #1{\if:w\token_if_macro_p:N#1}

\token_if_cs_p:N Check if token has same catcode as a control sequence. We use \scan_stop: for this.
\token_if_cs:NTF
\token_if_cs:NT 4692 \def_new:Npn \token_if_cs_p:N {\token_if_eq_catcode_p:NN \scan_stop:}
\token_if_cs:NF 4693 \def_test_function_new:npn {token_if_cs:N} #1{
4694   \if:w \token_if_eq_catcode_p:NN \scan_stop: #1}

\token_if_expandable_p:N Check if token is expandable. We use the fact that TEX will temporarily convert
\token_if_expandable:NTF \exp_not:N <token> into \scan_stop: if <token> is expandable.
\token_if_expandable:NT
\token_if_expandable:NF 4695 \def_new:Npn \token_if_expandable_p:N #1{
4696   \exp_after:NN \if_token_eq:NN \exp_not:N #1 \scan_stop:
4697   \c_true
4698   \else:
4699   \c_false
4700   \fi:
4701 }
4702 \def_test_function_new:npn {token_if_expandable:N} #1{
4703   \if:w\token_if_expandable_p:N#1}

\token_if_chardef_p:N Most of these functions have to check the meaning of the token in question so we need to
\token_if_chardef_p_aux:w do some checkups on which characters are output by \token_to_meaning:N. As usual,
\token_if_mathchardef_p:N these characters have catcode 12 so we must do some serious substitutions in the code
\token_if_mathchardef_p_aux:w below...
\token_if_int_register_p:N
\token_if_int_register_p_aux:w 4704 \group_begin:
\token_if_skip_register_p:N 4705   \char_set_lccode:nn {'\X}{'\n}
\token_if_skip_register_p_aux:w 4706   \char_set_lccode:nn {'\Y}{'\t}
\token_if_dim_register_p:N 4707   \char_set_lccode:nn {'\Z}{'\d}
\token_if_dim_register_p_aux:w 4708   \char_set_lccode:nn {'\?}{'\}
\token_if_toks_register_p:N 4709   \tlist_map_inline:nn{\X\Y\Z\M\T\C\H\A\R\O\U\S\K\I\P\L\G\P\E}
\token_if_toks_register_p_aux:w 4710   {\char_set_catcode:nn {'#1}{12}}

\token_if_protected_macro_p:N We convert the token list to lowercase and restore the catcode and lowercase code changes.
\token_if_protected_macro_p_aux:w
\token_if_long_macro_p:N 4711 \tlist_to_lowercase:n{
\token_if_long_macro_p_aux:w 4712   \group_end:

\token_if_protected_long_macro_p:N First up is checking if something has been defined with \tex_chardef:D or \tex_mathchardef:D.
\token_if_protected_long_macro_p_aux:w This is easy since TEX thinks of such tokens as hexadecimal so it stores them as
\char"<hex number> or \mathchar"<hex number>.

4713 \def_new:Npn \token_if_chardef_p:N #1 {
4714   \exp_after:NN \token_if_chardef_p_aux:w
4715   \token_to_meaning:N #1?CHAR"\q_nil
4716 }
4717 \def_new:Npn \token_if_chardef_p_aux:w #1?CHAR"#2\q_nil{
4718   \tlist_if_empty_p:n{#1}
4719 }
4720 \def_new:Npn \token_if_mathchardef_p:N #1 {

```



```

4721 \exp_after:NN \token_if_mathchardef_p_aux:w
4722 \token_to_meaning:N #1?MAYHCHAR"q_nil
4723 }
4724 \def_new:Npn \token_if_mathchardef_p_aux:w #1?MAYHCHAR"#2\q_nil{
4725 \tlist_if_empty_p:n{#1}
4726 }

```

Integer registers are a little more difficult since they expand to `\count⟨number⟩` and there is also a primitive `\countdef`. So we have to check for that primitive as well.

```

4727 \def:Npn \token_if_int_register_p:N #1{
4728 \if_meaning:NN \tex_countdef:D #1
4729 \c_false
4730 \else:
4731 \exp_after:NN \token_if_int_register_p_aux:w
4732 \token_to_meaning:N #1?COUXY\q_nil
4733 \fi:
4734 }
4735 \def_new:Npn \token_if_int_register_p_aux:w #1?COUXY#2\q_nil{
4736 \tlist_if_empty_p:n{#1}
4737 }

```

Skip registers are done the same way as the integer registers.

```

4738 \def:Npn \token_if_skip_register_p:N #1{
4739 \if_meaning:NN \tex_skipdef:D #1
4740 \c_false
4741 \else:
4742 \exp_after:NN \token_if_skip_register_p_aux:w
4743 \token_to_meaning:N #1?SKIP\q_nil
4744 \fi:
4745 }
4746 \def_new:Npn \token_if_skip_register_p_aux:w #1?SKIP#2\q_nil{
4747 \tlist_if_empty_p:n{#1}
4748 }

```

Dim registers. No news here

```

4749 \def:Npn \token_if_dim_register_p:N #1{
4750 \if_meaning:NN \tex_dimendef:D #1
4751 \c_false
4752 \else:
4753 \exp_after:NN \token_if_dim_register_p_aux:w
4754 \token_to_meaning:N #1?ZIMEX\q_nil
4755 \fi:
4756 }
4757 \def_new:Npn \token_if_dim_register_p_aux:w #1?ZIMEX#2\q_nil{
4758 \tlist_if_empty_p:n{#1}
4759 }

```

Toks registers. Ho-hum.

```

4760 \def:Npn \token_if_toks_register_p:N #1{
4761 \if_meaning:NN \tex_toksdef:D #1
4762 \c_false

```

```

4763 \else:
4764 \exp_after:NN \token_if_toks_register_p_aux:w
4765 \token_to_meaning:N #1?YOKS\q_nil
4766 \fi:
4767 }
4768 \def_new:Npn \token_if_toks_register_p_aux:w #1?YOKS#2\q_nil{
4769 \tlist_if_empty_p:n{#1}
4770 }

```

Protected macros.

```

4771 \def_new:Npn \token_if_protected_macro_p:N #1 {
4772 \exp_after:NN \token_if_protected_macro_p_aux:w
4773 \token_to_meaning:N #1?PROYECY EZ~MACRO\q_nil
4774 }
4775 \def_new:Npn \token_if_protected_macro_p_aux:w #1?PROYECY EZ~MACRO#2\q_nil{
4776 \tlist_if_empty_p:n{#1}
4777 }

```

Long macros.

```

4778 \def_new:Npn \token_if_long_macro_p:N #1 {
4779 \exp_after:NN \token_if_long_macro_p_aux:w
4780 \token_to_meaning:N #1?LOXG~MACRO\q_nil
4781 }
4782 \def_new:Npn \token_if_long_macro_p_aux:w #1?LOXG~MACRO#2\q_nil{
4783 \tlist_if_empty_p:n{#1}
4784 }

```

Finally protected long macros where we for once don't have to add an extra test since there is no primitive for the combined prefixes.

```

4785 \def_new:Npn \token_if_protected_long_macro_p:N #1 {
4786 \exp_after:NN \token_if_protected_long_macro_p_aux:w
4787 \token_to_meaning:N #1?PROYECY EZ~?LOXG~MACRO\q_nil
4788 }
4789 \def_new:Npn \token_if_protected_long_macro_p_aux:w #1
4790 ?PROYECY EZ~?LOXG~MACRO#2\q_nil{
4791 \tlist_if_empty_p:n{#1}
4792 }

```

Finally the \tlist_to_lowercase:n ends!

```

4793 }

```

```

\token_if_chardef:NTF
\token_if_chardef:NT
\token_if_chardef:NF 4794 \def_test_function_new:npn {token_if_chardef:N} {\if:w \token_if_chardef_p:N}

```

```

\token_if_mathchardef:NTF
\token_if_mathchardef:NT
\token_if_mathchardef:NF 4795 \def_test_function_new:npn {token_if_mathchardef:N} {
4796 \if:w \token_if_mathchardef_p:N}

```

```

\token_if_long_macro:NTF
\token_if_long_macro:NT
\token_if_long_macro:NF 4797 \def_test_function_new:npn {token_if_long_macro:N} {
4798   \if:w \token_if_long_macro_p:N}

\token_if_protected_macro:NTF
\token_if_protected_macro:NT
\token_if_protected_macro:NF 4799 \def_test_function_new:npn {token_if_protected_macro:N} {
4800   \if:w \token_if_protected_macro_p:N}

\token_if_protected_long_macro:NTF
\token_if_protected_long_macro:NT
\token_if_protected_long_macro:NF 4801 \def_test_function_new:npn {token_if_protected_long_macro:N} {
4802   \if:w \token_if_protected_long_macro_p:N}

\token_if_dim_register:NTF
\token_if_dim_register:NT
\token_if_dim_register:NF 4803 \def_test_function_new:npn {token_if_dim_register:N} {
4804   \if:w \token_if_dim_register_p:N}

\token_if_skip_register:NTF
\token_if_skip_register:NT
\token_if_skip_register:NF 4805 \def_test_function_new:npn {token_if_skip_register:N} {
4806   \if:w \token_if_skip_register_p:N}

\token_if_int_register:NTF
\token_if_int_register:NT
\token_if_int_register:NF 4807 \def_test_function_new:npn {token_if_int_register:N} {
4808   \if:w \token_if_int_register_p:N}

\token_if_toks_register:NTF
\token_if_toks_register:NT
\token_if_toks_register:NF 4809 \def_test_function_new:npn {token_if_toks_register:N} {
4810   \if:w \token_if_toks_register_p:N}

```

We do not provide a function for testing if a control sequence is “outer” since we don’t use that in L^AT_EX3.

`\prefix_arg_replacement_aux:w` In the `xparse` package we sometimes want to test if a control sequence can be expanded to reveal a hidden value. However, we cannot just expand the macro blindly as it may have arguments and none might be present. Therefore we define these functions to pick either the prefix(es), the argument specification, or the replacement text from a macro. All of this information is returned as characters with catcode 12. If the token in question isn’t a macro, the token `\scan_stop:` is returned instead.

```

4811 \group_begin:
4812 \char_set_lccode:nn {'\?}{\:}
4813 \char_set_catcode:nn{'M}{12}
4814 \char_set_catcode:nn{'A}{12}
4815 \char_set_catcode:nn{'C}{12}
4816 \char_set_catcode:nn{'R}{12}
4817 \char_set_catcode:nn{'O}{12}

```

```

4818 \tlist_to_lowercase:n{
4819   \group_end:
4820   \def_new:Npn \token_get_prefix_arg_replacement_aux:w #1MACRO?#2->#3\q_nil#4{
4821     #4{#1}{#2}{#3}
4822   }
4823   \def_new:Npn\token_get_prefix_spec:N #1{
4824     \token_if_macro:NTF #1{
4825       \exp_after:NN \token_get_prefix_arg_replacement_aux:w
4826       \token_to_meaning:N #1\q_nil\use_arg_i:nnn
4827     }\scan_stop:}
4828   }
4829   \def_new:Npn\token_get_arg_spec:N #1{
4830     \token_if_macro:NTF #1{
4831       \exp_after:NN \token_get_prefix_arg_replacement_aux:w
4832       \token_to_meaning:N #1\q_nil\use_arg_ii:nnn
4833     }\scan_stop:}
4834   }
4835   \def_new:Npn\token_get_replacement_spec:N #1{
4836     \token_if_macro:NTF #1{
4837       \exp_after:NN \token_get_prefix_arg_replacement_aux:w
4838       \token_to_meaning:N #1\q_nil\use_arg_iii:nnn
4839     }\scan_stop:}
4840   }
4841 }

```

Useless code: because we can!

\token_if_primitive_p:N It is rather hard to determine if a token is a primitive. First we can check if it is a
\token_if_primitive_p_aux:N control sequence or active character. If either, we check if it is a macro. Then we can go
\token_if_primitive:NTF through a tedious process of testing for different register types... I don't actually think
\token_if_primitive:NT this function is useful but you never know.
\token_if_primitive:NF

```

4842 \def_new:Npn \token_if_primitive_p:N #1{
4843   \if:w \token_if_cs_p:N #1\scan_stop:
4844   \if:w \token_if_macro_p:N #1
4845     \c_false
4846   \else:
4847     \token_if_primitive_p_aux:N #1
4848     \fi:
4849   \else:
4850     \if:w \token_if_active_p:N #1
4851       \if:w \token_if_macro_p:N #1
4852         \c_false
4853       \else:
4854         \token_if_primitive_p_aux:N #1
4855       \fi:
4856     \else:
4857       \c_false
4858     \fi:
4859   \fi:
4860 }
4861 \def_new:Npn \token_if_primitive_p_aux:N #1{
4862   \if:w \token_if_chardef_p:N #1 \c_false

```

```

4863 \else:
4864 \if:w \token_if_mathchardef_p:N #1 \c_false
4865 \else:
4866 \if:w \token_if_int_register_p:N #1 \c_false
4867 \else:
4868 \if:w \token_if_skip_register_p:N #1 \c_false
4869 \else:
4870 \if:w \token_if_dim_register_p:N #1 \c_false
4871 \else:
4872 \if:w \token_if_toks_register_p:N #1 \c_false
4873 \else:

```

We made it!

```

4874 \c_true
4875 \fi:
4876 \fi:
4877 \fi:
4878 \fi:
4879 \fi:
4880 \fi:
4881 }
4882 \def_test_function_new:npn {token_if_primitive:N} #1{
4883 \if:w\token_if_primitive_p:N#1}

```

24.4.3 Peeking ahead at the next token

```

\l_peek_token We define some other tokens which will initially be the character ?.
\g_peek_token
\l_peek_search_token
4884 \token_new:Nn \l_peek_token {?}
4885 \token_new:Nn \g_peek_token {?}
4886 \token_new:Nn \l_peek_search_token {?}

```

```

\peek_after:NN \peek_after:NN takes two argument where the first is a function acting on \l_peek_token
\peek_gafter:NN and the second is the next token in the input stream which \l_peek_token is set equal
to. \peek_gafter:NN does the same globally to \g_peek_token.

```

```

4887 \def_new:Npn \peek_after:NN {\tex_futurelet:D \l_peek_token }
4888 \def_new:Npn \peek_gafter:NN {
4889 \pref_global:D \tex_futurelet:D \g_peek_token
4890 }

```

For normal purposes there are four main cases:

1. peek at the next token.
2. peek at the next non-space token.
3. peek at the next token and remove it.
4. peek at the next non-space token and remove it.

The generic functions will take four arguments: The token to search for, the test function to run on it and the true/false cases. The general algorithm is this:

1. Store the token to search for in `\l_peek_search_token`.
2. In order to avoid doubling of hash marks where it seems unnatural we put the `<true>` and `<false>` cases through an `x` type expansion but using `\exp_not:n` to avoid any expansion. This has the same effect as putting it through a `<toks>` register but is faster. Also put in a special alignment safe group end.
3. Put in an alignment safe group begin.
4. Peek ahead and call the function which will act on the next token in the input stream.

`\l_peek_true_tlp` Two dedicated token list pointers that store the true and false cases.

```
\l_peek_false_tlp
4891 \tlp_new:Nn \l_peek_true_tlp {}
4892 \tlp_new:Nn \l_peek_false_tlp {}
```

`\peek_tmp:w` Scratch function used for storing the token to be removed if found.

```
4893 \def_new:Npn \peek_tmp:w{}
```

`\l_peek_search_tlp` We also use this token list pointer for storing the token we want to compare. This turns out to be useful.

```
4894 \tlp_new:Nn \l_peek_search_tlp{}
```

`\peek_token_generic:NNTF` #1 is the function to execute (obey or ignore spaces, etc.), #2 is the special token we're looking for, and #3 and #4 are the `<true>` and `<false>` branches.

```
4895 \def_long_new:Npn \peek_token_generic:NNTF #1#2#3#4{
4896   \let:NN \l_peek_search_token #2
4897   \tlp_set:Nn \l_peek_search_tlp {#2}
4898   \tlp_set:Nx \l_peek_true_tlp {\exp_not:n{\group_align_safe_end: #3}}
4899   \tlp_set:Nx \l_peek_false_tlp {\exp_not:n{\group_align_safe_end: #4}}
4900   \group_align_safe_begin:
4901     \peek_after:NN #1
4902 }
```

`\peek_token_remove_generic:NNTF` If we want to be able to remove any character from the input stream we might as well do it the same way for all characters so we define this as little differently from above.

```
4903 \def_long_new:Npn \peek_token_remove_generic:NNTF #1#2#3#4{
4904   \let:NN \l_peek_search_token #2
4905   \tlp_set:Nn \l_peek_search_tlp {#2}
4906   \tlp_set:Nx \l_peek_true_aux_tlp {\exp_not:n{ #3 } }
4907   \tlp_set_eq:NN \l_peek_true_tlp \c_peek_true_remove_next_tlp
4908   \tlp_set:Nx \l_peek_false_tlp {\exp_not:n{\group_align_safe_end: #4}}
4909   \group_align_safe_begin:
4910     \peek_after:NN #1
4911 }
```

`\l_peek_true_aux_tlp` Two token list pointers to help with removing the character from the input stream.
`\l_peek_true_remove_next_tlp`

```
4912 \tlp_new:Nn \l_peek_true_aux_tlp {}
4913 \tlp_new:Nn \c_peek_true_remove_next_tlp {\group_align_safe_end:
4914 \tex_afterassignment:D \l_peek_true_aux_tlp \let:NN \peek_tmp:w
4915 }
```

`\peek_execute_braches_meaning:` There are three major tests between tokens in T_EX: meaning, catcode and charcode.
`\peek_execute_braches_catcode:` Hence we define three basic test functions that set in after the ignoring phase is over and
`\peek_execute_braches_charcode:` done with.

`\peek_execute_braches_charcode_aux:NN`

```
4916 \def_new:Npn \peek_execute_braches_meaning: {
4917 \if_meaning:NN \l_peek_token \l_peek_search_token
4918 \exp_after:NN \l_peek_true_tlp
4919 \else:
4920 \exp_after:NN \l_peek_false_tlp
4921 \fi:
4922 }
4923 \def_new:Npn \peek_execute_braches_catcode: {
4924 \if_catcode:w \exp_not:N\l_peek_token \exp_not:N\l_peek_search_token
4925 \exp_after:NN \l_peek_true_tlp
4926 \else:
4927 \exp_after:NN \l_peek_false_tlp
4928 \fi:
4929 }
```

For the charcode version we do things a little differently. We want to check the token directly but if we do this we face problems if the next thing in the input stream is a braced group or a space token. The braced group would be read as a complete argument and the space would be gobbled by T_EX's argument reading routines. Hence we test for both of these and if one of them is found we just execute the false result directly since no one should ever try to use the `charcode` function for searching for `\c_group_begin_token` or `\c_space_token`.

```
4930 \def_new:Npn \peek_execute_braches_charcode: {
4931 \predicate:nTF {
4932 \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token ||
4933 \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
4934 }
4935 { \l_peek_false_tlp } }
```

Otherwise we call a small auxiliary function that just grabs the next token. We can do that because it really is a single token; we just have insert it again afterwards. Also we stored the token we were looking for in the token list pointer `\l_peek_search_tlp` so we unpack it again for this function.

```
4936 { \exp_after:NN \peek_execute_braches_charcode_aux:NN \l_peek_search_tlp }
4937 }
```

Then we just do the usual `\if_charcode:w` comparison. We also remember to insert #2 again after executing the true or false branches.

```
4938 \def_long_new:Npn \peek_execute_braches_charcode_aux:NN #1#2{
4939 \if_charcode:w \exp_not:N #1\exp_not:N#2
```

```

4940     \exp_after:NN \l_peek_true_tlp
4941     \else:
4942     \exp_after:NN \l_peek_false_tlp
4943     \fi:
4944     #2
4945 }

\peek_meaning:NTF Here we use meaning comparison with \if_meaning:NN.
\peek_meaning_ignore_spaces:NTF
\peek_meaning_remove:NTF 4946 \def_new:Npn \peek_meaning:NTF {
\meaning_remove_ignore_spaces:NTF 4947 \peek_token_generic:NNTF \peek_execute_branches_meaning:
4948 }
4949 \def_new:Npn \peek_meaning_ignore_spaces:NTF {
4950 \let:NN \peek_execute_branches: \peek_execute_branches_meaning:
4951 \peek_token_generic:NNTF \peek_ignore_spaces_execute_branches:
4952 }
4953 \def_new:Npn \peek_meaning_remove:NTF {
4954 \peek_token_remove_generic:NNTF \peek_execute_branches_meaning:
4955 }
4956 \def_new:Npn \peek_meaning_remove_ignore_spaces:NTF {
4957 \let:NN \peek_execute_branches: \peek_execute_branches_meaning:
4958 \peek_token_remove_generic:NNTF \peek_ignore_spaces_execute_branches:
4959 }

\peek_catcode:NTF Here we use catcode comparison with \if_catcode:w.
\peek_catcode_ignore_spaces:NTF
\peek_catcode_remove:NTF 4960 \def_new:Npn \peek_catcode:NTF {
\code_remove_ignore_spaces:NTF 4961 \peek_token_generic:NNTF \peek_execute_branches_catcode:
4962 }
4963 \def_new:Npn \peek_catcode_ignore_spaces:NTF {
4964 \let:NN \peek_execute_branches: \peek_execute_branches_catcode:
4965 \peek_token_generic:NNTF \peek_ignore_spaces_execute_branches:
4966 }
4967 \def_new:Npn \peek_catcode_remove:NTF {
4968 \peek_token_remove_generic:NNTF \peek_execute_branches_catcode:
4969 }
4970 \def_new:Npn \peek_catcode_remove_ignore_spaces:NTF {
4971 \let:NN \peek_execute_branches: \peek_execute_branches_catcode:
4972 \peek_token_remove_generic:NNTF \peek_ignore_spaces_execute_branches:
4973 }

\peek_charcode:NTF Here we use charcode comparison with \if_charcode:w.
\peek_charcode_ignore_spaces:NTF
\peek_charcode_remove:NTF 4974 \def_new:Npn \peek_charcode:NTF {
\code_remove_ignore_spaces:NTF 4975 \peek_token_generic:NNTF \peek_execute_branches_charcode:
4976 }
4977 \def_new:Npn \peek_charcode_ignore_spaces:NTF {
4978 \let:NN \peek_execute_branches: \peek_execute_branches_charcode:
4979 \peek_token_generic:NNTF \peek_ignore_spaces_execute_branches:
4980 }
4981 \def_new:Npn \peek_charcode_remove:NTF {
4982 \peek_token_remove_generic:NNTF \peek_execute_branches_charcode:
4983 }
4984

```



```

4985 \def_new:Npn \peek_charcode_remove_ignore_spaces:NTF {
4986   \let:NN \peek_execute_branches: \peek_execute_branches_charcode:
4987   \peek_token_remove_generic:NNTF \peek_ignore_spaces_execute_branches:
4988 }

```

`\peek_ignore_spaces_aux:` Throw away a space token and search again. We could define this in a more devious way where the auxiliary function gobbles the space token but then what do we do if we decide that a certain function should ignore more than one specific token? For example someone might find it interesting to define a `\peek_` function that ignores a’s and b’s! Or maybe different kinds of “funny spaces”... Therefore I have decided to use this version which uses `\tex_afterassignment:D` to call the auxiliary function after the next token has been removed by `\let:NN`. That way it is easily extensible.

```

4989 \def_new:Npn \peek_ignore_spaces_aux: {
4990   \peek_after:NN \peek_ignore_spaces_execute_branches:
4991 }
4992 \def_new:Npn \peek_ignore_spaces_execute_branches: {
4993   \token_if_eq_meaning:NNTF \l_peek_token \c_space_token
4994   { \tex_afterassignment:D \peek_ignore_spaces_aux:
4995     \let:NN \peek_tmp:w
4996   }
4997   \peek_execute_branches:
4998 }

4999 </initex | package>
5000 < *showmemory>
5001 \showMemUsage
5002 </showmemory>

```

25 Cross references

`\xref_set_label:n` `\xref_set_label:n {<name>}`

Sets a label in the text. Note that this function does not do anything else than setting the correct labels. In particular, it does not try to fix any spacing around the write node; this is a task for the `galley2` module.

`\xref_new:nn` `\xref_new:nn {<type>} {<value>}`

Defines a new cross reference type `<type>`. This defines the token list pointer `\l_xref_curr_<type>_tlp` with default value `<value>` which gets written fully expanded when `\xref_set_label:n` is called.

`\xref_deferred_new:nn` `\xref_deferred_new:nn {<type>} {<value>}`

Same as `\xref_new:n` except for this one, the value written happens when `TEX` ships out the page. Page numbers use this one obviously.

`\xref_get_value:nn` `\xref_get_value:nn {<type>} {<name>}`

Extracts the cross reference information of type `<type>` for the label `<name>`. This operation is expandable.

25.1 Implementation

We start by ensuring that the required packages are loaded.

```

5003 <*package>
5004 \ProvidesExplPackage
5005   {\filename}{\filedate}{\fileversion}{\filedescription}
5006 \RequirePackage{l3quark}
5007 \RequirePackage{l3toks}
5008 \RequirePackage{l3io}
5009 \RequirePackage{l3prop}
5010 \RequirePackage{l3int}
5011 \RequirePackage{l3token}
5012 </package>
5013 <*initex | package>

```

There are two kinds of information, namely information which is *immediate* like a section title and then there's *deferred* information like page numbers. Each reference type belong to one of these categories, which we save internally as the property lists `\g_xref_all_curr_immediate_fields_plist` and `\g_xref_all_curr_deferred_fields_plist` and the reference type `<xyz>` exists as the key-info pair `\xref_<xyz>_key {\l_xref_curr_<xyz>_tlp}` on one of these lists. This way each new entry type is just added as another key-info pair.

When the cross references are generated at the beginning of the document each will turn into a control sequence. Thus `\label{mylab}` will internally refer to the property list `\g_xref_mylab_plist`.

The extraction of values from this property list can be done in several different ways but we want to keep the operation expandable. Therefore we use a dedicated function for each type of cross reference, which looks like this:

```
\xref_get_value_xyz_aux:w -> #1 \xref_xyz_key #2#3\q_nil{#2}
```

This will throw away all the bits we don't need. In case `xyz` is the first on the `mylab` property list `#1` is empty, if it's the last key-info pair `#3` is empty. The value of the field can be extracted with the function `\xref_get_value:nn` where the first argument is the type and the second the label name so here it would be `\xref_get_value:nn {xyz}{mylab}`.

`\l_curr_immediate_fields_plist` The two main property lists for storing information. They contain key-info pairs for all
`\l_curr_deferred_fields_plist` known types.

```

5014 \prop_new:N \g_xref_all_curr_immediate_fields_plist
5015 \prop_new:N \g_xref_all_curr_deferred_fields_plist

```

`\xref_new:nn` Setting up a new cross reference type is fairly straight forward when we follow the game
`\xref_deferred_new:nn` plan mentioned earlier.

```

\xref_new_aux:nnn
5016 \def_new:Npn \xref_new:nn {\xref_new_aux:nnn{immediate}}
5017 \def_new:Npn \xref_deferred_new:nn {\xref_new_aux:nnn{deferred}}
5018 \def_new:Npn \xref_new_aux:nnn #1#2#3{

```

First put the new type in the relevant property list.

```
5019 \prop_gput:ccx {g_xref_all_curr_ #1 _fields_plist}
5020 { xref_ #2 _key }
5021 { \exp_not:c {l_xref_curr_#2_tlp }}}
```

Then define the key to be a protected macro.¹⁴

```
5022 \def_protected:cpn { xref_#2_key }{}
5023 \tlp_new:cn{l_xref_curr_#2_tlp}{#3}
```

Now for the function extracting the value of a reference. We could do this with a simple `\prop_if_in` thing put since we want to do things in an expandable way we make a separate grabber for each type—this is also faster. The grabber function can be defined by using an intricate construction of `\exp_after:NN` and other goodies but I prefer readable code. The end result for the input `xyz` is

```
\def:Npn\xref_get_value_xyz_aux:w #1\xref_xyz_key #2#3\q_nil{#2}
```

```
5024 \toks_set:Nx \l_tmpa_toks {
5025   \exp_not:n { \def:cpn {xref_get_value_#2_aux:w} ##1 }
5026   \exp_not:c { xref_#2_key }
5027 }
5028 \toks_use:N \l_tmpa_toks ##2 ##3\q_nil {##2}
5029 }
```

`\xref_get_value:nn` Getting the correct value for a given label-type pair is a matter of connecting the correct grabber functions and property list.

```
5030 \def_new:Npn \xref_get_value:nn #1#2 {
5031   \cs_if_really_free:cTF{g_xref_#2_plist}
5032   {??}
5033   {
```

This next expansion may look a little weird but it isn't if you think about it!

```
5034   \exp_args:NcNc \exp_after:NN {xref_get_value_#1_aux:w}
5035   \prop_use:N {g_xref_#2_plist}
```

Better put in the stop marker.

```
5036   \q_nil
5037 }
5038 }
5039 \def:NNn \exp_after:cc 2 {
5040   \exp_after:NN \exp_after:NN
5041   \cs:w #1\exp_after:NN\cs_end: \cs:w #2\cs_end:
5042 }
```

`\xref_define_label:nn` Define the property list for each label. We better do this in two steps because the special catcode regime is in effect and since some of the info fields are very likely to contain actual text, we better make sure spaces aren't ignored! As for the meaning of other characters

¹⁴We could also set it equal to `\scan_stop:` but this just feels “cleaner”.

then it is a possibility to also have a field containing catcode instructions which can then be activated with `\etex_scantokens:D`.

```

5043 \def_protected_new:Npn \xref_define_label:nn {
5044   \group_begin:
5045     \char_set_catcode:nn {'\ } \c_ten
5046     \xref_define_label_aux:nn
5047 }

```

If the label is already taken we have a multiply defined label and we should do something about it. For now we don't do anything spectacular.

```

5048 \def_new:Npn \xref_define_label_aux:nn #1#2 {
5049   \cs_if_really_free:cTF{g_xref_#1_plist}
5050   {\prop_new:c{g_xref_#1_plist}}{\WARNING}
5051   \toks_gset:cn{g_xref_#1_plist}{#2}
5052   \group_end:
5053 }

```

`\xref_set_label:n` Then the generic command for setting a label. We expand the immediate labels fully before calling the write function but make sure the deferred fields aren't expanded just yet. Due to property lists being implemented as token list registers we must expand the 'immediate' fields twice.

```

5054 \def:Npn \xref_set_label:n #1{
5055   \def:Npx \tmp:w{\prop_use:N \g_xref_all_curr_immediate_fields_plist}
5056   \exp_args:NNx \iow_deferred_expanded:Nn \xref_write{
5057     \xref_define_label:nn {#1} {
5058       \tmp:w
5059       \prop_use:N \g_xref_all_curr_deferred_fields_plist
5060     }
5061   }
5062 }

```

`\xref_write` A stream for writing cross references although they do not require to be in a separate file.

```

5063 \iow_new:N \xref_write

```

That's it (for now).

