

The L^AT_EX3 Sources

The L^AT_EX3 Project*

December 29, 2011

Abstract

This is the reference documentation for the `expl3` programming environment. The `expl3` modules set up an experimental naming scheme for L^AT_EX commands, which allow the L^AT_EX programmer to systematically name functions and variables, and specify the argument types of functions.

The T_EX and ε -T_EX primitives are all given a new name according to these conventions. However, in the main direct use of the primitives is not required or encouraged: the `expl3` modules define an independent low-level L^AT_EX3 programming language.

At present, the `expl3` modules are designed to be loaded on top of L^AT_EX 2 ε . In time, a L^AT_EX3 format will be produced based on this code. This allows the code to be used in L^AT_EX 2 ε packages *now* while a stand-alone L^AT_EX3 is developed.

While `expl3` is still experimental, the bundle is now regarded as broadly stable. The syntax conventions and functions provided are now ready for wider use. There may still be changes to some functions, but these will be minor when compared to the scope of `expl3`.

New modules will be added to the distributed version of `expl3` as they reach maturity.

*E-mail: latex-team@latex-project.org

Contents

I	Introduction to <code>expl3</code> and this document	1
1	Naming functions and variables	1
1.1	Terminological inexactitude	3
2	Documentation conventions	3
3	Formal language conventions which apply generally	5
II	The <code>l3bootstrap</code> package: Bootstrap code	6
4	Using the <code>l^AT_EX3</code> modules	6
III	The <code>l3names</code> package: Namespace for primitives	8
5	Setting up the <code>l^AT_EX3</code> programming language	8
IV	The <code>l3basics</code> package: Basic definitions	9
6	No operation functions	9
7	Grouping material	9
8	Control sequences and functions	10
8.1	Defining functions	10
8.2	Defining new functions using primitive parameter text	10
8.3	Defining new functions using the signature	12
8.4	Copying control sequences	14
8.5	Deleting control sequences	15
8.6	Showing control sequences	15
8.7	Converting to and from control sequences	16
9	Using or removing tokens and arguments	17
9.1	Selecting tokens from delimited arguments	18
9.2	Decomposing control sequences	19
10	Predicates and conditionals	19
10.1	Tests on control sequences	21
10.2	Testing string equality	21
10.3	Engine-specific conditionals	21
10.4	Primitive conditionals	22

11	Internal kernel functions	23
12	Experimental functions	23
V	The <code>l3expan</code> package: Argument expansion	24
13	Defining new variants	24
14	Methods for defining variants	25
15	Introducing the variants	25
16	Manipulating the first argument	26
17	Manipulating two arguments	27
18	Manipulating three arguments	28
19	Unbraced expansion	29
20	Preventing expansion	29
21	Internal functions and variables	31
VI	The <code>l3prg</code> package: Control structures	32
22	Defining a set of conditional functions	32
23	The boolean data type	34
24	Boolean expressions	36
25	Logical loops	37
26	Switching by case	38
27	Producing n copies	39
28	Detecting <code>T_EX</code> 's mode	40
29	Internal programming functions	41
30	Experimental programmings functions	42
VII	The <code>l3quark</code> package: Quarks	43

31	Defining quarks	43
32	Quark tests	44
33	Recursion	44
34	Scan marks	45
35	Internal quark functions	46
VIII The l3token package: Token manipulation		47
36	All possible tokens	47
37	Character tokens	48
38	Generic tokens	51
39	Converting tokens	51
40	Token conditionals	52
41	Peeking ahead at the next token	55
42	Decomposing a macro definition	57
43	Experimental token functions	58
IX The l3int package: Integers		60
44	Integer expressions	60
45	Creating and initialising integers	61
46	Setting and incrementing integers	62
47	Using integers	62
48	Integer expression conditionals	63
49	Integer expression loops	63
50	Formatting integers	65
51	Converting from other formats to integers	67
52	Viewing integers	68

53	Constant integers	68
54	Scratch integers	69
55	Internal functions	69
X	The <code>l3skip</code> package: Dimensions and skips	71
56	Creating and initialising <code>dim</code> variables	71
57	Setting <code>dim</code> variables	71
58	Utilities for dimension calculations	72
59	Dimension expression conditionals	73
60	Dimension expression loops	73
61	Using <code>dim</code> expressions and variables	75
62	Viewing <code>dim</code> variables	75
63	Constant dimensions	75
64	Scratch dimensions	76
65	Creating and initialising <code>skip</code> variables	76
66	Setting <code>skip</code> variables	76
67	Skip expression conditionals	77
68	Using <code>skip</code> expressions and variables	77
69	Viewing <code>skip</code> variables	78
70	Constant skips	78
71	Scratch skips	78
72	Creating and initialising <code>muskip</code> variables	78
73	Setting <code>muskip</code> variables	79
74	Using <code>muskip</code> expressions and variables	79
75	Inserting skips into the output	80

76	Viewing muskip variables	80
77	Internal functions	80
78	Experimental skip functions	81
79	Internal functions	81
XI	The l3tl package: Token lists	82
80	Creating and initialising token list variables	82
81	Adding data to token list variables	83
82	Modifying token list variables	83
83	Reassigning token list category codes	84
84	Reassigning token list character codes	85
85	Token list conditionals	85
86	Mapping to token lists	87
87	Using token lists	88
88	Working with the content of token lists	88
89	The first token from a token list	90
90	Viewing token lists	92
91	Constant token lists	92
92	Scratch token lists	93
93	Experimental token list functions	93
94	Internal functions	94
XII	The l3seq package: Sequences and stacks	95
95	Creating and initialising sequences	95
96	Appending data to sequences	96
97	Recovering items from sequences	96

98	Modifying sequences	97
99	Sequence conditionals	97
100	Mapping to sequences	98
101	Sequences as stacks	99
102	Viewing sequences	100
103	Experimental sequence functions	100
104	Internal sequence functions	102
XIII	The l3clist package: Comma separated lists	104
105	Creating and initialising comma lists	104
106	Adding data to comma lists	105
107	Using comma lists	105
108	Modifying comma lists	105
109	Comma list conditionals	106
110	Mapping to comma lists	107
111	Comma lists as stacks	109
112	Viewing comma lists	110
113	Scratch comma lists	110
114	Experimental comma list functions	110
115	Internal comma-list functions	111
XIV	The l3prop package: Property lists	112
116	Creating and initialising property lists	112
117	Adding entries to property lists	113
118	Recovering values from property lists	113
119	Modifying property lists	114

120	Property list conditionals	114
121	Recovering values from property lists with branching	114
122	Mapping to property lists	115
123	Viewing property lists	116
124	Experimental property list functions	116
125	Internal property list functions	117
XV	The l3box package: Boxes	118
126	Creating and initialising boxes	118
127	Using boxes	119
128	Measuring and setting box dimensions	119
129	Affine transformations	120
130	Viewing part of a box	122
131	Box conditionals	122
132	The last box inserted	123
133	Constant boxes	123
134	Scratch boxes	123
135	Viewing box contents	123
136	Horizontal mode boxes	123
137	Vertical mode boxes	125
138	Primitive box conditionals	127
139	Experimental box functions	127
XVI	The l3coffins package: Coffin code layer	128
140	Creating and initialising coffins	128
141	Setting coffin content and poles	128

142	Coffin transformations	129
143	Joining and using coffins	130
144	Measuring coffins	131
145	Coffin diagnostics	131
XVII	The l3color package: Colour support	132
146	Colour in boxes	132
XVIII	The l3io package: Input–output operations	133
147	Managing streams	133
148	Writing to files	134
149	Wrapping lines in output	136
150	Reading from files	137
151	Constants	138
152	Internal input–output functions	138
XIX	The l3msg package: Messages	139
153	Creating new messages	139
154	Contextual information for messages	140
155	Issuing messages	141
156	Redirecting messages	142
157	Low-level message functions	143
158	Kernel-specific functions	144
159	Expandable errors	145
160	Internal l3msg functions	146
XX	The l3keys package: Key–value interfaces	147

161	Creating keys	148
162	Sub-dividing keys	152
163	Choice and multiple choice keys	153
164	Setting keys	155
165	Setting known keys only	155
166	Utility functions for keys	156
167	Low-level interface for parsing key–val lists	156
XXI	The <code>l3file</code> package: File operations	158
168	File operation functions	158
169	Internal file functions	159
XXII	The <code>l3fp</code> package: Floating-point operations	160
170	Floating-point variables	160
171	Conversion of floating point values to other formats	161
172	Rounding floating point values	162
173	Floating-point conditionals	162
174	Unary floating-point operations	163
175	Floating-point arithmetic	164
176	Floating-point power operations	164
177	Exponential and logarithm functions	164
178	Trigonometric functions	165
179	Constant floating point values	165
180	Notes on the floating point unit	166
XXIII	The <code>l3luatex</code> package: LuaTeX-specific functions	167

181	Breaking out to Lua	167
182	Category code tables	168
XXIV	Implementation	169
183	l3bootstrap implementation	169
183.1	Format-specific code	169
183.2	Package-specific code	170
183.3	Dealing with package-mode meta-data	172
183.4	The <code>\pdfstrcmp</code> primitive in $X_{\text{E}}\text{TeX}$	175
183.5	Engine requirements	175
183.6	The $\text{L}^{\text{A}}\text{TeX}3$ code environment	176
184	l3names implementation	177
185	l3basics implementation	187
185.1	Renaming some $\text{T}_{\text{E}}\text{X}$ primitives (again)	188
185.2	Defining some constants	190
185.3	Defining functions	190
185.4	Selecting tokens	191
185.5	Gobbling tokens from input	192
185.6	Conditional processing and definitions	193
185.7	Dissecting a control sequence	198
185.8	Exist or free	200
185.9	Defining and checking (new) functions	202
185.10	More new definitions	203
185.11	Copying definitions	205
185.12	Undefining functions	206
185.13	Defining functions from a given number of arguments	206
185.14	Using the signature to define functions	208
185.15	Checking control sequence equality	210
185.16	Diagnostic wrapper functions	210
185.17	Engine specific definitions	210
185.18	Doing nothing functions	211
185.19	String comparisons	211
185.20	Breaking out of mapping functions	212
185.21	Deprecated functions	212

186	l3expan implementation	213
186.1	General expansion	214
186.2	Hand-tuned definitions	217
186.3	Definitions with the automated technique	219
186.4	Last-unbraced versions	220
186.5	Preventing expansion	222
186.6	Defining function variants	222
186.7	Variants which cannot be created earlier	225
187	l3prg implementation	226
187.1	Primitive conditionals	226
187.2	Defining a set of conditional functions	226
187.3	The boolean data type	226
187.4	Boolean expressions	228
187.5	Logical loops	233
187.6	Switching by case	234
187.7	Producing n copies	236
187.8	Detecting TeX's mode	239
187.9	Internal programming functions	239
187.10	Experimental programmings functions	241
187.11	Deprecated functions	244
188	l3quark implementation	244
188.1	Quarks	244
188.2	Scan marks	247
189	l3token implementation	248
189.1	Character tokens	248
189.2	Generic tokens	250
189.3	Token conditionals	251
189.4	Peeking ahead at the next token	259
189.5	Decomposing a macro definition	264
189.6	Experimental token functions	265
189.7	Deprecated functions	266
190	l3int implementation	268
190.1	Integer expressions	269
190.2	Creating and initialising integers	271
190.3	Setting and incrementing integers	272
190.4	Using integers	273
190.5	Integer expression conditionals	273
190.6	Integer expression loops	276
190.7	Formatting integers	277
190.8	Converting from other formats to integers	282
190.9	Viewing integer	286
190.10	Constant integers	286

190.1	S cratch integers	287
190.1	D eprecated functions	287
191	l3skip implementation	288
191.1	Length primitives renamed	289
191.2	Creating and initialising <code>dim</code> variables	289
191.3	Setting <code>dim</code> variables	289
191.4	Utilities for dimension calculations	290
191.5	Dimension expression conditionals	291
191.6	Dimension expression loops	292
191.7	Using <code>dim</code> expressions and variables	294
191.8	Viewing <code>dim</code> variables	295
191.9	Constant dimensions	295
191.1	S cratch dimensions	295
191.1	Creating and initialising <code>skip</code> variables	295
191.1	S etting <code>skip</code> variables	296
191.1	S kip expression conditionals	297
191.1	Using <code>skip</code> expressions and variables	297
191.1	I nserting skips into the output	298
191.1	V iewing <code>skip</code> variables	298
191.1	C onstant skips	298
191.1	S cratch skips	298
191.1	Creating and initialising <code>muskip</code> variables	299
191.2	Setting <code>muskip</code> variables	299
191.2	Using <code>muskip</code> expressions and variables	300
191.2	V iewing <code>muskip</code> variables	300
191.2	E xperimental skip functions	300
192	l3tl implementation	301
192.1	Functions	301
192.2	Adding to token list variables	302
192.3	Reassigning token list category codes	304
192.4	Reassigning token list character codes	305
192.5	Modifying token list variables	305
192.6	Token list conditionals	308
192.7	Mapping to token lists	311
192.8	Using token lists	312
192.9	Working with the contents of token lists	312
192.1	The first token from a token list	314
192.1	Viewing token lists	318
192.1	C onstant token lists	319
192.1	S cratch token lists	319
192.1	E xperimental functions	320
192.1	D eprecated functions	325

193	l3seq implementation	327
193.1	Allocation and initialisation	328
193.2	Appending data to either end	329
193.3	Modifying sequences	330
193.4	Sequence conditionals	331
193.5	Recovering data from sequences	332
193.6	Mapping to sequences	335
193.7	Sequence stacks	337
193.8	Viewing sequences	337
193.9	Experimental functions	338
193.10	Deprecated interfaces	343
194	l3clist implementation	344
194.1	Allocation and initialisation	344
194.2	Removing spaces around items	345
194.3	Adding data to comma lists	347
194.4	Comma lists as stacks	348
194.5	Using comma lists	349
194.6	Modifying comma lists	349
194.7	Comma list conditionals	350
194.8	Mapping to comma lists	351
194.9	Viewing comma lists	354
194.10	Scratch comma lists	354
194.11	Experimental functions	354
194.12	Deprecated interfaces	358
195	l3prop implementation	359
195.1	Allocation and initialisation	359
195.2	Accessing data in property lists	360
195.3	Property list conditionals	363
195.4	Recovering values from property lists with branching	365
195.5	Mapping to property lists	365
195.6	Viewing property lists	366
195.7	Experimental functions	366
195.8	Deprecated interfaces	368

196	l3box implementation	369
196.1	Creating and initialising boxes	369
196.2	Measuring and setting box dimensions	370
196.3	Using boxes	371
196.4	Box conditionals	371
196.5	The last box inserted	372
196.6	Constant boxes	372
196.7	Scratch boxes	372
196.8	Viewing box contents	373
196.9	Horizontal mode boxes	373
196.10	Vertical mode boxes	375
196.11	Affine transformations	377
196.12	Viewing part of a box	385
196.13	Deprecated functions	387
197	l3coffins Implementation	387
197.1	Coffins: data structures and general variables	387
197.2	Basic coffin functions	389
197.3	Measuring coffins	393
197.4	Coffins: handle and pole management	393
197.5	Coffins: calculation of pole intersections	396
197.6	Aligning and typesetting of coffins	400
197.7	Rotating coffins	404
197.8	Resizing coffins	409
197.9	Coffin diagnostics	412
197.10	Messages	418
198	l3color Implementation	418
199	l3io implementation	419
199.1	Primitives	419
199.2	Variables and constants	419
199.3	Stream management	421
199.4	Deferred writing	425
199.5	Immediate writing	426
199.6	Special characters for writing	426
199.7	Hard-wrapping lines based on length	427
199.8	Reading input	432
199.9	Deprecated functions	433

200	l3msg implementation	433
200.1	Creating messages	434
200.2	Messages: support functions and text	435
200.3	Showing messages: low level mechanism	436
200.4	Displaying messages	438
200.5	Kernel-specific functions	443
200.6	Expandable errors	448
200.7	Showing variables	450
200.8	Deprecated functions	451
201	l3keys Implementation	452
201.1	Low-level interface	453
201.2	Constants and variables	456
201.3	The key defining mechanism	457
201.4	Turning properties into actions	459
201.5	Creating key properties	463
201.6	Setting keys	467
201.7	Utilities	470
201.8	Messages	470
201.9	Deprecated functions	471
202	l3file implementation	472
203	l3fp Implementation	475
203.1	Constants	476
203.2	Variables	477
203.3	Parsing numbers	479
203.4	Internal utilities	482
203.5	Operations for fp variables	484
203.6	Transferring to other types	488
203.7	Rounding numbers	495
203.8	Unary functions	498
203.9	Basic arithmetic	499
203.10	Arithmetic for internal use	508
203.11	Trigonometric functions	515
203.12	Exponent and logarithm functions	528
203.13	Tests for special values	549
203.14	Floating-point conditionals	550
203.15	Messages	555
204	l3luatex implementation	556
204.1	Category code tables	557
204.2	Deprecated functions	560
	Index	561

Part I

Introduction to expl3 and this document

This document is intended to act as a comprehensive reference manual for the `expl3` language. A general guide to the `LATEX3` programming language is found in [expl3.pdf](#).

1 Naming functions and variables

`LATEX3` does not use `@` as a “letter” for defining internal macros. Instead, the symbols `_` and `:` are used in internal macro names to provide structure. The name of each *function* is divided into logical units using `_`, while `:` separates the *name* of the function from the *argument specifier* (“arg-spec”). This describes the arguments expected by the function. In most cases, each argument is represented by a single letter. The complete list of arg-spec letters for a function is referred to as the *signature* of the function.

Each function name starts with the *module* to which it belongs. Thus apart from a small number of very basic functions, all `expl3` function names contain at least one underscore to divide the module name from the descriptive name of the function. For example, all functions concerned with comma lists are in module `clist` and begin `\clist_`.

Every function must include an argument specifier. For functions which take no arguments, this will be blank and the function name will end `:`. Most functions take one or more arguments, and use the following argument specifiers:

- D** The **D** specifier means *do not use*. All of the `TEX` primitives are initially `\let` to a **D** name, and some are then given a second name. Only the kernel team should use anything with a **D** specifier!
- N and n** These mean *no manipulation*, of a single token for **N** and of a set of tokens given in braces for **n**. Both pass the argument though exactly as given. Usually, if you use a single token for an **n** argument, all will be well.
- c** This means *csname*, and indicates that the argument will be turned into a *csname* before being used. So `\foo:c {ArgumentOne}` will act in the same way as `\foo:N \ArgumentOne`.
- V and v** These mean *value of variable*. The **V** and **v** specifiers are used to get the content of a variable without needing to worry about the underlying `TEX` structure containing the data. A **V** argument will be a single token (similar to **N**), for example `\foo:V \MyVariable`; on the other hand, using **v** a *csname* is constructed first, and then the value is recovered, for example `\foo:v {MyVariable}`.
- o** This means *expansion once*. In general, the **V** and **v** specifiers are favoured over **o** for recovering stored information. However, **o** is useful for correctly processing information with delimited arguments.

- x** The **x** specifier stands for *exhaustive expansion*: the plain \TeX `\edef`.
- f** The **f** specifier stands for *full expansion*, and in contrast to *x* stops at the first non-expandable item without trying to execute it.
- T and F** For logic tests, there are the branch specifiers **T** (*true*) and **F** (*false*). Both specifiers treat the input in the same way as **n** (no change), but make the logic much easier to see.
- p** The letter **p** indicates \TeX *parameters*. Normally this will be used for delimited functions as `expl3` provides better methods for creating simple sequential arguments.
- w** Finally, there is the **w** specifier for *weird* arguments. This covers everything else, but mainly applies to delimited values (where the argument must be terminated by some arbitrary string).

Notice that the argument specifier describes how the argument is processed prior to being passed to the underlying function. For example, `\foo:c` will take its argument, convert it to a control sequence and pass it to `\foo:N`.

Variables are named in a similar manner to functions, but begin with a single letter to define the type of variable:

- c** Constant: global parameters whose value should not be changed.
- g** Parameters whose value should only be set globally.
- l** Parameters whose value should only be set locally.

Each variable name is then build up in a similar way to that of a function, typically starting with the module¹ name and then a descriptive part. Variables end with a short identifier to show the variable type:

- bool** Either true or false.
- box** Box register.
- clist** Comma separated list.
- coffin** a “box with handles” — a higher-level data type for carrying out **box** alignment operations.
- dim** “Rigid” lengths.
- fp** floating-point values;
- int** Integer-valued count register.
- prop** Property list.

¹The module names are not used in case of generic scratch registers defined in the data type modules, e.g., the **int** module contains some scratch variables called `\l_tmpa_int`, `\l_tmpb_int`, and so on. In such a case adding the module name up front to denote the module and in the back to indicate the type, as in `\l_int_tmpa_int` would be very unreadable.

seq “Sequence”: a data-type used to implement lists (with access at both ends) and stacks.

skip “Rubber” lengths.

stream An input or output stream (for reading from or writing to, respectively).

tl Token list variables: placeholder for a token list.

1.1 Terminological inexactitude

A word of warning. In this document, and others referring to the `expl3` programming modules, we often refer to “variables” and “functions” as if they were actual constructs from a real programming language. In truth, `TeX` is a macro processor, and functions are simply macros that may or may not take arguments and expand to their replacement text. Many of the common variables are *also* macros, and if placed into the input stream will simply expand to their definition as well — a “function” with no arguments and a “token list variable” are in truth one and the same. On the other hand, some “variables” are actually registers that must be initialised and their values set and retrieved with specific functions.

The conventions of the `expl3` code are designed to clearly separate the ideas of “macros that contain data” and “macros that contain code”, and a consistent wrapper is applied to all forms of “data” whether they be macros or actually registers. This means that sometimes we will use phrases like “the function returns a value”, when actually we just mean “the macro expands to something”. Similarly, the term “execute” might be used in place of “expand” or it might refer to the more specific case of “processing in `TeX`’s stomach” (if you are familiar with the `TeXbook` parlance).

If in doubt, please ask; chances are we’ve been hasty in writing certain definitions and need to be told to tighten up our terminology.

2 Documentation conventions

This document is typeset with the experimental `l3doc` class; several conventions are used to help describe the features of the code. A number of conventions are used here to make the documentation clearer.

Each group of related functions is given in a box. For a function with a “user” name, this might read:

```
\ExplSyntaxOn
\ExplSyntaxOff
```

```
\ExplSyntaxOn ... \ExplSyntaxOff
```

The textual description of how the function works would appear here. The syntax of the function is shown in mono-spaced text to the right of the box. In this example, the function takes no arguments and so the name of the function is simply reprinted.

For programming functions, which use `_` and `:` in their name there are a few additional conventions: If two related functions are given with identical names but different argument specifiers, these are termed *variants* of each other, and the latter functions are

printed in grey to show this more clearly. They will carry out the same function but will take different types of argument:

`\seq_new:N`
`\seq_new:c`

`\seq_new:N` $\langle sequence \rangle$

When a number of variants are described, the arguments are usually illustrated only for the base function. Here, $\langle sequence \rangle$ indicates that `\seq_new:N` expects the name of a sequence. From the argument specifier, `\seq_new:c` also expects a sequence name, but as a name rather than as a control sequence. Each argument given in the illustration should be described in the following text.

Fully expandable functions Some functions are fully expandable, which allows it to be used within an **x**-type argument (in plain \TeX terms, inside an `\edef`), as well as within an **f**-type argument. These fully expandable functions are indicated in the documentation by a star:

`\cs_to_str:N` ☆

`\cs_to_str:N` $\langle cs \rangle$

As with other functions, some text should follow which explains how the function works. Usually, only the star will indicate that the function is expandable. In this case, the function expects a $\langle cs \rangle$, shorthand for a $\langle control\ sequence \rangle$.

Restricted expandable functions A few functions are fully expandable but cannot be fully expanded within an **f**-type argument. In this case a hollow star is used to indicate this:

`\seq_map_function:NN` ☆

`\seq_map_function:NN` $\langle seq \rangle$ $\langle function \rangle$

Conditional functions Conditional (**if**) functions are normally defined in three variants, with **T**, **F** and **TF** argument specifiers. This allows them to be used for different “true”/“false” branches, depending on which outcome the conditional is being used to test. To indicate this without repetition, this information is given in a shortened form:

`\xetex_if_engineTF` ☆

`\xetex_if_engine:TF` $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

The underlining and italic of **TF** indicates that `\xetex_if_engine:T`, `\xetex_if_engine:F` and `\xetex_if_engine:TF` are all available. Usually, the illustration will use the **TF** variant, and so both $\langle true\ code \rangle$ and $\langle false\ code \rangle$ will be shown. The two variant forms **T** and **F** take only $\langle true\ code \rangle$ and $\langle false\ code \rangle$, respectively. Here, the star also shows that this function is expandable. With some minor exceptions, *all* conditional functions in the `expl3` modules should be defined in this way.

Variables, constants and so on are described in a similar manner:

`\l_tmpa_tl`

A short piece of text will describe the variable: there is no syntax illustration in this case.

In some cases, the function is similar to one in $\text{\LaTeX} 2_{\epsilon}$ or plain \TeX . In these cases, the text will include an extra “ **\TeX hackers note**” section:

<code>\token_to_str:N</code> ★	<code>\token_to_str:N</code> $\langle token \rangle$
--------------------------------	--

The normal description text.

T_EXhackers note: Detail for the experienced T_EX or L^AT_EX 2_ε programmer. In this case, it would point out that this function is the T_EX primitive `\string`.

3 Formal language conventions which apply generally

As this is a formal reference guide for L^AT_EX3 programming, the descriptions of functions are intended to be reasonably “complete”. However, there is also a need to avoid repetition. Formal ideas which apply to general classes of function are therefore summarised here.

For tests which have a **TF** argument specification, the test is evaluated to give a logically **TRUE** or **FALSE** result. Depending on this result, either the $\langle true\ code \rangle$ or the $\langle false\ code \rangle$ will be left in the input stream. In the case where the test is expandable, and a predicate (`_p`) variant is available, the logical value determined by the test is left in the input stream: this will typically be part of a larger logical construct.

Part II

The l3bootstrap package

Bootstrap code

4 Using the L^AT_EX3 modules

The modules documented in `source3` are designed to be used on top of L^AT_EX 2_ε and are loaded all as one with the usual `\usepackage{expl3}` or `\RequirePackage{expl3}` instructions. These modules will also form the basis of the L^AT_EX3 format, but work in this area is incomplete and not included in this documentation at present.

As the modules use a coding syntax different from standard L^AT_EX 2_ε it provides a few functions for setting it up.

`\ExplSyntaxOn`
`\ExplSyntaxOff`

Updated: 2011-08-13

`\ExplSyntaxOn` *<code>* `\ExplSyntaxOff`

The `\ExplSyntaxOn` function switches to a category code régime in which spaces are ignored and in which the colon (`:`) and underscore (`_`) are treated as “letters”, thus allowing access to the names of code functions and variables. Within this environment, `~` is used to input a space. The `\ExplSyntaxOff` reverts to the document category code régime.

`\ExplSyntaxNamesOn`
`\ExplSyntaxNamesOff`

`\ExplSyntaxNamesOn` *<code>* `\ExplSyntaxNamesOff`

The `\ExplSyntaxOn` function switches to a category code régime in which the colon (`:`) and underscore (`_`) are treated as “letters”, thus allowing access to the names of code functions and variables. In contrast to `\ExplSyntaxOn`, using `\ExplSyntaxNamesOn` does not cause spaces to be ignored. The `\ExplSyntaxNamesOff` reverts to the document category code régime.

`\ProvidesExplPackage`
`\ProvidesExplClass`
`\ProvidesExplFile`

`\RequirePackage{expl3}`
`\ProvidesExplPackage` *{<package>}* *{<date>}* *{<version>}* *{<description>}*

These functions act broadly in the same way as the L^AT_EX 2_ε kernel functions `\ProvidesPackage`, `\ProvidesClass` and `\ProvidesFile`. However, they also implicitly switch `\ExplSyntaxOn` for the remainder of the code with the file. At the end of the file, `\ExplSyntaxOff` will be called to reverse this. (This is the same concept as L^AT_EX 2_ε provides in turning on `\makeatletter` within package and class code.)

`\GetIdInfo`

`\RequirePackage{l3names}`
`\GetIdInfo` *\$Id: <SVN info field> \$* *{<description>}*

Extracts all information from a SVN field. Spaces are not ignored in these fields. The information pieces are stored in separate control sequences with `\ExplFileName` for the part of the file name leading up to the period, `\ExplFileDate` for date, `\ExplFileVersion` for version and `\ExplFileDescription` for the description.

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 $\text{\LaTeX} 2_{\epsilon}$ category codes and the $\text{\LaTeX} 3$ category code 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{\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescript
```

Part III

The l3names package Namespace for primitives

5 Setting up the L^AT_EX3 programming language

This module is at the core of the L^AT_EX3 programming language. It performs the following tasks:

- defines new names for all T_EX primitives;
- switches to the category code regime for programming;
- provides support settings for building the code as a T_EX format.

This module is entirely dedicated to primitives, which should not be used directly within L^AT_EX3 code (outside of “kernel-level” code). As such, the primitives are not documented here: *The T_EXbook*, *T_EX by Topic* and the manuals for pdfT_EX, X_YT_EX and LuaT_EX should be consulted for details of the primitives. These are named based on the engine which first introduced them:

`\tex_...` Introduced by T_EX itself;

`\etex_...` Introduced by the ε -T_EX extensions;

`\pdftex_...` Introduced by pdfT_EX;

`\xetex_...` Introduced by X_YT_EX;

`\luatex_...` Introduced by LuaT_EX.

Part IV

The l3basics package

Basic definitions

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

Here we describe those functions that are 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.

6 No operation functions

`\prg_do_nothing` ★**`\prg_do_nothing:`**

An expandable function which does nothing at all: leaves nothing in the input stream after a single expansion.

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

A non-expandable function which does nothing. Does not vanish on expansion but produces no typeset output.

7 Grouping material

`\group_begin`**`\group_begin:`**

`\group_end`**`\group_end:`**

These functions begin and end a group for definition purposes. Assignments are local to groups unless carried out in a global manner. (A small number of exceptions to this rule will be noted as necessary elsewhere in this document.) Each `\group_begin:` must be matched by a `\group_end:`, although this does not have to occur within the same function. Indeed, it is often necessary to start a group within one function and finish it within another, for example when seeking to use non-standard category codes.

`\group_insert_after:N`**`\group_insert_after:N` *<token>***

Adds *<token>* to the list of *<tokens>* to be inserted when the current group level ends. The list of *<tokens>* to be inserted will be empty at the beginning of a group: multiple applications of `\group_insert_after:N` may be used to build the inserted list one *<token>* at a time. The current group level may be closed by a `\group_end:` function or by a token with category code 2 (close-group). The later will be a `}` if standard category codes apply.

8 Control sequences and functions

As \TeX is a macro language, creating new functions means creating macros. At point of use, a function is replaced by the replacement text (“code”) in which each parameter in the code ($\#1$, $\#2$, *etc.*) is replaced the appropriate arguments absorbed by the function. In the following, $\langle code \rangle$ is therefore used as a shorthand for “replacement text”.

Functions which are not “protected” will be fully expanded inside an \mathbf{x} expansion. In contrast, “protected” functions are not expanded within \mathbf{x} expansions.

8.1 Defining functions

Functions can be created with no requirement that they are declared first (in contrast to variables, which must always be declared). Declaring a function before setting up the code means that the name chosen will be checked and an error raised if it is already in use. The name of a function can be checked at the point of definition using the `\cs_new...` functions: this is recommended for all functions which are defined for the first time.

8.2 Defining new functions using primitive parameter text

<code>\cs_new:Npn</code>
<code>\cs_new:(cpn Npx cpx)</code>

`\cs_new:Npn $\langle function \rangle$ $\langle parameters \rangle$ { $\langle code \rangle$ }`

Creates $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

<code>\cs_new_nopar:Npn</code>
<code>\cs_new_nopar:(cpn Npx cpx)</code>

`\cs_new_nopar:Npn $\langle function \rangle$ $\langle parameters \rangle$ { $\langle code \rangle$ }`

Creates $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain `\par` tokens. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

<code>\cs_new_protected:Npn</code>
<code>\cs_new_protected:(cpn Npx cpx)</code>

`\cs_new_protected:Npn $\langle function \rangle$ $\langle parameters \rangle$ { $\langle code \rangle$ }`

Creates $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. The $\langle function \rangle$ will not expand within an \mathbf{x} -type argument. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

$\backslash\text{cs_new_protected_nopar:Npn}$ $\backslash\text{cs_new_protected_nopar: (cpn Npx cpx)}$	$\backslash\text{cs_new_protected_nopar:Npn } \langle function \rangle \langle parameters \rangle \{ \langle code \rangle \}$
---	--

Creates $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ (#1, #2, etc.) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain $\backslash\text{par}$ tokens. The $\langle function \rangle$ will not expand within an x-type argument. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

$\backslash\text{cs_set:Npn}$ $\backslash\text{cs_set: (cpn Npx cpx)}$	$\backslash\text{cs_set:Npn } \langle function \rangle \langle parameters \rangle \{ \langle code \rangle \}$
---	--

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ (#1, #2, etc.) will be replaced by those absorbed by the function. The assignment of a meaning to $\langle function \rangle$ is restricted to the current $\text{T}_{\text{E}}\text{X}$ group level.

$\backslash\text{cs_set_nopar:Npn}$ $\backslash\text{cs_set_nopar: (cpn Npx cpx)}$	$\backslash\text{cs_set_nopar:Npn } \langle function \rangle \langle parameters \rangle \{ \langle code \rangle \}$
---	---

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ (#1, #2, etc.) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain $\backslash\text{par}$ tokens. The assignment of a meaning to $\langle function \rangle$ is restricted to the current $\text{T}_{\text{E}}\text{X}$ group level.

$\backslash\text{cs_set_protected:Npn}$ $\backslash\text{cs_set_protected: (cpn Npx cpx)}$	$\backslash\text{cs_set_protected:Npn } \langle function \rangle \langle parameters \rangle \{ \langle code \rangle \}$
---	---

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ (#1, #2, etc.) will be replaced by those absorbed by the function. The assignment of a meaning to $\langle function \rangle$ is restricted to the current $\text{T}_{\text{E}}\text{X}$ group level. The $\langle function \rangle$ will not expand within an x-type argument.

$\backslash\text{cs_set_protected_nopar:Npn}$ $\backslash\text{cs_set_protected_nopar: (cpn Npx cpx)}$	$\backslash\text{cs_set_protected_nopar:Npn } \langle function \rangle \langle parameters \rangle \{ \langle code \rangle \}$
---	--

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ (#1, #2, etc.) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain $\backslash\text{par}$ tokens. The assignment of a meaning to $\langle function \rangle$ is restricted to the current $\text{T}_{\text{E}}\text{X}$ group level. The $\langle function \rangle$ will not expand within an x-type argument.

$\backslash\text{cs_gset:Npn}$ $\backslash\text{cs_gset: (cpn Npx cpx)}$	$\backslash\text{cs_gset:Npn } \langle function \rangle \langle parameters \rangle \{ \langle code \rangle \}$
---	---

Globally sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ (#1, #2, etc.) will be replaced by those absorbed by the function. The assignment of a meaning to $\langle function \rangle$ is *not* restricted to the current $\text{T}_{\text{E}}\text{X}$ group level: the assignment is global.

```
\cs_gset_nopar:Npn
\cs_gset_nopar:(cpn|Npx|cpx)
```

```
\cs_gset_nopar:Npn <function> <parameters> {\<code>}
```

Globally sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain $\backslash par$ tokens. The assignment of a meaning to $\langle function \rangle$ is *not* restricted to the current \TeX group level: the assignment is global.

```
\cs_gset_protected:Npn
\cs_gset_protected:(cpn|Npx|cpx)
```

```
\cs_gset_protected:Npn <function> <parameters> {\<code>}
```

Globally sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. The assignment of a meaning to $\langle function \rangle$ is *not* restricted to the current \TeX group level: the assignment is global. The $\langle function \rangle$ will not expand within an x -type argument.