```

5064 </initex | package>
5065 < *showmemory>
5066 \showMemUsage
5067 </showmemory>

5068 < *testfile>
5069 \documentclass{article}
5070 \usepackage{l3xref}
5071 \ExplSyntaxOn
5072 \def:Npn \startrecording {\iow_open:Nn \xref_write {\jobname.xref}}
5073 \def:Npn \DefineCrossReferences {
5074   \group_begin:
5075     \NamesStart

```

```

5076 \InputIfFileExists{\jobname.xref}{\{}}
5077 \group_end:
5078 }
5079 \AtBeginDocument{\DefineCrossReferences\startrecording}
5080
5081 \xref_new:nn {name}{\}
5082 \def:Npn \setname{\tlp_set:Nn\l_xref_curr_name_tlp}
5083 \def:Npn \getname{\xref_get_value:nn{name}}
5084
5085 \xref_deferred_new:nn {page}{\thepage}
5086 \def:Npn \getpage{\xref_get_value:nn{page}}
5087
5088 \xref_deferred_new:nn {valuepage}{\number\value{page}}
5089 \def:Npn \getvaluepage{\xref_get_value:nn{valuepage}}
5090
5091 \let:NN \setlabel \xref_set_label:n
5092
5093 \ExplSyntaxOff
5094 \begin{document}
5095 \pagenumbering{roman}
5096
5097 Text\setname{This is a name}\setlabel{testlabel1}. More
5098 text\setname{This is another name}\setlabel{testlabel2}. \clearpage
5099
5100 Text\setname{This is a third name}\setlabel{testlabel3}. More
5101 text\setname{Hello World!}\setlabel{testlabel4}. \clearpage
5102
5103 \pagenumbering{arabic}
5104
5105 Text\setname{Name 5}\setlabel{testlabel5}. More text\setname{Name
5106 6}\setlabel{testlabel6}. \clearpage
5107
5108 Text\setname{Name 7}\setlabel{testlabel 7}. More text\setname{Name
5109 8}\setlabel{testlabel8}. \clearpage
5110
5111 Now let's extract some values. \getname{testlabel1} on page
5112 \getpage{testlabel1} with value \getvaluepage{testlabel1}.
5113
5114 Now let's extract some values. \getname{testlabel 7} on page
5115 \getpage{testlabel 7} with value \getvaluepage{testlabel 7}.
5116 \end{document}
5117 \end{testfile}

```

26 Infix notation arithmetic

This is pretty much a straight adaption of the `calc` package and as such has same syntax for the *calc expression*. However, there are some noticeable differences.

- The `calc` expression is expanded fully, which means there are no problems with unfinished conditionals. However, the contents of `\widthof` etc. is not expanded at all. This includes uses in traditional L^AT_EX as in the `array` package, which tries

to do an `\edef` several times. The code used in `l3calc` provides self-protection for these cases.

- Muskip registers are supported although they can only be used in `\ratio` if already evaluating a muskip expression. For the other three register types, you can use points.
- All results are rounded, not truncated. More precisely, the primitive `TEX` operations `\divide` and `\multiply` are not used. The only instance where one will observe an effect is when dividing integers.

This version of `l3calc` is now a complete replacement for the original `calc` package providing the same functionality and will prevent the original `calc` package from loading.

<code>\calc_int_set:Nn</code>
<code>\calc_int_gset:Nn</code>
<code>\calc_int_add:Nn</code>
<code>\calc_int_gadd:Nn</code>
<code>\calc_int_sub:Nn</code>
<code>\calc_int_gsub:Nn</code>

`\calc_int_set:Nn <int> {<calc expression>}`

Evaluates `<calc expression>` and either adds or subtracts it from `<int>` or sets `<int>` to it. These operations can also be global.

<code>\calc_dim_set:Nn</code>
<code>\calc_dim_gset:Nn</code>
<code>\calc_dim_add:Nn</code>
<code>\calc_dim_gadd:Nn</code>
<code>\calc_dim_sub:Nn</code>
<code>\calc_dim_gsub:Nn</code>

`\calc_dim_set:Nn <dim> {<calc expression>}`

Evaluates `<calc expression>` and either adds or subtracts it from `<dim>` or sets `<dim>` to it. These operations can also be global.

<code>\calc_skip_set:Nn</code>
<code>\calc_skip_gset:Nn</code>
<code>\calc_skip_add:Nn</code>
<code>\calc_skip_gadd:Nn</code>
<code>\calc_skip_sub:Nn</code>
<code>\calc_skip_gsub:Nn</code>

`\calc_skip_set:Nn <skip> {<calc expression>}`

Evaluates `<calc expression>` and either adds or subtracts it from `<skip>` or sets `<skip>` to it. These operations can also be global.

<code>\calc_muskip_set:Nn</code>
<code>\calc_muskip_gset:Nn</code>
<code>\calc_muskip_add:Nn</code>
<code>\calc_muskip_gadd:Nn</code>
<code>\calc_muskip_sub:Nn</code>
<code>\calc_muskip_gsub:Nn</code>

`\calc_muskip_set:Nn <muskip> {<calc expression>}`

Evaluates $\langle calc\ expression \rangle$ and either adds or subtracts it from $\langle muskip \rangle$ or sets $\langle muskip \rangle$ to it. These operations can also be global.

```
\calc_calculate_box_size:nnn {\dim-set}
{\langle item 1 \rangle \langle item 2 \rangle ... \langle item n \rangle} {\langle contents \rangle}
```

Sets $\langle contents \rangle$ in a temporary box $\backslash l_tmpa_box$. Then $\langle dim-set \rangle$ is put in front of a loop that inserts $+\langle item_i \rangle$ in front of $\backslash l_tmpa_box$ and this is evaluated. For instance, if we wanted to determine the total height of the text `xyz` and store it in $\backslash l_tmpa_dim$, we would call it as.

```
\calc_calculate_box_size:nnn
{\dim_set:Nn\l_tmpa_dim}{\box_ht:N\box_dp:N}{xyz}
```

Similarly, if we wanted the difference between height and depth, we could call it as

```
\calc_calculate_box_size:nnn
{\dim_set:Nn\l_tmpa_dim}{\box_ht:N{-\box_dp:N}}{xyz}
```

26.1 The Implementation

Since this is basically a re-worked version of the `calc` package, I haven't bothered with too many comments except for in the places where this package differs. This may (and should) change at some point.

We start by ensuring that the required packages are loaded.

```
5118 \*package>
5119 \ProvidesExplPackage
5120   {\filename}{\filedate}{\fileversion}{\filedescription}
5121 \RequirePackage{l3int}
5122 \RequirePackage{l3skip}
5123 \RequirePackage{l3box}
5124 \</package>
5125 \*initex | package>
```

```
\l_calc_expression_tlp Here we define some registers and pointers we will need.
  \g_calc_A_register
  \l_calc_B_register 5126 \tlp_new:Nn\l_calc_expression_tlp{}
\l_calc_current_type_int 5127 \def_new:Npn \g_calc_A_register{}
5128 \def_new:Npn \l_calc_B_register{}
5129 \int_new:N \l_calc_current_type_int
```

```
\g_calc_A_int For each type of register we will need three registers to do our manipulations.
\l_calc_B_int
\l_calc_C_int 5130 \int_new:N \g_calc_A_int
\g_calc_A_dim 5131 \int_new:N \l_calc_B_int
\l_calc_B_dim 5132 \int_new:N \l_calc_C_int
\l_calc_C_dim 5133 \dim_new:N \g_calc_A_dim
\g_calc_A_skip
\l_calc_B_skip
\l_calc_C_skip
\g_calc_A_muskip
\l_calc_B_muskip
\l_calc_C_muskip
```

```

5134 \dim_new:N \l_calc_B_dim
5135 \dim_new:N \l_calc_C_dim
5136 \skip_new:N \g_calc_A_skip
5137 \skip_new:N \l_calc_B_skip
5138 \skip_new:N \l_calc_C_skip
5139 \muskip_new:N \g_calc_A_muskip
5140 \muskip_new:N \l_calc_B_muskip
5141 \muskip_new:N \l_calc_C_muskip

```

\calc_assign_generic:NNNNnn The generic function. #1 is a number denoting which type we are doing. (0=int, 1=dim, 2=skip, 3=muskip), #2 = temp register A, #3 = temp register B, #4 is a function acting on #5 which is the register to be set. #6 is the calc expression. We do a little extra work so that \real and \ratio can still be used by the user.

```

5142 \def_long_new:Npn \calc_assign_generic:NNNNnn#1#2#3#4#5#6{
5143   \let:NN\g_calc_A_register#2
5144   \let:NN\l_calc_B_register#3
5145   \int_set:Nn \l_calc_current_type_int {#1}
5146   \group_begin:
5147     \let:NN \real \calc_real:n
5148     \let:NN \ratio\calc_ratio:nn
5149     \tlp_set:Nx\l_calc_expression_tlp{#6}
5150     \exp_after:NN
5151   \group_end:
5152   \exp_after:NN\calc_open:w\exp_after:NN(\l_calc_expression_tlp !
5153   \pref_global:D\g_calc_A_register\l_calc_B_register
5154   \group_end:
5155   #4{#5}\l_calc_B_register
5156 }

```

A simpler version relying on \real and \ratio having our definition is

```

\def_long_new:Npn \calc_assign_generic:NNNNnn#1#2#3#4#5#6{
  \let:NN\g_calc_A_register#2\let:NN\l_calc_B_register#3
  \int_set:Nn \l_calc_current_type_int {#1}
  \tlp_set:Nx\l_calc_expression_tlp{#6}
  \exp_after:NN\calc_open:w\exp_after:NN(\l_calc_expression_tlp !
  \pref_global:D\g_calc_A_register\l_calc_B_register
  \group_end:
  #4{#5}\l_calc_B_register
}

```

\calc_int_set:Nn Here are the individual versions for the different register types. First integer registers.

```

\calc_int_gset:Nn
\calc_int_add:Nn 5157 \def_new:Npn\calc_int_set:Nn{
\calc_int_gadd:Nn 5158   \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_set:Nn
5159 }
\calc_int_sub:Nn 5160 \def_new:Npn\calc_int_gset:Nn{
\calc_int_gsub:Nn 5161   \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_gset:Nn
5162 }
5163 \def_new:Npn\calc_int_add:Nn{
5164   \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_add:Nn
5165 }

```



```

5166 \def_new:Npn\calc_int_gadd:Nn{
5167   \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_gadd:Nn
5168 }
5169 \def_new:Npn\calc_int_sub:Nn{
5170   \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_sub:Nn
5171 }
5172 \def_new:Npn\calc_int_gsub:Nn{
5173   \calc_assign_generic:NNNNnn\c_zero\g_calc_A_int\l_calc_B_int\int_gsub:Nn
5174 }

```

\calc_dim_set:Nn Dimens.

```

\calc_dim_gset:Nn
\calc_dim_add:Nn 5175 \def_new:Npn\calc_dim_set:Nn{
\calc_dim_gadd:Nn 5176   \calc_assign_generic:NNNNnn\c_one\g_calc_A_dim\l_calc_B_dim\dim_set:Nn
5177 }
\calc_dim_sub:Nn 5178 \def_new:Npn\calc_dim_gset:Nn{
\calc_dim_gsub:Nn 5179   \calc_assign_generic:NNNNnn\c_one\g_calc_A_dim\l_calc_B_dim\dim_gset:Nn
5180 }
5181 \def_new:Npn\calc_dim_add:Nn{
5182   \calc_assign_generic:NNNNnn\c_one\g_calc_A_dim\l_calc_B_dim\dim_add:Nn
5183 }
5184 \def_new:Npn\calc_dim_gadd:Nn{
5185   \calc_assign_generic:NNNNnn\c_one\g_calc_A_dim\l_calc_B_dim\dim_gadd:Nn
5186 }
5187 \def_new:Npn\calc_dim_sub:Nn{
5188   \calc_assign_generic:NNNNnn\c_one\g_calc_A_int\l_calc_B_int\dim_sub:Nn
5189 }
5190 \def_new:Npn\calc_dim_gsub:Nn{
5191   \calc_assign_generic:NNNNnn\c_one\g_calc_A_int\l_calc_B_int\dim_gsub:Nn
5192 }

```

\calc_skip_set:Nn Skips.

```

\calc_skip_gset:Nn
\calc_skip_add:Nn 5193 \def_new:Npn\calc_skip_set:Nn{
\calc_skip_gadd:Nn 5194   \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_set:Nn
5195 }
\calc_skip_sub:Nn 5196 \def_new:Npn\calc_skip_gset:Nn{
\calc_skip_gsub:Nn 5197   \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_gset:Nn
5198 }
5199 \def_new:Npn\calc_skip_add:Nn{
5200   \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_add:Nn
5201 }
5202 \def_new:Npn\calc_skip_gadd:Nn{
5203   \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_gadd:Nn
5204 }
5205 \def_new:Npn\calc_skip_sub:Nn{
5206   \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_sub:Nn
5207 }
5208 \def_new:Npn\calc_skip_gsub:Nn{
5209   \calc_assign_generic:NNNNnn\c_two\g_calc_A_skip\l_calc_B_skip\skip_gsub:Nn
5210 }

```

\calc_muskip_set:Nn Muskips.

```

\calc_muskip_gset:Nn
\calc_muskip_add:Nn
\calc_muskip_gadd:Nn
\calc_muskip_sub:Nn
\calc_muskip_gsub:Nn

```

```

5211 \def_new:Npn\calc_muskip_set:Nn{
5212   \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
5213   \muskip_set:Nn
5214 }
5215 \def_new:Npn\calc_muskip_gset:Nn{
5216   \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
5217   \muskip_gset:Nn
5218 }
5219 \def_new:Npn\calc_muskip_add:Nn{
5220   \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
5221   \muskip_add:Nn
5222 }
5223 \def_new:Npn\calc_muskip_gadd:Nn{
5224   \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
5225   \muskip_gadd:Nn
5226 }
5227 \def_new:Npn\calc_muskip_sub:Nn{
5228   \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
5229   \muskip_add:Nn
5230 }
5231 \def_new:Npn\calc_muskip_gsub:Nn{
5232   \calc_assign_generic:NNNNnn\c_three\g_calc_A_muskip\l_calc_B_muskip
5233   \muskip_gadd:Nn
5234 }

```

\calc_pre_scan:N In case we found one of the special operations, this should just be executed.

```

5235 \def_new:Npn \calc_pre_scan:N #1{
5236   \if_meaning:NN(#1
5237   \exp_after:NN\calc_open:w
5238   \else:
5239   \if_meaning:NN \calc_textsize:Nn #1
5240   \else:
5241   \if_meaning:NN \calc_maxmin_operation:Nnn #1
5242   \else:

```

\calc_numeric: uses a primitive assignment so doesn't care about these dangling \fi:s.

```

5243   \calc_numeric:
5244   \fi:
5245   \fi:
5246   \fi:
5247   #1}

```

\calc_open:w

```

5248 \def_new:Npn \calc_open:w({
5249   \group_begin:\group_execute_after:N\calc_init_B:
5250   \group_begin:\group_execute_after:N\calc_init_B:
5251   \calc_pre_scan:N
5252 }

```

```

\calc_init_B:
\calc_numeric:
\calc_close:

```

```

5253 \def_new:Npn\calc_init_B:{\l_calc_B_register\g_calc_A_register}
5254 \def_new:Npn\calc_numeric:{
5255   \tex_afterassignment:D\calc_post_scan:N
5256   \pref_global:D\g_calc_A_register
5257 }
5258 \def_new:Npn\calc_close:{
5259   \group_end:\pref_global:D\g_calc_A_register\l_calc_B_register
5260   \group_end:\pref_global:D\g_calc_A_register\l_calc_B_register
5261   \calc_post_scan:N}

```

\calc_post_scan:N Look at what token we have and decide where to go.

```

5262 \def_new:Npn\calc_post_scan:N#1{
5263   \if_meaning:NN#1!\let:NN\calc_next:w\group_end: \else:
5264     \if_meaning:NN#1+\let:NN\calc_next:w\calc_add: \else:
5265       \if_meaning:NN#1-\let:NN\calc_next:w\calc_subtract: \else:
5266         \if_meaning:NN#1*\let:NN\calc_next:w\calc_multiply:N \else:
5267           \if_meaning:NN#1/\let:NN\calc_next:w\calc_divide:N \else:
5268             \if_meaning:NN#1)\let:NN\calc_next:w\calc_close: \else:
5269               \if_meaning:NN#1\scan_stop:\let:NN\calc_next:w\calc_post_scan:N
5270             \else:

```

If we get here, there is an error but let's also disable \calc_next:w since it is otherwise undefined. No need to give extra errors just for that.

```

5271           \let:NN \calc_next:w \use_noop:
5272           \calc_error:N#1
5273         \fi:
5274       \fi:
5275     \fi:
5276   \fi:
5277 \fi:
5278 \fi:
5279 \fi:
5280 \calc_next:w}

```

\calc_multiply:N The switches for multiplication and division.

\calc_divide:N

```

5281 \def_new:Npn \calc_multiply:N #1{
5282   \if_meaning:NN \calc_maxmin_operation:Nnn #1
5283   \let:NN \calc_next:w \calc_maxmin_multiply:
5284   \else:
5285     \if_meaning:NN \calc_ratio_multiply:nn #1
5286     \let:NN \calc_next:w \calc_ratio_multiply:nn
5287   \else:
5288     \if_meaning:NN \calc_real_evaluate:nn #1
5289     \let:NN \calc_next:w \calc_real_multiply:n
5290   \else:
5291     \def:Npn \calc_next:w{\calc_multiply: #1}
5292   \fi:
5293 \fi:
5294 \fi:
5295 \calc_next:w
5296 }

```

```

5297 \def_new:Npn \calc_divide:N #1{
5298   \if_meaning:NN \calc_maxmin_operation:Nnn #1
5299     \let:NN \calc_next:w \calc_maxmin_divide:
5300   \else:
5301     \if_meaning:NN \calc_ratio_multiply:nn #1
5302       \let:NN \calc_next:w \calc_ratio_divide:nn
5303     \else:
5304       \if_meaning:NN \calc_real_evaluate:nn #1
5305         \let:NN \calc_next:w \calc_real_divide:n
5306       \else:
5307         \def:Npn \calc_next:w{\calc_divide: #1}
5308       \fi:
5309     \fi:
5310   \fi:
5311   \calc_next:w
5312 }

```

\calc_generic_add:N Here is how we add and subtract.

```

\calc_add:
\calc_subtract: 5313 \def_new:Npn\calc_generic_add_or_subtract:N#1{
\calc_add_A_to_B: 5314   \group_end:
\calc_subtract_A_from_B: 5315   \pref_global:D\g_calc_A_register\l_calc_B_register\group_end:
5316   \group_begin:\group_execute_after:N#1\group_begin:
5317   \group_execute_after:N\calc_init_B:
5318   \calc_pre_scan:N}
5319 \def_new:Npn\calc_add:{\calc_generic_add_or_subtract:N\calc_add_A_to_B:}
5320 \def_new:Npn\calc_subtract:{
5321   \calc_generic_add_or_subtract:N\calc_subtract_A_from_B:}

```

Don't use \tex_advance:D since it allows overflows.

```

5322 \def_new:Npn\calc_add_A_to_B:{
5323   \l_calc_B_register
5324   \if_case:w\l_calc_current_type_int
5325     \etex_numexpr:D\or:
5326     \etex_dimexpr:D\or:
5327     \etex_glueexpr:D\or:
5328     \etex_muexpr:D\fi:
5329   \l_calc_B_register + \g_calc_A_register\scan_stop:
5330 }
5331 \def_new:Npn\calc_subtract_A_from_B:{
5332   \l_calc_B_register
5333   \if_case:w\l_calc_current_type_int
5334     \etex_numexpr:D\or:
5335     \etex_dimexpr:D\or:
5336     \etex_glueexpr:D\or:
5337     \etex_muexpr:D\fi:
5338   \l_calc_B_register - \g_calc_A_register\scan_stop:
5339 }

```

generic_multiply_or_divide:N And here is how we multiply and divide. Note that we do not use the primitive TeX operations but the expandable operations provided by ϵ -TeX. This means that all results are rounded not truncated!

```

\calc_multiply_B_by_A:
\calc_divide_B_by_A:
\calc_multiply:
\calc_divide:

```

```

5340 \def_new:Npn\calc_generic_multiply_or_divide:N#1{
5341   \group_end:
5342   \group_begin:
5343   \let:NN\g_calc_A_register\g_calc_A_int
5344   \let:NN\l_calc_B_register\l_calc_B_int
5345   \int_zero:N \l_calc_current_type_int
5346   \group_execute_after:N#1\calc_pre_scan:N
5347 }
5348 \def_new:Npn\calc_multiply_B_by_A:{
5349   \l_calc_B_register
5350   \if_case:w\l_calc_current_type_int
5351   \etex_numexpr:D\or:
5352   \etex_dimexpr:D\or:
5353   \etex_glueexpr:D\or:
5354   \etex_muexpr:D\fi:
5355   \l_calc_B_register*\g_calc_A_int\scan_stop:
5356 }
5357 \def_new:Npn\calc_divide_B_by_A:{
5358   \l_calc_B_register
5359   \if_case:w\l_calc_current_type_int
5360   \etex_numexpr:D\or:
5361   \etex_dimexpr:D\or:
5362   \etex_glueexpr:D\or:
5363   \etex_muexpr:D\fi:
5364   \l_calc_B_register/\g_calc_A_int\scan_stop:
5365 }
5366 \def_new:Npn\calc_multiply:{
5367   \calc_generic_multiply_or_divide:N\calc_multiply_B_by_A:}
5368 \def_new:Npn\calc_divide:{
5369   \calc_generic_multiply_or_divide:N\calc_divide_B_by_A:}

```

\calc_calculate_box_size:nnn Put something in a box and measure it. #1 is a list of \box_ht:N etc., #2 should be
 \calc_calculate_box_size_aux:n \dim_set:Nn<dim register> or \dim_gset:Nn<dim register> and #3 is the contents.

```

5370 \def_long_new:Npn \calc_calculate_box_size:nnn #1#2#3{
5371   \hbox_set:Nn \l_tmpa_box {#{#3}}
5372   #2{\c_zero_dim \tlist_map_function:nN{#1}\calc_calculate_box_size_aux:n}
5373 }

```

Helper for calculating the final dimension.

```

5374 \def:Npn \calc_calculate_box_size_aux:n#1{ + #1\l_tmpa_box}

```

\calc_textsize:Nn Now we can define \calc_textsize:Nn.

```

5375 \def_protected_long:Npn \calc_textsize:Nn#1#2{
5376   \group_begin:
5377   \let:NN\calc_widthof_aux:n\box_wd:N
5378   \let:NN\calc_heightof_aux:n\box_ht:N
5379   \let:NN\calc_depthof_aux:n\box_dp:N
5380   \def:Npn\calc_totalheightof_aux:n{\box_ht:N\box_dp:N}
5381   \exp_args:No\calc_calculate_box_size:nnn{#1}
5382   {\dim_gset:Nn\g_calc_A_register}

```

Restore the four user commands here since there might be a recursive call.

```

5383 {
5384   \let:NN \calc_depthof_aux:n \calc_depthof_auxi:n
5385   \let:NN \calc_widthof_aux:n \calc_widthof_auxi:n
5386   \let:NN \calc_heightof_aux:n \calc_heightof_auxi:n
5387   \let:NN \calc_totalheightof_aux:n \calc_totalheightof_auxi:n
5388   #2
5389 }
5390 \group_end:
5391 \calc_post_scan:N
5392 }

```

`\calc_ratio_multiply:nn` Evaluate a ratio. If we were already evaluation a *⟨muskip⟩* register, the ratio is probably
`\calc_ratio_divide:nn` also done with this type and we'll have to convert them to regular points.

```

5393 \def_protected_long:Npn\calc_ratio_multiply:nn#1#2{
5394   \group_end:\group_begin:
5395   \if_num:w\l_calc_current_type_int < \c_three
5396     \calc_dim_set:Nn\l_calc_B_int{#1}
5397     \calc_dim_set:Nn\l_calc_C_int{#2}
5398   \else:
5399     \calc_dim_muskip:Nn{\l_calc_B_int\etex_mutogluue:D}{#1}
5400     \calc_dim_muskip:Nn{\l_calc_C_int\etex_mutogluue:D}{#2}
5401   \fi:

```

Then store the ratio as a fraction, which we just pass on.

```

5402   \gdef:Npx\calc_calculated_ratio:{
5403     \int_use:N\l_calc_B_int/\int_use:N\l_calc_C_int
5404   }
5405   \group_end:

```

Here we set the new value of `\l_calc_B_register` and remember to evaluate it as the correct type. Note that the intermediate calculation is a scaled operation (meaning the intermediate value is 64-bit) so we don't get into trouble when first multiplying by a large number and then dividing.

```

5406   \l_calc_B_register
5407   \if_case:w\l_calc_current_type_int
5408     \etex_numexpr:D\or:
5409     \etex_dimexpr:D\or:
5410     \etex_glueexpr:D\or:
5411     \etex_muexpr:D\fi:
5412   \l_calc_B_register*\calc_calculated_ratio:\scan_stop:
5413   \group_begin:
5414   \calc_post_scan:N}

```

Division is just flipping the arguments around.

```

5415 \def_long_new:Npn \calc_ratio_divide:nn#1#2{\calc_ratio_multiply:nn{#2}{#1}}

```

`\calc_real_evaluate:nn` Although we could define the `\real` function as a subcase of `\ratio`, this is horribly
`\calc_real_multiply:n` inefficient since we just want to convert the decimal to a fraction.
`\calc_real_divide:n`

```

5416 \def_protected_new:Npn\calc_real_evaluate:nn #1#2{
5417   \group_end:
5418   \l_calc_B_register
5419   \if_case:w\l_calc_current_type_int
5420     \etex_numexpr:D\or:
5421     \etex_dimexpr:D\or:
5422     \etex_glueexpr:D\or:
5423     \etex_muexpr:D\fi:
5424   \l_calc_B_register *
5425     \tex_number:D \dim_eval:n{#1pt}/
5426     \tex_number:D\dim_eval:n{#2pt}
5427   \scan_stop:
5428   \group_begin:
5429   \calc_post_scan:N}
5430 \def_new:Npn \calc_real_multiply:n #1{\calc_real_evaluate:nn{#1}{1}}
5431 \def_new:Npn \calc_real_divide:n {\calc_real_evaluate:nn{1}{}}

```

\calc_maxmin_operation:Nnn The max and min functions.

\calc_maxmin_generic:Nnn

```

\calc_maxmin_div_or_mul:NNnn 5432 \def_protected_long:Npn\calc_maxmin_operation:Nnn#1#2#3{
\calc_maxmin_multiply: 5433   \group_begin:
\calc_maxmin_multiply: 5434   \calc_maxmin_generic:Nnn#1{#2}{#3}
\calc_maxmin_multiply: 5435   \group_end:
5436   \calc_post_scan:N
5437 }

```

#1 is either > or < and was expanded into this initially.

```

5438 \def_protected_long_new:Npn \calc_maxmin_generic:Nnn#1#2#3{
5439   \group_begin:
5440   \if_case:w\l_calc_current_type_int
5441     \calc_int_set:Nn\l_calc_C_int{#2}%
5442     \calc_int_set:Nn\l_calc_B_int{#3}%
5443     \pref_global:D\g_calc_A_register
5444     \if_num:w\l_calc_C_int#1\l_calc_B_int
5445       \l_calc_C_int\else:\l_calc_B_int\fi:
5446   \or:
5447     \calc_dim_set:Nn\l_calc_C_dim{#2}%
5448     \calc_dim_set:Nn\l_calc_B_dim{#3}%
5449     \pref_global:D\g_calc_A_register
5450     \if_dim:w\l_calc_C_dim#1\l_calc_B_dim
5451       \l_calc_C_dim\else:\l_calc_B_dim\fi:
5452   \or:
5453     \calc_skip_set:Nn\l_calc_C_skip{#2}%
5454     \calc_skip_set:Nn\l_calc_B_skip{#3}%
5455     \pref_global:D\g_calc_A_register
5456     \if_dim:w\l_calc_C_skip#1\l_calc_B_skip
5457       \l_calc_C_skip\else:\l_calc_B_skip\fi:
5458   \else:
5459     \calc_muskip_set:Nn\l_calc_C_muskip{#2}%
5460     \calc_muskip_set:Nn\l_calc_B_muskip{#3}%
5461     \pref_global:D\g_calc_A_register
5462     \if_dim:w\l_calc_C_muskip#1\l_calc_B_muskip
5463       \l_calc_C_muskip\else:\l_calc_B_muskip\fi:

```

```

5464 \fi:
5465 \group_end:
5466 }
5467 \def_long_new:Npn\calc_maxmin_div_or_mul:NNnn#1#2#3#4{
5468 \group_end:
5469 \group_begin:
5470 \int_zero:N\l_calc_current_type_int
5471 \group_execute_after:N#1
5472 \calc_maxmin_generic:Nnn#2{#3}{#4}
5473 \group_end:
5474 \group_begin:
5475 \calc_post_scan:N
5476 }
5477 \def_new:Npn\calc_maxmin_multiply:{
5478 \calc_maxmin_div_or_mul:NNnn\calc_multiply_B_by_A:}
5479 \def_new:Npn\calc_maxmin_divide: {
5480 \calc_maxmin_div_or_mul:NNnn\calc_divide_B_by_A:}

```

\calc_error:N The error message.

```

5481 \def_new:Npn\calc_error:N#1{
5482 \PackageError{calc}
5483 {'\token_to_string:N#1'~ invalid~ at~ this~ point}
5484 {I~ expected~ to~ see~ one~ of:~ +~ -~ *~ /~ )}
5485 }

```

26.2 Higher level commands

The various operations allowed.

\calc_maxof:nn Max and min operations

```

\calc_minof:nn
\maxof 5486 \def_long_new:Npn \calc_maxof:nn#1#2{
5487 \calc_maxmin_operation:Nnn > \exp_not:n{{#1}{#2}}
\minof 5488 }
5489 \def_long_new:Npn \calc_minof:nn#1#2{
5490 \calc_maxmin_operation:Nnn < \exp_not:n{{#1}{#2}}
5491 }
5492 \let:NN \maxof \calc_maxof:nn
5493 \let:NN \minof \calc_minof:nn

```

\calc_widthof:n Text dimension commands.

```

\calc_widthof_aux:n
\calc_widthof_auxi:n 5494 \def_long_new:Npn \calc_widthof:n#1{
5495 \calc_textsize:Nn \exp_not:N\calc_widthof_aux:n\exp_not:n{{#1}}
\calc_heightof:n 5496 }
\calc_heightof_aux:n 5497 \def_long_new:Npn \calc_heightof:n#1{
\calc_heightof_auxi:n 5498 \calc_textsize:Nn \exp_not:N\calc_heightof_aux:n\exp_not:n{{#1}}
\calc_depthof:n 5499 }
\calc_depthof_aux:n 5500 \def_long_new:Npn \calc_depthof:n#1{
\calc_depthof_auxi:n 5501 \calc_textsize:Nn \exp_not:N\calc_depthof_aux:n\exp_not:n{{#1}}
\calc_totalheightof:n 5502 }
\calc_totalheightof_aux:n
\calc_totalheightof_auxi:n

```



```

5503 \def_long_new:Npn \calc_totalheightof:n#1{
5504   \calc_textsize:Nn \exp_not:N\calc_totalheightof_aux:n \exp_not:n{{#1}}
5505 }
5506 \def_long_new:Npn \calc_widthof_aux:n #1{
5507   \exp_not:N\calc_widthof_aux:n\exp_not:n{{#1}}
5508 }
5509 \let_new:NN \calc_widthof_auxi:n \calc_widthof_aux:n
5510 \def_long_new:Npn \calc_depthof_aux:n #1{
5511   \exp_not:N\calc_depthof_aux:n\exp_not:n{{#1}}
5512 }
5513 \let_new:NN \calc_depthof_auxi:n \calc_depthof_aux:n
5514 \def_long_new:Npn \calc_heightof_aux:n #1{
5515   \exp_not:N\calc_heightof_aux:n\exp_not:n{{#1}}
5516 }
5517 \let_new:NN \calc_heightof_auxi:n \calc_heightof_aux:n
5518 \def_long_new:Npn \calc_totalheightof_aux:n #1{
5519   \exp_not:N\calc_totalheightof_aux:n\exp_not:n{{#1}}
5520 }
5521 \let_new:NN \calc_totalheightof_auxi:n \calc_totalheightof_aux:n

```

\calc_ratio:nn Ratio and real.

```

\calc_real:n
5522 \def_long_new:Npn \calc_ratio:nn#1#2{
5523   \calc_ratio_multiply:nn\exp_not:n{{#1}}{#2}}
5524 \def_new:Npn \calc_real:n {\calc_real_evaluate:nn}

```

We can implement real and ratio without actually using these names. We'll see.

```

\widthof User commands.
\heightof
\depthof 5525 \let:NN \depthof\calc_depthof:n
\totalheightof 5526 \let:NN \widthof\calc_widthof:n
5527 \let:NN \heightof\calc_heightof:n
\ratio 5528 \let:NN \totalheightof\calc_totalheightof:n
\real 5529 %%\let:NN \ratio\calc_ratio:nn
5530 %%\let:NN \real\calc_real:n

\setlength
\gsetlength
\addtolength 5531 \def_protected:Npn \setlength{\calc_skip_set:Nn}
\gaddtolength 5532 \def_protected:Npn \gsetlength{\calc_skip_gset:Nn}
5533 \def_protected:Npn \addtolength{\calc_skip_add:Nn}
5534 \def_protected:Npn \gaddtolength{\calc_skip_gadd:Nn}

```

\calc_setcounter:nn Document commands for L^AT_EX 2_ε counters. Also add support for amstext. Note that
\calc_addtocounter:nn when l3breqn is used, \mathchoice will no longer need this switch as the argument is
\calc_stepcounter:n only executed once.

```

\setcounter
\addtocounter 5535 \newif\iffirstchoice@ \firstchoice@true
\stepcounter 5536 \def_protected:Npn \calc_setcounter:nn#1#2{
5537   \calc_chk_document_counter:nn{{#1}}{
\calc_chk_document_counter:nn 5538   \exp_args:Nc\calc_int_gset:Nn {c@#1}}{#2}
5539   }

```

```

5540 }
5541 \def_protected:Npn \calc_addtocounter:nn#1#2{
5542   \iffirstchoice@
5543   \calc_chk_document_counter:nn{#1}{
5544     \exp_args:Nc\calc_int_gadd:Nn {c@#1}{#2}
5545   }
5546   \fi:
5547 }
5548 \def_protected:Npn \calc_stepcounter:n#1{
5549   \iffirstchoice@
5550   \calc_chk_document_counter:nn{#1}{
5551     \int_gincr:c {c@#1}
5552     \group_begin:
5553     \let:NN \@elt\@stpelt \cs_use:c{cl@#1}
5554     \group_end:
5555   }
5556   \fi:
5557 }
5558 \def_new:Npn \calc_chk_document_counter:nn#1{
5559   \cs_if_free:cTF{c@#1}{\@nocounterr {#1}}
5560 }
5561 \let:NN \setcounter \calc_setcounter:nn
5562 \let:NN \addtocounter \calc_addtocounter:nn
5563 \let:NN \stepcounter \calc_stepcounter:n
5564 \AtBeginDocument{
5565   \let:NN \setcounter \calc_setcounter:nn
5566   \let:NN \addtocounter \calc_addtocounter:nn
5567   \let:NN \stepcounter \calc_stepcounter:n
5568 }

```

Prevent the usual calc from loading.