```
\cs_gset_protected_nopar:Npn
\cs_gset_protected_nopar:(cpn|Npx|cpx)
```

```
\cs_gset_protected_nopar:Npn <function> <parameters> {\<code>}
```

Globally sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain $\backslash par$ tokens. The assignment of a meaning to $\langle function \rangle$ is *not* restricted to the current \TeX group level: the assignment is global. The $\langle function \rangle$ will not expand within an x -type argument.

8.3 Defining new functions using the signature

```
\cs_new:Nn
\cs_new:(cn|Nx|cx)
```

```
\cs_new:Nn <function> {\<code>}
```

Creates $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

```
\cs_new_nopar:Nn
\cs_new_nopar:(cn|Nx|cx)
```

```
\cs_new_nopar:Nn <function> {\<code>}
```

Creates $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain $\backslash par$ tokens. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

```
\cs_new_protected:Nn
\cs_new_protected:(cn|Nx|cx)
```

```
\cs_new_protected:Nn <function> {\<code>}
```

Creates $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. The $\langle function \rangle$ will not expand within an x -type argument. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

<code>\cs_new_protected_nopar:Nn</code>	<code>\cs_new_protected_nopar:Nn <function> {<code>}</code>
<code>\cs_new_protected_nopar:(cn Nx cx)</code>	

Creates $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain `\par` tokens. The $\langle function \rangle$ will not expand within an x-type argument. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

<code>\cs_set:Nn</code>	<code>\cs_set:Nn <function> {<code>}</code>
<code>\cs_set:(cn Nx cx)</code>	

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. The assignment of a meaning to $\langle function \rangle$ is restricted to the current T_EX group level.

<code>\cs_set_nopar:Nn</code>	<code>\cs_set_nopar:Nn <function> {<code>}</code>
<code>\cs_set_nopar:(cn Nx cx)</code>	

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain `\par` tokens. The assignment of a meaning to $\langle function \rangle$ is restricted to the current T_EX group level.

<code>\cs_set_protected:Nn</code>	<code>\cs_set_protected:Nn <function> {<code>}</code>
<code>\cs_set_protected:(cn Nx cx)</code>	

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. The $\langle function \rangle$ will not expand within an x-type argument. The assignment of a meaning to $\langle function \rangle$ is restricted to the current T_EX group level.

<code>\cs_set_protected_nopar:Nn</code>	<code>\cs_set_protected_nopar:Nn <function> {<code>}</code>
<code>\cs_set_protected_nopar:(cn Nx cx)</code>	

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain `\par` tokens. The $\langle function \rangle$ will not expand within an x-type argument. The assignment of a meaning to $\langle function \rangle$ is restricted to the current T_EX group level.

<code>\cs_gset:Nn</code>	<code>\cs_gset:Nn <function> {<code>}</code>
<code>\cs_gset:(cn Nx cx)</code>	

Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, *etc.*) will be replaced by those absorbed by the function. The assignment of a meaning to $\langle function \rangle$ is global.

<code>\cs_gset_nopar:Nn</code>	<code>\cs_gset_nopar:Nn <function> {<code>}</code>
<code>\cs_gset_nopar:(cn Nx cx)</code>	Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, <i>etc.</i>) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain <code>\par</code> tokens. The assignment of a meaning to $\langle function \rangle$ is global.

<code>\cs_gset_protected:Nn</code>	<code>\cs_gset_protected:Nn <function> {<code>}</code>
<code>\cs_gset_protected:(cn Nx cx)</code>	Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, <i>etc.</i>) will be replaced by those absorbed by the function. The $\langle function \rangle$ will not expand within an <i>x</i> -type argument. The assignment of a meaning to $\langle function \rangle$ is global.

<code>\cs_gset_protected_nopar:Nn</code>	<code>\cs_gset_protected_nopar:Nn <function> {<code>}</code>
<code>\cs_gset_protected_nopar:(cn Nx cx)</code>	Sets $\langle function \rangle$ to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the number of $\langle parameters \rangle$ is detected automatically from the function signature. These $\langle parameters \rangle$ ($\#1$, $\#2$, <i>etc.</i>) will be replaced by those absorbed by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain <code>\par</code> tokens. The $\langle function \rangle$ will not expand within an <i>x</i> -type argument. The assignment of a meaning to $\langle function \rangle$ is global.

<code>\cs_generate_from_arg_count:NNnn</code>	<code>\cs_generate_from_arg_count:NNnn <function> <creator> <number> <code></code>
<code>\cs_generate_from_arg_count:cNnn</code>	

Updated: 2011-09-05

Uses the $\langle creator \rangle$ function (which should have signature `Npn`, for example `\cs_new:Npn`) to define a $\langle function \rangle$ which takes $\langle number \rangle$ arguments and has $\langle code \rangle$ as replacement text. The $\langle number \rangle$ of arguments is an integer expression, evaluated as detailed for `\int_eval:n`.

8.4 Copying control sequences

Control sequences (not just functions as defined above) can be set to have the same meaning using the functions described here. Making two control sequences equivalent means that the second control sequence is a *copy* of the first (rather than a pointer to it). Thus the old and new control sequence are not tied together: changes to one are not reflected in the other.

In the following text “cs” is used as an abbreviation for “control sequence”.

<code>\cs_new_eq:NN</code>	<code>\cs_new_eq:NN <cs 1> <cs 2></code>
<code>\cs_new_eq:(Nc cN cc)</code>	<code>\cs_new_eq:NN <cs 1> <token></code>

Globally creates $\langle control\ sequence\ 1 \rangle$ and sets it to have the same meaning as $\langle control\ sequence\ 2 \rangle$ or $\langle token \rangle$. The second control sequence may subsequently be altered without affecting the copy.

```
\cs_set_eq:NN
\cs_set_eq:(Nc|cN|cc)
```

```
\cs_set_eq:NN <cs 1> <cs 2>
\cs_set_eq:NN <cs 1> <token>
```

Sets $\langle \textit{control sequence 1} \rangle$ to have the same meaning as $\langle \textit{control sequence 2} \rangle$ (or $\langle \textit{token} \rangle$). The second control sequence may subsequently be altered without affecting the copy. The assignment of a meaning to $\langle \textit{control sequence 1} \rangle$ is restricted to the current \TeX group level.

```
\cs_gset_eq:NN
\cs_gset_eq:(Nc|cN|cc)
```

```
\cs_gset_eq:NN <cs 1> <cs 2>
\cs_gset_eq:NN <cs 1> <token>
```

Globally sets $\langle \textit{control sequence 1} \rangle$ to have the same meaning as $\langle \textit{control sequence 2} \rangle$ (or $\langle \textit{token} \rangle$). The second control sequence may subsequently be altered without affecting the copy. The assignment of a meaning to $\langle \textit{control sequence 1} \rangle$ is *not* restricted to the current \TeX group level: the assignment is global.

8.5 Deleting control sequences

There are occasions where control sequences need to be deleted. This is handled in a very simple manner.

```
\cs_undefine:N
\cs_undefine:c
```

```
\cs_undefine:N <control sequence>
```

Sets $\langle \textit{control sequence} \rangle$ to be globally undefined.

Updated: 2011-09-15

8.6 Showing control sequences

```
\cs_meaning:N ★
\cs_meaning:c ★
```

```
\cs_meaning:N <control sequence>
```

This function expands to the *meaning* of the $\langle \textit{control sequence} \rangle$ control sequence. This will show the $\langle \textit{replacement text} \rangle$ for a macro.

\TeX hackers note: This is \TeX 's `\meaning` primitive. The `c` variant correctly reports undefined arguments.

```
\cs_show:N
\cs_show:c
```

```
\cs_show:N <control sequence>
```

Displays the definition of the $\langle \textit{control sequence} \rangle$ on the terminal.

\TeX hackers note: This is the \TeX primitive `\show`.

Updated: 2011-12-22

8.7 Converting to and from control sequences

`\use:c` ★ `\use:c {⟨control sequence name⟩}`

Converts the given *⟨control sequence name⟩* into a single control sequence token. This process requires two expansions. The content for *⟨control sequence name⟩* may be literal material or from other expandable functions. The *⟨control sequence name⟩* must, when fully expanded, consist of character tokens which are not active: typically, they will be of category code 10 (space), 11 (letter) or 12 (other), or a mixture of these.

As an example of the `\use:c` function, both

```
\use:c { a b c }
```

and

```
\tl_new:N \l_my_tl
\tl_set:Nn \l_my_tl { a b c }
\use:c { \tl_use:N \l_my_tl }
```

would be equivalent to

```
\abc
```

after two expansions of `\use:c`.

`\cs:w` ★ `\cs:w ⟨control sequence name⟩ \cs_end:`
`\cs_end` ★

Converts the given *⟨control sequence name⟩* into a single control sequence token. This process requires one expansion. The content for *⟨control sequence name⟩* may be literal material or from other expandable functions. The *⟨control sequence name⟩* must, when fully expanded, consist of character tokens which are not active: typically, they will be of category code 10 (space), 11 (letter) or 12 (other), or a mixture of these.

TeXhackers note: These are the TeX primitives `\csname` and `\endcsname`.

As an example of the `\cs:w` and `\cs_end:` functions, both

```
\cs:w a b c \cs_end:
```

and

```
\tl_new:N \l_my_tl
\tl_set:Nn \l_my_tl { a b c }
\cs:w \tl_use:N \l_my_tl \cs_end:
```

would be equivalent to

```
\abc
```


after one expansion of `\cs:w`.

<code>\cs_to_str:N</code>	<code>\star</code>	<code>\cs_to_str:N {\langle control sequence \rangle}</code>
---------------------------	--------------------	--

Converts the given $\langle control\ sequence \rangle$ into a series of characters with category code 12 (other), except spaces, of category code 10. The sequence will *not* include the current escape token, *cf.* `\token_to_str:N`. Full expansion of this function requires exactly 2 expansion steps, and so an `x`-type expansion, or two `o`-type expansions will be required to convert the $\langle control\ sequence \rangle$ to a sequence of characters in the input stream. In most cases, an `f`-expansion will be correct as well, but this loses a space at the start of the result.

9 Using or removing tokens and arguments

Tokens in the input can be read and used or read and discarded. If one or more tokens are wrapped in braces then in absorbing them the outer set will be removed. At the same time, the category code of each token is set when the token is read by a function (if it is read more than once, the category code is determined by the the situation in force when first function absorbs the token).

<code>\use:n</code>	<code>\star</code>	<code>\use:n {\langle group_1 \rangle}</code>
<code>\use:(nn nnn nnnn)</code>	<code>\star</code>	<code>\use:nn {\langle group_1 \rangle} {\langle group_2 \rangle}</code> <code>\use:nnn {\langle group_1 \rangle} {\langle group_2 \rangle} {\langle group_3 \rangle}</code> <code>\use:nnnn {\langle group_1 \rangle} {\langle group_2 \rangle} {\langle group_3 \rangle} {\langle group_4 \rangle}</code>

As illustrated, these functions will absorb between one and four arguments, as indicated by the argument specifier. The braces surrounding each argument will be removed leaving the remaining tokens in the input stream. The category code of these tokens will also be fixed by this process (if it has not already been by some other absorption). All of these functions require only a single expansion to operate, so that one expansion of

`\use:nn { abc } { { def } }`

will result in the input stream containing

`abc { def }`

i.e. only the outer braces will be removed.

<code>\use_i:nn</code>	<code>\star</code>	<code>\use_i:nn {\langle group_1 \rangle} {\langle group_2 \rangle}</code>
<code>\use_ii:nn</code>	<code>\star</code>	

These functions will absorb two groups and leave only the first or the second in the input stream. The braces surrounding the arguments will be removed as part of this process. The category code of these tokens will also be fixed (if it has not already been by some other absorption). A single expansion is needed for the functions to take effect.

<code>\use_i:nnn</code>	★	<code>\use_i:nnn {⟨group₁⟩} {⟨group₂⟩} {⟨group₃⟩}</code>
<code>\use_ii:nnn</code>	★	These functions will absorb three groups and leave only of these in the input stream. The braces surrounding the arguments will be removed as part of this process. The category code of these tokens will also be fixed (if it has not already been by some other absorption). A single expansion is needed for the functions to take effect.
<code>\use_iii:nnn</code>	★	

<code>\use_i:nnnn</code>	★	<code>\use_i:nnnn {⟨group₁⟩} {⟨group₂⟩} {⟨group₃⟩} {⟨group₄⟩}</code>
<code>\use_ii:nnnn</code>	★	These functions will absorb four groups and leave only of these in the input stream. The braces surrounding the arguments will be removed as part of this process. The category code of these tokens will also be fixed (if it has not already been by some other absorption). A single expansion is needed for the functions to take effect.
<code>\use_iii:nnnn</code>	★	
<code>\use_iv:nnnn</code>	★	

<code>\use_i_ii:nnn</code>	★	<code>\use_i_ii:nnn {⟨group₁⟩} {⟨group₂⟩} {⟨group₃⟩}</code>
		This functions will absorb three groups and leave the first and second in the input stream. The braces surrounding the arguments will be removed as part of this process. The category code of these tokens will also be fixed (if it has not already been by some other absorption). A single expansion is needed for the functions to take effect. An example:

`\use_i_ii:nnn { abc } { { def } } { ghi }`

will result in the input stream containing

`abc { def }`

i.e. the outer braces will be removed and the third group will be removed.

<code>\use_none:n</code>	★	<code>\use_none:n {⟨group₁⟩}</code>
<code>\use_none:(nn nnn nnnn nnnnn nnnnnn nnnnnnn nnnnnnnn nnnnnnnnn)</code>	★	

These functions absorb between one and nine groups from the input stream, leaving nothing on the resulting input stream. These functions work after a single expansion. One or more of the `n` arguments may be an unbraced single token (*i.e.* an `N` argument).

<code>\use:x</code>		<code>\use:x {⟨expandable tokens⟩}</code>
		Fully expands the <code>⟨expandable tokens⟩</code> and inserts the result into the input stream at the current location. Any hash characters (<code>#</code>) in the argument must be doubled.

9.1 Selecting tokens from delimited arguments

A different kind of function 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_nil:w ⟨balanced text⟩ \q_nil</code>
<code>\use_none_delimit_by_q_stop:w</code>	★	
<code>\use_none_delimit_by_q_recursion_stop:w</code>	★	

Absorb the `⟨balanced⟩` text form the input stream delimited by the marker given in the function name, leaving nothing in the input stream.

<code>\use_i_delimit_by_q_nil:nw</code>	★	<code>\use_i_delimit_by_q_nil:nw {\langle inserted tokens \rangle}</code>
<code>\use_i_delimit_by_q_stop:nw</code>	★	<code>\langle balanced text \rangle \q_nil</code>
<code>\use_i_delimit_by_q_recursion_stop:nw</code>	★	

Absorb the $\langle balanced \rangle$ text form the input stream delimited by the marker given in the function name, leaving $\langle inserted tokens \rangle$ in the input stream for further processing.

9.2 Decomposing control sequences

<code>\cs_get_arg_count_from_signature:N</code>	★	<code>\cs_get_arg_count_from_signature:N \langle function \rangle</code>
---	---	--

Splits the $\langle function \rangle$ into the name (*i.e.* the part before the colon) and the signature (*i.e.* after the colon). The $\langle number \rangle$ of tokens in the $\langle signature \rangle$ is then left in the input stream. If there was no $\langle signature \rangle$ then the result is the marker value -1 .

<code>\cs_get_function_name:N</code>	★	<code>\cs_get_function_name:N \langle function \rangle</code>
--------------------------------------	---	---

Splits the $\langle function \rangle$ into the name (*i.e.* the part before the colon) and the signature (*i.e.* after the colon). The $\langle name \rangle$ is then left in the input stream without the escape character present made up of tokens with category code 12 (other).

<code>\cs_get_function_signature:N</code>	★	<code>\cs_get_function_signature:N \langle function \rangle</code>
---	---	--

Splits the $\langle function \rangle$ into the name (*i.e.* the part before the colon) and the signature (*i.e.* after the colon). The $\langle signature \rangle$ is then left in the input stream made up of tokens with category code 12 (other).

<code>\cs_split_function:NN</code>	★	<code>\cs_split_function:NN \langle function \rangle \langle processor \rangle</code>
------------------------------------	---	---

Splits the $\langle function \rangle$ into the name (*i.e.* the part before the colon) and the signature (*i.e.* after the colon). This information is then placed in the input stream after the $\langle processor \rangle$ function in three parts: the $\langle name \rangle$, the $\langle signature \rangle$ and a logic token indicating if a colon was found (to differentiate variables from function names). The $\langle name \rangle$ will not include the escape character, and both the $\langle name \rangle$ and $\langle signature \rangle$ are made up of tokens with category code 12 (other). The $\langle processor \rangle$ should be a function with argument specification `:nnN` (plus any trailing arguments needed).

10 Predicates and conditionals

L^AT_EX3 has three concepts for conditional flow processing:

Branching conditionals 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_if_free:cTF{abc} {\langle true code \rangle} {\langle false code \rangle}`

a function that will turn the first argument into a control sequence (since it's 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).

These type of functions are known as “conditionals”; whenever a `TF` function is defined it will usually be accompanied by `T` and `F` functions as well. These are provided for convenience when the branch only needs to go a single way. Package writers are free to choose which types to define but the kernel definitions will always provide all three versions.

Important to note is that these branching 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.

These conditional functions may or may not be fully expandable, but if they are expandable they will be accompanied by a “predicate” for the same test as described below.

Predicates “Predicates” are functions that return a special type of boolean value which can be tested by the boolean expression parser. All functions of this type are expandable and have names that end with `_p` in the description part. For example,

```
\cs_if_free_p:N
```

would be a predicate function for the same type of test as the conditional described 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

```
\bool_if:nTF {
  \cs_if_free_p:N \l_tmpz_tl || \cs_if_free_p:N \g_tmpz_tl
} {\langle true code \rangle} {\langle false code \rangle}
```

For each predicate defined, a “branching conditional” will also exist that behaves like a conditional described above.

Primitive conditionals There is a third variety of conditional, which is the original concept used in plain `TeX` and `LATEX 2ε`. Their use is discouraged in `expl3` (although still used in low-level definitions) because they are more fragile and in many cases require more expansion control (hence more code) than the two types of conditionals described above.

```
\c_true_bool
\c_false_bool
```

Constants that represent `true` and `false`, respectively. Used to implement predicates.

10.1 Tests on control sequences

<code>\cs_if_eq_p:NN</code>	★	<code>\cs_if_eq_p:NN {<cs₁>} {<cs₂>}</code>
<code>\cs_if_eq:NNTF</code>	★	<code>\cs_if_eq:NNTF {<cs₁>} {<cs₂>} {<true code>} {<false code>}</code>

Compares the definition of two *<control sequences>* and is logically **true** if the two are the same.

<code>\cs_if_exist_p:N</code>	★	<code>\cs_if_exist_p:N <control sequence></code>
<code>\cs_if_exist_p:c</code>	★	<code>\cs_if_exist:NNTF <control sequence> {<true code>} {<false code>}</code>
<code>\cs_if_exist:NNTF</code>	★	Tests whether the <i><control sequence></i> is currently defined (whether as a function or another control sequence type). Any valid definition of <i><control sequence></i> will evaluate as true .
<code>\cs_if_exist:cTF</code>	★	

<code>\cs_if_free_p:N</code>	★	<code>\cs_if_free_p:N <control sequence></code>
<code>\cs_if_free_p:c</code>	★	<code>\cs_if_free:NNTF <control sequence> {<true code>} {<false code>}</code>
<code>\cs_if_free:NNTF</code>	★	Tests whether the <i><control sequence></i> is currently free to be defined. This test will be false if the <i><control sequence></i> currently exists (as defined by <code>\cs_if_exist:N</code>).
<code>\cs_if_free:cTF</code>	★	

10.2 Testing string equality

<code>\str_if_eq_p:nn</code>	★	<code>\str_if_eq_p:nn {<tl₁>} {<tl₂>}</code>
<code>\str_if_eq_p:(Vn on no nV VV xx)</code>	★	<code>\str_if_eq:nnTF {<tl₁>} {<tl₂>} {<true code>} {<false code>}</code>
<code>\str_if_eq:nnTF</code>	★	
<code>\str_if_eq:(Vn on no nV VV xx)TF</code>	★	

Compares the two *<token lists>* on a character by character basis, and is **true** if the two lists contain the same characters in the same order. Thus for example

`\str_if_eq_p:xx { abc } { \tl_to_str:n { abc } }`

is logically **true**. All versions of these functions are fully expandable (including those involving an **x**-type expansion).

10.3 Engine-specific conditionals

<code>\luatex_if_engine_p:</code>	★	<code>\luatex_if_luatex:TF {<true code>} {<false code>}</code>
<code>\luatex_if_engineTF</code>	★	Detects is the document is being compiled using LuaT _E X.

Updated: 2011-09-06

<code>\pdftex_if_engine_p:</code>	★	<code>\pdftex_if_engine:TF {<true code>} {<false code>}</code>
<code>\pdftex_if_engineTF</code>	★	Detects is the document is being compiled using pdfT _E X.

Updated: 2011-09-06

<code>\xetex_if_engine_p:</code>	★	<code>\xetex_if_engine:TF {<true code>} {<false code>}</code>
<code>\xetex_if_engine\overline{TF}:</code>	★	Detects if the document is being compiled using Xe _{La} TeX.

Updated: 2011-09-06

10.4 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_`.

<code>\if_true</code>	★	<code>\if_true: <true code> \else: <false code> \fi:</code>
<code>\if_false</code>	★	<code>\if_false: <true code> \else: <false code> \fi:</code>
<code>\or</code>	★	<code>\reverse_if:N <primitive conditional></code>
<code>\else</code>	★	<code>\if_true:</code> always executes <i><true code></i> , while <code>\if_false:</code> always executes <i><false code></i> .
<code>\fi</code>	★	<code>\reverse_if:N</code> reverses any two-way primitive conditional. <code>\else:</code> and <code>\fi:</code> delimit the branches of the conditional. <code>\or:</code> is used in case switches, see <code>\int</code> for more.
<code>\reverse_if:N</code>	★	

TeXhackers note: These are equivalent to their corresponding TeX primitive conditionals; `\reverse_if:N` is ϵ -TeX's `\unless`.

<code>\if_meaning:w</code>	★	<code>\if_meaning:w <arg₁> <arg₂> <true code> \else: <false code> \fi:</code>
----------------------------	---	---

`\if_meaning:w` executes *<true code>* when *<arg₁>* and *<arg₂>* are the same, otherwise it executes *<false code>*. *<arg₁>* and *<arg₂>* could be functions, variables, tokens; in all cases the *unexpanded* definitions are compared.

TeXhackers note: This is TeX's `\ifx`.

<code>\if:w</code>	★	<code>\if:w <token₁> <token₂> <true code> \else: <false code> \fi:</code>
<code>\if_charcode:w</code>	★	<code>\if_catcode:w <token₁> <token₂> <true code> \else: <false code> \fi:</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 <code>\exp_not:N</code> . <code>\if_catcode:w</code> tests if the category codes of the two tokens are the same whereas <code>\if:w</code> tests if the character codes are identical. <code>\if_charcode:w</code> is an alternative name for <code>\if:w</code> .

<code>\if_cs_exist:N</code>	★	<code>\if_cs_exist:N <cs> <true code> \else: <false code> \fi:</code>
<code>\if_cs_exist:w</code>	★	<code>\if_cs_exist:w <tokens> \cs_end: <true code> \else: <false code> \fi:</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_horizontal: <true code> \else: <false code> \fi:</code>
<code>\if_mode_vertical</code>	★	
<code>\if_mode_math</code>	★	Execute <i><true code></i> if currently in horizontal mode, otherwise execute <i><false code></i> . Similar for the other functions.
<code>\if_mode_inner</code>	★	

11 Internal kernel functions

<code>\chk_if_exist_cs:N</code>	<code>\chk_if_exist_cs:N <cs></code>
<code>\chk_if_exist_cs:c</code>	This function checks that <i><cs></i> exists according to the criteria for <code>\cs_if_exist_p:N</code> , and if not raises a kernel-level error.

<code>\chk_if_free_cs:N</code>	<code>\chk_if_free_cs:N <cs></code>
<code>\chk_if_free_cs:c</code>	This function checks that <i><cs></i> is free according to the criteria for <code>\cs_if_free_p:N</code> , and if not raises a kernel-level error.

12 Experimental functions

<code>\cs_if_exist_use:N</code>	★	<code>\cs_if_exist_use:N <control sequence></code>
<code>\cs_if_exist_use:c</code>	★	
<small>New: 2011-10-10</small> <code>\cs_if_exist_use:N</code>	★	<code>\cs_if_exist_use:N <control sequence></code>
<code>\cs_if_exist_use:c</code>	★	If the <i><control sequence></i> exists, leave it in the input stream, followed by the <i><true code></i> (unbraced). Otherwise, leave the <i><false></i> code in the input stream. For example,

```
\cs_set:Npn \mypkg_use_character:N #1
{ \cs_if_exist_use:cF { mypkg_#1:n } { \mypkg_default:N #1 } }
```

calls the function `\mypkg_#1:n` if it exists, and falls back to a default action otherwise. This could also be done (more slowly) using `\prg_case_str:xxn`.

T_EXhackers note: The `c` variants do not introduce the *<control sequence>* in the hash table if it is not there.

Part V

The l3expan package

Argument expansion

This module provides generic methods for expanding T_EX arguments in a systematic manner. The functions in this module all have prefix `exp`.

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.

13 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` was not defined the example above could be coded in the following way:

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

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.*

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

Providing variants in this way in style files is uncritical as the `\cs_new_nopar: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 steps above may be automated by using the function `\cs_generate_variant:Nn`, described next.

14 Methods for defining variants

`\cs_generate_variant:Nn`

Updated: 2011-09-15

`\cs_generate_variant:Nn` $\langle parent\ control\ sequence \rangle$ $\{ \langle variant\ argument\ specifiers \rangle \}$

This function is used to define argument-specifier variants of the $\langle parent\ control\ sequence \rangle$ for L^AT_EX3 code-level macros. The $\langle parent\ control\ sequence \rangle$ is first separated into the $\langle base\ name \rangle$ and $\langle original\ argument\ specifier \rangle$. The comma-separated list of $\langle variant\ argument\ specifiers \rangle$ is then used to define variants of the $\langle original\ argument\ specifier \rangle$ where these are not already defined. For each $\langle variant \rangle$ given, a function is created which will expand its arguments as detailed and pass them to the $\langle parent\ control\ sequence \rangle$. So for example

```
\cs_set:Npn \foo:Nn #1#2 { code here }
\cs_generate_variant:Nn \foo:Nn { c }
```

will create a new function `\foo:cn` which will expand its first argument into a control sequence name and pass the result to `\foo:Nn`. Similarly

```
\cs_generate_variant:Nn \foo:Nn { NV , cV }
```

would generate the functions `\foo:NV` and `\foo:cV` in the same way. The `\cs_generate_variant:Nn` function can only be applied if the $\langle parent\ control\ sequence \rangle$ is already defined. If the $\langle parent\ control\ sequence \rangle$ is protected then the new sequence will also be protected. The $\langle variant \rangle$ is created globally, as is any `\exp_args:N` $\langle variant \rangle$ function needed to carry out the expansion.

15 Introducing the variants

The available internal functions for argument expansion come in two flavours, some of them are faster than 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 expandably, *i.e.*, functions of this type will not work correctly in arguments that are itself subject to `x` expansion.
- In general, unless in the last position, multi-token arguments `n`, `f`, and `o` will need special processing which is not fast. Therefore it is best to use the optimized functions, namely those that contain only `N`, `c`, `V`, and `v`, and, in the last position, `o`, `f`, with possible trailing `N` or `n`, which are not expanded.

The `V` type returns the value of a register, which can be one of `tl`, `num`, `int`, `skip`, `dim`, `toks`, or built-in T_EX registers. The `v` type is the same except it first creates a

control sequence out of its argument before returning the value. This recent addition to the argument specifiers may shake things up a bit as most places where `o` is used will be replaced by `V`. The documentation you are currently reading will therefore require a fair bit of re-writing.

In general, the programmer should not need to be concerned with expansion control. When simply using the content of a variable, functions with a `V` specifier should be used. For those referred to by `(cs)name`, the `v` specifier is available for the same purpose. Only when specific expansion steps are needed, such as when using delimited arguments, should the lower-level functions with `o` specifiers be employed.

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_tl b` into a control sequence. Furthermore we want to store the execution of it in a *tl var*. In this example we assume `\l_tmpa_tl` contains the text string `lurb`. The straightforward approach is

```
\tl_set:Nc \l_tmpb_tl {\cs_set_eq:Nc \aaa { b \l_tmpa_tl b } }
```

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

```
\tl_set:Nf \l_tmpb_tl {\cs_set_eq:Nc \aaa { b \l_tmpa_tl b } }
```

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

```
\cs_set_nopar:Npn \tl_set:Nf { \exp_args:Nnf \tl_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!

16 Manipulating the first argument

These functions are described in detail: expansion of multiple tokens follows the same rules but is described in a shorter fashion.

```
\exp_args:No ★ \exp_args:No <function> {\<tokens>} {\<tokens_2>} ...
```

This function absorbs two arguments (the *<function>* name and the *<tokens>*). The *<tokens>* are expanded once, and the result is inserted in braces into the input stream *after* reinsertion of the *<function>*. Thus the *<function>* may take more than one argument: all others will be left unchanged.

```
\exp_args:Nc ★ \exp_args:Nc <function> {\<tokens>} {\<tokens_2>} ...
```

```
\exp_args:cc ★
```

This function absorbs two arguments (the *<function>* name and the *<tokens>*). The *<tokens>* are expanded until only characters remain, and are then turned into a control sequence. (An internal error will occur if such a conversion is not possible). The result is inserted into the input stream *after* reinsertion of the *<function>*. Thus the *<function>* may take more than one argument: all others will be left unchanged.

The `:cc` variant constructs the *<function>* name in the same manner as described for the *<tokens>*.

<hr/> <hr/>	<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:NV</code> ★	<code>\exp_args:NV</code>	<code><function> <variable> {\tokens_2} ...</code>
		This function absorbs two arguments (the names of the <code><function></code> and the <code><variable></code>). The content of the <code><variable></code> are recovered and placed inside braces into the input stream <i>after</i> reinsertion of the <code><function></code> . Thus the <code><function></code> may take more than one argument: all others will be left unchanged.
<hr/> <hr/>	<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:Nv</code> ★	<code>\exp_args:Nv</code>	<code><function> {\tokens} {\tokens_2} ...</code>
		This function absorbs two arguments (the <code><function></code> name and the <code><tokens></code>). The <code><tokens></code> are expanded until only characters remain, and are then turned into a control sequence. (An internal error will occur if such a conversion is not possible). This control sequence should be the name of a <code><variable></code> . The content of the <code><variable></code> are recovered and placed inside braces into the input stream <i>after</i> reinsertion of the <code><function></code> . Thus the <code><function></code> may take more than one argument: all others will be left unchanged.
<hr/> <hr/>	<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:Nf</code> ★	<code>\exp_args:Nf</code>	<code><function> {\tokens} {\tokens_2} ...</code>
		This function absorbs two arguments (the <code><function></code> name and the <code><tokens></code>). The <code><tokens></code> are fully expanded until the first non-expandable token or space is found, and the result is inserted in braces into the input stream <i>after</i> reinsertion of the <code><function></code> . Thus the <code><function></code> may take more than one argument: all others will be left unchanged.
<hr/> <hr/>	<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:Nx</code>	<code>\exp_args:Nx</code>	<code><function> {\tokens} {\tokens_2} ...</code>
		This function absorbs two arguments (the <code><function></code> name and the <code><tokens></code>) and exhaustively expands the <code><tokens></code> second. The result is inserted in braces into the input stream <i>after</i> reinsertion of the <code><function></code> . Thus the <code><function></code> may take more than one argument: all others will be left unchanged.

17 Manipulating two arguments

<hr/> <hr/>	<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:NNo</code> ★	<code>\exp_args:NNo</code>	<code><token1> <token2> {\tokens}</code>
<code>\exp_args:(Nnc NNv NNV NNf Nco Ncf Ncc NVV)</code> ★		
		These optimized functions absorb three arguments and expand the second and third as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second and third arguments.
<hr/> <hr/>	<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:Nno</code> ★	<code>\exp_args:Nno</code>	<code><token> {\tokens_1} {\tokens_2}</code>
<code>\exp_args:(NnV Nnf Noo Noc Nff Nfo Nnc)</code> ★		
		These functions absorb three arguments and expand the second and third as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second and third arguments. These functions need special (slower) processing.

<code>\exp_args:NNx</code>	<code>\exp_args:NNx <token1> <token2> {\tokens}</code>
<code>\exp_args:(Nnx Ncx Nox Nxo Nxx)</code>	

These functions absorb three arguments and expand the second and third as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second and third arguments. These functions are not expandable.

18 Manipulating three arguments

<code>\exp_args:NNNo</code>	★	<code>\exp_args:NNNo <token1> <token2> <token3> {\tokens}</code>
<code>\exp_args:(NNNV Nccc NcNc NcNo Ncco)</code>	★	

These optimized functions absorb four arguments and expand the second, third and fourth as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second argument, *etc.*

<code>\exp_args:NNoo</code>	★	<code>\exp_args:NNNo <token1> <token2> <token3> {\tokens}</code>
<code>\exp_args:(NNno Nnno Nnnc Nooo)</code>	★	

These functions absorb four arguments and expand the second, third and fourth as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second argument, *etc.* These functions need special (slower) processing.

<code>\exp_args:NNnx</code>	<code>\exp_args:NNnx <token1> <token2> {\tokens₁} {\tokens₂}</code>
<code>\exp_args:(NNox Nnnx Nnox Noox Ncnx Nccx)</code>	

These functions absorb four arguments and expand the second, third and fourth as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second argument, *etc.*

19 Unbraced expansion

<code>\exp_last_unbraced:Nf</code>	★	<code>\exp_last_unbraced:Nno <token></code>
<code>\exp_last_unbraced:(NV No Nv Nco NcV NNV NNo Nno Noo Nfo NNNV NNNo)</code>	★	<code><tokens1> <tokens2></code>

These functions absorb the number of arguments given by their specification, carry out the expansion indicated and leave the the results in the input stream, with the last argument not surrounded by the usual braces. Of these, the `:Nno`, `:Noo`, and `:Nfo` variants need special (slower) processing.

T_EXhackers note: As an optimization, the last argument is unbraced by some of those functions before expansion. This can cause problems if the argument is empty: for instance, `\exp_last_unbraced:Nf \mypkg_foo:w { } \q_stop` leads to an infinite loop, as the quark is f-expanded.

<code>\exp_last_unbraced:Nx</code>	<code>\exp_last_unbraced:Nx <function> {<tokens>}</code>
------------------------------------	--

This functions fully expands the `<tokens>` and leaves the result in the input stream after reinsertion of `<function>`. This function is not expandable.

<code>\exp_last_two_unbraced:Noo</code>	★	<code>\exp_last_two_unbraced:Noo <token> <tokens1> {<tokens2>}</code>
---	---	---

This function absorbs three arguments and expand the second and third once. The first argument of the function is then the next item on the input stream, followed by the expansion of the second and third arguments, which are not wrapped in braces. This function needs special (slower) processing.

<code>\exp_after:wN</code>	★	<code>\exp_after:wN <token1> <token2></code>
----------------------------	---	--

Carries out a single expansion of `<token2>` prior to expansion of `<token1>`. If `<token2>` is a T_EX primitive, it will be executed rather than expanded, while if `<token2>` has not expansion (for example, if it is a character) then it will be left unchanged. It is important to notice that `<token1>` may be *any* single token, including group-opening and -closing tokens (`{` or `}` assuming normal T_EX category codes). Unless specifically required, expansion should be carried out using an appropriate argument specifier variant or the appropriate `\exp_arg:N` function.

T_EXhackers note: This is the T_EX primitive `\expandafter` renamed.

20 Preventing expansion

Despite the fact that the following functions are all about preventing expansion, they're designed to be used in an expandable context and hence are all marked as being 'expandable' since they themselves will not appear after the expansion has completed.