```

5569 <package>\def:cpn{ver@calc.sty}{2005/08/06}

5570 </initex | package>
5571 <*showmemory>
5572 \showMemUsage
5573 </showmemory>

```

Change History

v2.0a	name for T _E X primitives	1
General: new consistent tex module		

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols	
<code>\#</code>	4, 3607
<code>\%</code>	2325
<code>*</code>	3925, 4545, 4547, 4551, 4556
<code>\-</code>	37
<code>\/</code>	36
<code>\:</code>	550, 554, 4812
<code>\::</code>	1875
<code>\:::</code>	1809, 1811, 1812, 1814, 1815, 1817, 1820, 1823, 1831, 1837, <u>1842</u> , 1843, 1854, 1857, 1860, 1870–1883, 1885–1915
<code>\::C</code>	<u>1843</u> , 1885, 1886
<code>\::E</code>	<u>1854</u>
<code>\::N</code>	<u>1814</u> , 1871–1875, 1886–1894, 1903
<code>\::O</code>	1873, 1876–1878
<code>\::c</code>	<u>1817</u> , 1874, 1877, 1879–1882, 1893–1899, 1904
<code>\::d</code>	<u>1860</u> , 1870, 1871
<code>\::e</code>	<u>1857</u>
<code>\::f</code>	<u>1823</u> , 1887, 1900–1902
<code>\::n</code>	<u>1811</u> , 1888, 1889, 1898, 1902–1909
<code>\::o</code>	<u>1820</u> , 1872, 1873, 1875, 1876, 1878, 1882, 1883, 1888, 1890, 1891, 1894–1896, 1901, 1905, 1907, 1908, 1910–1912, 1914
<code>\::x</code>	<u>1829</u> , 1889, 1891, 1892, 1897– 1899, 1906, 1908, 1909, 1911–1915
<code>\?</code>	4708, 4812
<code>\@</code>	611, 654
<code>\@end</code>	626
<code>\@hyph</code>	630
<code>\@input</code>	624
<code>\@italiccorr</code>	629
<code>\@underline</code>	625
<code>\@current</code>	653
<code>\@currname</code>	652
<code>\@currnamestack</code>	647, 655
<code>\@declaredoptions</code>	650
<code>\@elt</code>	5553
<code>\@empty</code>	649, 650
<code>\@gobble</code>	632
<code>\@nil</code>	647, 651
<code>\@nocounterr</code>	5559
<code>\@onefilewithoptions</code>	<u>603</u>
<code>\@p@pfilename</code>	647, 651
<code>\@popfilename</code>	<u>603</u>
<code>\@stpelt</code>	5553
<code>\@tempboxa</code>	3841
<code>\@unknownoptionerror</code>	648
<code>\@</code>	799, 801, 805, 813, 814, 3764, 4708
<code>\{</code>	2, 2326
<code>\}</code>	3, 2327
<code>\^</code>	5, 6, 1179
<code>_</code>	549, 553
<code>\ </code>	2936
<code>l3names</code>	4
<code>_</code>	35, 2309, 5045
A	
<code>\A</code>	4709, 4814
<code>\above</code>	156
<code>\abovedisplaysshortskip</code>	169
<code>\abovedisplayskip</code>	170
<code>\abovewithdelims</code>	157
<code>\absolutelytracingall</code>	<u>1214</u>
<code>\accent</code>	207
<code>\addtocounter</code>	<u>5535</u>
<code>\addtolength</code>	<u>5531</u>
<code>\adjdemerits</code>	244
<code>\advance</code>	51
<code>\afterassignment</code>	61
<code>\aftergroup</code>	62
<code>\aleph_textdir:D</code>	527, 1107
<code>\alloc_next_g:n</code>	<u>2242</u>
<code>\alloc_next_l:n</code>	<u>2242</u>
<code>\alloc_reg:NnNN</code>	<u>107</u> , <u>2266</u> , <u>2288</u> , <u>2330</u> , 2805, 2806, 3147, 3148, 3249, 3250, 3330, 3331, 3348, 3349, 3792, 3793
<code>\alloc_setup_type:nnn</code> <u>107</u> , <u>2235</u> , <u>2287</u> , <u>2329</u> , 2804, 3146, 3248, 3329, 3347, 3790
<code>\AtBeginDocument</code>	5079, 5564
<code>\AtEndOfPackage</code>	633
<code>\atop</code>	158
<code>\atopwithdelims</code>	159
B	
<code>\badcrerr</code>	3763
<code>\badlinearg</code>	3746
<code>\badmath</code>	3717
<code>\badness</code>	306
<code>\badpoptabs</code>	3727

\badtab	3734	\box_dp:c	184, 3828
\baselineskip	234	\box_dp:N	184, 3828, 5379, 5380
\batchmode	127	\box_gclear:c	184, 3824
\begin	2053, 3713, 5094	\box_gclear:N	184, 3824
\begingroup	65	\box_gset_eq:cc	184, 3811
\beginL	419	\box_gset_eq:cN	184, 3811
\beginR	421	\box_gset_eq:Nc	184, 3811
\belowdisplayshortskip	171	\box_gset_eq:NN	184, 3811
\belowdisplayskip	172	\box_gset_to_last:c	184, 3816
\binoppenalty	195	\box_gset_to_last:N	184, 3816
\bool_double_if:cnmnn	4314	\box_ht:c	184, 3828
\bool_double_if:cNnnnn	4314	\box_ht:N	184, 3828, 5378, 5380
\bool_double_if:Ncnmmn	4314	\box_if_empty:cF	183, 3800
\bool_double_if:NNnnnn	3054, 4314	\box_if_empty:cT	183, 3800
\bool_dowhile:cF	4306	\box_if_empty:cTF	183, 3800
\bool_dowhile:cT	4306	\box_if_empty:NF	183, 3800
\bool_dowhile:NF	204, 4306	\box_if_empty:NT	183, 3800
\bool_dowhile:NT	204, 4306	\box_if_empty:NTF	183, 3800
\bool_dowhiledo:cF	4313	\box_if_empty_p:c	183, 3800
\bool_gset_eq:cc	204, 4282	\box_if_empty_p:N	183, 3800
\bool_gset_eq:cN	204, 4282	\box_move_down:n	184, 3820
\bool_gset_eq:Nc	204, 4282	\box_move_left:n	184, 3820
\bool_gset_eq:NN	204, 4282	\box_move_right:n	184, 3820
\bool_gset_false:c	204, 4272	\box_move_up:n	184, 3820
\bool_gset_false:N	204, 4272	\box_new:c	183, 3789
\bool_gset_true:c	204, 4272	\box_new:N	183, 3789, 3842-3844
\bool_gset_true:N	204, 4272	\box_new_l:N	183, 3789
\bool_if:cF	4292	\box_set_eq:cc	183, 3807
\bool_if:cT	4292	\box_set_eq:cN	183, 3807
\bool_if:cTF	4292	\box_set_eq:Nc	183, 3807
\bool_if:NF	204, 3055, 3056, 4292, 4303, 4311	\box_set_eq:NN	183, 3807, 3811, 3824
\bool_if:NT	204, 4292, 4299, 4307	\box_set_to_last:c	184, 3816
\bool_if:NTF	204, 3046, 4292	\box_set_to_last:N	184, 3816
\bool_if_p:c	4296	\box_show:c	185, 3838
\bool_if_p:N	204, 4296	\box_show:N	185, 3838
\bool_new:c	204, 4272	\box_use:c	184, 3834
\bool_new:N	204, 4272, 4290, 4291	\box_use:N	184, 3834
\bool_set_eq:cc	204, 4282	\box_use_clear:c	184, 3834
\bool_set_eq:cN	204, 4282	\box_use_clear:N	184, 3834
\bool_set_eq:Nc	204, 4282	\box_wd:c	184, 3828
\bool_set_eq:NN	204, 4282	\box_wd:N	184, 3832, 3833, 5377
\bool_set_false:c	204, 4272	\box_wd:n	3828
\bool_set_false:N	204, 4272	\boxmaxdepth	312
\bool_set_true:c	204, 4272	\brokenpenalty	269
\bool_set_true:N	204, 4272		
\bool_whiledo:cF	4298		
\bool_whiledo:cT	4298		
\bool_whiledo:NF	204, 4298		
\bool_whiledo:NT	204, 4298		
\botmark	142		
\botmarks	368		
\box	350		
\box_clear:c	184, 3824		
\box_clear:N	184, 3824		
		C	
		\C	4709, 4815
		\c_active_char_token	221, 4542, 4650
		\c_alignment_tab_token	221, 4542, 4587
		\c_cs_dump_stream	
			192, 3907, 3922, 3923, 3931,
			3935, 3941, 3943, 3983, 3987, 3995
		\c_eleven	93, 2072, 2804
		\c_empty_box	185, 3824, 3840

- \c_empty_tlp 60, 1359, 1361, 1459,
1462, 1481, 2121, 2377, 2382, 3604
- \c_empty_toks 165, 3353, 3505
- \c_false 23, 731, 773, 788,
790, 794, 804, 806, 823, 1459, 1467,
1499, 1511, 1604, 1756, 1766, 1775,
1785, 2038, 3018, 3038, 3041, 3047,
3313, 3467, 3801, 4062, 4072, 4080,
4094, 4106, 4114, 4136, 4139, 4142,
4272, 4273, 4276, 4277, 4280, 4281,
4373, 4563, 4572, 4581, 4590, 4599,
4608, 4617, 4626, 4635, 4644, 4653,
4662, 4671, 4680, 4689, 4699, 4729,
4740, 4751, 4762, 4845, 4852, 4857,
4862, 4864, 4866, 4868, 4870, 4872
- \c_four 93, 2072
- \c_group_begin_token
. . . 221, 3857, 3882, 4542, 4560, 4932
- \c_group_end_token
221, 3858, 3861, 3885, 3890, 4542, 4569
- \c_hundred_one 93, 2072
- \c_io_term_stream . 112, 2296, 2302, 2303
- \c_iow_comment_char 112, 2325
- \c_iow_err_stream
. . . 173, 3582, 3587, 3595, 3602, 3616
- \c_iow_lbrace_char 112, 2325
- \c_iow_log_stream 112, 2296, 2301
- \c_iow_rbrace_char 112, 2325
- \C_job_name_tlp 60
- \c_kernel_err_tlp
. 173, 3635, 3637, 3667, 3779
- \c_letter_token 221, 4542, 4632
- \c_math_shift_token 221, 4542, 4578
- \c_math_subscript_token 221, 4542, 4614
- \c_math_superscript_token 221, 4542, 4605
- \c_max_dim 156, 3285
- \c_max_int 139, 2964
- \c_max_register_num 2053,
2804, 3146, 3248, 3329, 3347, 3790
- \c_max_skip 154, 3219, 3286
- \c_minus_one
. . 93, 733, 742, 1996, 1998, 2064,
2072, 2297, 2778, 2796, 2798, 3551
- \c_nine 93, 2072
- \c_one 93, 1230–1232, 1234,
1235, 1238, 2072, 2773, 2795, 2797,
5176, 5179, 5182, 5185, 5188, 5191
- \c_other_char_token 221, 4542, 4641
- \c_parameter_token 221, 4542
- \c_peek_true_remove_next_tlp 4907, 4913
- \c_relax_tlp 60, 1481
- \c_seven 93, 2072, 4154
- \c_six 93, 2072, 4150
- \c_sixteen
. 93, 733, 744, 2072, 2287, 2296, 2329
- \c_space_token 221, 4542, 4623, 4933, 4993
- \c_ten 93, 2072, 5045
- \c_ten_thousand 93, 1236, 1237, 2072
- \c_ten_thousand_four 2072
- \c_ten_thousand_one 2072
- \c_ten_thousand_three 2072
- \c_ten_thousand_two 2072
- \c_thirty_two 2072
- \c_thousand 93, 2072
- \c_three 93, 2072, 5212,
5216, 5220, 5224, 5228, 5232, 5395
- \c_true 23,
731, 775, 788, 794, 803, 823, 1459,
1467, 1497, 1511, 1604, 1754, 1764,
1773, 1783, 2036, 3016, 3035, 3038,
3041, 3048, 3311, 3465, 3801, 4061,
4070, 4080, 4093, 4104, 4114, 4136,
4139, 4142, 4274, 4275, 4278, 4279,
4373, 4561, 4570, 4579, 4588, 4597,
4606, 4615, 4624, 4633, 4642, 4651,
4660, 4669, 4678, 4689, 4697, 4874
- \c_twenty_thousand 93, 2072
- \c_two 93, 1228, 1229, 1233, 2072,
5194, 5197, 5200, 5203, 5206, 5209
- \c_twohundred_fifty_five 93, 2072
- \c_twohundred_fifty_six 93, 2072
- \c_undefined 19, 24, 50, 793, 1106
- \c_zero 93, 1247–
1257, 1604, 1609, 2042, 2072, 2287,
2329, 2823, 2825, 2977, 2981, 2982,
2988, 3065, 3146, 3231, 3232, 3248,
3329, 3347, 3535, 3790, 4079, 4113,
4147, 4148, 4152, 4200, 4227, 4251,
5158, 5161, 5164, 5167, 5170, 5173
- \c_zero_dim 156, 3285, 3863, 5372
- \c_zero_skip 154, 3165,
3219, 3238, 3239, 3261, 3285, 3892
- \calc_add: 5264, 5313
- \calc_add_A_to_B: 5313
- \calc_addtocounter:nn 5535
- \calc_assign_generic:NNNNnn . . 5142,
5158, 5161, 5164, 5167, 5170, 5173,
5176, 5179, 5182, 5185, 5188, 5191,
5194, 5197, 5200, 5203, 5206, 5209,
5212, 5216, 5220, 5224, 5228, 5232
- \calc_calculate_box_size:nnn
. 248, 5370, 5381
- \calc_calculate_box_size_aux:n . . 5370
- \calc_calculated_ratio: 5402, 5412
- \calc_chk_document_counter:nn . . . 5535
- \calc_close: 5253, 5268
- \calc_depthof:n 5494, 5525

- \calc_depthof_aux:n 5379, 5384, 5494
- \calc_depthof_auxi:n 5384, 5494
- \calc_dim_add:Nn 247, 5175
- \calc_dim_gadd:Nn 247, 5175
- \calc_dim_gset:Nn 247, 5175
- \calc_dim_gsub:Nn 247, 5175
- \calc_dim_muskip:Nn 5399, 5400
- \calc_dim_set:Nn
 . . . 247, 5175, 5396, 5397, 5447, 5448
- \calc_dim_sub:Nn 247, 5175
- \calc_divide: 5307, 5340
- \calc_divide:N 5267, 5281
- \calc_divide_B_by_A: 5340, 5480
- \calc_error:N 5272, 5481
- \calc_generic_add:N 5313
- \calc_generic_add_or_subtract:N
 5313, 5319, 5321
- \calc_generic_multiply_or_divide:N 5340
- \calc_heightof:n 5494, 5527
- \calc_heightof_aux:n . . . 5378, 5386, 5494
- \calc_heightof_auxi:n 5386, 5494
- \calc_init_B: 5249, 5250, 5253, 5317
- \calc_int_add:Nn 247, 5157
- \calc_int_gadd:Nn 247, 5157, 5544
- \calc_int_gset:Nn 247, 5157, 5538
- \calc_int_gsub:Nn 247, 5157
- \calc_int_set:Nn . . 247, 5157, 5441, 5442
- \calc_int_sub:Nn 247, 5157
- \calc_maxmin_div_or_mul:NNnn 5432
- \calc_maxmin_divide: 5299, 5479
- \calc_maxmin_generic:Nnn 5432
- \calc_maxmin_multiply: 5283, 5432
- \calc_maxmin_operation:Nnn
 . . 5241, 5282, 5298, 5432, 5487, 5490
- \calc_maxof:nn 5486
- \calc_minof:nn 5486
- \calc_multiply: 5291, 5340
- \calc_multiply:N 5266, 5281
- \calc_multiply_B_by_A: 5340, 5478
- \calc_muskip_add:Nn 247, 5211
- \calc_muskip_gadd:Nn 247, 5211
- \calc_muskip_gset:Nn 247, 5211
- \calc_muskip_gsub:Nn 247, 5211
- \calc_muskip_set:Nn 247, 5211, 5459, 5460
- \calc_muskip_sub:Nn 247, 5211
- \calc_next:w 5263–5269,
 5271, 5280, 5283, 5286, 5289, 5291,
 5295, 5299, 5302, 5305, 5307, 5311
- \calc_numeric: 5243, 5253
- \calc_open:w 5152, 5237, 5248
- \calc_post_scan:N 5255, 5261,
 5262, 5391, 5414, 5429, 5436, 5475
- \calc_pre_scan:N . . 5235, 5251, 5318, 5346
- \calc_ratio:nn 5148, 5522, 5529
- \calc_ratio_divide:nn 5302, 5393
- \calc_ratio_multiply:nn
 5285, 5286, 5301, 5393, 5523
- \calc_real:n 5147, 5522, 5530
- \calc_real_divide:n 5305, 5416
- \calc_real_evaluate:nn
 5288, 5304, 5416, 5524
- \calc_real_multiply:n 5289, 5416
- \calc_setcounter:nn 5535
- \calc_skip_add:Nn 247, 5193, 5533
- \calc_skip_gadd:Nn 247, 5193, 5534
- \calc_skip_gset:Nn 247, 5193, 5532
- \calc_skip_gsub:Nn 247, 5193
- \calc_skip_set:Nn
 247, 5193, 5453, 5454, 5531
- \calc_skip_sub:Nn 247, 5193
- \calc_stepcounter:n 5535
- \calc_subtract: 5265, 5313
- \calc_subtract_A_from_B: 5313
- \calc_textsize:Nn
 . . 5239, 5375, 5495, 5498, 5501, 5504
- \calc_totalheightof:n 5494, 5528
- \calc_totalheightof_aux:n 5380, 5387, 5494
- \calc_totalheightof_auxi:n . . 5387, 5494
- \calc_widthof:n 5494, 5526
- \calc_widthof_aux:n . . . 5377, 5385, 5494
- \calc_widthof_auxi:n 5385, 5494
- \catcode 2–6, 9–11, 13, 14, 354
- \char 208
- \char_gset_mathcode:nn 220, 4501
- \char_gset_mathcode:w 220, 4501
- \char_set_catcode:nn
 219, 3607, 4492, 4545, 4547,
 4551, 4556, 4710, 4813–4817, 5045
- \char_set_catcode:w 219, 4492
- \char_set_lccode:nn
 219, 4514, 4705–4708, 4812
- \char_set_lccode:w 219, 4514
- \char_set_mathcode:nn 220, 4501
- \char_set_mathcode:w 220, 4501
- \char_set_sfcode:nn 220, 4532
- \char_set_sfcode:w 220, 4532
- \char_set_uccode:nn 220, 4523
- \char_set_uccode:w 220, 4523
- \char_show_value_catcode:n . . . 219, 4492
- \char_show_value_catcode:w . . . 219, 4492
- \char_show_value_lccode:n . . . 219, 4514
- \char_show_value_lccode:w . . . 219, 4514
- \char_show_value_mathcode:n . . 220, 4501
- \char_show_value_mathcode:w . . 220, 4501
- \char_show_value_sfcode:n . . . 220, 4532
- \char_show_value_sfcode:w . . . 220, 4532
- \char_show_value_uccode:n . . . 220, 4523
- \char_show_value_uccode:w . . . 220, 4523

- \char_value_catcode:n 219, [4492](#)
- \char_value_catcode:w 219, [4492](#)
- \char_value_lccode:n 219, [4514](#)
- \char_value_lccode:w 219, [4514](#)
- \char_value_mathcode:n 220, [4501](#)
- \char_value_mathcode:w 220, [4501](#)
- \char_value_sfcode:n 220, [4532](#)
- \char_value_sfcode:w 220, [4532](#)
- \char_value_uccode:n 220, [4523](#)
- \char_value_uccode:w 220, [4523](#)
- \chardef 43
- \chk_exist_cs:c 23
- \chk_exist_cs:N 23, 1220, 1310, 1314, 1317, 1320, 1341, 1345
- \chk_global:c 50
- \chk_global:N 50, [1180](#), 1195, 1215, 1317, 1320, 1346, 3437, 3451
- \chk_global_aux:w 50, [1180](#)
- \chk_if_exist_cs:c 782
- \chk_if_exist_cs:c␣ [776](#)
- \chk_if_exist_cs:N 776, 783
- \chk_if_exist_cs:N␣ [776](#)
- \chk_local:c 50
- \chk_local:N 50, [1164](#), 1196, 1198, 1216, 1314, 3420, 3447
- \chk_local_aux:w 50, [1164](#)
- \chk_local_or_pref_global:N 50, [1193](#), 1217, 1311, 1342, 1389, 1394, 1398, 1403, 1408, 1412, 1423, 1428, 1433, 1438, 1442, 1446, 2344, 2775, 2780, 2812, 2830, 2837, 3154, 3167, 3181, 3188, 3355, 3392, 3397
- \chk_new_cs:N 23, [752](#), 825, 827, 829, 831, 833, 835, 837, 839, 841, 843, 845, 847, 849, 851, 853, 855, 1028, 1037, 1219, 1285, 1293, 2068, 2267, 3212
- \chk_var_or_const:c 50
- \chk_var_or_const:N 50, [1199](#), 1221, 1287, 1294, 1342, 1346, 1365, 1377, 3448, 3452
- \chk_var_or_const_aux:w 50, [1199](#)
- \cleaders 226
- \clearpage 5098, 5101, 5106, 5109
- \clearshortrefmaps 3569
- \clist_clear:c 117, [2367](#)
- \clist_clear:N 117, [2367](#), 2502
- \clist_concat:NNN 118, [2488](#)
- \clist_concat_aux:NNNN 121, [2488](#)
- \clist_gc_clear:c 117, [2367](#)
- \clist_gc_clear:N 117, [2367](#)
- \clist_gconcat:ccc 118, [2488](#)
- \clist_gconcat:NNc 118, [2488](#)
- \clist_gconcat:NNN 118, [2488](#)
- \clist_get:cN 118, [2387](#), 2551
- \clist_get:NN 118, [2387](#), 2550
- \clist_get_aux:w 120, 2389, 2390
- \clist_gpop:cN 121, [2545](#)
- \clist_gpop:NN 121, [2545](#)
- \clist_gpush:cn 121, [2545](#)
- \clist_gpush:Nn 121, [2545](#)
- \clist_gpush:No 121, [2545](#)
- \clist_gput_left:Nn 117, [2413](#), 2545
- \clist_gput_right:cc 117, [2422](#)
- \clist_gput_right:cn 117, [2422](#)
- \clist_gput_right:co 117, [2422](#)
- \clist_gput_right:NC 2422
- \clist_gput_right:Nn 117, [2422](#)
- \clist_gput_right:No 117, [2422](#)
- \clist_gremove_duplicates:N .. 118, [2501](#)
- \clist_gset_eq:cc 118, 2375
- \clist_gset_eq:cN 118, 2373
- \clist_gset_eq:Nc 118, 2374
- \clist_gset_eq:NN .. 118, 2372–2375, 2514
- \clist_if_empty:cF 120, [2377](#)
- \clist_if_empty:cT [2377](#)
- \clist_if_empty:cTF 120, [2377](#)
- \clist_if_empty:NF .. 120, [2377](#), 2431, 2450
- \clist_if_empty:NT [2377](#)
- \clist_if_empty:NTF 120, [2377](#), 2404
- \clist_if_empty_err:N 120, [2381](#), 2388, 2393
- \clist_if_empty_p:N 120, [2376](#)
- \clist_if_eq:NNTF 120, [2386](#)
- \clist_if_in:cnTF 120, [2526](#)
- \clist_if_in:coTF 120, [2526](#)
- \clist_if_in:NnTF 120, 2507, [2526](#)
- \clist_if_in:NoTF 120, [2526](#)
- \clist_map_break:w 119, [2447](#)
- \clist_map_function:cN 119, [2430](#)
- \clist_map_function:NN .. 119, [2430](#), 2503
- \clist_map_function:nN 119, [2430](#)
- \clist_map_function_aux:Nw 120, 2433, 2440, [2442](#), 2455, 2468
- \clist_map_inline:cn 119, [2448](#)
- \clist_map_inline:Nn 119, [2448](#)
- \clist_map_inline:nn 119, 1261, 1267, [2448](#)
- \clist_map_inline_aux:Nw 120
- \clist_map_variable:cNn 119, [2474](#)
- \clist_map_variable:NNn 119, [2474](#)
- \clist_map_variable:nNn 119, [2474](#)
- \clist_map_variable_aux:Nnw 120, 2477, [2483](#)
- \clist_new:c 117, [2365](#)
- \clist_new:N 117, [2365](#), 2516
- \clist_pop:cN 121, [2540](#)
- \clist_pop:NN 121, [2540](#)
- \clist_pop_aux:nnNN 120, [2392](#), 2543, 2548
- \clist_pop_aux:w 120, [2392](#)
- \clist_pop_auxi:w 120, [2392](#)

- \clist_push:cn 121, 2540
- \clist_push:Nn 121, 2540
- \clist_push:No 121, 2540
- \clist_put_aux:NNnnNn
... 120, 2403, 2408, 2414, 2417, 2423
- \clist_put_left:cn 117, 2407, 2542
- \clist_put_left:Nn 117, 2407, 2540
- \clist_put_left:No 117, 2407, 2541
- \clist_put_left:Nx 117, 2407
- \clist_put_right:cn 2416
- \clist_put_right:Nn 117, 2416, 2508
- \clist_put_right:No 117, 2416
- \clist_put_right:Nx 117, 2416
- \clist_remove_duplicates:N ... 118, 2501
- \clist_remove_duplicates_aux:n 121, 2501
- \clist_remove_duplicates_aux:NN
... 121, 2501
- \clist_set_eq:cc 2372
- \clist_set_eq:cN 2372
- \clist_set_eq:Nc 2372
- \clist_set_eq:NN .. 118, 2371, 2372, 2511
- \clist_top:cN 121, 2550
- \clist_top:NN 121, 2550
- \clist_use:c 118, 2517
- \clist_use:N 118, 2517
- \closein 102
- \closeout 97
- \clubpenalties 410
- \clubpenalty 237
- \cmd_arg_list_build 3532, 3543
- \cmd_declare:Nnn 3542, 3621, 3694
- \CodeStart 528, 3926
- \CodeStop 528, 3567
- \const_new:Nn .. 93, 2053, 2075–2095, 2964
- \const_new_aux:Nw 2053
- \copy 294
- \count 345
- \countdef 44
- \cr 69
- \crrcr 70
- \cs:w 24, 690, 783, 857–865, 867,
869, 871, 873, 875, 877, 879, 881,
883, 885, 887, 889, 891, 893, 895,
897, 899, 901, 903, 905, 907, 909,
911, 930, 1025, 1091, 1103, 1109,
1110, 1112, 1457, 1603, 1608, 1818,
1844, 1869, 1924, 1938, 1940, 1942,
1944, 1946–1948, 1953, 1959, 1963,
2856, 4160, 4166, 4169, 4596, 5041
- \cs_dump: 192, 3920, 3954
- \cs_end: .. 24, 690, 783, 857–865, 867,
869, 871, 873, 875, 877, 879, 881,
883, 885, 887, 889, 891, 893, 895,
897, 899, 901, 903, 905, 907, 909,
911, 930, 1025, 1091, 1096, 1103,
1105, 1109, 1110, 1112, 1457, 1603,
1608, 1818, 1844, 1869, 1924, 1938,
1940, 1942, 1944, 1946, 1947, 1949,
1956, 1962, 1965, 2856, 4162, 4163,
4172–4174, 4176, 4178, 4180, 4182,
4184, 4186, 4188–4198, 4596, 5041
- \cs_exist_p:N 1366, 1378
- \cs_free:cF 1094, 1262, 1268
- \cs_free:cT 1093
- \cs_free:cTF 1092
- \cs_free:NF 1089
- \cs_free:NT 1088
- \cs_free:NTF 1087, 2305
- \cs_free_p:N 797, 3689
- \cs_gen_sym:N 191, 3958
- \cs_ggen_sym:N 191, 3958
- \cs_gundefine:N 28, 1106
- \cs_gundefine:N_□ 1106
- \cs_if_eq:ccF 22, 1144
- \cs_if_eq:ccT 22, 1144
- \cs_if_eq:ccTF 22, 1144
- \cs_if_eq:cnF 22, 1144
- \cs_if_eq:cNT 22, 1144
- \cs_if_eq:cNTF 22, 1144
- \cs_if_eq:NcF 22, 1144
- \cs_if_eq:NcT 22, 1144
- \cs_if_eq:NcTF 22, 1144
- \cs_if_eq:NNF 22, 1144
- \cs_if_eq:NNT 22, 1144
- \cs_if_eq:NNTF 22, 1144
- \cs_if_eq_name_p:NN 787, 793, 807
- \cs_if_eq_p:NN 21
- \cs_if_exist:cF 22, 1100
- \cs_if_exist:cT 22, 1100
- \cs_if_exist:cTF 22, 1100
- \cs_if_exist:NF 22, 1100
- \cs_if_exist:NT 22, 1100
- \cs_if_exist:NTF 22, 1100
- \cs_if_exist_p:N 22, 771, 777, 1100
- \cs_if_free:cF 22, 1090
- \cs_if_free:cT 22, 1090, 1967
- \cs_if_free:cTF 22, 1090, 5559
- \cs_if_free:NF 22, 1086
- \cs_if_free:NT 22, 1086
- \cs_if_free:NTF 22, 1086, 1830
- \cs_if_free_p:N .. 22, 753, 772, 784, 1086
- \cs_if_really_exist:cF 23, 1100
- \cs_if_really_exist:cT 23, 1100
- \cs_if_really_exist:cTF .. 23, 923, 1100
- \cs_if_really_free:cF 22, 1095
- \cs_if_really_free:cT ... 22, 1095, 1614
- \cs_if_really_free:cTF
... 22, 1095, 4064, 4098, 5031, 5049

- \cs_load_dump:n 191, 3947
 - \cs_meaning:c 690
 - \cs_meaning:N 690
 - \cs_really_free:cF 1099
 - \cs_really_free:cT 1098
 - \cs_really_free:cTF 1097
 - \cs_record_meaning:N
..... 747, 765, 1224, 2272, 3975
 - \cs_record_name:c 1223, 3912
 - \cs_record_name:N
..... 191, 1222, 3912, 3962, 3969
 - \cs_show:c 690
 - \cs_show:N 690
 - \cs_to_str:N 32, 1143, 2325–2327
 - \cs_use:c 925, 1109,
2029, 2275, 2879, 4297, 4359, 5553
 - \csname 132
 - \currentgrouplevel 384
 - \currentgrouptype 385
 - \currentifbranch 381
 - \currentiflevel 380
 - \currentiftype 382
- D**
- \d 4707
 - \dagger 2932, 2938
 - \day 340
 - \ddagger 2933, 2939
 - \deadcycles 274
 - \DeclareOption 22, 25
 - \def 18, 23, 26, 31, 39
 - \def:cNn 28, 951
 - \def:cNx 28, 951
 - \def:cpn 30, 857, 939–
947, 949, 1063, 2859, 2860, 5025, 5569
 - \def:cpx 30, 857, 1066, 1969
 - \def:NNn 28, 951, 5039
 - \def:NNx 28, 951
 - \def:No 1042
 - \def:Npn 30, 707, 731, 732, 741, 743, 745,
748, 752, 771, 776, 782, 784, 798,
801, 815, 818, 822, 826, 857, 951,
953, 1042, 1310, 1324, 1514, 1600,
1866–1883, 1885–1915, 1950, 1952,
1995–1998, 2177, 2181, 2196, 2301,
2302, 2484, 2501, 2506, 2527, 2569,
2586, 2606, 2625, 2627, 2687, 2691,
2702, 2706, 2732, 2734, 2736, 2738,
2740, 2742, 2744, 2746, 2748, 2750,
2752, 2754, 2756, 2795–2802, 2882,
2965, 2966, 3064, 3089, 3092, 3095,
3106, 3123, 3532, 3542, 3544, 3553,
3556, 3559, 3562, 3572, 3577, 3670,
3676, 3679, 3685, 3883, 3888, 3975,
3978, 4259, 4267, 4727, 4738, 4749,
4760, 5054, 5072, 5073, 5082, 5083,
5086, 5089, 5291, 5307, 5374, 5380
 - \def:Npx 30, 707, 828, 858,
952, 954, 1314, 1325, 1833, 3540, 5055
 - \def_arg_number_error_msg:Nn 912
 - \def_aux:Ncnn
..... 912, 953, 954, 957, 958, 961,
962, 965, 966, 969, 970, 973, 974,
977, 978, 981, 982, 985, 986, 989,
990, 993, 994, 997, 998, 1001, 1002,
1005, 1006, 1012, 1014, 1020, 1022
 - \def_aux:NNnn
..... 912, 951, 952, 955, 956, 959,
960, 963, 964, 967, 968, 971, 972,
975, 976, 979, 980, 983, 984, 987,
988, 991, 992, 995, 996, 999, 1000,
1003, 1004, 1008, 1010, 1016, 1018
 - \def_aux_0:NNn 912
 - \def_aux_1:NNn 912
 - \def_aux_2:NNn 912
 - \def_aux_3:NNn 912
 - \def_aux_4:NNn 912
 - \def_aux_5:NNn 912
 - \def_aux_6:NNn 912
 - \def_aux_7:NNn 912
 - \def_aux_8:NNn 912
 - \def_aux_9:NNn 912
 - \def_aux_use_0_parameter: 939
 - \def_aux_use_1_parameter: 939
 - \def_aux_use_2_parameter: 939
 - \def_aux_use_3_parameter: 939
 - \def_aux_use_4_parameter: 939
 - \def_aux_use_5_parameter: 939
 - \def_aux_use_6_parameter: 939
 - \def_aux_use_7_parameter: 939
 - \def_aux_use_8_parameter: 939
 - \def_aux_use_9_parameter: 939
 - \def_long:cNn 29, 967
 - \def_long:cNx 29, 967, 4406,
4410, 4418, 4431, 4449, 4458, 4460,
4462, 4464, 4466, 4468, 4470, 4472
 - \def_long:cpn 30, 865, 1069
 - \def_long:cpx
30, 865, 1072, 4414, 4422, 4440, 4474
 - \def_long:NNn 29, 967, 1709
 - \def_long:NNx 29, 967
 - \def_long:Npn
30, 707, 750, 751, 807, 830, 865,
967, 969, 1440, 1444, 1661, 1662,
1668, 1669, 1675, 1676, 1683, 1691,
1695, 1711, 4085, 4375, 4385, 4395
 - \def_long:Npx . 30, 707, 832, 867, 968, 970
 - \def_long_new:cNn 26, 967

- \def_long_new:cNx 26, 967, 1601
- \def_long_new:cpn 27, 865, 1078,
4172–4175, 4177, 4179, 4181, 4183,
4185, 4187, 4189–4198, 4363, 4365
- \def_long_new:cpx 27, 865, 1084
- \def_long_new:NNn 26, 967, 1115–1122,
1131–1139, 1576, 1579, 1582, 1585,
1588, 1591, 1594, 1597, 1690, 1731,
1732, 2572, 2574, 2576, 2580, 2581,
2583, 2589, 2596, 2602, 2620, 2622,
2624, 2635, 2637, 4199, 4205, 4214,
4224, 4234, 4242, 4250, 4256, 4264
- \def_long_new:NNx 26, 967
- \def_long_new:Npn 27, 825, 873,
975, 977, 1044, 1053, 1062, 1065,
1068, 1071, 1074, 1077, 1080, 1083,
1108, 1113, 1114, 1123–1130, 1284,
1292, 1309, 1313, 1316, 1319, 1324–
1327, 1386, 1391, 1396, 1400, 1405,
1410, 1414–1420, 1425, 1430, 1435,
1495, 1507, 1520, 1527, 1532, 1541,
1552, 1557, 1564, 1682, 1735, 1738,
1743, 1745, 1747, 1750–1752, 1761,
1771, 1780, 1790, 1793, 1808, 1811,
1814, 1817, 1820, 1823, 1831, 1837,
1842, 1843, 1845, 1854, 1857, 1860,
1916, 1917, 1920, 1924–1926, 1928,
1930, 1932, 1934, 1936, 1938, 1939,
1941, 1943, 1945, 1948, 2046, 2047,
2442, 2449, 2462, 2568, 3022, 3025,
3028, 3031, 3389, 3395, 3407, 3410,
3413, 3423, 3424, 3430, 3511, 3615,
3820–3823, 3846, 3850, 3862, 3863,
3872, 3876, 3891, 3892, 4018, 4024,
4032, 4041, 4052, 4054, 4056, 4060,
4065, 4077, 4092, 4099, 4111, 4298,
4302, 4306, 4310, 4332, 4335, 4338,
4341, 4344, 4372, 4379, 4389, 4399,
4895, 4903, 4938, 5142, 5370, 5415,
5467, 5486, 5489, 5494, 5497, 5500,
5503, 5506, 5510, 5514, 5518, 5522
- \def_long_new:Npx . 27, 825, 875, 976, 978
- \def_long_test_function:npn
..... 32, 1044, 1615, 1620, 1625,
1630, 1635, 1640, 1645, 1650, 1655
- \def_long_test_function:npx 1044
- \def_long_test_function_new:npn
..... 32, 1044, 1086,
1090, 1095, 1100, 1101, 1104, 1503,
1505, 1512, 1515, 1759, 1769, 1778,
1788, 4058, 4083, 4096, 4117, 4349
- \def_long_test_function_new:npx
..... 1044, 1606
- \def_new:cNn 26, 951
- \def_new:cNx 26, 951
- \def_new:cpn 27, 857,
912–921, 1075, 4360, 4367, 4370, 4371
- \def_new:cpx 27, 857, 1081
- \def_new:NNn
.. 26, 951, 2629, 2631, 2655, 2658,
2661, 3474, 3477, 3480, 3492, 4405
- \def_new:NNx 26, 951
- \def_new:Npn 27, 825, 857–866, 868, 870,
872, 874, 876, 878, 880, 882, 884,
886, 888, 890, 892, 894, 896, 898,
900, 902, 904, 906, 908, 910, 929,
931, 951–1007, 1009, 1011, 1013,
1015, 1017, 1019, 1021, 1025–1028,
1030–1032, 1036, 1037, 1039–1043,
1106, 1109–1111, 1140–1143, 1145–
1154, 1164, 1167, 1180, 1182, 1193,
1199, 1201, 1214, 1227, 1242, 1246,
1260, 1266, 1291, 1297–1299, 1307,
1329–1338, 1340, 1344, 1353–1362,
1364, 1374, 1376, 1385, 1448–1451,
1457, 1458, 1460, 1463–1466, 1468–
1470, 1472–1480, 1491, 1493, 1494,
1502, 1510, 1523, 1531, 1550, 1555,
1556, 1569, 1570, 1667, 1674, 1681,
1688, 1705–1708, 1725, 1727, 1728,
1730, 1733, 1734, 1741, 1742, 1746,
1748, 1828, 1923, 1973, 1999–2010,
2013–2015, 2017–2019, 2021–2027,
2033, 2034, 2041, 2044, 2045, 2063,
2106, 2107, 2118, 2120, 2121, 2124,
2127, 2129, 2130, 2133, 2135, 2137,
2138, 2140–2143, 2145–2147, 2153,
2159–2164, 2170, 2174, 2176, 2180,
2184, 2186, 2188–2191, 2194, 2195,
2207–2211, 2215, 2216, 2218–2222,
2235, 2242, 2254, 2266, 2288, 2291,
2292, 2294, 2295, 2298, 2299, 2303,
2304, 2306, 2308, 2310, 2312, 2315,
2319, 2324, 2330, 2334, 2336, 2340,
2343, 2346, 2347, 2365, 2366, 2373–
2375, 2378–2381, 2387, 2390–2392,
2395, 2402, 2403, 2407, 2410–2413,
2416, 2419–2422, 2425–2430, 2437,
2438, 2461, 2474, 2481–2483, 2488,
2497–2500, 2510, 2513, 2517, 2525,
2526, 2537–2539, 2543, 2544, 2546–
2549, 2562, 2563, 2565, 2567, 2575,
2578, 2579, 2594, 2610–2619, 2633,
2634, 2664–2673, 2677, 2718–2720,
2722, 2730, 2773, 2778, 2783, 2789,
2805, 2806, 2809, 2810, 2815, 2821–
2827, 2833, 2834, 2840, 2846, 2852–
2854, 2856–2858, 2870, 2874, 2878,

- 2892, 2901, 2910, 2914, 2918, 2929,
2942, 2969, 2972, 2975, 2999, 3002–
3004, 3014, 3034, 3037, 3040, 3043,
3061, 3076, 3112, 3115, 3147, 3148,
3151, 3152, 3157, 3163–3165, 3170,
3176–3178, 3184, 3185, 3191, 3197,
3204, 3205, 3207, 3208, 3210, 3211,
3229, 3235, 3249, 3250, 3253–3265,
3268–3276, 3278, 3287, 3289, 3296,
3302, 3309, 3316, 3319, 3322, 3325,
3330, 3331, 3334–3339, 3348, 3349,
3352, 3353, 3358, 3365, 3366, 3372,
3378, 3381, 3382, 3388, 3400, 3406,
3409, 3412, 3416–3418, 3420, 3426–
3429, 3432–3435, 3437, 3440–3444,
3446, 3450, 3457–3463, 3471–3473,
3483–3491, 3495–3497, 3506, 3529,
3530, 3546, 3550, 3552, 3555, 3558,
3561, 3564, 3571, 3576, 3581, 3584,
3591, 3606, 3619, 3634–3636, 3638,
3642, 3647, 3652, 3656, 3661, 3792,
3793, 3796, 3800, 3802, 3804–3814,
3816–3819, 3824–3827, 3829, 3831,
3833, 3835, 3837, 3839, 3845, 3847–
3849, 3852–3856, 3859, 3864, 3868,
3870, 3871, 3873–3875, 3878–3881,
3886, 3894, 3896, 3912, 3919, 3920,
3947, 3958, 3965, 3981, 4010, 4029,
4049, 4119–4122, 4135, 4138, 4141,
4146, 4148, 4149, 4159, 4165, 4168,
4171, 4272–4281, 4293–4296, 4301,
4305, 4309, 4313, 4314, 4329–4331,
4354, 4358, 4493, 4496–4499, 4502,
4506, 4509–4512, 4515, 4518–4521,
4524, 4527–4530, 4533, 4536–4539,
4541, 4559, 4568, 4577, 4586, 4595,
4604, 4613, 4622, 4631, 4640, 4649,
4658, 4667, 4676, 4685, 4688, 4692,
4695, 4713, 4717, 4720, 4724, 4735,
4746, 4757, 4768, 4771, 4775, 4778,
4782, 4785, 4789, 4820, 4823, 4829,
4835, 4842, 4861, 4887, 4888, 4893,
4916, 4923, 4930, 4946, 4949, 4953,
4956, 4960, 4963, 4967, 4970, 4974,
4977, 4981, 4985, 4989, 4992, 5016–
5018, 5030, 5048, 5127, 5128, 5157,
5160, 5163, 5166, 5169, 5172, 5175,
5178, 5181, 5184, 5187, 5190, 5193,
5196, 5199, 5202, 5205, 5208, 5211,
5215, 5219, 5223, 5227, 5231, 5235,
5248, 5253, 5254, 5258, 5262, 5281,
5297, 5313, 5319, 5320, 5322, 5331,
5340, 5348, 5357, 5366, 5368, 5430,
5431, 5477, 5479, 5481, 5524, 5558
- `\def_new:Npx` 27, 825, 862, 960, 962
`\def_protected:cNn` 29, 983
`\def_protected:cNx` 29, 983
`\def_protected:cpn` 30, 880, 5022
`\def_protected:cpx` 30, 880
`\def_protected:NNn` 29, 983
`\def_protected:NNx` 29, 983
`\def_protected:Npn`
. 30, 707, 721–725, 728, 825, 827,
829, 831, 833–835, 837, 839, 881,
983, 985, 5531–5534, 5536, 5541, 5548
`\def_protected:Npx`
. 30, 707, 836, 883, 984, 986
`\def_protected_long:cNn` 29, 999
`\def_protected_long:cNx` 29, 999
`\def_protected_long:cpn` 31, 896
`\def_protected_long:cpx` 31, 896
`\def_protected_long:NNn` 29, 999
`\def_protected_long:NNx` 29, 999
`\def_protected_long:Npn` 31, 707,
838, 897, 999, 1001, 5375, 5393, 5432
`\def_protected_long:Npx`
. 31, 707, 840, 899, 1000, 1002
`\def_protected_long_new:cNn` 26, 999
`\def_protected_long_new:cNx` 26, 999
`\def_protected_long_new:cpn` 28, 896
`\def_protected_long_new:cpx` 28, 896
`\def_protected_long_new:NNn` 26, 999
`\def_protected_long_new:NNx` 26, 999
`\def_protected_long_new:Npn` 28,
825, 905, 922, 1008, 1012, 1023, 5438
`\def_protected_long_new:Npx`
. 28, 825, 907, 1010, 1014
`\def_protected_new:cNn` 26, 983
`\def_protected_new:cNx` 26, 983
`\def_protected_new:cpn` 27, 880
`\def_protected_new:cpx` 27, 880
`\def_protected_new:NNn` 26, 983
`\def_protected_new:NNx` 26, 983
`\def_protected_new:Npn`
. 27, 825, 841, 843,
845, 847, 849, 851, 853, 855, 889,
991, 993, 1033–1035, 4505, 5043, 5416
`\def_protected_new:Npx`
. 27, 825, 891, 992, 994
`\def_test_function:npn` 32, 1044
`\def_test_function:npx` 1044
`\def_test_function_aux:Nnnn` 1044
`\def_test_function_aux:Nnnx` 1044
`\def_test_function_new:npn` 32,
1044, 1107, 1144, 1461, 1471, 2030,
2377, 3021, 3470, 3803, 4137, 4140,
4143, 4144, 4292, 4566, 4575, 4584,
4593, 4602, 4611, 4620, 4629, 4638,

4647, 4656, 4665, 4674, 4683, 4691, 4693, 4702, 4794, 4795, 4797, 4799, 4801, 4803, 4805, 4807, 4809, 4882	\discretionary 209
\def_test_function_new:npx 1044	\displayindent 174
\default@ds 648	\displaylimits 184
\defaultthyphenchar 324	\displaystyle 162
\defaultskewchar 325	\displaywidowpenalties 412
\DefineCrossReferences 5073, 5079	\displaywidowpenalty 173
\delcode 355	\displaywidth 175
\delimiter 149	\divide 52
\delimiterfactor 198	\documentclass 5069
\delimitershortfall 197	\donotcheck 1214, 1243
\depthof 5525	\doublehyphendemerits 242
\detokenize 372	\dp 353
\dim_add:cn 155, 3265	\ds@ 649
\dim_add:Nc 3265	\dump 336
\dim_add:Nn 155, 3265, 5182	\dumpLaTeXstate 3981
\dim_compare:nNnF 155, 3289, 3320, 3326	
\dim_compare:nNnT 155, 3289, 3317, 3323	E
\dim_compare:nNnTF 155, 3289, 4391	\E 4709
\dim_compare_p:nNn 156, 3309	\edef 40
\dim_dowhile:nNnF 156, 3316	\efcode 437
\dim_dowhile:nNnT 156, 3316	\else 93
\dim_eval:n 155, 3254, 3266, 3287, 3290, 3297, 3303, 3310, 3820– 3823, 3862, 3877, 4386, 5425, 5426	\else: 20, 670, 755, 774, 778, 788, 789, 792, 794, 803, 804, 806, 823, 1046, 1048, 1050, 1055, 1057, 1059, 1128, 1172, 1174, 1187, 1189, 1203, 1205, 1207, 1209, 1303, 1368, 1380, 1459, 1467, 1498, 1511, 1604, 1755, 1765, 1774, 1784, 1976, 2037, 2200, 2337, 2521, 2530, 2979, 2984, 2987, 2990, 3017, 3104, 3129, 3292, 3305, 3312, 3466, 3609, 3801, 4036, 4044, 4062, 4071, 4080, 4088, 4094, 4105, 4114, 4136, 4139, 4142, 4318, 4321, 4324, 4373, 4562, 4571, 4580, 4589, 4598, 4607, 4616, 4625, 4634, 4643, 4652, 4661, 4670, 4679, 4689, 4698, 4730, 4741, 4752, 4763, 4846, 4849, 4853, 4856, 4863, 4865, 4867, 4869, 4871, 4873, 4919, 4926, 4941, 5238, 5240, 5242, 5263–5268, 5270, 5284, 5287, 5290, 5300, 5303, 5306, 5398, 5445, 5451, 5457, 5458, 5463
\dim_gadd:cn 155, 3265	\emergencystretch 257
\dim_gadd:Nn 155, 3265, 5185	\end 131, 5116
\dim_gset:cc 3254	\endcsname 133
\dim_gset:cn 154, 3254	\endgroup 66
\dim_gset:Nc 3254	\endinput 105
\dim_gset:Nn 154, 3254, 5179, 5382	\endL 420
\dim_gsub:cn 3272	\endlinechar 12, 147
\dim_gsub:Nn 155, 3272, 5191	\endR 422
\dim_gzero:c 154, 3264	\engine_if_aleph:TF 33, 1107, 2054
\dim_gzero:N 154, 3261	\eqno 167
\dim_new:c 154, 3247	\err_display_aux:w 172, 3570, 3621, 3634
\dim_new:N 154, 3247, 3279–3284, 5133–5135	\err_fatal:nn 171, 3571, 3648
\dim_new_l:N 154, 3247	
\dim_set:cn 154, 3254	
\dim_set:Nc 3254	
\dim_set:Nn 154, 3254, 5176	
\dim_sub:cn 3272	
\dim_sub:Nc 3272	
\dim_sub:Nn 155, 3272, 5188	
\dim_use:c 155, 3277	
\dim_use:N 155, 3277, 4386	
\dim_while:nNnF 156, 3316	
\dim_while:nNnT 156, 3316	
\dim_zero:c 154, 3261	
\dim_zero:N 154, 3261	
\dimen 346	
\dimendef 45	
\dimexpr 399	

- \err_fatal_noline:nn 3571, 3662
- \err_file_close:N 171, 3591, 3779
- \err_file_new:Nn .. 171, 3584, 3667, 3668
- \err_help_ignored: 3670, 3719, 3776
- \err_help_return_or_X:
..... 3677, 3679, 3687, 3765, 3771
- \err_help_textlost: 3676, 3750, 3755, 3760
- \err_help_trouble:
.. 3685, 3714, 3724, 3731, 3736, 3743
- \err_info:nn 170, 3552, 3639
- \err_info_noline:nn 3558, 3653
- \err_interrupt:NNw 171, 3564, 3635
- \err_interrupt_new:NNnnn 172, 3606, 3637
- \err_interrupt_new_aux:w 172, 3614, 3615
- \err_kernel_fatal:n 173, 3638
- \err_kernel_fatal_noline:n 3638
- \err_kernel_info:n . 173, 3588, 3603, 3638
- \err_kernel_info_noline:n 3638
- \err_kernel_interrupt:Nw 173, 3635
- \err_kernel_interrupt_new:NNnnn
.... 173, 3636, 3689, 3699, 3705,
3712, 3717, 3722, 3727, 3734, 3739,
3746, 3753, 3758, 3763, 3768, 3774
- \err_kernel_warn:n 173, 3638
- \err_kernel_warn_noline:n 3638
- \err_latex_bug:x
... 173, 745, 759, 779, 932, 1169,
1175, 1184, 1190, 1210, 1301, 2123,
2384, 2519, 3586, 3593, 3595, 3610
- \err_message:x 172, 3528, 3630
- \err_msgline_aux:NNnnn . 172, 3617, 3619
- \err_newline: 171, 3553,
3556, 3559, 3562, 3572, 3577, 3581
- \err_warn:nn 170, 3552, 3643
- \err_warn_noline:nn 3558, 3657
- \errhelp 113
- \errmessage 107
- \ERROR .. 2104, 2105, 2178, 2182, 4481, 4482
- \errorcontextlines 114
- \errorstopmode 128
- \escapechar 146
- \etex_beginL:D 419
- \etex_beginR:D 421
- \etex_botmarks:D 368
- \etex_clubpenalties:D 410
- \etex_currentgrouplevel:D 384
- \etex_currentgrouptype:D 385, 4150
- \etex_currentifbranch:D 381
- \etex_currentiflevel:D 380
- \etex_currentifttype:D 382
- \etex_detokenize:D 372, 1519
- \etex_dimexpr:D 399, 3287,
5326, 5335, 5352, 5361, 5409, 5421
- \etex_displaywidowpenalties:D 412
- \etex_endL:D 420
- \etex_endR:D 422
- \etex_eTeXrevision:D 364
- \etex_eTeXversion:D 363
- \etex_everyeof:D 424
- \etex_firstmarks:D 367
- \etex_fontchardp:D 392
- \etex_fontcharht:D 391
- \etex_fontcharic:D 394
- \etex_fontcharwd:D 393
- \etex_glueexpr:D 400, 3211,
5327, 5336, 5353, 5362, 5410, 5422
- \etex_glueshrink:D 403, 3244
- \etex_glueshrinkorder:D 405, 3232
- \etex_gluestretch:D 402, 3243
- \etex_gluestretchorder:D 404, 3231
- \etex_gluetomu:D 406
- \etex_ifcsname:D 361, 686
- \etex_ifdefined:D 360, 685
- \etex_iffontchar:D 390
- \etex_interactionmode:D 388
- \etex_interlinepenalties:D 409
- \etex_lastlinefit:D 408
- \etex_lastnodetype:D 389, 4152, 4154
- \etex_marks:D 365
- \etex_middle:D 413
- \etex_muexpr:D . 401, 3334, 3336, 3338,
5328, 5337, 5354, 5363, 5411, 5423
- \etex_mutogluue:D 407, 5399, 5400
- \etex_numexpr:D 398, 923, 925, 934, 1989,
5325, 5334, 5351, 5360, 5408, 5420
- \etex_pagediscards:D 416
- \etex_parshapedimen:D 397
- \etex_parshapeindent:D 395
- \etex_parshapelength:D 396
- \etex_predisplaydirection:D 423
- \etex_protected:D 425, 706
- \etex_readline:D 375
- \etex_savinghyphcodes:D 414
- \etex_savingvdiscards:D 415
- \etex_scantokens:D 373
- \etex_showgroups:D 386
- \etex_showifs:D 387
- \etex_showtokens:D 374
- \etex_splitbotmarks:D 370
- \etex_splitdiscards:D 417
- \etex_splitfirstmarks:D 369
- \etex_TeXETstate:D 418
- \etex_topmarks:D 366
- \etex_tracingassigns:D 376
- \etex_tracinggroups:D 383
- \etex_tracingifs:D 379
- \etex_tracingnesting:D 378
- \etex_tracingscantokens:D 377

- `\etex_unexpanded:D` 371, 689
- `\etex_unless:D` 362, 674
- `\etex_widowpenalties:D` 411
- `\eTeXrevision` 364
- `\eTeXversion` 363
- `\everycr` 75
- `\everydisplay` 176
- `\everyeof` 424
- `\everyhbox` 315
- `\everyjob` 344
- `\everymath` 200
- `\everypar` 263
- `\everyvbox` 316
- `\ExecuteOptions` 27
- `\exhyphenpenalty` 239
- `\exp_after:cc` 5039
- `\exp_after:NN` 83, 687, 696, 698, 754, 756, 783, 808–813, 819, 857–865, 867, 869, 871, 873, 875, 877, 879, 881, 883, 885, 887, 889, 891, 893, 895, 897, 899, 901, 903, 905, 907, 909, 911, 930, 1025, 1046, 1048, 1050, 1052, 1055, 1057, 1059, 1061, 1091, 1102, 1103, 1112, 1143, 1165, 1180, 1199, 1304, 1491, 1496, 1508, 1524, 1525, 1586, 1589, 1592, 1595, 1598, 1603, 1608, 1665, 1672, 1679, 1702, 1721, 1753, 1762, 1772, 1781, 1818, 1821, 1824, 1825, 1834, 1838, 1839, 1844, 1846, 1848, 1850, 1855, 1858, 1861–1863, 1865, 1869, 1916, 1918, 1921, 1923–1939, 1941–1946, 1948, 1949, 1951, 1955, 1957, 1961, 1977, 2042, 2126, 2132, 2136, 2172, 2199, 2201, 2204, 2269, 2337, 2338, 2341, 2389, 2394, 2433, 2434, 2522, 2529, 2531, 2534, 2535, 2570, 2674, 2675, 2679, 2688, 2689, 2703, 2704, 2708, 2871, 2875, 3096–3104, 3291, 3293, 3298, 3304, 3306, 3367, 3369, 3379, 3390, 3395, 3407, 3408, 3411, 3414, 3415, 3423, 3425, 3431, 3507, 3508, 3962, 3964, 3969, 3971, 4019, 4021, 4026, 4033, 4035, 4037, 4043, 4045, 4087, 4089, 4161, 4317, 4319, 4323, 4325, 4346, 4351, 4355, 4361, 4368, 4596, 4686, 4696, 4714, 4721, 4731, 4742, 4753, 4764, 4772, 4779, 4786, 4825, 4831, 4837, 4918, 4920, 4925, 4927, 4936, 4940, 4942, 5034, 5040, 5041, 5150, 5152, 5237
- `\exp_arg:x` 81, 1829, 1830, 1839
- `\exp_arg_next:nnn` 1808, 1818, 1821, 1824, 1834, 1838, 1847, 1855, 1858, 1861
- `\exp_args:NC` 82, 1872
- `\exp_args:Nc` 81, 1030, 1035, 1039, 1145–1147, 1291, 1307, 1332, 1334, 1337, 1338, 1354, 1357, 1360, 1362, 1374, 1385, 1414–1419, 1448, 1449, 1460, 1463–1465, 1469, 1472–1474, 1494, 1531, 1536, 1550, 1556, 1667, 1674, 1681, 1706, 1708, 1727, 1730, 1733, 1734, 1741, 1742, 1879, 1938, 1999–2002, 2005, 2006, 2008, 2014, 2015, 2017, 2021, 2025, 2027, 2107, 2118, 2120, 2129, 2140, 2161, 2174, 2184, 2188, 2208, 2211, 2216, 2219, 2222, 2291, 2294, 2307, 2366, 2373, 2378–2380, 2391, 2410, 2419, 2426, 2437, 2461, 2468, 2482, 2525, 2539, 2544, 2547, 2549, 2563, 2565, 2567, 2575, 2579, 2616, 2667–2669, 2718, 2730, 2799–2802, 2809, 2821, 2822, 2824, 2826, 2833, 2852–2854, 3151, 3163, 3164, 3176, 3177, 3184, 3204, 3207, 3210, 3253, 3256, 3258, 3263, 3264, 3268, 3271, 3273, 3276, 3278, 3352, 3365, 3432–3435, 3442–3444, 3458, 3461, 3471–3473, 3486–3488, 3495, 3796, 3802, 3804–3806, 3808, 3812, 3817, 3819, 3825, 3827, 3829, 3831, 3833, 3835, 3837, 3839, 3847, 3849, 3852, 3854, 3868, 3870, 3873, 3875, 3878, 3880, 3883, 3888, 3894, 3896, 3919, 4293–4295, 4301, 4305, 4309, 4313, 4329, 5538, 5544
- `\exp_args:Ncc` 82, 1027, 1032, 1036, 1041, 1151–1153, 1355, 1358, 1450, 1470, 1478–1480, 1880, 1938, 2019, 2023, 2163, 2190, 2375, 2428, 2594, 2617, 2634, 2670–2672, 2720, 3260, 3459, 3462, 3489–3491, 3497, 3810, 3814, 3855, 4331
- `\exp_args:Nccc` . 83, 1881, 1938, 2194, 2500
- `\exp_args:Ncco` 83, 1872, 2618
- `\exp_args:Nccx` 1872, 2619
- `\exp_args:NcE` 1545, 1869, 2455
- `\exp_args:NcNc` 83, 1872, 2033, 5034
- `\exp_args:NcNo` 83, 1894
- `\exp_args:Ncnx` 83, 1872
- `\exp_args:Nco` ... 82, 1333, 1882, 1948, 2162, 2207, 2427, 2538, 4228, 4229
- `\exp_args:Ncx` 82, 1872, 2209
- `\exp_args:Nd` 1870
- `\exp_args:NE` 1865–1868
- `\exp_args:NEE` 1868
- `\exp_args:Nf` ... 82, 1746, 1748, 1872, 3109

- `\exp_args:Nfo` 1872, 3093
`\exp_args:NNC` 82, 1872, 2220, 2429
`\exp_args:NNc` 82,
1026, 1031, 1034, 1040, 1148–1150,
1353, 1356, 1468, 1475–1477, 1874,
1938, 2018, 2022, 2160, 2186, 2189,
2374, 2578, 2614, 2664–2666, 2719,
3257, 3259, 3269, 3274, 3457, 3460,
3483–3485, 3496, 3809, 3813, 4330
`\exp_args:NNd` 1330, 1336, 1871, 3409, 3427
`\exp_args:NNE` 1866
`\exp_args:NNf` .. 82, 1331, 1872, 3412, 3429
`\exp_args:Nnf` 1872
`\exp_args:Nnnc` 83, 1872, 2499
`\exp_args:NNNE` 1867
`\exp_args:NnnN` 83, 1872
`\exp_args:NNNo` 1872, 1925
`\exp_args:NNno` 1872, 2610
`\exp_args:NNno` 83, 1872
`\exp_args:NNnx` 1872, 2611
`\exp_args:Nnnx` 83, 1872
`\exp_args:NNO` 1925, 3544
`\exp_args:NNo` 82, 1042, 1043, 1329, 1335,
1875, 1925, 2141, 2145, 2159, 2218,
2411, 2425, 2537, 2546, 2633, 3381,
3406, 3417, 3426, 3440, 4238, 4246
`\exp_args:Nno` 82, 1872, 2420, 3090
`\exp_args:NNOo` 83, 1873, 1925
`\exp_args:NNoo` 1872, 2613
`\exp_args:NNox` 1872, 2612
`\exp_args:Nnox` 83, 1872
`\exp_args:NNx` 82, 1872, 2146,
3388, 3416, 3418, 3428, 3441, 5056
`\exp_args:Nnx` .. 82, 1872, 2142, 2412, 2421
`\exp_args:No` 81, 1502, 1514,
1555, 1569, 1688, 1883, 1925, 1941,
2481, 2885, 2888, 2890, 2965, 3062,
3118, 4030, 4050, 4066, 4100, 4119–
4122, 4201, 4202, 4209, 4218, 4252,
4253, 4260, 4268, 4376, 4386, 5381
`\exp_args:NOc` 82, 1877, 1938
`\exp_args:NOo` 82, 1878, 1925
`\exp_args:Noo` 82, 1872, 2967
`\exp_args:N00o` 83, 1876, 1925
`\exp_args:NOoo` 2615
`\exp_args:Noox` 83, 1872
`\exp_args:Nox` 82, 1872
`\exp_args:Nx` 82, 1872
`\exp_args:Nxo` 82, 1914
`\exp_args:Nxx` 82, 1872
`\exp_args_form:x:w` 1970, 1973
`\exp_C_aux:nnn` 1844, 1845
`\exp_def_form:nnn` 1950
`\exp_not:c` 83,
1923, 1974, 4407, 4411, 4416, 4420,
4423, 4425–4427, 4432, 4434–4436,
4441, 4443–4445, 4450, 4452–4454,
4459, 4461, 4463, 4465, 4467, 4469,
4471, 4473, 4475–4477, 5021, 5026
`\exp_not:d` 83, 1393, 1407, 1432, 1437, 1916
`\exp_not:E` 84, 1923
`\exp_not:f` 83, 1916
`\exp_not:N` 83, 687, 1602, 1603,
1607, 1608, 1762, 1763, 1772, 1781,
1782, 1923, 1924, 4079, 4113, 4412,
4415, 4419, 4423, 4432, 4441, 4450,
4560, 4569, 4578, 4587, 4596, 4605,
4614, 4623, 4632, 4641, 4650, 4668,
4677, 4696, 4924, 4939, 5495, 5498,
5501, 5504, 5507, 5511, 5515, 5519
`\exp_not:n` . 83, 687, 1055, 1057, 1059,
1061, 1577, 1580, 1583, 1586, 1589,
1592, 1595, 1598, 1604, 1609, 1616,
1617, 1621, 1626, 1632, 1647, 1916,
1918, 1921, 2171, 2300, 2316, 2574,
2580, 2584, 3932, 3935, 3938, 4079,
4113, 4408, 4898, 4899, 4906, 4908,
5025, 5487, 5490, 5495, 5498, 5501,
5504, 5507, 5511, 5515, 5519, 5523
`\exp_not:o` 83,
1388, 1393, 1397, 1402, 1407, 1411,
1422, 1427, 1432, 1437, 1441, 1445,
1622, 1631, 1636, 1637, 1641, 1652,
1694, 1696, 1714, 1717, 1916, 2192
`\exp_stop_f:` 84, 1823
`\expandafter` 63
`\expanded` 1829
`\ExplSyntaxOff` 4, 537,
547, 606, 612, 620, 633, 638, 646, 5093
`\ExplSyntaxOn`
. 4, 528, 546, 597, 601, 618, 660, 5071
`\ExplSyntaxPopStack` 608, 615
`\ExplSyntaxStack` 605, 608, 615
`\ExplSyntaxStatus` 529, 538, 605
- F**
- `\fam` 55
`\fi` 94
`\fi:` ... 20, 670, 757, 775, 781, 788, 791,
794, 795, 805, 806, 819, 820, 823,
1046, 1048, 1050, 1052, 1055, 1057,
1059, 1061, 1127–1130, 1177, 1178,
1192, 1212, 1213, 1305, 1370, 1382,
1459, 1467, 1500, 1511, 1604, 1757,
1767, 1776, 1786, 1978, 2039, 2042,
2123, 2202, 2270, 2338, 2341, 2385,
2523, 2532, 2680, 2709, 2899, 2908,

2940, 2953, 2986, 2992, 2993, 2995, 3019, 3105, 3131, 3294, 3299, 3307, 3314, 3468, 3613, 3633, 3741, 3801, 3954, 4022, 4027, 4038, 4046, 4062, 4073, 4080, 4090, 4094, 4107, 4114, 4136, 4139, 4142, 4147, 4148, 4320, 4326, 4327, 4373, 4564, 4573, 4582, 4591, 4600, 4609, 4618, 4627, 4636, 4645, 4654, 4663, 4672, 4681, 4689, 4700, 4733, 4744, 4755, 4766, 4848, 4855, 4858, 4859, 4875–4880, 4921, 4928, 4943, 5244–5246, 5273–5279, 5292–5294, 5308–5310, 5328, 5337, 5354, 5363, 5401, 5411, 5423, 5445, 5451, 5457, 5463, 5464, 5546, 5556	\frozen@everymath 627
\file_input_stop: 2763	\futurelet 50
\file_not_found:nTF 1154, 3948	
\fileauthor 4, 566	
\filedate 4, 579, 583, 588, 591, 667, 1160, 1280, 1804, 1984, 2098, 2230, 2282, 2358, 2557, 2766, 3140, 3342, 3517, 3785, 3902, 4007, 4129, 4488, 5005, 5120	
\filedescription 565, 583, 591, 667, 1160, 1280, 1804, 1984, 2098, 2230, 2282, 2358, 2557, 2766, 3140, 3342, 3517, 3785, 3902, 4007, 4129, 4488, 5005, 5120	
\filename 4, 563, 583, 591, 667, 1160, 1280, 1804, 1984, 2098, 2230, 2282, 2358, 2557, 2766, 3140, 3342, 3517, 3785, 3902, 4007, 4129, 4488, 5005, 5120	
\filenameext 4, 580, 587	
\filetimestamp 4, 581, 589	
\fileversion 4, 564, 583, 591, 667, 1160, 1280, 1804, 1984, 2098, 2230, 2282, 2358, 2557, 2766, 3140, 3342, 3517, 3785, 3902, 4007, 4129, 4488, 5005, 5120	
\finalhyphendemerits 243	
\firstchoice@true 5535	
\firstmark 141	
\firstmarks 367	
\floatingpenalty 288	
\fltovf 3758	
\font 54	
\fontcharhp 392	
\fontcharht 391	
\fontcharic 394	
\fontcharwd 393	
\fontdimen 321	
\fontname 145	
\frozen@everydisplay 628	
	G
	\G 4709
	\g@addto@macro 610
	\g_box_allocation_seq 3791
	\g_calc_A_dim 5130, 5176, 5179, 5182, 5185
	\g_calc_A_int 5130, 5158, 5161, 5164, 5167, 5170, 5173, 5188, 5191, 5343, 5355, 5364
	\g_calc_A_muskip 5130, 5212, 5216, 5220, 5224, 5228, 5232
	\g_calc_A_register 5126, 5143, 5153, 5253, 5256, 5259, 5260, 5315, 5329, 5338, 5343, 5382, 5443, 5449, 5455, 5461
	\g_calc_A_skip 5130, 5194, 5197, 5200, 5203, 5206, 5209
	\g_clist_inline_level_int 2448, 2452, 2453, 2456, 2458, 2465, 2466, 2469, 2471
	\g_cs_dump_name_tlp 3910, 3921, 3922, 3950
	\g_cs_dump_seq 192, 3911, 3918, 3929
	\g_cs_trace_seq .. 3915, 3972, 3977, 3985
	\g_err_curr_fname 173, 3583, 3585, 3586, 3589, 3592, 3594, 3603, 3604, 3608, 3611
	\g_err_help_toks 173, 3549, 3623
	\g_file_curr_name_tlp 3527, 3548
	\g_gen_sym_num 192, 3936, 3937, 3956, 3959, 3960
	\g_ggen_sym_num 192, 3938, 3939, 3956, 3966, 3967
	\g_peek_token 225, 4884, 4889
	\g_prg_inline_level_int 4223
	\g_prop_inline_level_num 130, 2721
	\g_register_trace_seq .. 3974, 3980, 3993
	\g_testa_tlp 60, 1485
	\g_testb_tlp 60, 1485
	\g_tlp_inline_level_num 1532
	\g_tmpa_bool 205, 4290
	\g_tmpa_dim 156, 3279
	\g_tmpa_int 139, 2955
	\g_tmpa_num 93, 2048
	\g_tmpa_skip 154, 3212
	\g_tmpa_tlp 60, 1483
	\g_tmpa_toks 165, 3498
	\g_tmpb_dim 156, 3279
	\g_tmpb_int 139, 2955
	\g_tmpb_num 93, 2048
	\g_tmpb_skip 154, 3212
	\g_tmpb_tlp 60, 1483
	\g_tmpb_toks 165, 3498

- \g_tmpc_toks 3498
- \g_toks_allocation_seq 3499
- \g_trace_box_breadth_int 1236, 1256
- \g_trace_box_depth_int 1237, 1257
- \g_trace_chars_status 1232, 1252
- \g_trace_commands_status 1228, 1248
- \g_trace_macros_status 1233, 1253
- \g_trace_online_status 1238, 1247
- \g_trace_output_status 1231, 1251
- \g_trace_pages_status 1230, 1250
- \g_trace_paragraphs_status .. 1234, 1254
- \g_trace_restores_status 1235, 1255
- \g_trace_statistics_status .. 1229, 1249
- \g_xref_all_curr_deferred_fields_plist
..... 5014, 5059
- \g_xref_all_curr_immediate_fields_plist
..... 5014, 5055
- \gaddtolength 5531
- \gdef 41
- \gdef:cNn 29, 951
- \gdef:cNx 29, 951
- \gdef:cpn 30, 857, 4226
- \gdef:cpx 30, 857
- \gdef:NNn 29, 951
- \gdef:NNx 29, 951
- \gdef:No 1042
- \gdef:Npn 30, 719, 842, 859, 955,
957, 1043, 1194, 1289, 1317, 1326, 3932
- \gdef:Npx 30, 719, 844,
860, 956, 958, 1295, 1320, 1327, 5402
- \gdef_long:cNn 29, 967
- \gdef_long:cNx 29, 967
- \gdef_long:cpn
30, 865, 1534, 1543, 2453, 2466, 2724
- \gdef_long:cpx 30, 865
- \gdef_long:NNn 29, 967
- \gdef_long:NNx 29, 967
- \gdef_long:Npn 30, 719, 846, 869, 971, 973
- \gdef_long:Npx 30, 719, 848, 871, 972, 974
- \gdef_long_new:cNn 26, 967
- \gdef_long_new:cNx 26, 967
- \gdef_long_new:cpn 27, 865
- \gdef_long_new:cpx 27, 865
- \gdef_long_new:NNn 26, 967
- \gdef_long_new:NNx 26, 967
- \gdef_long_new:Npn . 27, 841, 877, 979, 981
- \gdef_long_new:Npx . 27, 841, 879, 980, 982
- \gdef_new:cNn 26, 951
- \gdef_new:cNx 26, 951
- \gdef_new:cpn 27, 857
- \gdef_new:cpx 27, 857
- \gdef_new:NNn 26, 951
- \gdef_new:NNx 26, 951
- \gdef_new:Npn 27, 841, 863, 963, 965
- \gdef_new:Npx 27, 841, 864, 964, 966
- \gdef_protected:cNn 29, 983
- \gdef_protected:cNx 29, 983
- \gdef_protected:cpn 31, 880
- \gdef_protected:cpx 31, 880
- \gdef_protected:NNn 29, 983
- \gdef_protected:NNx 29, 983
- \gdef_protected:Npn
..... 31, 719, 850, 885, 987, 989
- \gdef_protected:Npx
..... 31, 719, 852, 887, 988, 990
- \gdef_protected_long:cNn 29, 999
- \gdef_protected_long:cNx 29, 999
- \gdef_protected_long:cpn 31, 896
- \gdef_protected_long:cpx 31, 896
- \gdef_protected_long:NNn 29, 999
- \gdef_protected_long:NNx 29, 999
- \gdef_protected_long:Npn
..... 31, 719, 854, 901, 1003, 1005
- \gdef_protected_long:Npx
..... 31, 719, 856, 903, 1004, 1006
- \gdef_protected_long_new:cNn ... 27, 999
- \gdef_protected_long_new:cNx ... 27, 999
- \gdef_protected_long_new:cpn ... 28, 896
- \gdef_protected_long_new:cpx ... 28, 896
- \gdef_protected_long_new:NNn ... 27, 999
- \gdef_protected_long_new:NNx ... 27, 999
- \gdef_protected_long_new:Npn
..... 28, 841, 909, 1016, 1020
- \gdef_protected_long_new:Npx
..... 28, 841, 911, 1018, 1022
- \gdef_protected_new:cNn 26, 983
- \gdef_protected_new:cNx 26, 983
- \gdef_protected_new:cpn 28, 880
- \gdef_protected_new:cpx 28, 880
- \gdef_protected_new:NNn 26, 983
- \gdef_protected_new:NNx 26, 983
- \gdef_protected_new:Npn
..... 28, 841, 893, 995, 997
- \gdef_protected_new:Npx
..... 28, 841, 895, 996, 998
- \GetIdInfo 4, 556