<hr/> <hr/>	<code>\exp_not:N</code> ★	<code>\exp_not:N</code> $\langle token \rangle$
		Prevents expansion of the $\langle token \rangle$ in a context where it would otherwise be expanded, for example an x -type argument.
		T_EXhackers note: This is the T _E X <code>\noexpand</code> primitive.
<hr/> <hr/>	<code>\exp_not:c</code> ★	<code>\exp_not:c</code> $\{\langle tokens \rangle\}$
		Expands the $\langle tokens \rangle$ until only unexpandable content remains, and then converts this into a control sequence. Further expansion of this control sequence is then inhibited.
<hr/> <hr/>	<code>\exp_not:n</code> ★	<code>\exp_not:n</code> $\{\langle tokens \rangle\}$
		Prevents expansion of the $\langle tokens \rangle$ in a context where they would otherwise be expanded, for example an x -type argument.
		T_EXhackers note: This is the ε -T _E X <code>\unexpanded</code> primitive.
<hr/> <hr/>	<code>\exp_not:V</code> ★	<code>\exp_not:V</code> $\langle variable \rangle$
		Recovers the content of the $\langle variable \rangle$, then prevents expansion of the this material in a context where it would otherwise be expanded, for example an x -type argument.
<hr/> <hr/>	<code>\exp_not:v</code> ★	<code>\exp_not:v</code> $\{\langle tokens \rangle\}$
		Expands the $\langle tokens \rangle$ until only unexpandable content remains, and then converts this into a control sequence (which should be a $\langle variable \rangle$ name). The content of the $\langle variable \rangle$ is recovered, and further expansion is prevented in a context where it would otherwise be expanded, for example an x -type argument.
<hr/> <hr/>	<code>\exp_not:o</code> ★	<code>\exp_not:o</code> $\{\langle tokens \rangle\}$
		Expands the $\langle tokens \rangle$ once, then prevents any further expansion in a context where they would otherwise be expanded, for example an x -type argument.
<hr/> <hr/>	<code>\exp_not:f</code> ★	<code>\exp_not:f</code> $\{\langle tokens \rangle\}$
		Expands $\langle tokens \rangle$ fully until the first unexpandable token is found. Expansion then stops, and the result of the expansion (including any tokens which were not expanded) is protected from further expansion.
<hr/> <hr/>	<code>\exp_stop_f</code> ★	<code>\function:f</code> $\langle tokens \rangle$ <code>\exp_stop_f:</code> $\langle more\ tokens \rangle$
Updated: 2011-06-03		This function terminates an f -type expansion. Thus if a function <code>\function:f</code> starts an f -type expansion and all of $\langle tokens \rangle$ are expandable <code>\exp_stop:f</code> will terminate the expansion of tokens even if $\langle more\ tokens \rangle$ are also expandable. The function itself is an implicit space token. Inside an x -type expansion, it will retain its form, but when typeset it produces the underlying space (\sqcup).

21 Internal functions and variables

`\l_exp_tl`

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.

`\exp_eval_register:N` ★
`\exp_eval_register:c` ★

`\exp_eval_register:N` $\langle variable \rangle$

These functions evaluates a $\langle variable \rangle$ as part of a `V` or `v` expansion (respectively), preceded by `\c_zero` which stops the expansion of a previous `\romannumeral`. A $\langle variable \rangle$ might exist as one of two things: a parameter-less non-long, non-protected macro or a built-in `TEX` register such as `\count`.

`\::n` `\cs_set_nopar:Npn \exp_args:Ncof { \::c \::o \::f \:: }`

`\::N`

`\::c`

`\::o`

`\::f`

`\::x`

`\::v`

`\::V`

`\:::`

Internal forms for the base expansion types. These names do *not* conform to the general `LATEX3` approach as this makes them more readily visible in the log and so forth.

`\cs_generate_internal_variant:n` `\cs_generate_internal_variant:n` $\langle arg spec \rangle$

Tests if the function `\exp_args:N` $\langle arg spec \rangle$ exists, and defines it if it does not. The $\langle arg spec \rangle$ should be a series of one or more of the letters `N`, `c`, `n`, `o`, `V`, `v`, `f` and `x`.

Part VI

The l3prg package

Control structures

Conditional processing in L^AT_EX3 is defined as something that performs a series of tests, possibly involving assignments and calling other functions that do not read further ahead in the input stream. After processing the input, a *state* is returned. The typical states returned are *⟨true⟩* and *⟨false⟩* but other states are possible, say an *⟨error⟩* state for erroneous input, *e.g.*, text as input in a function comparing integers.

L^AT_EX3 has two primary forms of conditional flow processing based on these states. One type is predicate functions that turn the returned state into a boolean *⟨true⟩* or *⟨false⟩*. For example, the function `\cs_if_free_p:N` checks whether the control sequence given as its argument is free and then returns the boolean *⟨true⟩* or *⟨false⟩* values to be used in testing with `\if_predicate:w` or in functions to be described below. The other type is the kind of functions choosing a particular argument from the input stream based on the result of the testing as in `\cs_if_free:NTF` which also takes one argument (the N) and then executes either *⟨true⟩* or *⟨false⟩* depending on the result. Important to note here is that the arguments are executed after exiting the underlying `\if... \fi:` structure.

22 Defining a set of conditional functions

```
\prg_new_conditional:Npnn
\prg_new_conditional:Nnn
\prg_set_conditional:Npnn
\prg_set_conditional:Nnn
```

```
\prg_set_conditional:Npnn \<name>:<arg spec> <parameters> {<conditions>} {<code>}
\prg_set_conditional:Nnn \<name>:<arg spec> {<conditions>} {<code>}
```

These functions creates a family of conditionals using the same `{<code>}` to perform the test created. The **new** version will check for existing definitions (*cf.* `\cs_new:Npn`) whereas the **set** version will not (*cf.* `\cs_set:Npn`). The conditionals created are dependent on the comma-separated list of *⟨conditions⟩*, which should be one or more of **p**, **T**, **F** and **TF**.

The conditionals are defined by `\prg_new_conditional:Npnn` and friends as:

- `\<name>_p:<arg spec>` — a predicate function which will supply either a logical **true** or logical **false**. This function is intended for use in cases where one or more logical tests are combined to lead to a final outcome.
- `\<name>:<arg spec>T` — a function with one more argument than the original *⟨arg spec⟩* demands. The *⟨true branch⟩* code in this additional argument will be left on the input stream only if the test is **true**.
- `\<name>:<arg spec>F` — a function with one more argument than the original *⟨arg spec⟩* demands. The *⟨false branch⟩* code in this additional argument will be left on the input stream only if the test is **false**.
- `\<name>:<arg spec>TF` — a function with two more argument than the original *⟨arg spec⟩* demands. The *⟨true branch⟩* code in the first additional argument will

be left on the input stream if the test is `true`, while the *⟨false branch⟩* code in the second argument will be left on the input stream if the test is `false`.

The *⟨code⟩* of the test may use *⟨parameters⟩* as specified by the second argument to `\prg_set_conditional:Npnn`: this should match the *⟨argument specification⟩* but this is not enforced. The `Nnn` versions infer the number of arguments from the argument specification given (cf. `\cs_new:Nn`, etc.). Within the *⟨code⟩*, the functions `\prg_return_true:` and `\prg_return_false:` are used to indicate the logical outcomes of the test. If *⟨code⟩* is expandable then `\prg_set_conditional:Npnn` will generate a family of conditionals which are also expandable. All of the functions are created globally.

An example can easily clarify matters here:

```
\prg_set_conditional:Nnn \foo_if_bar:NN { p , T , TF }
{
  \if_meaning:w \l_tmpa_tl #1
  \prg_return_true:
\else:
  \if_meaning:w \l_tmpa_tl #2
  \prg_return_true:
\else:
  \prg_return_false:
\fi:
\fi:
}
```

This defines the function `\foo_if_bar_p:NN`, `\foo_if_bar:NNTF`, `\foo_if_bar:NNT` but not `\foo_if_bar:NNF` (because `F` is missing from the *⟨conds⟩* list). The return statements take care of resolving the remaining `\else:` and `\fi:` before returning the state. There must be a return statement for each branch, failing to do so will result in an error if that branch is executed.

<code>\prg_new_protected_conditional:Npnn</code>	<code>\prg_set_protected_conditional:Npnn</code>
<code>\prg_new_protected_conditional:Nnn</code>	<code>\⟨name⟩:⟨arg spec⟩ ⟨parameters⟩ {⟨conditions⟩} {⟨code⟩}</code>
<code>\prg_set_protected_conditional:Npnn</code>	<code>\prg_set_protected_conditional:Nnn</code>
<code>\prg_set_protected_conditional:Nnn</code>	<code>\⟨name⟩:⟨arg spec⟩ {⟨conditions⟩} {⟨code⟩}</code>

These functions creates a family of conditionals using the same *⟨code⟩* to perform the test created. The `new` version will check for existing definitions (cf. `\cs_new:Npn`) whereas the `set` version will not (cf. `\cs_set:Npn`). The conditionals created are depended on the comma-separated list of *⟨conditions⟩*, which should be one or more of `T`, `F` and `TF`.

The conditionals are defined by `\prg_new_protected_conditional:Npnn` and friends as:

- `\⟨name⟩:⟨arg spec⟩T` — a function with one more argument than the original *⟨arg spec⟩* demands. The *⟨true branch⟩* code in this additional argument will be left on the input stream only if the test is `true`.

- `\<name>:<arg spec>F` — a function with one more argument than the original `<arg spec>` demands. The `<false branch>` code in this additional argument will be left on the input stream only if the test is `false`.
- `\<name>:<arg spec>TF` — a function with two more argument than the original `<arg spec>` demands. The `<true branch>` code in the first additional argument will be left on the input stream if the test is `true`, while the `<false branch>` code in the second argument will be left on the input stream if the test is `false`.

The `<code>` of the test may use `<parameters>` as specified by the second argument to `\prg_set_conditional:Npn`: this should match the `<argument specification>` but this is not enforced. The `Nnn` versions infer the number of arguments from the argument specification given (cf. `\cs_new:Nn`, etc.). Within the `<code>`, the functions `\prg_return_true:` and `\prg_return_false:` are used to indicate the logical outcomes of the test. `\prg_set_protected_conditional:Npn` will generate a family of protected conditional functions, and so `<code>` does not need to be expandable. All of the functions are created globally.

<code>\prg_new_eq_conditional:Nnn</code>	<code>\prg_new_eq_conditional:Nnn \<name1>:<arg spec1> \<name2>:<arg spec2></code>
<code>\prg_set_eq_conditional:Nnn</code>	<code>{<conditions>}</code>

These functions copies a family of conditionals. The `new` version will check for existing definitions (cf. `\cs_new:Npn`) whereas the `set` version will not (cf. `\cs_set:Npn`). The conditionals copied are depended on the comma-separated list of `<conditions>`, which should be one or more of `p`, `T`, `F` and `TF`.

<code>\prg_return_true</code> ★	<code>\prg_return_true:</code>
<code>\prg_return_false</code> ★	<code>\prg_return_false:</code>

These functions define the logical state at the end of a conditional. As such, they should appear within the code for a conditional statement generated by `\prg_set_conditional:Npnn`, etc.

23 The boolean data type

This section describes a boolean data type which is closely connected to conditional processing 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_predicate:w` test. The problem of the primitive `\if_false:` and `\if_true:` tokens is that it is not always safe to pass them around as they may interfere with scanning for termination of primitive conditional processing. Therefore, we employ two canonical booleans: `\c_true_bool` or `\c_false_bool`. Besides preventing problems as described above, it also allows us to implement a simple boolean parser supporting the logical operations And, Or, Not, etc. which can then be used on both the boolean type and predicate functions.

All conditional `\bool_` functions are expandable and expect the input to also be fully expandable (which will generally mean being constructed from predicate functions, possibly nested).

<hr/> <hr/>	<hr/>
<code>\bool_new:N</code>	<code>\bool_new:N <boolean></code>
<code>\bool_new:c</code>	Creates a new <i><boolean></i> or raises an error if the name is already taken. The declaration is global. The <i><boolean></i> will initially be false .
<hr/> <hr/>	
<code>\bool_set_false:N</code>	<code>\bool_set_false:N <boolean></code>
<code>\bool_set_false:c</code>	Sets <i><boolean></i> logically false within the current TeX group.
<hr/> <hr/>	
<code>\bool_gset_false:N</code>	<code>\bool_gset_false:N <boolean></code>
<code>\bool_gset_false:c</code>	Sets <i><boolean></i> logically false globally.
<hr/> <hr/>	
<code>\bool_set_true:N</code>	<code>\bool_set_true:N <boolean></code>
<code>\bool_set_true:c</code>	Sets <i><boolean></i> logically true within the current TeX group.
<hr/> <hr/>	
<code>\bool_gset_true:N</code>	<code>\bool_gset_true:N <boolean></code>
<code>\bool_gset_true:c</code>	Sets <i><boolean></i> logically true globally.
<hr/> <hr/>	
<code>\bool_set_eq:NN</code>	<code>\bool_set_eq:NN <boolean1> <boolean2></code>
<code>\bool_set_eq:(cN Nc cc)</code>	Sets the content of <i><boolean1></i> equal to that of <i><boolean2></i> . This assignment is restricted to the current TeX group level.
<hr/> <hr/>	
<code>\bool_gset_eq:NN</code>	<code>\bool_gset_eq:NN <boolean1> <boolean2></code>
<code>\bool_gset_eq:(cN Nc cc)</code>	Sets the content of <i><boolean1></i> equal to that of <i><boolean2></i> . This assignment is global and so is not limited by the current TeX group level.
<hr/> <hr/>	
<code>\bool_set:Nn</code>	<code>\bool_set:Nn <boolean> {<boolexpr>}</code>
<code>\bool_set:cn</code>	Evaluates the <i><boolean expression></i> as described for <code>\bool_if:n(TF)</code> , and sets the <i><boolean></i> variable to the logical truth of this evaluation. This assignment is local.
<hr/> <hr/>	
<code>\bool_gset:Nn</code>	<code>\bool_gset:Nn <boolean> {<boolexpr>}</code>
<code>\bool_gset:cn</code>	Evaluates the <i><boolean expression></i> as described for <code>\bool_if:n(TF)</code> , and sets the <i><boolean></i> variable to the logical truth of this evaluation. This assignment is global.
<hr/> <hr/>	
<code>\bool_if_p:N</code> ★	<code>\bool_if_p:N {<boolean>}</code>
<code>\bool_if_p:c</code> ★	<code>\bool_if:NTF {<boolean>} {<true code>} {<false code>}</code>
<code>\bool_if:NTF</code> ★	Tests the current truth of <i><boolean></i> , and continues expansion based on this result.
<code>\bool_if:cTF</code> ★	
<hr/> <hr/>	
<code>\l_tmpa_bool</code>	A scratch boolean for local assignment. It is never used by the kernel code, and so is safe for use with any L ^A T _E X3-defined function. However, it may be overwritten by other non-kernel code and so should only be used for short-term storage.

`\g_tmpa_bool`

A scratch boolean for global assignment. It is never used by the kernel code, and so is safe for use with any L^AT_EX3-defined function. However, it may be overwritten by other non-kernel code and so should only be used for short-term storage.

24 Boolean expressions

As we have a boolean datatype and predicate functions returning boolean $\langle true \rangle$ or $\langle false \rangle$ values, it seems only fitting that we also provide a parser for $\langle boolean\ expressions \rangle$.

A boolean expression is an expression which given input in the form of predicate functions and boolean variables, return boolean $\langle true \rangle$ or $\langle false \rangle$. It supports the logical operations And, Or and Not as the well-known infix operators `&&`, `||` and `!`. In addition to this, parentheses can be used to isolate sub-expressions. For example,

```
\int_compare_p:n { 1 = 1 } &&
(
  \int_compare_p:n { 2 = 3 } ||
  \int_compare_p:n { 4 = 4 } ||
  \int_compare_p:n { 1 = \error } % is skipped
) &&
! ( \int_compare_p:n { 2 = 4 } )
```

is a valid boolean expression. Note that minimal evaluation is carried out whenever possible so that whenever a truth value cannot be changed any more, the remaining tests within the current group are skipped.

<code>\bool_if_p:n</code> ☆	<code>\bool_if_p:n {<boolean expression>}</code>
<code>\bool_if:nTF</code> ☆	<code>\bool_if:nTF {<boolean expression>} {<true code>} {<false code>}</code>

Tests the current truth of *<boolean expression>*, and continues expansion based on this result. The *<boolean expression>* should consist of a series of predicates or boolean variables with the logical relationship between these defined using `&&` (“And”), `||` (“Or”), `!` (“Not”) and parentheses. Minimal evaluation is used in the processing, so that once a result is defined there is not further expansion of the tests. For example

```

\bool_if_p:n
{
  \int_compare_p:nNn { 1 } = { 1 }
  &&
  (
    \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 } = { 4 } )
}

```

will be **true** and will not evaluate `\int_compare_p:nNn { 1 } = { \error }`. The logical Not applies to the next single predicate or group. As shown above, this means that any predicates requiring an argument have to be given within parentheses.

<code>\bool_not_p:n</code> ☆	<code>\bool_not_p:n {<boolean expression>}</code>
------------------------------	---

Function version of `!(<boolean expression>)` within a boolean expression.

<code>\bool_xor_p:nn</code> ☆	<code>\bool_xor_p:nn {<boolexpr₁>} {<boolexpr₁>}</code>
-------------------------------	---

Implements an “exclusive or” operation between two boolean expressions. There is no infix operation for this logical operator.

25 Logical loops

Loops using either boolean expressions or stored boolean values.

<code>\bool_until_do:Nn</code> ☆	<code>\bool_until_do:Nn {<boolean>} {<code>}</code>
<code>\bool_until_do:cn</code> ☆	

This function firsts checks the logical value of the *<boolean>*. If it is **false** the *<code>* is placed in the input stream and expanded. After the completion of the *<code>* the truth of the *<boolean>* is re-evaluated. The process will then loop until the *<boolean>* is **true**.

<code>\bool_while_do:Nn</code> ☆	<code>\bool_while_do:Nn {<boolean>} {<code>}</code>
<code>\bool_while_do:cn</code> ☆	

This function firsts checks the logical value of the *<boolean>*. If it is **true** the *<code>* is placed in the input stream and expanded. After the completion of the *<code>* the truth of the *<boolean>* is re-evaluated. The process will then loop until the *<boolean>* is **false**.

`\bool_until_do:nn` ☆ `\bool_until_do:nn {\boolean expression} {\code}`

This function firsts checks the logical value of the *\boolean expression* (as described for `\bool_if:nTF`). If it is `false` the *\code* is placed in the input stream and expanded. After the completion of the *\code* the truth of the *\boolean expression* is re-evaluated. The process will then loop until the *\boolean expression* is `true`.