- \GetIdInfoAuxCVS:w 556
- \GetIdInfoAuxi:w 556
- \GetIdInfoAuxii:w 556
- \GetIdInfoAuxSVN:w 556
- \getname 5083, 5111, 5114
- \getpage 5086, 5112, 5115
- \getvaluepage 5089, 5112, 5115
- \glet:cc 31, 1033, 4289
- \glet:cN 31, 1033, 4279, 4281, 4288
- \glet:Nc 31, 1033, 4287
- \glet:NN .. 31, 1033, 1106, 1196, 1345,
1351, 2187, 2372, 4278, 4280, 4286

- \glet_new:cc 28, 1033
 - \glet_new:cN 28, 1033
 - \glet_new:Nc 28, 1033
 - \glet_new:NN 28, 1033, 4541
 - \global 56
 - \globaldefs 60
 - \glueexpr 400
 - \glueshrink 403
 - \glueshrinkorder 405
 - \gluestretch 402
 - \gluestretchorder 404
 - \gluetomu 406
 - \group_align_safe_begin:
... 202, 4146, 4345, 4350, 4900, 4909
 - \group_align_safe_end:
..... 202, 4146, 4364, 4366,
4370, 4371, 4898, 4899, 4908, 4913
 - \group_begin: 32,
699, 2309, 3565, 3607, 3924, 4544,
4704, 4811, 5044, 5074, 5146, 5249,
5250, 5316, 5342, 5376, 5394, 5413,
5428, 5433, 5439, 5469, 5474, 5552
 - \group_end: . 32, 699, 2309, 3601, 3618,
3622, 3942, 4558, 4712, 4819, 5052,
5077, 5151, 5154, 5259, 5260, 5263,
5314, 5315, 5341, 5390, 5394, 5405,
5417, 5435, 5465, 5468, 5473, 5554
 - \group_execute_after:N
702, 5249, 5250, 5316, 5317, 5346, 5471
 - \gsetlength 5531
 - \gtmp:w 1140
- H**
- \H 4709
 - \halign 67
 - \hangafter 245
 - \hangindent 246
 - \hbadness 307
 - \hbox 302
 - \hbox:n 185, 3871
 - \hbox_gset:cn 185, 3872
 - \hbox_gset:Nn 185, 3872
 - \hbox_gset_inline_begin:c 186, 3881
 - \hbox_gset_inline_begin:N 186, 3881
 - \hbox_gset_inline_end: 186, 3890
 - \hbox_gset_to_wd:cnn 185, 3876
 - \hbox_gset_to_wd:Nnn 185, 3876
 - \hbox_set:cn 185, 3872
 - \hbox_set:Nn 185, 3872, 5371
 - \hbox_set_inline_begin:c 186, 3881
 - \hbox_set_inline_begin:N 186, 3881
 - \hbox_set_inline_end: 186, 3881
 - \hbox_set_to_wd:cnn 185, 3876
 - \hbox_set_to_wd:Nnn 185, 3876
 - \hbox_to_wd:nn 186, 3891
 - \hbox_to_zero:n 3891
 - \hbox_unpack:c 3893
 - \hbox_unpack:N 186, 3893
 - \hbox_unpack_clear:c 3893
 - \hbox_unpack_clear:N 186, 3893
 - \heightof 5525
 - \hfil 210
 - \hfill 212
 - \hfilneg 211
 - \hfuzz 309
 - \hoffset 284
 - \holdinginserts 287
 - \hrule 223
 - \hsize 248
 - \hskip 213
 - \hss 214
 - \ht 352
 - \hyphenation 338
 - \hyphenchar 322
 - \hyphenpenalty 240
- I**
- \I 4709
 - \if 76
 - \if:w 21, 670, 753, 772, 777,
787, 793, 816, 819, 820, 1086, 1100,
1366, 1378, 1504, 1506, 1513, 1516,
1760, 1770, 1779, 1789, 2268, 2678,
3127, 3464, 3470, 4066, 4084, 4086,
4100, 4118, 4292, 4351, 4373, 4567,
4576, 4585, 4594, 4603, 4612, 4621,
4630, 4639, 4648, 4657, 4675, 4684,
4689, 4691, 4694, 4703, 4794, 4796,
4798, 4800, 4802, 4804, 4806, 4808,
4810, 4843, 4844, 4850, 4851, 4862,
4864, 4866, 4868, 4870, 4872, 4883
 - \if_box_empty:N ... 183, 3797, 3801, 3803
 - \if_case:w
. 94, 1988, 2893, 2902, 2930, 2943,
3095, 3740, 4315, 4316, 4322, 5324,
5333, 5350, 5359, 5407, 5419, 5440
 - \if_catcode:w 21, 670, 4560,
4569, 4578, 4587, 4596, 4605, 4614,
4623, 4632, 4641, 4650, 4668, 4924
 - \if_charcode:w 21,
670, 1208, 1762, 1772, 1781, 4677, 4939
 - \if_cs_exist:N 21, 685, 785, 1107
 - \if_cs_exist:w 21, 685, 1096, 1105
 - \if_cs_meaning_eq:NN 20, 678
 - \if_dim:w 155, 3288, 3290,
3297, 3303, 3310, 5450, 5456, 5462
 - \if_eof:w 111, 2336, 2339
 - \if_false: 20, 670, 3955, 4147

- \if_hbox:N 183, 3797
- \if_meaning:NN
20, 678, 786, 802–804, 1091, 1103,
1144, 1300, 1459, 1462, 1467, 1471,
1496, 1511, 1618, 1623, 1628, 1633,
1638, 1643, 1648, 1653, 1658, 1753,
1975, 2121, 2198, 2377, 2382, 2518,
2528, 2707, 3608, 3620, 4019, 4025,
4033, 4042, 4059, 4061, 4093, 4097,
4659, 4666, 4728, 4739, 4750, 4761,
4917, 5236, 5239, 5241, 5263–5269,
5282, 5285, 5288, 5298, 5301, 5304
- \if_mode_horizontal: . 21, 681, 4139, 4140
- \if_mode_inner: 21, 681, 4142, 4143
- \if_mode_math: 21, 681, 4145
- \if_mode_vertical: ... 21, 681, 4136, 4137
- \if_num:w
. 94, 1168, 1173, 1183, 1188, 1202,
1204, 1206, 1602, 1607, 1988, 2031,
2035, 2042, 2977, 2981, 2982, 2988,
4078, 4112, 4147, 4148, 5395, 5444
- \if_num_odd:w 94, 1988, 3015, 3021
- \if_token_eq:NN 20, 678, 4696
- \if_true: 20, 670
- \if_vbox:N 183, 3797
- \ifcase 77
- \ifcat 78
- \ifcsname 361
- \ifdefined 360
- \ifdim 81
- \ifeof 82
- \iffalse 87
- \IfFileExists 1154
- \iffirstchoice@ 5535, 5542, 5549
- \iffontchar 390
- \ifhbox 83
- \ifhmode 89
- \ifinner 92
- \ifmmode 90
- \ifnum 79
- \ifodd 80
- \iftrue 88
- \ifvbox 84
- \ifvmode 91
- \ifvoid 85
- \ifx 86
- \ignorespaces 134
- \immediate 96
- \indent 230
- \input 104, 3953
- \InputIfFileExists 5076
- \inputlineno 106
- \insert 286
- \insertpenalties 289
- \int_abs:n 140, 3010
- \int_abs:nn 3012
- \int_add:cn 137, 2827
- \int_add:Nn ... 137, 2795, 2796, 2827, 5164
- \int_advance:w 2770, 2773, 2778, 2828, 2835
- \int_Alph_default_conversion_rule:n
..... 139, 2892, 2916
- \int_alph_default_conversion_rule:n
..... 139, 2892, 2912
- \int_compare:nNnF . 140, 3007, 3026, 3032
- \int_compare:nNnT . 140, 3007, 3023, 3029
- \int_compare:nNnTF . 140, 2883, 3007, 3124
- \int_compare_p:nNn
..... 140, 3013, 3127, 3231, 3232
- \int_convert_from_base_ten:nn 141, 3064
- \int_convert_from_base_ten_aux:fon 3064
- \int_convert_from_base_ten_aux:nnn 3064
- \int_convert_from_base_ten_aux:non 3064
- \int_convert_letter_to_number:N
..... 3119, 3123
- \int_convert_number_to_letter:n
..... 3078, 3082, 3095
- \int_convert_number_with_rule:nnN ..
... 139, 2882, 2911, 2915, 2921, 2925
- \int_convert_to_base_ten:nn .. 141, 3106
- \int_convert_to_base_ten_aux:nn
..... 3109, 3112
- \int_convert_to_base_ten_auxi:nnN ..
..... 3113, 3115, 3118
- \int_decr:c 137, 2773
- \int_decr:N 137, 2773, 3538
- \int_div_round:nn 139, 2969
- \int_div_round_raw:nn 2969
- \int_div_truncate:nn ... 139, 2886, 2969
- \int_div_truncate_raw:nn 2969, 3085
- \int_dowhile:nNnF 140, 3022
- \int_dowhile:nNnT 140, 3022
- \int_eval:n 139, 2857,
2872, 2876, 2888, 2890, 2961, 2973,
3002, 3005, 3015, 3021, 3068, 3072,
3107, 3119, 3126, 4162, 4203, 4210,
4219, 4231, 4239, 4247, 4254, 4261,
4269, 4376, 4494, 4497, 4500, 4503,
4507, 4510, 4513, 4516, 4519, 4522,
4525, 4528, 4531, 4534, 4537, 4540
- \int_eval:w
2810, 2828, 2835, 2961, 2965, 2967,
2976, 2977, 2981, 2982, 2988, 3095
- \int_eval_end: 2810, 2828, 2835, 2961, 2997
- \int_gadd:cn 137, 2827
- \int_gadd:Nn .. 137, 2797, 2798, 2827, 5167
- \int_gdecr:c 137, 1269, 2773
- \int_gdecr:N .. 137, 2458, 2471, 2773, 4232
- \int_get_digits:n 3034, 3109

- \int_get_sign:n 3034, 3108
- \int_get_sign_and_digits:n 3034
- \int_get_sign_and_digits_aux:nNNN 3034
- \int_get_sign_and_digits_aux:oNNN 3034
- \int_gincr:c 137, 1263, 2773, 5551
- \int_gincr:N .. 137, 2452, 2465, 2773, 4225
- \int_gset:cn 137, 2810
- \int_gset:Nn 137, 2810, 5161
- \int_gsub:cn 137, 2827
- \int_gsub:Nn 137, 2827, 5173
- \int_gzero:c 137, 2823
- \int_gzero:N 137, 2823
- \int_if_odd:nF 3014
- \int_if_odd:nT 3014
- \int_if_odd:nTF 140, 3014
- \int_if_odd:p:n 140, 3014
- \int_incr:c 137, 2773
- \int_incr:N 137, 2773
- \int_max_of:nn 140, 3010
- \int_min_of:nn 140, 3010
- \int_mod:nn 139, 2888, 2969
- \int_mod_raw:nn 2969, 3082
- \int_new:c 137, 2803
- \int_new:N 137, 2067, 2070, 2448,
2803, 2955–2960, 4223, 5129–5132
- \int_new_l:N 137, 2803
- \int_pre_eval_one_arg:Nn 2965
- \int_pre_eval_one_arg:Nnn 2965
- \int_pre_eval_two_args:Nnn
..... 2965, 2970, 3000, 3003
- \int_roman_lcuc_mapping:Nnn
..... 139, 2858, 2862–2869
- \int_set:cn 137, 2810
- \int_set:Nn 137, 2067,
2070, 2810, 3534, 3551, 5145, 5158
- \int_sub:cn 137, 2827
- \int_sub:Nn 137, 2827, 5170
- \int_symbol_math_conversion_rule:n .
..... 139, 2922, 2929
- \int_symbol_text_conversion_rule:n .
..... 139, 2926, 2929
- \int_to_Alph:n 138, 2910
- \int_to_alph:n 138, 2910
- \int_to_arabic:n 138, 2857
- \int_to_number:w 2770, 2857
- \int_to_Roman:n 138, 2870
- \int_to_roman:n 138, 2870
- \int_to_roman:w 138,
1825, 1921, 2770, 2872, 2876, 3415,
3431, 4347, 4352, 4356, 4361, 4368
- \int_to_roman_lcuc:NN 139, 2870
- \int_to_symbol:n 138, 2918
- \int_use:c 138, 2855
- \int_use:N 138, 2453, 2456, 2466,
2469, 2855, 2886, 2888, 2890, 2967,
3068, 3072, 3082, 3085, 3107, 3119,
4162, 4203, 4210, 4219, 4226, 4230,
4231, 4239, 4247, 4254, 4261, 4269,
4496, 4509, 4518, 4527, 4536, 5403
- \int_while:nNnT 3535
- \int_whiledo:nNnF 140, 3022
- \int_whiledo:nNnT 140, 3022
- \int_zero:c 137, 2823
- \int_zero:N 137, 2823, 5345, 5470
- \interactionmode 388
- \interlinepenalties 409
- \interlinepenalty 268
- \io_put_deferred:Nx ... 25, 690, 742, 744
- \io_put_log:x 25, 741, 767, 2273, 3554, 3560
- \io_put_term:x 25, 741,
746, 766, 2245, 2249, 2257, 2261,
2263, 3557, 3563, 3573, 3578, 3624
- \io_show_file_lineno:
..... 3546, 3554, 3557, 3573, 3624
- \ior_close:N 111, 2333, 2334
- \ior_close:Nn 2333
- \ior_eof:Nf 111, 2339
- \ior_eof:Nf 111, 2336
- \ior_gto:NN 111, 2342
- \ior_new:N 111, 2328
- \ior_open:Nn 111, 2333
- \ior_to:NN 111, 2342
- \iow_close:N 2292, 2295, 3602, 3943
- \iow_deferred_expanded:Nn
... 110, 2298, 2314, 2316, 2322, 5056
- \iow_deferred_unexpanded:Nn .. 110, 2315
- \iow_expanded:Nn
... 110, 2298, 2300–2302, 2311,
3923, 3931, 3935, 3940, 3987, 3995
- \iow_expanded_log:n 110, 2301
- \iow_expanded_term:n 110, 2301, 3921, 3982
- \iow_long_expanded:Nn 2308
- \iow_long_expanded:Nx 110
- \iow_long_expanded_aux:w 112, 2308
- \iow_long_unexpanded:Nn
..... 110, 2308, 3595, 3616
- \iow_new:c 109, 2286
- \iow_new:N ... 109, 2286, 3582, 3909, 5063
- \iow_newline:
111, 2324, 3553, 3556, 3559, 3562,
3572, 3577, 3596, 3598, 3624, 3626,
3671, 3673, 3681, 3692, 3695, 3708
- \iow_open:cn 110, 2292
- \iow_open:Nn
... 110, 2292, 3587, 3922, 3983, 5072
- \iow_unexpanded:Nn
..... 110, 2299, 2303, 2305, 2313

- \iow_unexpanded_if_avail:cn .. 110, [2304](#)
- \iow_unexpanded_if_avail:Nn .. 110, [2304](#)
- \iow_unexpanded_term:n 110, [2303](#)
- \item 3770
- J**
- \jobname 343, 5072, 5076
- K**
- \K 4709
- \kern 221
- L**
- \L 4709
- \l_calc_B_dim [5130](#), 5176, 5179, 5182, 5185, 5448, 5450, 5451
- \l_calc_B_int . [5130](#), 5158, 5161, 5164, 5167, 5170, 5173, 5188, 5191, 5344, 5396, 5399, 5403, 5442, 5444, 5445
- \l_calc_B_muskip [5130](#), 5212, 5216, 5220, 5224, 5228, 5232, 5460, 5462, 5463
- \l_calc_B_register .. [5126](#), 5144, 5153, 5155, 5253, 5259, 5260, 5315, 5323, 5329, 5332, 5338, 5344, 5349, 5355, 5358, 5364, 5406, 5412, 5418, 5424
- \l_calc_B_skip [5130](#), 5194, 5197, 5200, 5203, 5206, 5209, 5454, 5456, 5457
- \l_calc_C_dim [5130](#), 5447, 5450, 5451
- \l_calc_C_int [5130](#), 5397, 5400, 5403, 5441, 5444, 5445
- \l_calc_C_muskip . [5130](#), 5459, 5462, 5463
- \l_calc_C_skip ... [5130](#), 5453, 5456, 5457
- \l_calc_current_type_int [5126](#), 5145, 5324, 5333, 5345, 5350, 5359, 5395, 5407, 5419, 5440, 5470
- \l_calc_expression_tlp . [5126](#), 5149, 5152
- \l_clist_remove_duplicates_clist 121, 2502, 2504, 2507, 2508, [2516](#)
- \l_cmd_arg_list 3540, 3544
- \l_err_label_token 173, [3550](#), 3564, 3600, 3620
- \l_exp_tlp 84, 1387, 1388, 1392, 1393, 1401, 1402, 1406, 1407, 1421, 1422, 1426, 1427, 1431, 1432, 1436, 1437, [1807](#), 1832–1834
- \l_loop_int [2955](#)
- \l_peek_false_tlp 227, [4891](#), 4899, 4908, 4920, 4927, 4935, 4942
- \l_peek_search_tlp [4894](#), 4897, 4905, 4936
- \l_peek_search_token 225, [4884](#), 4896, 4904, 4917, 4924
- \l_peek_token [225](#), [4884](#), 4887, 4917, 4924, 4932, 4933, 4993
- \l_peek_true_aux_tlp ... 227, 4906, [4912](#)
- \l_peek_true_remove_next_tlp . 227, [4912](#)
- \l_peek_true_tlp 227, [4891](#), 4898, 4907, 4918, 4925, 4940
- \l_replace_tlp 60
- \l_testa_tlp 60, [1485](#), 1616, 1618, 1621, 1623, 1626, 1628, 1631, 1633, 1636, 1638, 1641, 1643, 1646, 1648, 1651, 1653, 1656, 1658, 2122, 2383
- \l_testb_tlp 60, [1485](#), 1617, 1618, 1622, 1623, 1627, 1628, 1632, 1633, 1637, 1638, 1642, 1643, 1647, 1648, 1652, 1653, 1657, 1658
- \l_tlp_replace_tlp [1689](#), 1694, 1696, 1700, 1710, 1714, 1717, 1723
- \l_tmpa_bool [205](#), [4290](#)
- \l_tmpa_box 185, [3840](#), 5371, 5374
- \l_tmpa_dim 156, [3279](#)
- \l_tmpa_int ... 139, [2955](#), 3534–3536, 3538
- \l_tmpa_num 93, [2048](#)
- \l_tmpa_skip 154, [3212](#)
- \l_tmpa_tlp 60, [1489](#), 2320, 2322
- \l_tmpa_toks 165, 2489, 2492, 2493, [3498](#), 5024, 5028
- \l_tmppb_box 185, [3840](#)
- \l_tmppb_dim 156, [3279](#)
- \l_tmppb_int 139, [2955](#)
- \l_tmppb_num 93, [2048](#)
- \l_tmppb_skip 154, [3212](#)
- \l_tmppb_tlp 60, [1489](#), 2321, 2323
- \l_tmppb_toks 165, 2490, 2493, 2494, [3498](#), 3533, 3536, 3537, 3540
- \l_tmppc_dim 156, [3279](#)
- \l_tmppc_int 139, [2955](#)
- \l_tmppc_num 93, [2048](#)
- \l_tmppc_skip 154, [3212](#)
- \l_tmppc_toks [3498](#)
- \l_tmppd_dim 156, [3279](#)
- \l_xref_curr_name_tlp 5082
- \language 138
- \lastbox 295
- \lastkern 228
- \lastlinefit 408
- \lastnodetype 389
- \lastpenalty 334
- \lastskip 229
- \lccode 357
- \leaders 225
- \left 193
- \lefthyphenmin 249
- \leftmarginkern 471
- \leftskip 251
- \leqno 168
- \let 15, 38
- \let:cc 31, [1023](#), 4285

<code>\let:cN</code>	31, 1023 , 4275, 4277, 4284	<code>\mag</code>	137
<code>\let:Nc</code>	31, 1023 , 4283	<code>\mark</code>	139
<code>\let:NN</code>	31, 1023 , 1033, 1038, 1087– 1089, 1092–1094, 1097–1099, 1215– 1225, 1341, 1350, 1865, 2178, 2182, 2185, 2290, 2314, 2332, 2333, 2339, 2371, 2808, 3150, 3226, 3227, 3252, 3351, 3838, 3840, 3954, 4274, 4276, 4282, 4481, 4482, 4896, 4904, 4914, 4950, 4957, 4964, 4971, 4978, 4986, 4995, 5091, 5143, 5144, 5147, 5148, 5263–5269, 5271, 5283, 5286, 5289, 5299, 5302, 5305, 5343, 5344, 5377– 5379, 5384–5387, 5492, 5493, 5525– 5530, 5553, 5561–5563, 5565–5567	<code>\marks</code>	365
<code>\let:NwN</code>	31, 666, 670–695, 697, 699–708, 719, 720, 734, 797, 1024, 3549, 3564, 3566	<code>\mathaccent</code>	150
<code>\let_new:cc</code>	28, 1023	<code>\mathbin</code>	180
<code>\let_new:cN</code>	28, 1023 , 4273	<code>\mathchar</code>	151
<code>\let_new:Nc</code>	28, 1023	<code>\mathchardef</code>	48
<code>\let_new:NN</code>	28, 1023 , 1198, 1273, 1274, 1350, 1351, 1373, 1384, 1517–1519, 1562, 1563, 1575, 1744, 1749, 1829, 1988–1994, 2016, 2020, 2028, 2029, 2056, 2060, 2104, 2105, 2108–2117, 2119, 2175, 2185, 2187, 2212–2214, 2217, 2223, 2224, 2296, 2297, 2367–2372, 2376, 2386, 2447, 2540–2542, 2545, 2550, 2551, 2564, 2566, 2639–2654, 2731, 2770–2772, 2855, 2961–2963, 3007– 3013, 3203, 3206, 3209, 3277, 3285, 3286, 3288, 3333, 3364, 3422, 3439, 3455, 3456, 3528, 3795, 3797–3799, 3815, 3828, 3830, 3832, 3834, 3836, 3841, 3858, 3861, 3867, 3869, 3885, 3890, 3893, 3895, 4272, 4282–4289, 4297, 4492, 4501, 4514, 4523, 4532, 4542, 4543, 5509, 5513, 5517, 5521	<code>\mathchoice</code>	148
<code>\limits</code>	185	<code>\mathclose</code>	181
<code>\line</code>	3747	<code>\mathcode</code>	359
<code>\linepenalty</code>	241	<code>\mathinner</code>	182
<code>\lineskip</code>	235	<code>\mathop</code>	183
<code>\lineskiplimit</code>	236	<code>\mathopen</code>	187
<code>\long</code>	18, 23, 26, 31, 57	<code>\mathord</code>	188
<code>\looseness</code>	253	<code>\mathparagraph</code>	2935
<code>\lower</code>	290	<code>\mathpunct</code>	189
<code>\lowercase</code>	329	<code>\mathrel</code>	190
<code>\lpcode</code>	438	<code>\mathsection</code>	2934
M		<code>\mathsurround</code>	201
<code>\M</code>	4709, 4813	<code>\maxdeadcycles</code>	271
<code>\m@ne</code>	734	<code>\maxdepth</code>	272
		<code>\maxdimen</code>	3227
		<code>\maxof</code>	5486
		<code>\meaning</code>	331
		<code>\medmuskip</code>	202
		<code>\message</code>	108
		<code>\middle</code>	413
		<code>\minof</code>	5486
		<code>\mkern</code>	155
		<code>\mode_if_horizontal:F</code>	202, 4138
		<code>\mode_if_horizontal:T</code>	202, 4138
		<code>\mode_if_horizontal:TF</code>	202, 4138
		<code>\mode_if_horizontal_p:</code>	202, 4138
		<code>\mode_if_inner:F</code>	202, 4141
		<code>\mode_if_inner:T</code>	202, 4141
		<code>\mode_if_inner:TF</code>	202, 4141
		<code>\mode_if_inner_p:</code>	202, 4141
		<code>\mode_if_math:F</code>	202, 4144
		<code>\mode_if_math:T</code>	202, 4144
		<code>\mode_if_math:TF</code>	202, 2919, 4144
		<code>\mode_if_vertical:F</code>	202, 4135
		<code>\mode_if_vertical:T</code>	202, 4135
		<code>\mode_if_vertical:TF</code>	202, 4135
		<code>\mode_if_vertical_p:</code>	202, 4135
		<code>\month</code>	341
		<code>\moveleft</code>	291
		<code>\moveright</code>	292
		<code>\mskip</code>	152
		<code>\muexpr</code>	401
		<code>\multiply</code>	53
		<code>\muskip</code>	349
		<code>\muskip_add:Nn</code>	157, 3334 , 5221, 5229
		<code>\muskip_gadd:Nn</code>	157, 3334 , 5225, 5233
		<code>\muskip_gset:Nn</code>	156, 3334 , 5217

- `\muskip_gsub:Nn` 157, 3334
`\muskip_new:N` 156, 3328, 5139–5141
`\muskip_new_l:N` 156, 3328
`\muskip_set:Nn` 156, 3334, 5213
`\muskip_sub:Nn` 157, 3334
`\muskipdef` 47
`\mutoglu` 407
- N**
- `\n` 4705
`\name_pop_stack:w` 634, 645, 664
`\name_primitive:NN`
 ... 31, 35–425, 427–447, 449–455,
 457–460, 462–483, 485–496, 498–527
`\name_tmp:` 664
`\name_undefine:N` 18, 23, 26, 33, 632
`\NamesStart` 548, 5075
`\NamesStart:` 3568
`\NamesStop` 548
`\NeedsTeXFormat` 656, 2360
`\newbox` 3795
`\newcnt` 3705
`\newcount` 2808
`\newdimen` 3252
`\newif` 5535
`\newline` 3699
`\newlinechar` 103
`\newmuskip` 3333
`\newread` 2332
`\newskip` 3150
`\newtoks` 3351
`\newwrite` 2290
`\noalign` 71
`\noboundary` 206
`\nodocument` 3712
`\noexpand` 64
`\noindent` 232
`\noitemerr` 3768
`\nolimits` 186
`\nonscript` 166
`\nonstopmode` 129
`\notprerr` 3774
`\nulldelimiterspace` 199
`\nullfont` 317
`\num_abs:n` 92, 2041
`\num_abs:nn` 3012
`\num_add:cn` 91, 2024
`\num_add:Nn` 91, 1995, 1996, 2024
`\num_compare:cNcTF` .. 92, 2033, 2244, 2256
`\num_compare:nNnF`
 ... 92, 2030, 3009, 4152, 4154,
 4206, 4215, 4235, 4243, 4257, 4265
`\num_compare:nNnT` .. 92, 2030, 3008, 4150
`\num_compare:nNnTF` 92,
 2030, 2044, 2045, 2064, 2066, 3007,
 3065, 3077, 4200, 4227, 4251, 4381
`\num_compare_p:nNn` 92, 2034, 3013
`\num_decr:c` 91, 1999
`\num_decr:N` 91, 1995, 2000
`\num_elt_count:n` ... 92, 1566, 1572, 2046
`\num_elt_count_prop:Nn` 2046
`\num_eval:n`
 92, 2009, 2011, 2031, 2035, 2042, 2961
`\num_eval:w` 94, 1565, 1571,
 1988, 2009, 2962, 4315, 4316, 4322
`\num_eval_end:` 1567, 1573, 1988, 2009, 2963
`\num_gadd:cn` 91, 2024
`\num_gadd:Nn` 91, 1997, 1998, 2024
`\num_gdecr:c` 91, 1999, 2255
`\num_gdecr:N` 91, 1539, 1548, 1995, 2002, 2728
`\num_gincr:c` 91, 1999, 2243
`\num_gincr:N` 91, 1533,
 1542, 1995, 2001, 2723, 3959, 3966
`\num_gset:cn` 91, 2010
`\num_gset:Nn`
 91, 2004, 2010, 3935, 3938, 3956, 3957
`\num_gset_eq:cc` 91, 2020
`\num_gset_eq:cN` 91, 2020
`\num_gset_eq:Nc` 91, 2020
`\num_gset_eq:NN` 91, 2020
`\num_gzero:c` 91, 2003
`\num_gzero:N` 91, 2003
`\num_incr:c` 91, 1999
`\num_incr:N` 91, 1995, 1999
`\num_max_of:nn` 92, 2041, 3010
`\num_min_of:nn` 92, 2041, 3011
`\num_new:c` 91, 2007, 2236, 2237
`\num_new:N`
 91, 2007, 2048–2052, 2721, 3956, 3957
`\num_set:cn` 91, 2010, 2239, 2240
`\num_set:Nn` 91, 2003, 2010, 2024
`\num_set_eq:cc` 2016
`\num_set_eq:cN` 2016
`\num_set_eq:Nc` 2016
`\num_set_eq:NN` 2016
`\num_use:c` 92, 2028, 2248,
 2249, 2260, 2261, 2263, 2271, 2274
`\num_use:N`
 . 92, 1534, 1537, 1543, 1546, 2028,
 2724, 2727, 3937, 3939, 3960, 3967
`\num_value:w` ... 93, 1565, 1571, 1988, 4376
`\num_zero:c` 91, 2003
`\num_zero:N` 91, 2003
`\number` 326, 5088
`\numexpr` 398

O

\O	4709, 4817	\pdf_destmargin:D	454
\omit	72	\pdf_efcode:D	437
\openin	98	\pdf_elapsedtime:D	494
\openout	99	\pdf_endlink:D	506
\or	95	\pdf_endthread:D	511
\or:	94, 1129, 1988, 2894–2898, 2903–2907, 2931–2939, 2944–2952, 3096–3104, 3740, 3741, 5325–5327, 5334–5336, 5351–5353, 5360–5362, 5408–5410, 5420–5422, 5446, 5452	\pdf_escapehex:D	475
\org@onefilewithoptions	603	\pdf_escapename:D	474
\outer	58	\pdf_escapestring:D	473
\output	273	\pdf_filedump:D	483
\outputpenalty	283	\pdf_filemoddate:D	481
\over	160	\pdf_filesize:D	482
\overfullrule	311	\pdf_fontattr:D	518
\overline	191	\pdf_fontexpand:D	520
\overwithdelims	161	\pdf_fontname:D	467
		\pdf_fontobjnum:D	468
		\pdf_fontsize:D	469
		\pdf_forcepagebox:D	440
		\pdf_gamma:D	446
		\pdf_horigin:D	449
		\pdf_imageapplygamma:D	445
		\pdf_imagegamma:D	447
		\pdf_imagehicolor:D	444
		\pdf_imageresolution:D	431
		\pdf_includechars:D	470
		\pdf_inclusionerrorlevel:D	442
		\pdf_info:D	513
		\pdf_lastannot:D	490
		\pdf_lastdemerits:D	493
		\pdf_lastobj:D	486
		\pdf_lastxform:D	487
		\pdf_lastximage:D	488
		\pdf_lastximagepages:D	489
		\pdf_lastxpos:D	491
		\pdf_lastypos:D	492
		\pdf_leftmarginkern:D	471
		\pdf_linkmargin:D	453
		\pdf_literal:D	522
		\pdf_lpcode:D	438
		\pdf_mapfile:D	516
		\pdf_mapline:D	517
		\pdf_mdffivesum:D	480
		\pdf_minorversion:D	428
		\pdf_names:D	515
		\pdf_noligatures:D	526
		\pdf_normaldeviate:D	479
		\pdf_obj:D	498
		\pdf_optionalalwaysusepdfpagebox:D ..	441
		\pdf_optionpdfinclusionerrorlevel:D ..	443
		\pdf_outline:D	507
		\pdf_output:D	427
		\pdf_pageattr:D	458
		\pdf_pageheight:D	452
		\pdf_pageref:D	465
		\pdf_pageresources:D	459

P

\P	4709		
\package_provides:w	639, 640		
\PackageError	5482		
\pagedepth	275		
\pagediscards	416		
\pagefilllstretch	279		
\pagefillstretch	278		
\pagefilstretch	277		
\pagegoal	281		
\pagenumbering	5095, 5103		
\pageshrink	280		
\pagestretch	276		
\pagetotal	282		
\par	231, 631, 1798, 1985, 1986, 2233, 2283, 2284, 2558, 2559, 3343, 3344, 3566, 3786, 3787, 4008		
\parfillskip	262		
\parindent	255		
\parmoderr	3753		
\parshape	247		
\parshapedimen	397		
\parshapeindent	395		
\parshapelength	396		
\parskip	254		
\patterns	337		
\pausing	124		
\pdf_adjustspacing:D	435		
\pdf_annot:D	504		
\pdf_catalog:D	514		
\pdf_compresslevel:D	429		
\pdf_creationdate:D	464		
\pdf_decimaldigits:D	430		
\pdf_dest:D	508		

\pdf_pagesattr:D	457	\pdfforcepagebox	440
\pdf_pagewidth:D	451	\pdfgamma	446
\pdf_pkmode:D	460	\pdfhorigin	449
\pdf_pkresolution:D	432	\pdfimageapplygamma	445
\pdf_protrudechars:D	436	\pdfimagegamma	447
\pdf_randomseed:D	495	\pdfimagehicolor	444
\pdf_refobj:D	499	\pdfimageresolution	431
\pdf_refxform:D	501	\pdfincludechars	470
\pdf_refximage:D	503	\pdfinclusionerrorlevel	442
\pdf_resettimer:D	524	\pdfinfo	513
\pdf_rightmarginkern:D	472	\pdflastannot	490
\pdf_rpcode:D	439	\pdflastdemerits	493
\pdf_savepos:D	512	\pdflastobj	486
\pdf_setrandomseed:D	525	\pdflastxform	487
\pdf_shellescape:D	496	\pdflastximage	488
\pdf_startlink:D	505	\pdflastximagepages	489
\pdf_startthread:D	510	\pdflastxpos	491
\pdf_strcmp:D	477, 4079, 4113	\pdflastypos	492
\pdf_texbanner:D	463	\pdflinkmargin	453
\pdf_texrevision:D	462	\pdfliteral	522
\pdf_texversion:D	485	\pdfmapfile	516
\pdf_thread:D	509	\pdfmapline	517
\pdf_threadmargin:D	455	\pdfmdfivesum	480
\pdf_tracingfonts:D	433	\pdfminorversion	428
\pdf_trailer:D	519	\pdfnames	515
\pdf_unescapehex:D	476	\pdfnoligatures	526
\pdf_uniformdeviate:D	478	\pdfnormaldeviate	479
\pdf_uniqueresname:D	434	\pdfobj	498
\pdf_vorigin:D	450	\pdfoptionalwaysusepdfpagebox	441
\pdf_xform:D	500	\pdfoptionpdfinclusionerrorlevel	443
\pdf_xformname:D	466	\pdfoutline	507
\pdf_ximage:D	502	\pdfoutput	427
\pdfadjustspacing	435	\pdfpageattr	458
\pdfannot	504	\pdfpageheight	452
\pdfcatalog	514	\pdfpageref	465
\pdfcompresslevel	429	\pdfpageresources	459
\pdfcreationdate	464	\pdfpagesattr	457
\pdfdecimaldigits	430	\pdfpagewidth	451
\pdfdest	508	\pdfpkmode	460
\pdfdestmargin	454	\pdfpkresolution	432
\pdfelapsedtime	494	\pdfprotrudechars	436
\pdfendlink	506	\pdfrandomseed	495
\pdfendthread	511	\pdfrefobj	499
\pdfescapehex	475	\pdfrefxform	501
\pdfescapename	474	\pdfrefximage	503
\pdfescapestring	473	\pdfresettimer	524
\pdffiledump	483	\pdfsavepos	512
\pdffilemoddate	481	\pdfsetrandomseed	525
\pdffilesize	482	\pdfshellescape	496
\pdffontattr	518	\pdfstartlink	505
\pdffontexpand	520	\pdfstartthread	510
\pdffontname	467	\pdfstrcmp	477, 1575
\pdffontobjnum	468	\pdftexbanner	463
\pdffontsize	469	\pdftexrevision	462

- \pdftexversion 485
- \pdfthread 509
- \pdfthreadmargin 455
- \pdftracingfonts 433
- \pdftrailer 519
- \pdfunescapehex 476
- \pdfuniformdeviate 478
- \pdfuniquestring 434
- \pdfvorigin 450
- \pdfxform 500
- \pdfxformname 466
- \pdfximage 502
- \peek_after:NN 226, 4887, 4901, 4910, 4990
- \peek_catcode:NTF 226, 4960
- \peek_catcode_ignore_spaces:NTF 226, 4960
- \peek_catcode_remove:NTF 226, 4960
- \peek_catcode_remove_ignore_spaces:NTF 226, 4960
- \peek_charcode:NTF 226, 4974
- \peek_charcode_ignore_spaces:NTF 226, 4974
- \peek_charcode_remove:NTF 226, 4974
- \peek_charcode_remove_ignore_spaces:NTF 226, 4974
- \peek_execute_branches_catcode: .. 4916
- \peek_execute_branches_charcode: .. 4916
- \peek_execute_branches_meaning: .. 4916
- \peek_execute_branches: 4950, 4957, 4964, 4971, 4978, 4986, 4997
- \peek_execute_branches_catcode: 226, 4923, 4961, 4964, 4968, 4971
- \peek_execute_branches_charcode: ... 226, 4930, 4975, 4978, 4982, 4986
- \peek_execute_branches_charcode_aux:NN 4916
- \peek_execute_branches_meaning: 226, 4916, 4947, 4950, 4954, 4957
- \peek_gafter:NN 226, 4887
- \peek_ignore_spaces_aux: 227, 4989
- \peek_ignore_spaces_execute_branches: 227, 4951, 4958, 4965, 4972, 4979, 4987, 4989
- \peek_meaning:NTF 226, 4946
- \peek_meaning_ignore_spaces:NTF 226, 4946
- \peek_meaning_remove:NTF 226, 4946
- \peek_meaning_remove_ignore_spaces:NTF 226, 4946
- \peek_tmp:w 227, 4893, 4914, 4995
- \peek_token_generic:NNTF .. 226, 4895, 4947, 4951, 4961, 4965, 4975, 4979
- \peek_token_remove_generic:NNTF 226, 4903, 4954, 4958, 4968, 4972, 4982, 4987
- \penalty 332
- \poptabs 3729
- \postdisplaypenalty 179
- \preamerr 3739
- \predicate:nF 205, 4344
- \predicate:nT 205, 4344
- \predicate:nTF 205, 3230, 4344, 4931
- \predicate_02_0:w 4344
- \predicate_02_1:w 4344
- \predicate_88_0:w 4344
- \predicate_88_1:w 4344
- \predicate_auxii:NNN 4344
- \predicate_II_0:w 4344
- \predicate_II_1:w 4344
- \predicate_not_p:n 205, 4372
- \predicate_p:n 205, 4344, 4373
- \predisplaydirection 423
- \predisdisplaypenalty 178
- \predisplaysize 177
- \pref_global:D 25, 703, 1033, 1197, 1218, 1228–1238, 1247–1257, 1455, 2013, 2026, 2151, 2157, 2269, 2351, 2787, 2793, 2819, 2825, 2844, 2850, 3161, 3174, 3195, 3201, 3255, 3262, 3270, 3275, 3335, 3337, 3339, 3362, 3376, 3386, 3404, 3437, 3439, 3453, 3456, 3811, 3818, 3826, 3848, 3853, 3860, 3874, 3879, 3887, 4505, 4889, 5153, 5256, 5259, 5260, 5315, 5443, 5449, 5455, 5461
- \pref_global_chk: 50, 1193, 1218, 1453, 2149, 2155, 2349, 2785, 2791, 2817, 2842, 2848, 3159, 3172, 3193, 3199, 3360, 3374, 3384, 3402
- \pref_long:D 25, 703, 709, 710, 714, 717, 721, 722, 726, 729
- \pref_protected:D 25, 703, 709–712, 714, 717, 723, 724, 726, 729
- \pretolerance 258
- \prevdepth 305
- \prevgraf 264
- \prg_case_dim:nnn 206, 4385
- \prg_case_dim_aux:nnn 4385
- \prg_case_int:nnn 206, 4375
- \prg_case_int_aux:nnn 4375
- \prg_case_str:nnn 206, 4395
- \prg_case_str_aux:nnn 4395
- \prg_define_quicksort:nnn ... 4405, 4480
- \prg_dowhile:nF 206, 4332
- \prg_dowhile:nT 206, 4332

\prg_quicksort:n	207, <u>4480</u>	\prop_gput:NNo	2736
\prg_quicksort_compare:nnTF	207, <u>4481</u>	\prop_gput:Nno	128, <u>2589</u> , 2737
\prg_quicksort_function:n	207, <u>4481</u>	\prop_gput:Nnx	<u>2589</u>
\prg_replicate:nn	203, <u>4159</u>	\prop_gput:Noo	128, <u>2589</u>
\prg_replicate_	4171	\prop_gput:Nox	128, <u>2589</u>
\prg_replicate_aux:N	<u>4159</u>	\prop_gput:Ooo	128, <u>2589</u>
\prg_replicate_first_aux:N	<u>4159</u>	\prop_gput_if_new:NNn	2740
\prg_stepwise_function:nnnN	203, <u>4199</u>	\prop_gput_if_new:Nnn	129, <u>2635</u> , 2741
\prg_stepwise_function_decr:nnnN	<u>4199</u>	\prop_gset_eq:cc	129, <u>2639</u>
\prg_stepwise_function_incr:nnnN	<u>4199</u>	\prop_gset_eq:cN	<u>2639</u>
\prg_stepwise_inline:nnnn	203, <u>4223</u>	\prop_gset_eq:Nc	<u>2639</u>
\prg_stepwise_inline_decr:Nnnn	4228, 4242, 4246	\prop_gset_eq:NN	129, <u>2639</u>
\prg_stepwise_inline_decr:nnnn	<u>4223</u>	\prop_if_empty:cF	<u>2647</u>
\prg_stepwise_inline_incr:Nnnn	4229, 4234, 4238	\prop_if_empty:cT	<u>2647</u>
\prg_stepwise_inline_incr:nnnn	<u>4223</u>	\prop_if_empty:cTF	<u>2647</u>
\prg_stepwise_variable:nnnNn	203, <u>4250</u>	\prop_if_empty:NF	<u>2647</u>
\prg_stepwise_variable_decr:nnnNn	<u>4250</u>	\prop_if_empty:NT	<u>2647</u>
\prg_stepwise_variable_incr:nnnNn	<u>4250</u>	\prop_if_empty:NTF	130, <u>2647</u>
\prg_whiledo:nF	206, <u>4332</u>	\prop_if_empty_p:c	<u>2647</u>
\prg_whiledo:nT	206, <u>4332</u>	\prop_if_empty_p:N	<u>2647</u>
\ProcessOptions	28	\prop_if_eq:ccF	130, <u>2655</u>
\prop_clear:c	<u>2564</u>	\prop_if_eq:ccT	<u>2655</u>
\prop_clear:N	128, <u>2564</u> , 2591	\prop_if_eq:ccTF	<u>2655</u>
\prop_del:NN	2752	\prop_if_eq:cNF	<u>2655</u>
\prop_del:Nn	129, <u>2620</u> , 2753	\prop_if_eq:cNT	<u>2655</u>
\prop_del_aux:w	130, <u>2620</u>	\prop_if_eq:cNTF	<u>2655</u>
\prop_gclear:c	<u>2564</u>	\prop_if_eq:NcF	<u>2655</u>
\prop_gclear:N	128, <u>2564</u> , 2598	\prop_if_eq:NcT	<u>2655</u>
\prop_gdel:NN	2754	\prop_if_eq:NcTF	<u>2655</u>
\prop_gdel:Nn	129, <u>2620</u> , 2755	\prop_if_eq:NNF	130, <u>2655</u>
\prop_get:cNN	2744	\prop_if_eq:NNT	<u>2655</u>
\prop_get:cnN	129, <u>2572</u> , 2745	\prop_if_eq:NNTF	<u>2655</u>
\prop_get:NNN	2742	\prop_if_in:ccTF	130, <u>2629</u>
\prop_get:NnN	129, <u>2572</u> , 2743	\prop_if_in:NNTF	2756
\prop_get_aux:w	130, <u>2572</u>	\prop_if_in:NnTF	130, <u>2629</u> , 2757
\prop_get_del_aux:w	130, <u>2581</u>	\prop_if_in:NoTF	130, <u>2629</u>
\prop_get_gdel:NNN	<u>2581</u> , 2750	\prop_if_in_aux:w	130, <u>2629</u>
\prop_get_gdel:NnN	129, <u>2581</u> , 2751	\prop_map_break:w	130, 2679, 2708, <u>2731</u>
\prop_gget:cNN	2748	\prop_map_function:cc	129, <u>2673</u>
\prop_gget:cnN	129, <u>2576</u> , 2749	\prop_map_function:cN	<u>2673</u>
\prop_gget:NcN	129, <u>2576</u>	\prop_map_function:Nc	129, <u>2673</u> , 2726
\prop_gget:NNN	2746	\prop_map_function:NN	129, <u>2673</u>
\prop_gget:NnN	129, <u>2576</u> , 2747	\prop_map_function_aux:NNn	130
\prop_gget_aux:w	<u>2576</u>	\prop_map_function_aux:w	<u>2673</u>
\prop_gput:ccn	128, <u>2589</u>	\prop_map_inline:cN	2730
\prop_gput:cco	128, <u>2589</u>	\prop_map_inline:cn	130, <u>2721</u>
\prop_gput:ccx	128, <u>2589</u> , 5019	\prop_map_inline:NN	2730
\prop_gput:cNn	<u>2589</u> , 2738	\prop_map_inline:Nn	130, <u>2721</u>
\prop_gput:cnN	128, 2616, 2739	\prop_map_inline_aux:Nn	130
\prop_gput:Ncn	128, <u>2589</u>	\prop_new:c	128, <u>2562</u> , 5050
\prop_gput:NNn	2734	\prop_new:N	128, <u>2562</u> , 5014, 5015
\prop_gput:Nnn	128, <u>2589</u> , 2735	\prop_put:ccn	128, <u>2589</u>
		\prop_put:NNn	2732
		\prop_put:Nnn	128, <u>2589</u> , 2733

- \prop_put_aux:w 130, 2589
 - \prop_put_if_new_aux:w . 130, 2636, 2637
 - \prop_set_eq:cc 129, 2639
 - \prop_set_eq:cN 2639
 - \prop_set_eq:Nc 2639
 - \prop_set_eq:NN 129, 2639
 - \prop_split_aux:Nnn
 - 131, 2568, 2573, 2577, 2582,
 - 2590, 2597, 2621, 2623, 2630, 2636
 - \prop_use:N 5035, 5055, 5059
 - \protected 425
 - \ProvidesClass 600
 - \ProvidesExplClass 4, 594
 - \ProvidesExplPackage
 - 4, 594, 666, 1159, 1279,
 - 1803, 1983, 2097, 2229, 2281, 2357,
 - 2556, 2765, 3139, 3341, 3516, 3784,
 - 3901, 4006, 4128, 4487, 5004, 5119
 - \ProvidesPackage 596, 636
 - \pushtabs 3728
- Q**
- \q 4347, 4352, 4356, 4361, 4368
 - \q_error 4014
 - \q_mark . 1698, 1702, 1718, 1721, 3379, 4014
 - \q_nil 198,
 - 1123, 1125, 1496, 1508, 1510, 1743,
 - 1745, 1747, 1750, 1751, 1753, 1763,
 - 1772, 1782, 2172, 2394, 2402, 2704,
 - 2707, 2715, 3113, 3509, 3511, 4011,
 - 4020, 4034, 4053, 4093, 4097, 4101,
 - 4102, 4113, 4408, 4412, 4686, 4688,
 - 4715, 4717, 4722, 4724, 4732, 4735,
 - 4743, 4746, 4754, 4757, 4765, 4768,
 - 4773, 4775, 4780, 4782, 4787, 4790,
 - 4820, 4826, 4832, 4838, 5028, 5036
 - \q_no_value 198,
 - 1665, 1672, 1679, 1686, 1695, 1703,
 - 1722, 2198, 2206, 2528, 2535, 2570,
 - 2586, 2606, 2627, 2675, 2689, 2704,
 - 4011, 4059, 4061, 4067, 4068, 4079
 - \q_prop 2561, 2569,
 - 2570, 2586, 2603, 2606, 2627, 2638,
 - 2675, 2677, 2689, 2691, 2704, 2706
 - \q_recursion_stop
 - 197, 1521, 1525, 1538, 1547, 1553,
 - 1791, 2434, 2440, 2457, 2470, 2478,
 - 4016, 4055, 4057, 4377, 4387, 4397
 - \q_recursion_tail .. 197, 1521, 1525,
 - 1538, 1547, 1553, 1791, 2434, 2440,
 - 2457, 2470, 2478, 4016, 4020, 4025,
 - 4034, 4042, 4053, 4377, 4387, 4397
 - \q_stop ... 198, 1124, 1126, 1165, 1167,
 - 1181, 1182, 1200, 1201, 1662, 1665,
 - 1669, 1672, 1676, 1679, 1683, 1686,
 - 1691, 1703, 1711, 1718, 1722, 2126,
 - 2128, 2132, 2134, 2136, 2137, 2172,
 - 2197, 2206, 2389, 2390, 2394, 2395,
 - 2527, 2535, 2569, 2570, 2675, 2704,
 - 3380, 3389, 4011, 4408, 4412, 4474
 - \quark_if_nil:Nf 197, 4092
 - \quark_if_nil:nf 197, 4098
 - \quark_if_nil:NT 197, 2166, 4092
 - \quark_if_nil:nT .. 197, 4098, 4415, 4419
 - \quark_if_nil:NTF . 197, 2398, 3116, 4092
 - \quark_if_nil:nTF
 - ... 197, 4098, 4423, 4432, 4441, 4450
 - \quark_if_nil:oF 197, 4098
 - \quark_if_nil:oT 197, 4098
 - \quark_if_nil:OTF 197, 4098
 - \quark_if_nil_p:N 197, 4092
 - \quark_if_nil_p:n 197, 4098
 - \quark_if_nil_p:o 197, 4098
 - \quark_if_no_value:Nf
 - 196, 2585, 2626, 2699, 4058
 - \quark_if_no_value:nf
 - ... 196, 1670, 1692, 2692, 2698, 4058
 - \quark_if_no_value:nFT
 - 1663, 1684, 2631, 4058
 - \quark_if_no_value:NT 196, 4058
 - \quark_if_no_value:nT .. 196, 1677, 4058
 - \quark_if_no_value:NTF 196, 4058
 - \quark_if_no_value:nTF . 196, 1712, 4058
 - \quark_if_no_value_p:N 196, 4058
 - \quark_if_no_value_p:n 196, 4058
 - \quark_if_recursion_tail_aux:w
 - 4020, 4034, 4052
 - \quark_if_recursion_tail_stop:N
 - 197, 1559, 2485, 4018
 - \quark_if_recursion_tail_stop:n
 - 197, 1528, 2443, 4018
 - \quark_if_recursion_tail_stop:o
 - 197, 4018
 - \quark_if_recursion_tail_stop_do:Nn
 - 198, 4032
 - \quark_if_recursion_tail_stop_do:nn
 - ... 198, 1794, 4032, 4380, 4390, 4400
 - \quark_if_recursion_tail_stop_do:on
 - 198, 4032
 - \quark_new:N .. 196, 2561, 4010, 4011–4017
- R**
- \R 4709, 4816
 - \R_last_box 185, 3815, 3816
 - \radical 153
 - \raise 293
 - \ratio 5148, 5525
 - \read 100

- \readline 375
 - \real 5147, 5525
 - \register_record_name:N 1225, 3978
 - \relax 9–14, 135, 645
 - \relpenalty 196
 - \RequirePackage
 - 657, 1161, 1162, 1281, 1282,
 - 1797, 1798, 1805, 1985, 1986, 2099–
 - 2102, 2231–2233, 2283, 2284, 2361,
 - 2362, 2558, 2559, 2767, 2768, 3141–
 - 3143, 3343, 3344, 3518–3525, 3786,
 - 3787, 3903–3906, 4008, 4130–4132,
 - 4489, 4490, 5006–5011, 5121–5123
 - \reverse_if:N 20, 670, 1096, 1102
 - \right 194
 - \rightthyphenmin 250
 - \rightmarginkern 472
 - \rightskip 252
 - \romannumeral 327
 - \rPCODE 439
- S**
- \S 4709
 - \savingshyphcodes 414
 - \savingsvdiscards 415
 - \scan_align_safe_stop: . 202, 4145, 4149
 - \scan_stop: 32, 699,
 - 737, 738, 740, 786, 787, 799, 803,
 - 804, 812, 814, 816, 819, 823, 934,
 - 1091, 1103, 1168, 1173, 1183, 1188,
 - 1202, 1204, 1206, 1208, 1300, 1482,
 - 1990, 2057, 2061, 2068, 2072–2074,
 - 2293, 2334, 2335, 2518, 3096, 3165,
 - 3211, 3243, 3244, 3266, 3272, 3287,
 - 3334, 3336, 3338, 3845, 3871, 3925,
 - 4155, 4315, 4316, 4322, 4347, 4352,
 - 4363, 4365, 4370, 4371, 4692, 4694,
 - 4696, 4827, 4833, 4839, 4843, 5269,
 - 5329, 5338, 5355, 5364, 5412, 5427
 - \scantokens 373
 - \scriptfont 319
 - \scriptscriptfont 320
 - \scriptscriptstyle 165
 - \scriptspace 205
 - \scriptstyle 164
 - \scrollmode 130
 - \seq_clear:c 98, 2108
 - \seq_clear:N 98, 2108
 - \seq_clear_new:c 2112
 - \seq_clear_new:N 2112
 - \seq_elt:w 101, 2103,
 - 2127, 2133, 2139, 2144, 2164, 2172,
 - 2177, 2178, 2181, 2182, 2197, 2205
 - \seq_elt_end:
 - 101, 2103, 2127, 2133, 2139, 2144,
 - 2164, 2172, 2177, 2181, 2197, 2206
 - \seq_gclear:c 98, 2108
 - \seq_gclear:N 98, 2108
 - \seq_gclear_new:c 2112
 - \seq_gclear_new:N 2112
 - \seq_gconcat:ccc 99, 2191
 - \seq_gconcat:NNN 99, 2191
 - \seq_get:cN 99, 2124, 2224
 - \seq_get:NN 99, 2124, 2223
 - \seq_get_aux:w 101, 2126, 2127
 - \seq_gpop:cN 101, 2217
 - \seq_gpop:NN 101, 2217
 - \seq_gpush:cn 101, 2217
 - \seq_gpush:NC 2217
 - \seq_gpush:Nn 101, 2217
 - \seq_gpush:No 101, 2217
 - \seq_gput_left:Nn
 - 99, 2147, 2217, 3914, 3917, 3976, 3979
 - \seq_gput_right:cc 99, 2147
 - \seq_gput_right:cn 99, 2147
 - \seq_gput_right:co 99, 2147
 - \seq_gput_right:Nc 99, 2147
 - \seq_gput_right:Nn 99, 2147
 - \seq_gput_right:No 99, 2147
 - \seq_gset_eq:cc 99, 2187
 - \seq_gset_eq:cN 99, 2187
 - \seq_gset_eq:Nc 99, 2187
 - \seq_gset_eq:NN 99, 2187
 - \seq_if_empty:cF 100, 2117
 - \seq_if_empty:cTF 100, 2117
 - \seq_if_empty:NF 100, 2117
 - \seq_if_empty:NTF 100, 2117
 - \seq_if_empty_err:N 100, 2121, 2125, 2131
 - \seq_if_empty_p:N 100, 2116
 - \seq_if_in:cnF 100, 2195
 - \seq_if_in:cnTF 100, 2195
 - \seq_if_in:coTF 100, 2195
 - \seq_if_in:cxTF ... 100, 2195, 2247, 2259
 - \seq_if_in:NnF 100, 2195
 - \seq_if_in:NnTF 100, 2195
 - \seq_map:NN 99, 2176
 - \seq_map_break:w 2164
 - \seq_map_inline:cn 100, 2180
 - \seq_map_inline:Nn
 - 100, 2180, 3928, 3984, 3992
 - \seq_map_variable:cNn 99, 2164
 - \seq_map_variable:Nn 2174
 - \seq_map_variable:NNn 99, 2164
 - \seq_map_variable_aux:Nnw 2164, 2168, 2171
 - \seq_map_variable_aux:nw 2164
 - \seq_new:c 98, 2106, 2238
 - \seq_new:N 98, 2106, 3911, 3973, 3974

- \seq_pop:cN 101, [2212](#)
 - \seq_pop:NN 101, [2212](#)
 - \seq_pop_aux:nnNN . 100, [2130](#), 2215, 2221
 - \seq_pop_aux:w 101, [2130](#)
 - \seq_push:cn 101, [2212](#)
 - \seq_push:Nn 101, [2212](#)
 - \seq_push:No 101, [2212](#)
 - \seq_put_aux:Nnn .. 101, [2135](#), 2139, 2144
 - \seq_put_aux:w 101, 2136, 2137
 - \seq_put_left:cn 98, [2138](#), 2214
 - \seq_put_left:Nn ... 98, [2138](#), 2152, 2212
 - \seq_put_left:No 98, [2138](#), 2213
 - \seq_put_left:Nx 98, [2138](#)
 - \seq_put_right:Nn 98, [2138](#), 2158, 3499, 3791
 - \seq_put_right:No 98, [2138](#)
 - \seq_put_right:Nx 98, [2138](#)
 - \seq_set_eq:Nc 2185
 - \seq_set_eq:NN 99, [2185](#)
 - \seq_top:cN 101, [2223](#)
 - \seq_top:NN 101, [2223](#)
 - \setbox 301
 - \setcounter [5535](#)
 - \setlabel 5091, 5097, 5098,
5100, 5101, 5105, 5106, 5108, 5109
 - \setlanguage 59
 - \setlength [5531](#)
 - \setname 5082,
5097, 5098, 5100, 5101, 5105, 5108
 - \sfcode 356
 - \shipout 266
 - \show 109
 - \showbox 111
 - \showboxbreadth 125
 - \showboxdepth 126
 - \showgroups 386
 - \showifs 387
 - \showlists 112
 - \showMemUsage 1157,
1277, 1801, 1981, 2227, 2278, 2355,
2554, 2760, 3137, 3514, 3782, 3899,
4004, 4125, 4485, 5001, 5066, 5572
 - \showthe 110
 - \showtokens 374
 - \skewchar 323
 - \skip 347
 - \skip@ 3213
 - \skip_add:cn [153](#), [3178](#)
 - \skip_add:Nn [153](#), [3178](#), 5200
 - \skip_eval:n
[153](#), 3152, 3179, 3186, 3205, 3208, [3211](#)
 - \skip_gadd:cn [153](#)
 - \skip_gadd:Nn [153](#), [3178](#), 5203
 - \skip_gset:cn [152](#), [3152](#)
 - \skip_gset:Nn [152](#), [3152](#), 5197
 - \skip_gsub:Nn [153](#), [3178](#), 5209
 - \skip_gzero:c [152](#), [3165](#)
 - \skip_gzero:N [152](#), [3165](#)
 - \skip_horizontal:c [153](#), [3203](#)
 - \skip_horizontal:N [153](#), 3203
 - \skip_horizontal:n [153](#), 3203
 - \skip_infinite_glue:nTF [153](#), [3229](#), 3236
 - \skip_new:c [152](#), [3145](#)
 - \skip_new:N [152](#), [3145](#),
3214–3218, 3220, 3222, 5136–5138
 - \skip_new_l:N [152](#), 3145
 - \skip_set:cn [152](#), 3152
 - \skip_set:Nn .. [152](#), [3152](#), 3221, 3223, 5194
 - \skip_split_finite_else_action:nnNN
..... [153](#), [3235](#)
 - \skip_sub:Nn [153](#), [3178](#), 5206
 - \skip_use:c [153](#), 3209
 - \skip_use:N [153](#), 3209
 - \skip_vertical:c [153](#), [3203](#)
 - \skip_vertical:N [153](#), [3203](#)
 - \skip_vertical:n [153](#), [3203](#)
 - \skip_vertital:c 3207
 - \skip_zero:c [152](#), [3165](#)
 - \skip_zero:N [152](#), [3165](#)
 - \skipdef 46
 - \space 4067, 4101
 - \spacefactor 265
 - \spaceskip 260
 - \span 73
 - \special 335, 523
 - \splitbotmark 144
 - \splitbotmarks 370
 - \splitdiscards 417
 - \splitfirstmark 143
 - \splitfirstmarks 369
 - \splitmaxdepth 313
 - \splittopskip 314
 - \startrecording 5072, 5079
 - \stepcounter [5535](#)
 - \str_if_eq_p:nn [798](#)
 - \str_if_eq_p_aux:w [798](#), 809
 - \str_if_eq_var_p:nf [815](#), 4066, 4100
 - \str_if_eq_var_start:nnN [815](#)
 - \str_if_eq_var_stop:w [815](#)
 - \string 328, 2733,
2735, 2737, 2739, 2741, 2743, 2745,
2747, 2749, 2751, 2753, 2755, 2757
- T**
- \T 4709
 - \t 4706
 - \tabskip 74
 - T_EX and L^AT_EX 2_ε commands:
 - \@Roman 138

<code>\@alph</code>	138	<code>\newread</code>	111
<code>\@arabic</code>	138	<code>\newskip</code>	152
<code>\@empty</code>	60	<code>\newtoks</code>	163
<code>\@fnsymbol</code>	138	<code>\newwrite</code>	110
<code>\@for</code>	119	<code>\noexpand</code>	83
<code>\@ifundefined</code>	22	<code>\number</code>	93
<code>\@namedef</code>	30	<code>\numexpr</code>	94, 140
<code>\@roman</code>	138	<code>\openin</code>	112
<code>\@tfor</code>	59	<code>\openout</code>	112
<code>\Alph</code>	138	<code>\or</code>	94
<code>\begingroup</code>	32	<code>\outer</code>	25
<code>\box</code>	184	<code>\protected</code>	25
<code>\catcode</code>	219	<code>\read</code>	112
<code>\closein</code>	111	<code>\relax</code>	32
<code>\closeout</code>	112	<code>\romannumeral</code>	139
<code>\copy</code>	184	<code>\sfcode</code>	220
<code>\csname</code>	24	<code>\showbox</code>	185
<code>\def</code>	30	<code>\the</code>	164
<code>\detokenize</code>	58	<code>\thr@@</code>	93
<code>\dimexpr</code>	155	<code>\toksdef</code>	165
<code>\dp</code>	185	<code>\tw@</code>	93
<code>\edef</code>	30	<code>\uccode</code>	220
<code>\empty</code>	60	<code>\unexpanded</code>	83
<code>\endcsname</code>	24	<code>\unhbox</code>	186
<code>\endgroup</code>	32	<code>\unhcopy</code>	186
<code>\errhelp</code>	173	<code>\unvbox</code>	187
<code>\errorcontextlines</code>	173	<code>\unvcopy</code>	187
<code>\errormessage</code>	172	<code>\uppercase</code>	58
<code>\expandafter</code>	83	<code>\vskip</code>	153
<code>\futurelet</code>	226	<code>\vsplit</code>	187
<code>\gdef</code>	30	<code>\wd</code>	185
<code>\global</code>	25	<code>\write</code>	111
<code>\glueexpr</code>	154	<code>\xdef</code>	30
<code>\hskip</code>	153	<code>\tex_above:D</code>	156
<code>\ht</code>	185	<code>\tex_abovedisplaysshortskip:D</code>	169
<code>\ifcase</code>	94	<code>\tex_abovedisplayskip:D</code>	170
<code>\ifdim</code>	155, 156	<code>\tex_abovewithdelims:D</code>	157
<code>\ifeof</code>	111	<code>\tex_accent:D</code>	207
<code>\ifhbox</code>	183	<code>\tex_adjdemerits:D</code>	244
<code>\ifnum</code>	94, 140	<code>\tex_advance:D</code>	51, 2772, 3179, 3186, 3266, 3272, 3336, 3338
<code>\ifodd</code>	94, 140	<code>\tex_afterassignment:D</code>	61, 4914, 4994, 5255
<code>\ifvbox</code>	183	<code>\tex_aftergroup:D</code>	62, 702
<code>\ifvoid</code>	183	<code>\tex_atop:D</code>	158
<code>\immediate</code>	112	<code>\tex_atopwithdelims:D</code>	159
<code>\jobname</code>	60	<code>\tex_badness:D</code>	306
<code>\lccode</code>	220	<code>\tex_baselineskip:D</code>	234
<code>\let</code>	31	<code>\tex_batchmode:D</code>	127
<code>\long</code>	25	<code>\tex_begingroup:D</code>	65, 557, 637, 700
<code>\lowercase</code>	58	<code>\tex_belowdisplaysshortskip:D</code>	171
<code>\mathcode</code>	220	<code>\tex_belowdisplayskip:D</code>	172
<code>\newbox</code>	183	<code>\tex_binoppenalty:D</code>	195
<code>\newcount</code>	137	<code>\tex_botmark:D</code>	142
<code>\newdimen</code>	154	<code>\tex_box:D</code>	350, 3807, 3834
<code>\newmuskip</code>	156		

- \tex_boxmaxdepth:D 312
- \tex_brokenpenalty:D 269
- \tex_catcode:D
 - 354, 530–532, 534, 535, 539–
 - 541, 543, 544, 549, 550, 553, 554,
 - 558, 611, 654, 3925, 4492, 4496, 4498
- \tex_char:D 208
- \tex_chardef:D
 - 43, 740, 2056, 2074, 2288, 2330
- \tex_cleaders:D 226
- \tex_closein:D 102, 2333
- \tex_closeout:D 97, 112, 2295
- \tex_clubpenalty:D 237
- \tex_copy:D 294, 3836
- \tex_count:D 345
- \tex_countdef:D
 - 44, 737, 2072, 2805, 2806, 4728
- \tex_cr:D 69
- \tex_crcr:D 70
- \tex_csname:D 132, 644, 659, 693
- \tex_day:D 340
- \tex_deadcycles:D 274
- \tex_def:D 39, 528, 529, 537, 538,
 - 548, 552, 556, 561, 563–566, 570,
 - 577, 579–581, 585, 587–589, 595,
 - 599, 604, 615, 616, 623, 634, 636,
 - 640, 645, 651, 656, 657, 696, 698, 707
- \tex_defaulthyphenchar:D 324
- \tex_defaultskewchar:D 325
- \tex_delcode:D 355
- \tex_delimiter:D 149
- \tex_delimiterfactor:D 198
- \tex_delimitershortfall:D 197
- \tex_dimen:D 346
- \tex_dimendef:D 45, 3249, 3250, 4750
- \tex_discretionary:D 209
- \tex_displayindent:D 174
- \tex_displaylimits:D 184
- \tex_displaystyle:D 162
- \tex_displaywidowpenalty:D 173
- \tex_displaywidth:D 175
- \tex_divide:D 52
- \tex_doublehyphendemerits:D 242
- \tex_dp:D 353, 3830
- \tex_dump:D 336
- \tex_edef:D 40, 605, 708
- \tex_else:D 93, 573, 619, 635, 672
- \tex_emergencystretch:D 257
- \tex_end:D 131, 626, 3574, 3579
- \tex_endcsname:D 133, 644, 659, 694
- \tex_endgroup:D 66, 562, 641, 701
- \tex_endinput:D 105, 3632
- \tex_endlinechar:D 147, 533, 542
- \tex_eqno:D 167
- \tex_errhelp:D 113, 3549
- \tex_errmessage:D .. 107, 746, 3528, 3952
- \tex_errorcontextlines:D . 114, 173, 3551
- \tex_errorstopmode:D 128, 1239
- \tex_escapechar:D 146
- \tex_everycr:D 75
- \tex_everydisplay:D 176, 628
- \tex_everyhbox:D 315
- \tex_everyjob:D 344
- \tex_everymath:D 200, 627
- \tex_everypar:D 263
- \tex_everyvbox:D 316
- \tex_exhyphenpenalty:D 239
- \tex_expandafter:D
 - . 63, 572, 574, 608, 643, 647, 658, 687
- \tex_fam:D 55
- \tex_fi:D . 94, 575, 613, 621, 662, 663, 673
- \tex_finalhyphendemerits:D 243
- \tex_firstmark:D 141
- \tex_floatingpenalty:D 288
- \tex_font:D 54
- \tex_fontdimen:D 321
- \tex_fontname:D 145
- \tex_futurelet:D 50, 664, 4887, 4889
- \tex_gdef:D 41, 652, 653, 655, 719
- \tex_global:D 56, 649, 650, 704, 1038
- \tex_globaldefs:D 60
- \tex_halign:D 67
- \tex_hangafter:D 245
- \tex_hangindent:D 246
- \tex_hbadness:D 307
- \tex_hbox:D
 - 302, 3871, 3872, 3877, 3882, 3891, 3892
- \tex_hfil:D 210
- \tex_hfill:D 212
- \tex_hfilneg:D 211
- \tex_hfuzz:D 309
- \tex_hoffset:D 284
- \tex_holdinginserts:D 287
- \tex_hrule:D 223
- \tex_hsize:D 248
- \tex_hskip:D 213, 3203
- \tex_hss:D 214
- \tex_ht:D 352, 3828
- \tex_hyphen:D 37, 630
- \tex_hyphenation:D 338
- \tex_hyphenchar:D 322
- \tex_hyphenpenalty:D 240
- \tex_if:D 76, 617, 675, 676
- \tex_ifcase:D 77, 1993
- \tex_ifcat:D 78, 677
- \tex_ifdim:D 81, 3288
- \tex_ifeof:D 82, 2339
- \tex_iffalse:D 87, 671

<code>\tex_ifhbox:D</code>	83, 3797	<code>\tex_mathinner:D</code>	182
<code>\tex_ifhmode:D</code>	89, 682	<code>\tex_mathop:D</code>	183
<code>\tex_ifinner:D</code>	92, 684	<code>\tex_mathopen:D</code>	187
<code>\tex_ifmmode:D</code>	90, 681	<code>\tex_mathord:D</code>	188
<code>\tex_ifnum:D</code>	79, 611, 1991	<code>\tex_mathpunct:D</code>	189
<code>\tex_ifodd:D</code>	80, 1992	<code>\tex_mathrel:D</code>	190
<code>\tex_iftrue:D</code>	88, 670	<code>\tex_mathsurround:D</code>	201
<code>\tex_ifvbox:D</code>	84, 3798	<code>\tex_maxdeadcycles:D</code>	271
<code>\tex_ifvmode:D</code>	91, 683	<code>\tex_maxdepth:D</code>	272
<code>\tex_ifvoid:D</code>	85, 3799	<code>\tex_meaning:D</code>	331, 691, 695
<code>\tex_ifx:D</code>	86, 571, 632, 658, 678–680	<code>\tex_medmuskip:D</code>	202
<code>\tex_ignorespaces:D</code>	134	<code>\tex_message:D</code>	
<code>\tex_immediate:D</code>	96, 112, 582, 590, 642, 742, 744, 2293, 2295, 2298	<code>\tex_mkern:D</code>	155
<code>\tex_indent:D</code>	230	<code>\tex_month:D</code>	341
<code>\tex_input:D</code>	104, 624, 661, 3570	<code>\tex_moveleft:D</code>	291, 3820
<code>\tex_inputlineno:D</code>	106, 768, 1171, 1177, 1186, 1192, 1212, 3547	<code>\tex_moveright:D</code>	292, 3821
<code>\tex_insert:D</code>	286	<code>\tex_mskip:D</code>	152
<code>\tex_insertpenalties:D</code>	289	<code>\tex_multiply:D</code>	53
<code>\tex_interlinepenalty:D</code>	268	<code>\tex_muskip:D</code>	349
<code>\tex_italiccor:D</code>	36, 629	<code>\tex_muskipdef:D</code>	47, 3330, 3331
<code>\tex_jobname:D</code>	343	<code>\tex_newlinechar:D</code>	103, 1179, 2309
<code>\tex_kern:D</code>	221	<code>\tex_noalign:D</code>	71
<code>\tex_language:D</code>	138	<code>\tex_noboundary:D</code>	206
<code>\tex_lastbox:D</code>	295, 3815	<code>\tex_noexpand:D</code>	64, 688
<code>\tex_lastkern:D</code>	228	<code>\tex_noindent:D</code>	232
<code>\tex_lastpenalty:D</code>	334	<code>\tex_nolimits:D</code>	186
<code>\tex_lastskip:D</code>	229	<code>\tex_nonscript:D</code>	166
<code>\tex_lccode:D</code>	357, 4514, 4518, 4520	<code>\tex_nonstopmode:D</code>	129
<code>\tex_leaders:D</code>	225	<code>\tex_nulldelimiterspace:D</code>	199
<code>\tex_left:D</code>	193	<code>\tex_nullfont:D</code>	317
<code>\tex_lefthyphenmin:D</code>	249	<code>\tex_number:D</code>	
<code>\tex_leftskip:D</code>	251	<code>\tex_omit:D</code>	326, 816, 1988, 2011, 2771, 5425, 5426
<code>\tex_leqno:D</code>	168	<code>\tex_omit:D</code>	72
<code>\tex_let:D</code>	15, 19, 24, 32, 38, 546, 547, 603, 624–631, 648–650, 669	<code>\tex_openin:D</code>	98, 112, 2335
<code>\tex_limits:D</code>	185	<code>\tex_openout:D</code>	99, 112, 2293
<code>\tex_linepenalty:D</code>	241	<code>\tex_or:D</code>	95, 1994
<code>\tex_lineskip:D</code>	235	<code>\tex_outer:D</code>	58
<code>\tex_lineskiplimit:D</code>	236	<code>\tex_output:D</code>	273
<code>\tex_long:D</code>	57, 705	<code>\tex_outputpenalty:D</code>	283
<code>\tex_looseness:D</code>	253	<code>\tex_over:D</code>	160
<code>\tex_lower:D</code>	290, 3823	<code>\tex_overfullrule:D</code>	311
<code>\tex_lowercase:D</code>	329, 1517	<code>\tex_overline:D</code>	191
<code>\tex_mag:D</code>	137	<code>\tex_overwithdelims:D</code>	161
<code>\tex_mark:D</code>	139	<code>\tex_pagedepth:D</code>	275
<code>\tex_mathaccent:D</code>	150	<code>\tex_pagefilllstretch:D</code>	279
<code>\tex_mathbin:D</code>	180	<code>\tex_pagefillstretch:D</code>	278