`\bool_while_do:nn` ☆ `\bool_while_do:nn {\boolean expression} {\code}`

This function firsts checks the logical value of the *\boolean expression* (as described for `\bool_if:nTF`). If it is `true` the *\code* is placed in the input stream and expanded. After the completion of the *\code* the truth of the *\boolean expression* is re-evaluated. The process will then loop until the *\boolean expression* is `false`.

26 Switching by case

For cases where a number of cases need to be considered a family of case-selecting functions are available.

`\prg_case_int:nnn` ☆

Updated: 2011-09-17

`\prg_case_int:nnn {\test integer expression}`
`{`
 `{\intexpr case1} {\code case1}`
 `{\intexpr case2} {\code case2}`
 `...`
 `{\intexpr casen} {\code casen}`
`}`
`{\else case}`

This function evaluates the *\test integer expression* and compares this in turn to each of the *\integer expression cases*. If the two are equal then the associated *\code* is left in the input stream. If none of the tests are `true` then the `else` code will be left in the input stream.

As an example of `\prg_case_int:nnn`:

```
\prg_case_int:nnn
{ 2 * 5 }
{
  { 5 }      { Small }
  { 4 + 6 }  { Medium }
  { -2 * 10 } { Negative }
}
{ No idea! }
```

will leave “Medium” in the input stream.

<code>\prg_case_dim:nnn</code> ★ <hr/> Updated: 2011-07-06 <hr/>	<pre> \prg_case_dim:nnn {<test dimension expression>} { {<dimexpr case₁>} {<code case₁>} {<dimexpr case₂>} {<code case₂>} ... {<dimexpr case_n>} {<code case_n>} } {<else case>} </pre>
--	---

This function evaluates the *<test dimension expression>* and compares this in turn to each of the *<dimension expression cases>*. If the two are equal then the associated *<code>* is left in the input stream. If none of the tests are **true** then the **else** code will be left in the input stream.

<code>\prg_case_str:nnn</code> ★ <code>\prg_case_str:(onn xxn)</code> ★ <hr/> Updated: 2011-09-17 <hr/>	<pre> \prg_case_str:nnn {<test string>} { {<string case₁>} {<code case₁>} {<string case₂>} {<code case₂>} ... {<string case_n>} {<code case_n>} } {<else case>} </pre>
--	--

This function compares the *<test string>* in turn with each of the *<string cases>*. If the two are equal (as described for `\str_if_eq:nnTF` then the associated *<code>* is left in the input stream. If none of the tests are **true** then the **else** code will be left in the input stream. The **xx** variant is fully expandable, in the same way as the underlying `\str_if_eq:xxTF` test.

<code>\prg_case_tl:Nnn</code> ★ <code>\prg_case_tl:cnn</code> ★ <hr/> Updated: 2011-09-17 <hr/>	<pre> \prg_case_tl:Nnn <test token list variable> { <token list variable case₁> {<code case₁>} <token list variable case₂> {<code case₂>} ... <token list variable case_n> {<code case_n>} } {<else case>} </pre>
--	---

This function compares the *<test token list variable>* in turn with each of the *<token list variable cases>*. If the two are equal (as described for `\tl_if_eq:nnTF` then the associated *<code>* is left in the input stream. If none of the tests are **true** then the **else** code will be left in the input stream.

27 Producing n copies

<code>\prg_replicate:nn</code> ★ <hr/> Updated: 2011-07-04 <hr/>	<pre> \prg_replicate:nn {<integer expression>} {<tokens>} </pre>
--	--

Evaluates the *<integer expression>* (which should be zero or positive) and creates the resulting number of copies of the *<tokens>*. The function is both expandable and safe for nesting. It yields its result after two expansion steps.

<code>\prg_stepwise_function:nnnN</code> ★	<code>\prg_stepwise_function:nnnN {<initial value>} {<step>} {<final value>} {<function>}</code>
--	--

Updated: 2011-09-06

This function first evaluates the *<initial value>*, *<step>* and *<final value>*, all of which should be integer expressions. The *<function>* is then placed in front of each *<value>* from the *<initial value>* to the *<final value>* in turn (using *<step>* between each *<value>*). Thus *<function>* should absorb one numerical argument. For example

```
\cs_set_nopar:Npn \my_func:n #1 { [I~saw~#1] \quad }
\prg_stepwise_function:nnnN { 1 } { 5 } { 1 } \my_func:n
```

would print

```
[I saw 1]   [I saw 2]   [I saw 3]   [I saw 4]   [I saw 5]
```

<code>\prg_stepwise_inline:nnnn</code>	<code>\prg_stepwise_inline:nnnn {<initial value>} {<step>} {<final value>} {<code>}</code>
--	--

Updated: 2011-09-06

This function first evaluates the *<initial value>*, *<step>* and *<final value>*, all of which should be integer expressions. The *<code>* is then placed in front of each *<value>* from the *<initial value>* to the *<final value>* in turn (using *<step>* between each *<value>*). Thus the *<code>* should define a function of one argument (*#1*).

<code>\prg_stepwise_variable:nnnNn</code>	<code>\prg_stepwise_variable:nnnNn {<initial value>} {<step>} {<final value>} <tl var> {<code>}</code>
---	--

Updated: 2011-09-06

This function first evaluates the *<initial value>*, *<step>* and *<final value>*, all of which should be integer expressions. The *<code>* is inserted into the input stream, with the *<tl var>* defined as the current *<value>*. Thus the *<code>* should make use of the *<tl var>*.

28 Detecting T_EX's mode

<code>\mode_if_horizontal_p:</code> ★	<code>\mode_if_horizontal_p:</code>
<code>\mode_if_horizontalTF</code> ★	<code>\mode_if_horizontal:TF {<true code>} {<false code>}</code>

Detects if T_EX is currently in horizontal mode.

<code>\mode_if_inner_p:</code> ★	<code>\mode_if_inner_p:</code>
<code>\mode_if_innerTF</code> ★	<code>\mode_if_inner:TF {<true code>} {<false code>}</code>

Detects if T_EX is currently in inner mode.

<code>\mode_if_math_p:</code> ★	<code>\mode_if_math:TF {<true code>} {<false code>}</code>
<code>\mode_if_mathTF</code> ★	

Detects if T_EX is currently in maths mode.

Updated: 2011-09-05

<code>\mode_if_vertical_p:</code>	★	<code>\mode_if_vertical_p:</code>
<code>\mode_if_vertical\underline{TF}:</code>	★	<code>\mode_if_vertical:TF {\langle true code \rangle} {\langle false code \rangle}</code>

Detects if $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ is currently in vertical mode.

29 Internal programming functions

<code>\group_align_safe_begin</code>	★	<code>\group_align_safe_begin:</code>
<code>\group_align_safe_end</code>	★	<code>...</code>
		<code>\group_align_safe_end:</code>

Updated: 2011-08-11

These functions are used to enclose material in a $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ alignment environment within a specially-constructed group. This group is designed in such a way that it does not add brace groups to the output but does act as a group for the `&` token inside `\halign`. This is necessary to allow grabbing of tokens for testing purposes, as $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ uses group level to determine the effect of alignment tokens. Without the special grouping, the use of a function such as `\peek_after:Nw` will result in a forbidden comparison of the internal `\endtemplate` token, yielding a fatal error. Each `\group_align_safe_begin:` must be matched by a `\group_align_safe_end:`, although this does not have to occur within the same function.

<code>\scan_align_safe_stop</code>		<code>\scan_align_safe_stop:</code>
------------------------------------	--	-------------------------------------

Updated: 2011-09-06

Stops $\mathrm{T}_{\mathrm{E}}\mathrm{X}$'s scanner looking for expandable control sequences at the beginning of an alignment cell. This function is required, for example, to obtain the expected output when testing `\mode_if_math:TF` at the start of a math array cell: placing `\scan_align_safe_stop:` before `\mode_if_math:TF` will give the correct result. This function does not destroy any kerning if used in other locations, but *does* render functions non-expandable.

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ hackers note: This is a protected version of `\prg_do_nothing:`, which therefore stops $\mathrm{T}_{\mathrm{E}}\mathrm{X}$'s scanner in the circumstances described without producing any affect on the output.

<code>\prg_variable_get_scope:N</code>	★	<code>\prg_variable_get_scope:N \langle variable \rangle</code>
--	---	---

Returns the scope (`g` for global, blank otherwise) for the `\langle variable \rangle`.

<code>\prg_variable_get_type:N</code>	★	<code>\prg_variable_get_type:N \langle variable \rangle</code>
---------------------------------------	---	--

Returns the type of `\langle variable \rangle` (`tl`, `int`, *etc.*)

<code>\if_predicate:w</code>	★	<code>\if_predicate:w \langle predicate \rangle \langle true code \rangle \else: \langle false code \rangle \fi:</code>
------------------------------	---	---

This function takes a predicate function and branches according to the result. (In practice this function would also accept a single boolean variable in place of the `\langle predicate \rangle` but to make the coding clearer this should be done through `\if_bool:N`.)

```
\if_bool:N ★ \if_bool:N <boolean> <true code> \else: <false code> \fi:
```

This function takes a boolean variable and branches according to the result.

```
\prg_break_point:n ★ \prg_break_point:n <tokens>
```

Used to mark the end of a recursion or mapping: the functions `\prg_map_break:` and `\prg_map_break:n` use this to break out of the loop. After the loop ends, the *<tokens>* are inserted into the input stream. This occurs even if the the break functions are *not* applied: `\prg_break_point:n` is functionally-equivalent in these cases to `\use:n`.

```
\prg_map_break:n ★ \prg_map_break:n {<user code>}
...
\prg_break_point:n {<ending code>}
```

Breaks a recursion in mapping contexts, inserting in the input stream the *<user code>* after the *<ending code>* for the loop.

30 Experimental programmings functions

```
\prg_quicksort:n \prg_quicksort:n { {<item1>} {<item2>} ... {<itemn>} }
```

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 items are given as argument to the function `\prg_quicksort_function:n` which the programmer also controls.

```
\prg_quicksort_function:n \prg_quicksort_function:n {<element>}
\prg_quicksort_compare:nnTF \prg_quicksort_compare:nnTF {<element1>} {<element2>}
```

The two functions the programmer must define before calling `\prg_quicksort:n`. As an example we could define

```
\cs_set_nopar:Npn\prg_quicksort_function:n #1{{#1}}
\cs_set_nopar:Npn\prg_quicksort_compare:nnTF #1#2 {\int_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

```
\cs_set_nopar:Npn\prg_quicksort_compare:nnTF #1#2 {
  \int_compare:nNnTF{\tl_compare:nn{#1}{#2}}>\c_zero }
```

Part VII

The l3quark package

Quarks

Two special types of constants in L^AT_EX3 are “quarks” and “scan marks”. By convention all constants of type quark start out with `\q_`, and scan marks start with `\s_`.

Scan marks are an experimental feature.

Quarks 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 in 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:w`. 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.

Scan marks are control sequences set equal to `\scan_stop:`, hence will never expand.

Since they are harmless when executed by T_EX in non-expandable contexts, they can be used to mark the end of a set of instructions. This allows to skip to that point if the end of the instructions should not be performed (see l3regex).

31 Defining quarks

`\quark_new:N`

`\quark_new:N <quark>`

Creates a new `<quark>` which expands only to `<quark>`. The `<quark>` will be defined globally, and an error message will be raised if the name was already taken.

`\q_stop`

Used as a marker for delimited arguments, such as

`\cs_set:Npn \tmp:w #1#2 \q_stop {#1}`

`\q_mark`

Used as a marker for delimited arguments when `\q_stop` is already in use.

Quark to mark a null value in structured variables or functions. Used as an end delimiter when this may itself may need to be tested (in contrast to `\q_stop`, which is only ever used as a delimiter).

`\q_no_value`

A canonical value for a missing value, when one is requested from a data structure. This is therefore used as a “return” value by functions such as `\prop_get:NnN` if there is no data to return.

32 Quark tests

The method used to define quarks means that the single token (N) tests are faster than the multi-token (n) tests. The later should therefore only be used when the argument can definitely take more than a single token.

<code>\quark_if_nil_p:N</code> *	<code>\quark_if_nil_p:N <token></code>
<code>\quark_if_nil:N\underline{TF}</code> *	<code>\quark_if_nil:N\underline{TF} <token> {\true code} {\false code}</code>

Tests if the $\langle token \rangle$ is equal to `\q_nil`.

<code>\quark_if_nil_p:n</code> *	<code>\quark_if_nil_p:n {\token list}</code>
<code>\quark_if_nil_p:(o V)</code> *	<code>\quark_if_nil:N\underline{TF} {\token list} {\true code} {\false code}</code>
<code>\quark_if_nil:n\underline{TF}</code> *	Tests if the $\langle token list \rangle$ contains only <code>\q_nil</code> (distinct from $\langle token list \rangle$ being empty or containing <code>\q_nil</code> plus one or more other tokens).
<code>\quark_if_nil:(o V)\underline{TF}</code> *	

<code>\quark_if_no_value_p:N</code> *	<code>\quark_if_no_value_p:N <token></code>
<code>\quark_if_no_value_p:c</code> *	<code>\quark_if_no_value:N\underline{TF} <token> {\true code} {\false code}</code>
<code>\quark_if_no_value:N\underline{TF}</code> *	Tests if the $\langle token \rangle$ is equal to <code>\q_no_value</code> .
<code>\quark_if_no_value:c\underline{TF}</code> *	

<code>\quark_if_no_value_p:n</code> *	<code>\quark_if_no_value_p:n {\token list}</code>
<code>\quark_if_no_value:n\underline{TF}</code> *	<code>\quark_if_no_value:N\underline{TF} {\token list} {\true code} {\false code}</code>

Tests if the $\langle token list \rangle$ contains only `\q_no_value` (distinct from $\langle token list \rangle$ being empty or containing `\q_no_value` plus one or more other tokens).

33 Recursion

This module provides a uniform interface to intercepting and terminating loops as when one is doing tail recursion. The building blocks follow below.

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 <token></code>
--	--

Tests if $\langle token \rangle$ contains only the marker `\q_recursion_tail`, and if so terminates the recursion this is part of using `\use_none_delimit_by_q_recursion_stop:w`. The recursion input must include the marker tokens `\q_recursion_tail` and `\q_recursion_stop` as the last two items.

```
\quark_if_recursion_tail_stop:n \quark_if_recursion_tail_stop:n {\token list}
\quark_if_recursion_tail_stop:o
```

Updated: 2011-09-06

Tests if the $\langle token list \rangle$ contains only `\q_recursion_tail`, and if so terminates the recursion this is part of using `\use_none_delimit_by_q_recursion_stop:w`. The recursion input must include the marker tokens `\q_recursion_tail` and `\q_recursion_stop` as the last two items.

```
\quark_if_recursion_tail_stop_do:Nn \quark_if_recursion_tail_stop_do:Nn \token {\insertion}
```

Tests if $\langle token \rangle$ contains only the marker `\q_recursion_tail`, and if so terminates the recursion this is part of using `\use_none_delimit_by_q_recursion_stop:w`. The recursion input must include the marker tokens `\q_recursion_tail` and `\q_recursion_stop` as the last two items. The $\langle insertion \rangle$ code is then added to the input stream after the recursion has ended.

```
\quark_if_recursion_tail_stop_do:nn \quark_if_recursion_tail_stop_do:nn {\token list} {\insertion}
\quark_if_recursion_tail_stop_do:on
```

Updated: 2011-09-06

Tests if the $\langle token list \rangle$ contains only `\q_recursion_tail`, and if so terminates the recursion this is part of using `\use_none_delimit_by_q_recursion_stop:w`. The recursion input must include the marker tokens `\q_recursion_tail` and `\q_recursion_stop` as the last two items. The $\langle insertion \rangle$ code is then added to the input stream after the recursion has ended.

```
\quark_if_recursion_tail_break:N \quark_if_recursion_tail_break:n {\token list}
\quark_if_recursion_tail_break:n
```

Tests if $\langle token list \rangle$ contains only `\q_recursion_tail`, and if so terminates the recursion using `\prg_map_break:.` The recursion end should be marked by `\prg_break_point:n`.

34 Scan marks

```
\scan_new:N \scan_new:N \scan mark
```

Creates a new $\langle scan mark \rangle$ which is set equal to `\scan_stop:.` The $\langle scan mark \rangle$ will be defined globally, and an error message will be raised if the name was already taken by another scan mark.

```
\s_stop \s_stop Used at the end of a set of instructions, as a marker that can be jumped to using \use_
none_delimit_by_s_stop:w.
```

`\use_none_delimit_by_s_stop:w`

`\cs`

 `use_none_delimit_by_s_stop:w` $\langle tokens \rangle$ `\s_stop`
Removes the $\langle tokens \rangle$ and `\s_stop` from the input stream. This leads to a low-level TeX error if `\s_stop` is absent.

35 Internal quark functions

`\use_none_delimit_by_q_recursion_stop:w` `\use_none_delimit_by_q_recursion_stop:w` $\langle tokens \rangle$
`\q_recursion_stop`

Used to prematurely terminate a recursion using `\q_recursion_stop` as the end marker, removing any remaining $\langle tokens \rangle$ from the input stream.

`\use_i_delimit_by_q_recursion_stop:nw` `\use_i_delimit_by_q_recursion_stop:nw` $\{\langle insertion \rangle\}$
 $\langle tokens \rangle$ `\q_recursion_stop`

Used to prematurely terminate a recursion using `\q_recursion_stop` as the end marker, removing any remaining $\langle tokens \rangle$ from the input stream. The $\langle insertion \rangle$ is then made into the input stream after the end of the recursion.

Part VIII

The l3token package

Token manipulation

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 T_EX, 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: `\token` for anything that deals with tokens and `\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 variable” `tl var`. Functions for these two types are found in the `l3tl` module.

36 All possible tokens

Let us start by reviewing every case that a given token can fall into. It is very important to distinguish two aspects of a token: its meaning, and what it looks like.

For instance, `\if:w`, `\if_charcode:w`, and `\tex_if:D` are three for the same internal operation of T_EX, namely the primitive testing the next two characters for equality of their character code. They behave identically in many situations. However, T_EX distinguishes them when searching for a delimited argument. Namely, the example function `\show_until_if:w` defined below will take everything until `\if:w` as an argument, despite the presence of other copies of `\if:w` under different names.