<code>\tex_mathchar:D</code>	151	<code>\tex_pagefilstretch:D</code>	277
<code>\tex_mathchardef:D</code> ..	48, 2060, 3792, 3793	<code>\tex_pagegoal:D</code>	281
<code>\tex_mathchoice:D</code>	148	<code>\tex_pageshrink:D</code>	280
<code>\tex_mathclose:D</code>	181	<code>\tex_pagestretch:D</code>	276
<code>\tex_mathcode:D</code> 359, 4501, 4505, 4509, 4511		<code>\tex_pagetotal:D</code>	282
		<code>\tex_par:D</code>	231, 631
		<code>\tex_parfillskip:D</code>	262

<code>\tex_parindent:D</code>	255	<code>\tex_string:D</code>	328, 692, 3690
<code>\tex_parshape:D</code>	247	<code>\tex_tabskip:D</code>	74
<code>\tex_parskip:D</code>	254	<code>\tex_textfont:D</code>	318
<code>\tex_patterns:D</code>	337	<code>\tex_textstyle:D</code>	163
<code>\tex_pausing:D</code>	124	<code>\tex_the:D</code>	136, 611, 703, 768, 923, 925, 934, 1171, 1177, 1186, 1192, 1212, 2855, 3209, 3277
<code>\tex_penalty:D</code>	332	<code>\tex_thickmuskip:D</code>	204
<code>\tex_postdisplaypenalty:D</code>	179	<code>\tex_thinmuskip:D</code>	203
<code>\tex_predisdisplaypenalty:D</code>	178	<code>\tex_time:D</code>	339
<code>\tex_predisplaysize:D</code>	177	<code>\tex_toks:D</code>	348
<code>\tex_pretolerance:D</code>	258	<code>\tex_toksdef:D</code> 49, 165, 3348, 3349, 3498, 4761
<code>\tex_prevdepth:D</code>	305	<code>\tex_tolerance:D</code>	259
<code>\tex_prevgraf:D</code>	264	<code>\tex_topmark:D</code>	140
<code>\tex_radical:D</code>	153	<code>\tex_topskip:D</code>	270
<code>\tex_raise:D</code>	293, 3822	<code>\tex_tracingcommands:D</code>	115
<code>\tex_read:D</code>	100, 112, 2343, 2346	<code>\tex_tracinglostchars:D</code>	116
<code>\tex_relax:D</code>	135, 530–535, 539–544, 549, 550, 553, 554, 558, 567, 568, 570, 577, 578, 585, 586, 608, 611, 615, 659, 661, 699	<code>\tex_tracingmacros:D</code>	117
<code>\tex_relpenalty:D</code>	196	<code>\tex_tracingonline:D</code>	118
<code>\tex_right:D</code>	194	<code>\tex_tracingoutput:D</code>	119
<code>\tex_righthyphenmin:D</code>	250	<code>\tex_tracingpages:D</code>	120
<code>\tex_rightskip:D</code>	252	<code>\tex_tracingparagraphs:D</code>	121
<code>\tex_romannumeral:D</code>	327, 2770, 3960, 3967	<code>\tex_tracingrestores:D</code>	122
<code>\tex_scriptfont:D</code>	319	<code>\tex_tracingstats:D</code>	123
<code>\tex_scriptscriptfont:D</code>	320	<code>\tex_uccode:D</code> 358, 1168, 1173, 1183, 1188, 1202, 1204, 1206, 4523, 4527, 4529, 4536
<code>\tex_scriptscriptstyle:D</code>	165	<code>\tex_uchyph:D</code>	256
<code>\tex_scriptspace:D</code>	205	<code>\tex_underline:D</code>	192, 625
<code>\tex_scriptstyle:D</code>	164	<code>\tex_unhbox:D</code>	297, 3895
<code>\tex_scrollmode:D</code>	130	<code>\tex_unhcopy:D</code>	298, 3893
<code>\tex_setbox:D</code> ..	301, 3807, 3816, 3846, 3851, 3857, 3865, 3872, 3877, 3882	<code>\tex_unkern:D</code>	222
<code>\tex_setlanguage:D</code>	59	<code>\tex_unpenalty:D</code>	333
<code>\tex_sfcode:D</code>	356, 4532, 4538	<code>\tex_unskip:D</code>	220
<code>\tex_shipout:D</code>	266	<code>\tex_unvbox:D</code>	299, 3869
<code>\tex_show:D</code>	109, 697	<code>\tex_unvcopy:D</code>	300, 3867
<code>\tex_showbox:D</code>	111, 3838	<code>\tex_uppercase:D</code>	330, 1518
<code>\tex_showboxbreadth:D</code>	125	<code>\tex_vadjust:D</code>	233
<code>\tex_showboxdepth:D</code>	126	<code>\tex_valign:D</code>	68
<code>\tex_showlists:D</code>	112	<code>\tex_vbadness:D</code>	308
<code>\tex_showthe:D</code> 110, 4498, 4511, 4520, 4529, 4538	<code>\tex_vbox:D</code> 303, 3845, 3846, 3851, 3857, 3862, 3863
<code>\tex_skewchar:D</code>	323	<code>\tex_vcenter:D</code>	154
<code>\tex_skip:D</code>	347	<code>\tex_vfil:D</code>	215
<code>\tex_skipdef:D</code> ..	46, 3147, 3148, 3213, 4739	<code>\tex_vfill:D</code>	217
<code>\tex_space:D</code>	35	<code>\tex_vfilneg:D</code>	216
<code>\tex_spacefactor:D</code>	265	<code>\tex_vfuzz:D</code>	310
<code>\tex_spaceskip:D</code>	260	<code>\tex_voffset:D</code>	285
<code>\tex_span:D</code>	73	<code>\tex_vrule:D</code>	224
<code>\tex_special:D</code>	335	<code>\tex_vsize:D</code>	267
<code>\tex_splitbotmark:D</code>	144	<code>\tex_vskip:D</code>	218, 3206
<code>\tex_splitfirstmark:D</code>	143	<code>\tex_vsplit:D</code>	296, 3865
<code>\tex_splitmaxdepth:D</code>	313	<code>\tex_vss:D</code>	219
<code>\tex_splittopskip:D</code>	314		

- \tex_vtop:D 304
- \tex_wd:D 351, 3832
- \tex_widowpenalty:D 238
- \tex_write:D . 101, 582, 590, 642, 690, 2314
- \tex_xdef:D 42, 643, 720
- \tex_xleaders:D 227
- \tex_xspaceskip:D 261
- \tex_year:D 342
- \text_put_four_sp:
 - 1170, 1176, 1211, 3529, 3625, 3627,
 - 3640, 3645, 3650, 3654, 3659, 3664
- \text_put_sp:
 - 3529, 3547, 3625, 3627, 3628, 3644,
 - 3649, 3658, 3663, 3672–3674, 3677,
 - 3680, 3681, 3683, 3687, 3694, 3728,
 - 3730, 3741, 3748, 3749, 4068, 4102
- \textasteriskcentered 2944, 2950
- \textbardbl 2949
- \textdagger 2945, 2951
- \textdaggerdbl 2946, 2952
- \textdir 527
- \textfont 318
- \textparagraph 2948
- \textsection 2947
- \textstyle 163
- \TeXETstate 418
- \the 136
- \the_internal:D
 - ... 703, 3364, 3536, 3540, 3547, 3997
- \thepage 5085
- \thickmuskip 204
- \thinmuskip 203
- \time 339
- \tlist_compare:nn 1576
- \tlist_compare:no 1585
- \tlist_compare:nx 1579
- \tlist_compare:on 1588
- \tlist_compare:oo 1591
- \tlist_compare:ox 1597
- \tlist_compare:xn 1582
- \tlist_compare:xo 1594
- \tlist_compare:xx .. 1575, 1577, 1580,
 - 1583, 1586, 1589, 1592, 1595, 1598
- \tlist_elt_count:n 59, 1564
- \tlist_elt_count:o 59, 1564
- \tlist_head:n 62, 1743
- \tlist_head:w
 - 62, 1743, 1753, 1763, 1772, 1782
- \tlist_head_i:n 1743
- \tlist_head_iii:f 62, 1743
- \tlist_head_iii:n 62, 1743
- \tlist_head_iii:w 62, 1743
- \tlist_if_blank:nF 58, 1512, 2439
- \tlist_if_blank:nT 58, 1512
- \tlist_if_blank:nTF 58, 1512
- \tlist_if_blank:oF 58, 1512
- \tlist_if_blank:oT 58, 1512
- \tlist_if_blank:oTF 58, 1512
- \tlist_if_blank_p:n . 58, 1507, 1513, 1514
- \tlist_if_blank_p:o 58, 1512
- \tlist_if_blank_p_aux:w 60, 1507
- \tlist_if_empty:nF
 - 58, 1502, 2405, 2463, 2475, 2604
- \tlist_if_empty:nT 58, 1502, 2638
- \tlist_if_empty:nTF 58, 1502
- \tlist_if_empty:oF 58, 1502
- \tlist_if_empty:oT 58, 1502
- \tlist_if_empty:oTF 58, 1502
- \tlist_if_empty_p:n .. 58, 1495, 1502,
 - 1504, 2678, 4689, 4718, 4725, 4736,
 - 4747, 4758, 4769, 4776, 4783, 4791
- \tlist_if_empty_p:o 58, 1502, 3464
- \tlist_if_eq:nnF 58, 1575
- \tlist_if_eq:nnT 58, 1575
- \tlist_if_eq:nnTF 58, 1575
- \tlist_if_eq:noF 58
- \tlist_if_eq:noT 58
- \tlist_if_eq:noTF 58
- \tlist_if_eq:xxF 2662, 3481
- \tlist_if_eq:xxT 2659, 3478
- \tlist_if_eq:xxTF 2656, 3475, 4401
- \tlist_if_eq_p:xx 3493
- \tlist_if_head_eq_catcode:nNF ... 1752
- \tlist_if_head_eq_catcode:nNT ... 1752
- \tlist_if_head_eq_catcode:nNTF 62, 1752
- \tlist_if_head_eq_catcode_p:nN
 - 62, 1780, 1789
- \tlist_if_head_eq_catcode_p:nNTF . 1752
- \tlist_if_head_eq_charcode:nNF
 - 3044, 3051
- \tlist_if_head_eq_charcode:nNF .. 1752
- \tlist_if_head_eq_charcode:nNT .. 1752
- \tlist_if_head_eq_charcode:nNTF 62, 1752
- \tlist_if_head_eq_charcode_p:nF
 - 1771, 1779
- \tlist_if_head_eq_charcode_p:nN
 - 62, 1761, 1770
- \tlist_if_head_eq_charcode_p:nNTF 1752
- \tlist_if_head_eq_meaning:nNF ... 1752
- \tlist_if_head_eq_meaning:nNT ... 1752
- \tlist_if_head_eq_meaning:nNTF 62, 1752
- \tlist_if_head_eq_meaning_p:nN 62, 1752
- \tlist_if_in:nnTF 61, 1661
- \tlist_if_in:onTF 61, 1661
- \tlist_map_break:w 59, 1562
- \tlist_map_function:nN 58, 1520, 1566, 5372
- \tlist_map_function_aux:NN 1520

- \tlist_map_function_aux:Nn 59, 1521, 1524, 1527, 1529, 1536, 1545
- \tlist_map_inline:nn 59, 1532, 4709
- \tlist_map_inline_aux:n 1532
- \tlist_map_inline_aux:Nn 59
- \tlist_map_variable:nNn 59, 1552
- \tlist_map_variable_aux:NnN 1557
- \tlist_map_variable_aux:Nnn 59, 1553, 1557, 1560
- \tlist_reverse:n 59, 1790
- \tlist_reverse_aux:nN 1790
- \tlist_tail:f 1743
- \tlist_tail:n 62, 1743
- \tlist_tail:w 62, 1743
- \tlist_to_lowercase:n 58, 1517, 4711, 4818
- \tlist_to_str:n 58, 1496, 1508, 1519, 4069, 4103
- \tlist_to_uppercase:n 58, 1517
- \tlp_clear:c 55, 1359, 2109, 2368
- \tlp_clear:N 55, 1359, 1367, 1373, 1710, 2108, 2122, 2367, 2383
- \tlp_clear_new:c 56, 1363, 2113
- \tlp_clear_new:N 56, 1363, 2112, 3964, 3971
- \tlp_elt_count:N 59, 1570
- \tlp_elt_count:n 1564
- \tlp_gclear:c 55, 1359, 2111, 2370
- \tlp_gclear:N 55, 1359, 1379, 1384, 2110, 2369
- \tlp_gclear_new:c 56, 1375, 2115
- \tlp_gclear_new:N 56, 1375, 2114
- \tlp_gput_left:cn 1417
- \tlp_gput_left:co 1418
- \tlp_gput_left:cx 1419
- \tlp_gput_left:Nn 56, 1386, 2414
- \tlp_gput_left:No 56, 1386
- \tlp_gput_left:Nx 56, 1386
- \tlp_gput_right:cn 56, 1420
- \tlp_gput_right:co 56, 1420
- \tlp_gput_right:Nn 56, 1420, 2423
- \tlp_gput_right:No 56, 1420
- \tlp_gput_right:Nx 56, 1420
- \tlp_gremove_all_in:cn 61, 1742
- \tlp_gremove_all_in:Nn 61, 1735
- \tlp_gremove_in:cn 61, 1731
- \tlp_gremove_in:Nn 61, 1731
- \tlp_greplace_all_in:cnn 61, 1709
- \tlp_greplace_all_in:Nnn 61, 1728, 1730, 1739
- \tlp_greplace_in:cnn 61, 1690
- \tlp_greplace_in:Nnn 61, 1690, 1732
- \tlp_gset:cn 55, 1308
- \tlp_gset:cx 55, 1308
- \tlp_gset:Nc 55, 1451
- \tlp_gset:Nd 55, 1308
- \tlp_gset:Nn 55, 1308, 2221, 2414, 2423, 2548, 3589, 3950
- \tlp_gset:No 55, 1308
- \tlp_gset:Nx 55, 1308, 1402, 1407, 1411, 1427, 1437, 1445, 2192, 2498, 2580
- \tlp_gset_eq:cc 56, 1339
- \tlp_gset_eq:cN 56, 1339
- \tlp_gset_eq:Nc 56, 1339
- \tlp_gset_eq:NN 56, 1339, 1361, 1707, 1729, 2020, 3604
- \tlp_if_empty:cF 57, 1461
- \tlp_if_empty:cT 57, 1461
- \tlp_if_empty:cTF 57, 1461
- \tlp_if_empty:NF ... 57, 1461, 2119, 3585
- \tlp_if_empty:NT 57, 1461
- \tlp_if_empty:NTF 57, 1461, 2117
- \tlp_if_empty_p:c 57, 1458
- \tlp_if_empty_p:N .. 57, 1458, 2116, 2376
- \tlp_if_empty_p:NN 1468–1470
- \tlp_if_eq:ccF 57, 1471
- \tlp_if_eq:ccT 57, 1471
- \tlp_if_eq:ccTF 57, 1471
- \tlp_if_eq:cNF 57, 1471
- \tlp_if_eq:cNT 57, 1471
- \tlp_if_eq:cNTF 57, 1471
- \tlp_if_eq:NcF 57, 1471
- \tlp_if_eq:NcT 57, 1471
- \tlp_if_eq:NcTF 57, 1471
- \tlp_if_eq:NNF 57, 1471, 3592
- \tlp_if_eq:NNT 57, 1471
- \tlp_if_eq:NNTF 57, 1471, 2386
- \tlp_if_eq_p:cc 57, 1466
- \tlp_if_eq_p:cN 57, 1466
- \tlp_if_eq_p:Nc 57, 1466
- \tlp_if_eq_p:NN 57, 1466
- \tlp_if_in:cnF 61, 1661
- \tlp_if_in:cnT 61, 1661
- \tlp_if_in:cnTF 61, 1661
- \tlp_if_in:NnF 61, 1661
- \tlp_if_in:NnT 61, 1661
- \tlp_if_in:NnTF 61, 1661
- \tlp_map_break:w 59, 1562
- \tlp_map_function:cN 58, 1520
- \tlp_map_function:NN 58, 1520, 1572
- \tlp_map_inline:cN 1550
- \tlp_map_inline:cn 59, 1532
- \tlp_map_inline:NN 1550
- \tlp_map_inline:Nn 59, 1532
- \tlp_map_variable:cNn 59, 1552
- \tlp_map_variable:Nnn 59, 1552
- \tlp_new:c 55, 1284
- \tlp_new:cn 55, 1284, 5023
- \tlp_new:N 55, 1284

- \tlp_new:Nn 55, 1284, 1369, 1381, 1481–1490, 1551, 1689, 1807, 2007, 2106, 2365, 3527, 3583, 3590, 3910, 4010, 4891, 4892, 4894, 4912, 4913, 5126
- \tlp_new:Nx 55, 1284, 2325–2327
- \tlp_put_left:cn 1414
- \tlp_put_left:co 1415
- \tlp_put_left:cx 1416
- \tlp_put_left:Nn 56, 1386, 2408
- \tlp_put_left:No 56, 1386
- \tlp_put_left:Nx 56, 1386
- \tlp_put_right:cc 56, 1420
- \tlp_put_right:Nn 56, 1420, 2417
- \tlp_put_right:No 56, 1420
- \tlp_put_right:Nx 56, 1420, 1696, 1714, 1717
- \tlp_remove_all_in:cn 61, 1741
- \tlp_remove_all_in:Nn 61, 1735
- \tlp_remove_in:cn 61, 1731
- \tlp_remove_in:Nn 61, 1731
- \tlp_replace_all_in:cnn 61, 1727
- \tlp_replace_all_in:Nnn . 61, 1709, 1736
- \tlp_replace_all_in_aux:NNnn 1709
- \tlp_replace_in:cnn 61, 1690
- \tlp_replace_in:Nnn 61, 1690, 1731
- \tlp_replace_in_aux:NNnn 1690
- \tlp_set:cn 1308
- \tlp_set:co 1308
- \tlp_set:cx 1308
- \tlp_set:Nc 55, 1451, 3960, 3967
- \tlp_set:Nd 55, 1308
- \tlp_set:Nf 55, 1308
- \tlp_set:Nn 55, 1308, 1387, 1392, 1401, 1406, 1421, 1426, 1431, 1436, 1558, 2128, 2137, 2165, 2215, 2221, 2320, 2390, 2408, 2417, 2543, 2548, 4897, 4905, 5082
- \tlp_set:No 55, 1308, 1457, 2011
- \tlp_set:Nx 55, 1308, 1388, 1393, 1397, 1422, 1432, 1441, 1616, 1617, 1621, 1622, 1626, 1627, 1631, 1632, 1636, 1637, 1641, 1642, 1646, 1647, 1651, 1652, 1656, 1657, 1694, 1832, 2171, 2321, 2497, 2574, 2584, 4898, 4899, 4906, 4908, 5149
- \tlp_set_eq:cc 56, 1339
- \tlp_set_eq:cN 56, 1339
- \tlp_set_eq:Nc 56, 1339
- \tlp_set_eq:NN 56, 1339, 1359, 1705, 1726, 2016, 4907
- \tlp_to_str:c 57, 1491
- \tlp_to_str:N 57, 1491, 2322, 3933
- \tlp_to_str_aux:w 60, 1491
- \tlp_use:c 55, 1299
- \tlp_use:N 55, 1299
- \tmp:w 1141, 1600, 1612, 1613, 1662, 1665, 1669, 1672, 1676, 1679, 1683, 1686, 1691, 1695, 1698, 1702, 1711, 1718, 1721, 2196, 2205, 2527, 2534, 2569, 2570, 2586, 2587, 2606, 2607, 2625–2628, 5055, 5058
- \token_get_arg_spec:N 225, 4811
- \token_get_prefix_arg_replacement_aux:w 4811
- \token_get_prefix_spec:N 225, 4811
- \token_get_replacement_spec:N 225, 4811
- \token_if_active_char:Nf 222, 4649
- \token_if_active_char:NT 222, 4649
- \token_if_active_char:NTF 222, 4649
- \token_if_active_char_p:N 222, 4649
- \token_if_active_p:N 4850
- \token_if_aligment_tab:Nf 221
- \token_if_aligment_tab:NT 221
- \token_if_aligment_tab:NTF 221
- \token_if_aligment_tab_p:N 221
- \token_if_alignment_tab:Nf 4586
- \token_if_alignment_tab:NT 4586
- \token_if_alignment_tab:NTF 4586
- \token_if_alignment_tab_p:N 4586
- \token_if_charcode_eq_p:NN 4676
- \token_if_chardef:Nf 224, 4794
- \token_if_chardef:NT 224, 4794
- \token_if_chardef:NTF 224, 4794
- \token_if_chardef_p:N 224, 4704, 4794, 4862
- \token_if_chardef_p_aux:w 4704
- \token_if_cs:Nf 223, 4692
- \token_if_cs:NT 223, 4692
- \token_if_cs:NTF 223, 4692
- \token_if_cs_p:N 223, 4692, 4843
- \token_if_dim_register:Nf 224, 4803
- \token_if_dim_register:NT 224, 4803
- \token_if_dim_register:NTF ... 224, 4803
- \token_if_dim_register_p:N 224, 4704, 4804, 4870
- \token_if_dim_register_p_aux:w .. 4704
- \token_if_eq_catcode:NNF 223, 4667
- \token_if_eq_catcode:NNT 223, 4667
- \token_if_eq_catcode:NNTF 223, 4667
- \token_if_eq_catcode_p:NN 223, 4667, 4692, 4694, 4932
- \token_if_eq_charcode:NNF 4676
- \token_if_eq_charcode:NNT 4676
- \token_if_eq_charcode:NNTF 4676
- \token_if_eq_charcode_p:NN ... 223, 4676
- \token_if_eq_meaning:NNF 223, 4658
- \token_if_eq_meaning:NNT 223, 4658
- \token_if_eq_meaning:NNTF 223, 4658, 4993
- \token_if_eq_meaning_p:NN 223, 4658, 4933
- \token_if_expandable:Nf 223, 4695

- \token_if_expandable:NT 223, [4695](#)
- \token_if_expandable:NTF 223, [4695](#)
- \token_if_expandable_p:N 223, [4695](#)
- \token_if_group_begin:Nf 221, [4559](#)
- \token_if_group_begin:NT 221, [4559](#)
- \token_if_group_begin:NTF 221, [4559](#)
- \token_if_group_begin_p:N 221, [4559](#)
- \token_if_group_end:Nf 221, [4568](#)
- \token_if_group_end:NT 221, [4568](#)
- \token_if_group_end:NTF 221, [4568](#)
- \token_if_group_end_p:N 221, [4568](#)
- \token_if_int_register:Nf 224, [4807](#)
- \token_if_int_register:NT 224, [4807](#)
- \token_if_int_register:NTF 224, [4807](#)
- \token_if_int_register_p:N 224, [4704](#), [4808](#), [4866](#)
- \token_if_int_register_p_aux:w .. 4704
- \token_if_letter:Nf 222, [4631](#)
- \token_if_letter:NT 222, [4631](#)
- \token_if_letter:NTF 222, [4631](#)
- \token_if_letter_p:N 222, [4631](#)
- \token_if_long_macro:Nf 224, [4797](#)
- \token_if_long_macro:NT 224, [4797](#)
- \token_if_long_macro:NTF 224, [4797](#)
- \token_if_long_macro_p:N 224, [4704](#), [4798](#)
- \token_if_long_macro_p_aux:w 4704
- \token_if_macro:Nf 223, [4685](#)
- \token_if_macro:NT 223, [4685](#)
- \token_if_macro:NTF 223, [4685](#), [4824](#), [4830](#), [4836](#)
- \token_if_macro_p:N 223, [4685](#), [4844](#), [4851](#)
- \token_if_macro_p_aux:w 4685
- \token_if_math_shift:Nf 221, [4577](#)
- \token_if_math_shift:NT 221, [4577](#)
- \token_if_math_shift:NTF 221, [4577](#)
- \token_if_math_shift_p:N 221, [4577](#)
- \token_if_math_subscript:Nf .. 222, [4613](#)
- \token_if_math_subscript:NT .. 222, [4613](#)
- \token_if_math_subscript:NTF .. 222, [4613](#)
- \token_if_math_subscript_p:N .. 222, [4613](#)
- \token_if_math_superscript:Nf 222, [4604](#)
- \token_if_math_superscript:NT 222, [4604](#)
- \token_if_math_superscript:NTF 222, [4604](#)
- \token_if_math_superscript_p:N 222, [4604](#)
- \token_if_mathchardef:Nf 224, [4795](#)
- \token_if_mathchardef:NT 224, [4795](#)
- \token_if_mathchardef:NTF 224, [4795](#)
- \token_if_mathchardef_p:N 224, [4704](#), [4796](#), [4864](#)
- \token_if_mathchardef_p_aux:w ... 4704
- \token_if_other_char:Nf 222, [4640](#)
- \token_if_other_char:NT 222, [4640](#)
- \token_if_other_char:NTF 222, [4640](#)
- \token_if_other_char_p:N 222, [4640](#)
- \token_if_parameter:Nf 222, [4595](#)
- \token_if_parameter:NT 222, [4595](#)
- \token_if_parameter:NTF 222, [4595](#)
- \token_if_parameter_p:N 222, [4595](#)
- \token_if_primitive:Nf 225, [4842](#)
- \token_if_primitive:NT 225, [4842](#)
- \token_if_primitive:NTF 225, [4842](#)
- \token_if_primitive_p:N 225, [4842](#)
- \token_if_primitive_p_aux:N [4842](#)
- \token_if_protected_long_macro:Nf .. 224, [4801](#)
- \token_if_protected_long_macro:NT .. 224, [4801](#)
- \token_if_protected_long_macro:NTF .. 224, [4801](#)
- \token_if_protected_long_macro_p:N .. 224, [4704](#), [4802](#)
- \token_if_protected_long_macro_p_aux:w 4704
- \token_if_protected_macro:Nf .. 224, [4799](#)
- \token_if_protected_macro:NT .. 224, [4799](#)
- \token_if_protected_macro:NTF 224, [4799](#)
- \token_if_protected_macro_p:N 224, [4704](#), [4800](#)
- \token_if_protected_macro_p_aux:w 4704
- \token_if_skip_register:Nf [4805](#)
- \token_if_skip_register:NT ... 225, [4805](#)
- \token_if_skip_register:NTF .. 225, [4805](#)
- \token_if_skip_register_p:N 225, [4704](#), [4806](#), [4868](#)
- \token_if_skip_register_p_aux:w .. 4704
- \token_if_space:Nf 222, [4622](#)
- \token_if_space:NT 222, [4622](#)
- \token_if_space:NTF 222, [4622](#)
- \token_if_space_p:N 222, [4622](#)
- \token_if_toks_register:Nf ... 225, [4809](#)
- \token_if_toks_register:NT ... 225, [4809](#)
- \token_if_toks_register:NTF .. 225, [4809](#)
- \token_if_toks_register_p:N 225, [4704](#), [4810](#), [4872](#)
- \token_if_toks_register_p_aux:w .. 4704
- \token_ifskip_register:Nf 225
- \token_new:Nn .. 221, [4541](#), [4546](#), [4548](#)–[4550](#), [4552](#)–[4555](#), [4557](#), [4884](#)–[4886](#)
- \token_to_meaning:N 32, [690](#), [761](#), [1492](#), [3599](#), [3989](#), [4686](#), [4715](#), [4722](#), [4732](#), [4743](#), [4754](#), [4765](#), [4773](#), [4780](#), [4787](#), [4826](#), [4832](#), [4838](#)
- \token_to_string:N 32, [690](#), [759](#), [766](#), [767](#), [779](#), [810](#), [814](#), [933](#), [1143](#), [1165](#), [1181](#), [1200](#), [1301](#), [2123](#), [2274](#), [2384](#), [2519](#), [3599](#), [3600](#), [3694](#), [3713](#), [3728](#), [3729](#), [3747](#),

- 3748, 3764, 3770, 3925, 3926, 3988,
3996, 4067, 4068, 4101, 4102, 5483
- \toks 348
- \toks_clear:N . 164, 2564, 3353, 3368, 3533
- \toks_gclear:N 164, 2566, 3353
- \toks_gput_left:Nn 164, 3378
- \toks_gput_left:Nx 3378
- \toks_gput_right:Nn 164, 2599, 3395
- \toks_gput_right:No 164, 3395
- \toks_gput_right:Nx 164, 3395
- \toks_gset:cn 3436, 5051
- \toks_gset:co 3436
- \toks_gset:cx 3436
- \toks_gset:Nn 163, 2582, 2623, 3436
- \toks_gset:No 163, 3436
- \toks_gset:Nx 163, 3436, 3623
- \toks_gset_eq:cc 2646, 3445
- \toks_gset_eq:cN 2645, 3445
- \toks_gset_eq:Nc 2644, 3445
- \toks_gset_eq:NN 164, 2643, 3445
- \toks_if_empty:cF 165, 2654, 3463
- \toks_if_empty:cT 165, 3463
- \toks_if_empty:cTF . 165, 2652, 2653, 3463
- \toks_if_empty:Nf . 165, 2493, 2651, 3463
- \toks_if_empty:NT 165, 2650, 3463
- \toks_if_empty:NTF 165, 2649, 3463
- \toks_if_empty_p:c 165, 2648, 3463
- \toks_if_empty_p:N 165, 2647, 3463
- \toks_if_eq:ccF 3474
- \toks_if_eq:ccT 3474
- \toks_if_eq:ccTF 3474
- \toks_if_eq:cNF 3474
- \toks_if_eq:cNT 3474
- \toks_if_eq:cNTF 3474
- \toks_if_eq:NcF 3474
- \toks_if_eq:NcT 3474
- \toks_if_eq:NcTF 3474
- \toks_if_eq:NNF 3474
- \toks_if_eq:NNTF 3474
- \toks_if_eq_p:cc 3474
- \toks_if_eq_p:cN 3474
- \toks_if_eq_p:Nc 3474
- \toks_if_eq_p:NN 3474
- \toks_new:c 163, 3346
- \toks_new:N ... 163, 2562, 3346, 3500–3505
- \toks_new_l:N 163, 3346
- \toks_put_left:Nn 164, 3378, 3537
- \toks_put_left:No 3378, 3536
- \toks_put_left_aux:w 165, 3378
- \toks_put_right:Nd 3395
- \toks_put_right:Nf 3395
- \toks_put_right:Nn 164, 2592, 3395
- \toks_put_right:No 164, 3395
- \toks_put_right:Nx 164, 3395
- \toks_remove_extra_brace_group:N . 3506
- \toks_remove_extra_brace_group_aux:NNw
..... 3506
- \toks_set:cf 163, 3419
- \toks_set:cn 163, 3419
- \toks_set:co 163, 3419
- \toks_set:cx 163, 3419
- \toks_set:Nd 163, 3419
- \toks_set:Nf 163, 3419
- \toks_set:Nn 163, 2621, 3419, 3508
- \toks_set:No 163, 2489, 2490, 3419
- \toks_set:Nx 163, 3419, 5024
- \toks_set_eq:cc 2642, 3445
- \toks_set_eq:cN 2641, 3445
- \toks_set_eq:Nc 2640, 3445
- \toks_set_eq:NN 2639, 3445
- \toks_use:c 3364
- \toks_use:N 164, 2492, 2494, 2570, 2656,
2659, 2662, 2675, 2689, 2704, 3364,
3371, 3380, 3395, 3408, 3415, 3464,
3475, 3478, 3481, 3493, 3509, 5028
- \toks_use_clear:N 164, 3366
- \toks_use_gclear:N 164, 3366
- \toksdef 49
- \tolerance 259
- \toodeep 3722
- \topmark 140
- \topmarks 366
- \topskip 270
- \totalheightof 5525
- \traceoff 1259
- \tracelon 1259
- \tracingall 1214
- \tracingassigns 376
- \tracingcommands 115
- \tracinggroups 383
- \tracingifs 379
- \tracinglostchars 116
- \tracingmacros 117
- \tracingnesting 378
- \tracingoff 1246
- \tracingonline 118
- \tracingoutput 119
- \tracingpages 120
- \tracingparagraphs 121
- \tracingrestores 122
- \tracingscantokens 377
- \tracingstats 123
- \typeout 2732,
2734, 2736, 2738, 2740, 2742, 2744,
2746, 2748, 2750, 2752, 2754, 2756

U

`\U` 4709
`\uccode` 358
`\uchyph` 256
`\underline` 192
`\unexpanded` 371
`\unhbox` 297
`\unhcopy` 298
`\unkern` 222
`\unless` 362
`\unpenalty` 333
`\unskip` 220
`\unvbox` 299
`\unvcopy` 300
`\uppercase` 330
`\use:c` 1109
`\use:cc` 1109
`\use_arg_i:n` .. 23, 751, 756, 934, 1050,
 1052, 1059, 1061, 1108, 1702, 1721,
 2028, 2341, 3306, 4380, 4390, 4400
`\use_arg_i:nn` 23, 1046, 1048,
 1055, 1057, 1113, 2201, 2337, 2531,
 3291, 3390, 4089, 4382, 4392, 4402
`\use_arg_i:nnn` 24, 1115, 4826
`\use_arg_i:nnnn` 24, 1115, 4317
`\use_arg_i_after_else:nw` 24, 1127
`\use_arg_i_after_fi:nw` .. 24, 1127, 3105
`\use_arg_i_after_or:nw` 24, 1127
`\use_arg_i_after_orelse:nw` 24, 1130
`\use_arg_i_delimit_by_q_nil:nw`
 24, 1125, 1749
`\use_arg_i_delimit_by_q_recursion_stop:nw`
 .. 4035, 4043, 4052, 4382, 4392, 4402
`\use_arg_i_delimit_by_q_stop:nw` 24, 1125
`\use_arg_i_ii:nn` 1122
`\use_arg_ii:nn`
 . 23, 1046, 1048, 1055, 1057, 1113,
 2199, 2338, 2529, 3293, 3298, 4087
`\use_arg_ii:nnn` 24, 1115, 4832
`\use_arg_ii:nnnn` 24, 1115, 4319
`\use_arg_iii:nnn` 24, 1115, 4838
`\use_arg_iii:nnnn` 24, 1115, 4323
`\use_arg_iv:nnnn` 24, 1115, 4325
`\use_none:n` .. 23, 750, 754, 1131, 1143,
 1215–1217, 1219–1225, 1273, 1274,
 1694, 1696, 1714, 1717, 2305, 3047,
 3048, 3052, 3300, 3304, 4037, 4045
`\use_none:nn` 23, 1050,
 1052, 1059, 1061, 1131, 2341, 2869
`\use_none:nnn` 23, 1131
`\use_none:nnnn` 23, 1131
`\use_none:nnnnn` 23, 1131
`\use_none:nnnnnn` 23, 1131
`\use_none:nnnnnnn` 23, 1131

`\use_none:nnnnnnnn` 23, 1131
`\use_none:nnnnnnnnn` 23, 1131
`\use_none_delimit_by_q_nil:w` .. 24, 1123
`\use_none_delimit_by_q_recursion_stop:w`
 1562, 2447, 4021, 4026, 4052
`\use_none_delimit_by_q_stop:w`
 24, 1123, 2175, 2731, 4415, 4419
`\use_noop:` 1142, 2210,
 2251, 3422, 3455, 3566, 3617, 5271
`\usepackage` 5070

V

`\vadjust` 233, 521
`\valign` 68
`\value` 5088
`\vbadness` 308
`\vbox` 303
`\vbox:n` 187, 3845
`\vbox_gset:cn` 186, 3846
`\vbox_gset:Nn` 186, 3846
`\vbox_gset_inline_begin:N` 186, 3856
`\vbox_gset_inline_end:` 186, 3861
`\vbox_gset_to_ht:ccn` 186, 3850
`\vbox_gset_to_ht:cnn` 186, 3850
`\vbox_gset_to_ht:Nnn` 186, 3850
`\vbox_set:cn` 186, 3846
`\vbox_set:Nn` 186, 3846
`\vbox_set_inline_begin:N` 186, 3856
`\vbox_set_inline_end:` 186, 3856
`\vbox_set_split_to_ht:Nnn` 187, 3864
`\vbox_set_to_ht:cnn` 186, 3850
`\vbox_set_to_ht:Nnn` 186, 3850
`\vbox_to_ht:nn` 187, 3862
`\vbox_to_zero:n` 187, 3862
`\vbox_unpack:c` 3867
`\vbox_unpack:N` 187, 3867
`\vbox_unpack_clear:c` 3867
`\vbox_unpack_clear:N` 187, 3867
`\vcenter` 154
`\vector` 3748
`\vfil` 215
`\vfill` 217
`\vfilneg` 216
`\vfuzz` 310
`\voffset` 285
`\voidb@x` 3840
`\vrule` 224
`\vsize` 267
`\vskip` 218
`\vsplit` 296
`\vss` 219
`\vtop` 304

W			
\WARNING	5050	\xref_get_value:nn	242, 5030, 5083, 5086, 5089
\wd	351	\xref_new:nn	242, 5016, 5081
\widowpenalties	411	\xref_new_aux:nnn	5016
\widowpenalty	238	\xref_set_label:n	242, 5054, 5091
\widthof	5525	\xref_write	5056, 5063, 5072
\write	101	\xspaceskip	261
X		Y	
\X	4705, 4709	\Y	4706, 4709
\xdef	42	\year	342
\xleaders	227	Z	
\xref_deferred_new:nn	242, 5016, 5085, 5088	\Z	4707, 4709
\xref_define_label:nn	5043, 5057	\z@	3226
\xref_define_label_aux:nn	5043		