```
\cs_new:Npn \show_until_if:w #1 \if:w { \tl_show:n {#1} }  
\show_until_if:w \tex_if:D \if_charcode:w \if:w
```

37 Character tokens

<code>\char_set_catcode_escape:N</code>	<code>\char_set_catcode_letter:N</code> $\langle character \rangle$
<code>\char_set_catcode_group_begin:N</code>	
<code>\char_set_catcode_group_end:N</code>	
<code>\char_set_catcode_math_toggle:N</code>	
<code>\char_set_catcode_alignment:N</code>	
<code>\char_set_catcode_end_line:N</code>	
<code>\char_set_catcode_parameter:N</code>	
<code>\char_set_catcode_math_superscript:N</code>	
<code>\char_set_catcode_math_subscript:N</code>	
<code>\char_set_catcode_ignore:N</code>	
<code>\char_set_catcode_space:N</code>	
<code>\char_set_catcode_letter:N</code>	
<code>\char_set_catcode_other:N</code>	
<code>\char_set_catcode_active:N</code>	
<code>\char_set_catcode_comment:N</code>	
<code>\char_set_catcode_invalid:N</code>	

Sets the category code of the $\langle character \rangle$ to that indicated in the function name. Depending on the current category code of the $\langle token \rangle$ the escape token may also be needed:

`\char_set_catcode_other:N \%`

The assignment is local.

<code>\char_set_catcode_escape:n</code>	<code>\char_set_catcode_letter:n</code> $\{ \langle integer\ expression \rangle \}$
<code>\char_set_catcode_group_begin:n</code>	
<code>\char_set_catcode_group_end:n</code>	
<code>\char_set_catcode_math_toggle:n</code>	
<code>\char_set_catcode_alignment:n</code>	
<code>\char_set_catcode_end_line:n</code>	
<code>\char_set_catcode_parameter:n</code>	
<code>\char_set_catcode_math_superscript:n</code>	
<code>\char_set_catcode_math_subscript:n</code>	
<code>\char_set_catcode_ignore:n</code>	
<code>\char_set_catcode_space:n</code>	
<code>\char_set_catcode_letter:n</code>	
<code>\char_set_catcode_other:n</code>	
<code>\char_set_catcode_active:n</code>	
<code>\char_set_catcode_comment:n</code>	
<code>\char_set_catcode_invalid:n</code>	

Sets the category code of the $\langle character \rangle$ which has character code as given by the $\langle integer\ expression \rangle$. This version can be used to set up characters which cannot otherwise be given (*cf.* the N-type variants). The assignment is local.

<hr/> <hr/> <code>\char_set_catcode:nn</code>	<code>\char_set_catcode:nn {⟨integer₁⟩} {⟨integer₂⟩}</code>
	These functions set the category code of the <i>⟨character⟩</i> which has character code as given by the <i>⟨integer expression⟩</i> . The first <i>⟨integer expression⟩</i> is the character code and the second is the category code to apply. The setting applies within the current T _E X group. In general, the symbolic functions <code>\char_set_catcode_⟨type⟩</code> should be preferred, but there are cases where these lower-level functions may be useful.
<hr/> <hr/> <code>\char_value_catcode:n</code> ★	<code>\char_value_catcode:n {⟨integer expression⟩}</code>
	Expands to the current category code of the <i>⟨character⟩</i> with character code given by the <i>⟨integer expression⟩</i> .
<hr/> <hr/> <code>\char_show_value_catcode:n</code>	<code>\char_show_value_catcode:n {⟨integer expression⟩}</code>
	Displays the current category code of the <i>⟨character⟩</i> with character code given by the <i>⟨integer expression⟩</i> on the terminal.
<hr/> <hr/> <code>\char_set_lccode:nn</code>	<code>\char_set_lccode:nn {⟨integer₁⟩} {⟨integer₂⟩}</code>
	This function set up the behaviour of <i>⟨character⟩</i> when found inside <code>\tl_to_lowercase:n</code> , such that <i>⟨character₁⟩</i> will be converted into <i>⟨character₂⟩</i> . The two <i>⟨characters⟩</i> may be specified using an <i>⟨integer expression⟩</i> for the character code concerned. This may include the T _E X ‘ <i>⟨character⟩</i> ’ method for converting a single character into its character code:
	<pre> \char_set_lccode:nn { ‘\A } { ‘\a } % Standard behaviour \char_set_lccode:nn { ‘\A } { ‘\A + 32 } \char_set_lccode:nn { 50 } { 60 } </pre>
	The setting applies within the current T _E X group.
<hr/> <hr/> <code>\char_value_lccode:n</code> ★	<code>\char_value_lccode:n {⟨integer expression⟩}</code>
	Expands to the current lower case code of the <i>⟨character⟩</i> with character code given by the <i>⟨integer expression⟩</i> .
<hr/> <hr/> <code>\char_show_value_lccode:n</code>	<code>\char_show_value_lccode:n {⟨integer expression⟩}</code>
	Displays the current lower case code of the <i>⟨character⟩</i> with character code given by the <i>⟨integer expression⟩</i> on the terminal.

<code>\char_set_uccode:nn</code>	<code>\char_set_uccode:nn {<intexpr₁>} {<intexpr₂>}</code>
----------------------------------	--

This function set up the behaviour of $\langle character \rangle$ when found inside `\tl_to_uppercase:n`, such that $\langle character1 \rangle$ will be converted into $\langle character2 \rangle$. The two $\langle characters \rangle$ may be specified using an $\langle integer expression \rangle$ for the character code concerned. This may include the T_EX ‘ $\langle character \rangle$ ’ method for converting a single character into its character code:

```
\char_set_uccode:nn { '\a } { '\A } % Standard behaviour
\char_set_uccode:nn { '\A } { '\A - 32 }
\char_set_uccode:nn { 60 } { 50 }
```

The setting applies within the current T_EX group.

<code>\char_value_uccode:n</code> ★	<code>\char_value_uccode:n {<integer expression>}</code>
-------------------------------------	--

Expands to the current upper case code of the $\langle character \rangle$ with character code given by the $\langle integer expression \rangle$.

<code>\char_show_value_uccode:n</code>	<code>\char_show_value_uccode:n {<integer expression>}</code>
--	---

Displays the current upper case code of the $\langle character \rangle$ with character code given by the $\langle integer expression \rangle$ on the terminal.

<code>\char_set_mathcode:nn</code>	<code>\char_set_mathcode:nn {<intexpr₁>} {<intexpr₂>}</code>
------------------------------------	--

This function sets up the math code of $\langle character \rangle$. The $\langle character \rangle$ is specified as an $\langle integer expression \rangle$ which will be used as the character code of the relevant character. The setting applies within the current T_EX group.

<code>\char_value_mathcode:n</code> ★	<code>\char_value_mathcode:n {<integer expression>}</code>
---------------------------------------	--

Expands to the current math code of the $\langle character \rangle$ with character code given by the $\langle integer expression \rangle$.

<code>\char_show_value_mathcode:n</code>	<code>\char_show_value_mathcode:n {<integer expression>}</code>
--	---

Displays the current math code of the $\langle character \rangle$ with character code given by the $\langle integer expression \rangle$ on the terminal.

<code>\char_set_sfcode:nn</code>	<code>\char_set_sfcode:nn {<intexpr₁>} {<intexpr₂>}</code>
----------------------------------	--

This function sets up the space factor for the $\langle character \rangle$. The $\langle character \rangle$ is specified as an $\langle integer expression \rangle$ which will be used as the character code of the relevant character. The setting applies within the current T_EX group.

<code>\char_value_sfcode:n</code> ★	<code>\char_value_sfcode:n {<integer expression>}</code>
-------------------------------------	--

Expands to the current space factor for the $\langle character \rangle$ with character code given by the $\langle integer expression \rangle$.

`\char_show_value_sfcode:n``\char_show_value_sfcode:n {{integer expression}}`

Displays the current space factor for the $\langle character \rangle$ with character code given by the $\langle integer expression \rangle$ on the terminal.

38 Generic tokens

`\token_new:Nn``\token_new:Nn \langle token1 \rangle {\langle token2 \rangle}`

Defines $\langle token1 \rangle$ to globally be a snapshot of $\langle token2 \rangle$. This will be an implicit representation of $\langle token2 \rangle$.

`\c_group_begin_token`
`\c_group_end_token`
`\c_math_toggle_token`
`\c_alignment_token`
`\c_parameter_token`
`\c_math_superscript_token`
`\c_math_subscript_token`
`\c_space_token`

These are implicit tokens which have the category code described by their name. They are used internally for test purposes but are also available to the programmer for other uses.

`\c_catcode_letter_token`
`\c_catcode_other_token`

These are implicit tokens which have the category code described by their name. They are used internally for test purposes and should not be used other than for category code tests.

`\c_catcode_active_tl`

A token list containing an active token. This is used internally for test purposes and should not be used other than in appropriately-constructed category code tests.

39 Converting tokens

`\token_to_meaning:N` ★`\token_to_meaning:N \langle token \rangle`

Inserts the current meaning of the $\langle token \rangle$ into the input stream as a series of characters of category code 12 (other). This will be the primitive \TeX description of the $\langle token \rangle$, thus for example both functions defined by `\cs_set_nopar:Npn` and token list variables defined using `\tl_new:N` will be described as **macros**.

\TeX hackers note: This is the \TeX primitive `\meaning`.

<code>\token_to_str:N</code>	★	<code>\token_to_str:N</code>	$\langle token \rangle$
<code>\token_to_str:c</code>	★		

Converts the given $\langle token \rangle$ into a series of characters with category code 12 (other). The current escape character will be the first character in the sequence, although this will also have category code 12 (the escape character is part of the $\langle token \rangle$). This function requires only a single expansion.

T_EXhackers note: `\token_to_str:N` is the T_EX primitive `\string` renamed.

40 Token conditionals

<code>\token_if_group_begin_p:N</code>	★	<code>\token_if_group_begin_p:N</code>	$\langle token \rangle$
<code>\token_if_group_begin:NTF</code>	★	<code>\token_if_group_begin:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if $\langle token \rangle$ has the category code of a begin group token (`{` when normal T_EX category codes are in force). Note that an explicit begin group token cannot be tested in this way, as it is not a valid N-type argument.

<code>\token_if_group_end_p:N</code>	★	<code>\token_if_group_end_p:N</code>	$\langle token \rangle$
<code>\token_if_group_end:NTF</code>	★	<code>\token_if_group_end:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if $\langle token \rangle$ has the category code of an end group token (`}` when normal T_EX category codes are in force). Note that an explicit end group token cannot be tested in this way, as it is not a valid N-type argument.

<code>\token_if_math_toggle_p:N</code>	★	<code>\token_if_math_toggle_p:N</code>	$\langle token \rangle$
<code>\token_if_math_toggle:NTF</code>	★	<code>\token_if_math_toggle:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if $\langle token \rangle$ has the category code of a math shift token (`$` when normal T_EX category codes are in force).

<code>\token_if_alignment_p:N</code>	★	<code>\token_if_alignment_p:N</code>	$\langle token \rangle$
<code>\token_if_alignment:NTF</code>	★	<code>\token_if_alignment:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if $\langle token \rangle$ has the category code of an alignment token (`&` when normal T_EX category codes are in force).

<code>\token_if_parameter_p:N</code>	★	<code>\token_if_parameter_p:N</code>	$\langle token \rangle$
<code>\token_if_parameter:NTF</code>	★	<code>\token_if_parameter:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if $\langle token \rangle$ has the category code of a macro parameter token (`#` when normal T_EX category codes are in force).

<code>\token_if_math_superscript_p:N</code>	★	<code>\token_if_math_superscript_p:N</code>	$\langle token \rangle$
<code>\token_if_math_superscript:NTF</code>	★	<code>\token_if_math_superscript:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if $\langle token \rangle$ has the category code of a superscript token (`^` when normal T_EX category codes are in force).

<code>\token_if_math_subscript_p:N</code>	<code>*</code>	<code>\token_if_math_subscript_p:N</code>	<code><token></code>
<code>\token_if_math_subscript:N</code>	<code>*</code>	<code>\token_if_math_subscript:N</code>	<code><token> {\true code} {\false code}</code>

Tests if $\langle token \rangle$ has the category code of a subscript token (`_` when normal \TeX category codes are in force).

<code>\token_if_space_p:N</code>	<code>*</code>	<code>\token_if_space_p:N</code>	<code><token></code>
<code>\token_if_space:N</code>	<code>*</code>	<code>\token_if_space:N</code>	<code><token> {\true code} {\false code}</code>

Tests if $\langle token \rangle$ has the category code of a space token. Note that an explicit space token with character code 32 cannot be tested in this way, as it is not a valid N-type argument.

<code>\token_if_letter_p:N</code>	<code>*</code>	<code>\token_if_letter_p:N</code>	<code><token></code>
<code>\token_if_letter:N</code>	<code>*</code>	<code>\token_if_letter:N</code>	<code><token> {\true code} {\false code}</code>

Tests if $\langle token \rangle$ has the category code of a letter token.

<code>\token_if_other_p:N</code>	<code>*</code>	<code>\token_if_other_p:N</code>	<code><token></code>
<code>\token_if_other:N</code>	<code>*</code>	<code>\token_if_other:N</code>	<code><token> {\true code} {\false code}</code>

Tests if $\langle token \rangle$ has the category code of an “other” token.

<code>\token_if_active_p:N</code>	<code>*</code>	<code>\token_if_active_p:N</code>	<code><token></code>
<code>\token_if_active:N</code>	<code>*</code>	<code>\token_if_active:N</code>	<code><token> {\true code} {\false code}</code>

Tests if $\langle token \rangle$ has the category code of an active character.

<code>\token_if_eq_catcode_p:NN</code>	<code>*</code>	<code>\token_if_eq_catcode_p:NN</code>	<code><token1> <token2></code>
<code>\token_if_eq_catcode:N</code>	<code>*</code>	<code>\token_if_eq_catcode:N</code>	<code><token1> <token2> {\true code} {\false code}</code>

Tests if the two $\langle tokens \rangle$ have the same category code.

<code>\token_if_eq_charcode_p:NN</code>	<code>*</code>	<code>\token_if_eq_charcode_p:NN</code>	<code><token1> <token2></code>
<code>\token_if_eq_charcode:N</code>	<code>*</code>	<code>\token_if_eq_charcode:N</code>	<code><token1> <token2> {\true code} {\false code}</code>

Tests if the two $\langle tokens \rangle$ have the same character code.

<code>\token_if_eq_meaning_p:NN</code>	<code>*</code>	<code>\token_if_eq_meaning_p:NN</code>	<code><token1> <token2></code>
<code>\token_if_eq_meaning:N</code>	<code>*</code>	<code>\token_if_eq_meaning:N</code>	<code><token1> <token2> {\true code} {\false code}</code>

Tests if the two $\langle tokens \rangle$ have the same meaning when expanded.

<code>\token_if_macro_p:N</code>	<code>*</code>	<code>\token_if_macro_p:N</code>	<code><token></code>
<code>\token_if_macro:N</code>	<code>*</code>	<code>\token_if_macro:N</code>	<code><token> {\true code} {\false code}</code>

Updated: 2001-05-23

Tests if the $\langle token \rangle$ is a \TeX macro.

<code>\token_if_cs_p:N</code>	<code>*</code>	<code>\token_if_cs_p:N</code>	<code><token></code>
<code>\token_if_cs:N</code>	<code>*</code>	<code>\token_if_cs:N</code>	<code><token> {\true code} {\false code}</code>

Tests if the $\langle token \rangle$ is a control sequence.

<code>\token_if_expandable_p:N</code>	★	<code>\token_if_expandable_p:N</code>	$\langle token \rangle$
<code>\token_if_expandable:NTF</code>	★	<code>\token_if_expandable:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is expandable. This test returns $\langle false \rangle$ for an undefined token.

<code>\token_if_long_macro_p:N</code>	★	<code>\token_if_long_macro_p:N</code>	$\langle token \rangle$
<code>\token_if_long_macro:NTF</code>	★	<code>\token_if_long_macro:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is a long macro.

<code>\token_if_protected_macro_p:N</code>	★	<code>\token_if_protected_macro_p:N</code>	$\langle token \rangle$
<code>\token_if_protected_macro:NTF</code>	★	<code>\token_if_protected_macro:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is a protected macro: a macro which is both protected and long will return logical false.

<code>\token_if_protected_long_macro_p:N</code>	★	<code>\token_if_protected_long_macro_p:N</code>	$\langle token \rangle$
<code>\token_if_protected_long_macro:NTF</code>	★	<code>\token_if_protected_long_macro:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is a protected long macro.

<code>\token_if_chardef_p:N</code>	★	<code>\token_if_chardef_p:N</code>	$\langle token \rangle$
<code>\token_if_chardef:NTF</code>	★	<code>\token_if_chardef:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is defined to be a chardef.

<code>\token_if_mathchardef_p:N</code>	★	<code>\token_if_mathchardef_p:N</code>	$\langle token \rangle$
<code>\token_if_mathchardef:NTF</code>	★	<code>\token_if_mathchardef:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is defined to be a mathchardef.

<code>\token_if_dim_register_p:N</code>	★	<code>\token_if_dim_register_p:N</code>	$\langle token \rangle$
<code>\token_if_dim_register:NTF</code>	★	<code>\token_if_dim_register:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is defined to be a dimension register.

<code>\token_if_int_register_p:N</code>	★	<code>\token_if_int_register_p:N</code>	$\langle token \rangle$
<code>\token_if_int_register:NTF</code>	★	<code>\token_if_int_register:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is defined to be a integer register.

<code>\token_if_skip_register_p:N</code>	★	<code>\token_if_skip_register_p:N</code>	$\langle token \rangle$
<code>\token_if_skip_register:NTF</code>	★	<code>\token_if_skip_register:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is defined to be a skip register.

<code>\token_if_toks_register_p:N</code>	★	<code>\token_if_toks_register_p:N</code>	$\langle token \rangle$
<code>\token_if_toks_register:NTF</code>	★	<code>\token_if_toks_register:NTF</code>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if the $\langle token \rangle$ is defined to be a toks register (not used by L^AT_EX3).

<code>\token_if_primitive_p:N</code> ★	<code>\token_if_primitive_p:N</code> $\langle token \rangle$
<code>\token_if_primitive:NTF</code> ★	<code>\token_if_primitive:NTF</code> $\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Updated: 2001-05-23

Tests if the $\langle token \rangle$ is an engine primitive.

41 Peeking ahead at the next token

There is often a need to look ahead at the next token in the input stream while leaving it in place. This is handled using the “peek” functions. The generic `\peek_after:Nw` is provided along with a family of predefined tests for common cases. As peeking ahead does *not* skip spaces the predefined tests include both a space-respecting and space-skipping version.

<code>\peek_after:Nw</code>	<code>\peek_after:Nw</code> $\langle function \rangle$ $\langle token \rangle$
-----------------------------	--

Locally sets the test variable `\l_peek_token` equal to $\langle token \rangle$ (as an implicit token, *not* as a token list), and then expands the $\langle function \rangle$. The $\langle token \rangle$ will remain in the input stream as the next item after the $\langle function \rangle$. The $\langle token \rangle$ here may be \sqcup , $\{$ or $\}$ (assuming normal T_EX category codes), *i.e.* it is not necessarily the next argument which would be grabbed by a normal function.

<code>\peek_gafter:Nw</code>	<code>\peek_gafter:Nw</code> $\langle function \rangle$ $\langle token \rangle$
------------------------------	---

Globally sets the test variable `\g_peek_token` equal to $\langle token \rangle$ (as an implicit token, *not* as a token list), and then expands the $\langle function \rangle$. The $\langle token \rangle$ will remain in the input stream as the next item after the $\langle function \rangle$. The $\langle token \rangle$ here may be \sqcup , $\{$ or $\}$ (assuming normal T_EX category codes), *i.e.* it is not necessarily the next argument which would be grabbed by a normal function.

<code>\l_peek_token</code>	Token set by <code>\peek_after:Nw</code> and available for testing as described above.
----------------------------	--

<code>\g_peek_token</code>	Token set by <code>\peek_gafter:Nw</code> and available for testing as described above.
----------------------------	---

<code>\peek_catcode:NTF</code>	<code>\peek_catcode:NTF</code> $\langle test token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$
--------------------------------	--

Updated: 2011-07-02

Tests if the next $\langle token \rangle$ in the input stream has the same category code as the $\langle test token \rangle$ (as defined by the test `\token_if_eq_catcode:NNTF`). Spaces are respected by the test and the $\langle token \rangle$ will be left in the input stream after the $\langle true code \rangle$ or $\langle false code \rangle$ (as appropriate to the result of the test).

<code>\peek_catcode_ignore_spaces:NTF</code>	<code>\peek_catcode_ignore_spaces:NTF</code> $\langle test token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$
--	--

Updated: 2011-07-02

Tests if the next $\langle token \rangle$ in the input stream has the same category code as the $\langle test token \rangle$ (as defined by the test `\token_if_eq_catcode:NNTF`). Spaces are ignored by the test and the $\langle token \rangle$ will be left in the input stream after the $\langle true code \rangle$ or $\langle false code \rangle$ (as appropriate to the result of the test).

<code>\peek_catcode_remove:NTF</code>	<code>\peek_catcode_remove:NTF <test token> {(true code)} {(false code)}</code>
Updated: 2011-07-02	Tests if the next <i><token></i> in the input stream has the same category code as the <i><test token></i> (as defined by the test <code>\token_if_eq_catcode:NNTF</code>). Spaces are respected by the test and the <i><token></i> will be removed from the input stream if the test is true. The function will then place either the <i><true code></i> or <i><false code></i> in the input stream (as appropriate to the result of the test).

<code>\peek_catcode_remove_ignore_spaces:NTF</code>	<code>\peek_catcode_remove_ignore_spaces:NTF <test token> {(true code)} {(false code)}</code>
Updated: 2011-07-02	Tests if the next <i><token></i> in the input stream has the same category code as the <i><test token></i> (as defined by the test <code>\token_if_eq_catcode:NNTF</code>). Spaces are ignored by the test and the <i><token></i> will be removed from the input stream if the test is true. The function will then place either the <i><true code></i> or <i><false code></i> in the input stream (as appropriate to the result of the test).

<code>\peek_charcode:NTF</code>	<code>\peek_charcode:NTF <test token> {(true code)} {(false code)}</code>
Updated: 2011-07-02	Tests if the next <i><token></i> in the input stream has the same character code as the <i><test token></i> (as defined by the test <code>\token_if_eq_charcode:NNTF</code>). Spaces are respected by the test and the <i><token></i> will be left in the input stream after the <i><true code></i> or <i><false code></i> (as appropriate to the result of the test).

<code>\peek_charcode_ignore_spaces:NTF</code>	<code>\peek_charcode_ignore_spaces:NTF <test token> {(true code)} {(false code)}</code>
Updated: 2011-07-02	Tests if the next <i><token></i> in the input stream has the same character code as the <i><test token></i> (as defined by the test <code>\token_if_eq_charcode:NNTF</code>). Spaces are ignored by the test and the <i><token></i> will be left in the input stream after the <i><true code></i> or <i><false code></i> (as appropriate to the result of the test).

<code>\peek_charcode_remove:NTF</code>	<code>\peek_charcode_remove:NTF <test token> {(true code)} {(false code)}</code>
Updated: 2011-07-02	Tests if the next <i><token></i> in the input stream has the same character code as the <i><test token></i> (as defined by the test <code>\token_if_eq_charcode:NNTF</code>). Spaces are respected by the test and the <i><token></i> will be removed from the input stream if the test is true. The function will then place either the <i><true code></i> or <i><false code></i> in the input stream (as appropriate to the result of the test).

<code>\peek_charcode_remove_ignore_spaces:NTF</code>	<code>\peek_charcode_remove_ignore_spaces:NTF <test token> {(true code)} {(false code)}</code>
Updated: 2011-07-02	Tests if the next <i><token></i> in the input stream has the same character code as the <i><test token></i> (as defined by the test <code>\token_if_eq_charcode:NNTF</code>). Spaces are ignored by the test and the <i><token></i> will be removed from the input stream if the test is true. The function will then place either the <i><true code></i> or <i><false code></i> in the input stream (as appropriate to the result of the test).

<code>\peek_meaning:N</code> <u><code>TF</code></u>	<code>\peek_meaning:N</code> <code>TF</code> $\langle test\ token \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
Updated: 2011-07-02	Tests if the next $\langle token \rangle$ in the input stream has the same meaning as the $\langle test\ token \rangle$ (as defined by the test <code>\token_if_eq_meaning:N</code> <code>NTF</code>). Spaces are respected by the test and the $\langle token \rangle$ will be left in the input stream after the $\langle true\ code \rangle$ or $\langle false\ code \rangle$ (as appropriate to the result of the test).

<code>\peek_meaning_ignore_spaces:N</code> <u><code>TF</code></u>	<code>\peek_meaning_ignore_spaces:N</code> <code>TF</code> $\langle test\ token \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
Updated: 2011-07-02	Tests if the next $\langle token \rangle$ in the input stream has the same meaning as the $\langle test\ token \rangle$ (as defined by the test <code>\token_if_eq_meaning:N</code> <code>NTF</code>). Spaces are ignored by the test and the $\langle token \rangle$ will be left in the input stream after the $\langle true\ code \rangle$ or $\langle false\ code \rangle$ (as appropriate to the result of the test).

<code>\peek_meaning_remove:N</code> <u><code>TF</code></u>	<code>\peek_meaning_remove:N</code> <code>TF</code> $\langle test\ token \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
Updated: 2011-07-02	Tests if the next $\langle token \rangle$ in the input stream has the same meaning as the $\langle test\ token \rangle$ (as defined by the test <code>\token_if_eq_meaning:N</code> <code>NTF</code>). Spaces are respected by the test and the $\langle token \rangle$ will be removed from the input stream if the test is true. The function will then place either the $\langle true\ code \rangle$ or $\langle false\ code \rangle$ in the input stream (as appropriate to the result of the test).

<code>\peek_meaning_remove_ignore_spaces:N</code> <u><code>TF</code></u>	<code>\peek_meaning_remove_ignore_spaces:N</code> <code>TF</code> $\langle test\ token \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
Updated: 2011-07-02	Tests if the next $\langle token \rangle$ in the input stream has the same meaning as the $\langle test\ token \rangle$ (as defined by the test <code>\token_if_eq_meaning:N</code> <code>NTF</code>). Spaces are ignored by the test and the $\langle token \rangle$ will be removed from the input stream if the test is true. The function will then place either the $\langle true\ code \rangle$ or $\langle false\ code \rangle$ in the input stream (as appropriate to the result of the test).

42 Decomposing a macro definition

These functions decompose \TeX macros into their constituent parts: if the $\langle token \rangle$ passed is not a macro then no decomposition can occur. In the later case, all three functions leave `\scan_stop:` in the input stream.

<code>\token_get_arg_spec:N</code>	★	<code>\token_get_arg_spec:N</code>	$\langle token \rangle$
------------------------------------	---	------------------------------------	-------------------------

If the $\langle token \rangle$ is a macro, this function will leave the primitive \TeX argument specification in input stream as a string of tokens of category code 12 (with spaces having category code 10). Thus for example for a token `\next` defined by

`\cs_set:Npn \next #1#2 { x #1 y #2 }`

will leave `#1#2` in the input stream. If the $\langle token \rangle$ is not a macro then `\scan_stop:` will be left in the input stream

\TeX hackers note: If the arg spec. contains the string `->`, then the `spec` function will produce incorrect results.

<code>\token_get_replacement_text:N</code>	★	<code>\token_get_replacement_text:N</code>	$\langle token \rangle$
--	---	--	-------------------------

If the $\langle token \rangle$ is a macro, this function will leave the replacement text in input stream as a string of tokens of category code 12 (with spaces having category code 10). Thus for example for a token `\next` defined by

`\cs_set:Npn \next #1#2 { x #1~y #2 }`

will leave `x#1 y#2` in the input stream. If the $\langle token \rangle$ is not a macro then `\scan_stop:` will be left in the input stream

<code>\token_get_prefix_spec:N</code>	★	<code>\token_get_prefix_spec:N</code>	$\langle token \rangle$
---------------------------------------	---	---------------------------------------	-------------------------

If the $\langle token \rangle$ is a macro, this function will leave the \TeX prefixes applicable in input stream as a string of tokens of category code 12 (with spaces having category code 10). Thus for example for a token `\next` defined by

`\cs_set:Npn \next #1#2 { x #1~y #2 }`

will leave `\long` in the input stream. If the $\langle token \rangle$ is not a macro then `\scan_stop:` will be left in the input stream

43 Experimental token functions

<code>\char_set_active:Npn</code>	<code>\char_set_active:Npn</code>	$\langle char \rangle$	$\langle parameters \rangle$	$\{ \langle code \rangle \}$
-----------------------------------	-----------------------------------	------------------------	------------------------------	------------------------------

`\char_set_active:Npx`

New: 2011-12-27

Makes $\langle char \rangle$ an active character to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ (`#1`, `#2`, etc.) will be replaced by those absorbed. The $\langle char \rangle$ is made active within the current \TeX group level, and the definition is also local.

<hr/> <code>\char_gset_active:Npn</code> <hr/>	<code>\char_gset_active:Npn <char> <parameters> {<code>}</code>
<code>\char_gset_active:Npx</code> <hr/>	
New: 2011-12-27	Makes $\langle char \rangle$ an active character to expand to $\langle code \rangle$ as replacement text. Within the $\langle code \rangle$, the $\langle parameters \rangle$ ($\#1$, $\#2$, <i>etc.</i>) will be replaced by those absorbed. The $\langle char \rangle$ is made active within the current T _E X group level, but the definition is global. This function is therefore suited to cases where an active character definition should be applied only in some context (where the $\langle char \rangle$ is again made active).
<hr/> <code>\char_set_active_eq:NN</code> <hr/>	<code>\char_set_active_eq:NN <char> <function></code>
New: 2011-12-27	Makes $\langle char \rangle$ an active character equivalent in meaning to the $\langle function \rangle$ (which may itself be an active character). The $\langle char \rangle$ is made active within the current T _E X group level, and the definition is also local.
<hr/> <code>\char_gset_active_eq:NN</code> <hr/>	<code>\char_gset_active_eq:NN <char> <function></code>
New: 2011-12-27	Makes $\langle char \rangle$ an active character equivalent in meaning to the $\langle function \rangle$ (which may itself be an active character). The $\langle char \rangle$ is made active within the current T _E X group level, but the definition is global. This function is therefore suited to cases where an active character definition should be applied only in some context (where the $\langle char \rangle$ is again made active).
<hr/> <code>\peek_N_typeTF</code> <hr/>	<code>\peek_N_type:TF {<true code>} {<false code>}</code>
New: 2011-08-14	Tests if the next $\langle token \rangle$ in the input stream can be safely grabbed as an N-type argument. The test will be $\langle false \rangle$ if the next $\langle token \rangle$ is either an explicit or implicit begin-group or end-group token (with any character code), or an explicit or implicit space character (with character code 32 and category code 10), and $\langle true \rangle$ in all other cases. Note that a $\langle true \rangle$ result ensures that the next $\langle token \rangle$ is a valid N-type argument. However, if the next $\langle token \rangle$ is for instance <code>\c_space_token</code> , the test will take the $\langle false \rangle$ branch, even though the next $\langle token \rangle$ is in fact a valid N-type argument. The $\langle token \rangle$ will be left in the input stream after the $\langle true code \rangle$ or $\langle false code \rangle$ (as appropriate to the result of the test).

Part IX

The l3int package

Integers

Calculation and comparison of integer values can be carried out using literal numbers, `int` registers, constants and integers stored in token list variables. The standard operators `+`, `-`, `/` and `*` and parentheses can be used within such expressions to carry arithmetic operations. This module carries out these functions on *integer expressions* (“`int expr`”).

44 Integer expressions

`\int_eval:n` ★ `\int_eval:n {⟨integer expression⟩}`

Evaluates the *⟨integer expression⟩*, expanding any integer and token list variables within the *⟨expression⟩* to their content (without requiring `\int_use:N/\tl_use:N`) and applying the standard mathematical rules. For example both

```
\int_eval:n { 5 + 4 * 3 - ( 3 + 4 * 5 ) }
```

and

```
\tl_new:N \l_my_tl
\tl_set:Nn \l_my_tl { 5 }
\int_new:N \l_my_int
\int_set:Nn \l_my_int { 4 }
\int_eval:n { \l_my_tl + \l_my_int * 3 - ( 3 + 4 * 5 ) }
```

both evaluate to -6 . The *⟨integer expression⟩* may contain the operators `+`, `-`, `*` and `/`, along with parenthesis `(` and `)`. After two expansions, `\int_eval:n` yields a *⟨integer denotation⟩* which is left in the input stream. This is *not* an *⟨internal integer⟩*, and therefore requires suitable termination if used in a TeX-style integer assignment.

`\int_abs:n` ★ `\int_abs:n {⟨integer expression⟩}`

Evaluates the *⟨integer expression⟩* as described for `\int_eval:n` and leaves the absolute value of the result in the input stream as an *⟨integer denotation⟩* after two expansions.

`\int_div_round:nn` ★ `\int_div_round:nn {⟨intexpr1⟩} {⟨intexpr2⟩}`

Evaluates the two *⟨integer expressions⟩* as described earlier, then calculates the result of dividing the first value by the second, round any remainder. Note that this is identical to using `/` directly in an *⟨integer expression⟩*. The result is left in the input stream as a *⟨integer denotation⟩* after two expansions.

<code>\int_div_truncate:nn</code> ★	<code>\int_div_truncate:nn {\langle integer \rangle_1} {\langle integer \rangle_2}</code>
-------------------------------------	---

Evaluates the two $\langle integer expressions \rangle$ as described earlier, then calculates the result of dividing the first value by the second, truncating any remainder. Note that division using `/` rounds the result. The result is left in the input stream as a $\langle integer denotation \rangle$ after two expansions.

<code>\int_max:nn</code> ★	<code>\int_max:nn {\langle integer \rangle_1} {\langle integer \rangle_2}</code>
<code>\int_min:nn</code> ★	<code>\int_min:nn {\langle integer \rangle_1} {\langle integer \rangle_2}</code>

Evaluates the $\langle integer expressions \rangle$ as described for `\int_eval:n` and leaves either the larger or smaller value in the input stream as an $\langle integer denotation \rangle$ after two expansions.

<code>\int_mod:nn</code> ★	<code>\int_mod:nn {\langle integer \rangle_1} {\langle integer \rangle_2}</code>
----------------------------	--

Evaluates the two $\langle integer expressions \rangle$ as described earlier, then calculates the integer remainder of dividing the first expression by the second. This is left in the input stream as an $\langle integer denotation \rangle$ after two expansions.

45 Creating and initialising integers

<code>\int_new:N</code>	<code>\int_new:N \langle integer \rangle</code>
<code>\int_new:c</code>	

Creates a new $\langle integer \rangle$ or raises an error if the name is already taken. The declaration is global. The $\langle integer \rangle$ will initially be equal to 0.

<code>\int_const:Nn</code>	<code>\int_const:Nn \langle integer \rangle {\langle integer expression \rangle}</code>
<code>\int_const:cn</code>	

Creates a new constant $\langle integer \rangle$ or raises an error if the name is already taken. The value of the $\langle integer \rangle$ will be set globally to the $\langle integer expression \rangle$.

Updated: 2011-10-22

<code>\int_zero:N</code>	<code>\int_zero:N \langle integer \rangle</code>
<code>\int_zero:c</code>	
<code>\int_gzero:N</code>	
<code>\int_gzero:c</code>	

Sets $\langle integer \rangle$ to 0.

<code>\int_zero_new:N</code>	<code>\int_zero_new:N \langle integer \rangle</code>
<code>\int_zero_new:c</code>	
<code>\int_gzero_new:N</code>	
<code>\int_gzero_new:c</code>	

Ensures that the $\langle integer \rangle$ exists globally by applying `\int_new:N` if necessary, then applies `\int_(g)zero:N` to leave the $\langle integer \rangle$ is set to zero.

New: 2011-12-13

<code>\int_set_eq:NN</code>	<code>\int_set_eq:NN \langle integer1 \rangle \langle integer2 \rangle</code>
<code>\int_set_eq:(cN Nc cc)</code>	
<code>\int_gset_eq:NN</code>	
<code>\int_gset_eq:(cN Nc cc)</code>	

Sets the content of $\langle integer1 \rangle$ equal to that of $\langle integer2 \rangle$.

46 Setting and incrementing integers

<code>\int_add:Nn</code>	<code>\int_add:Nn <integer> {<integer expression>}</code>
<code>\int_add:cn</code>	
<code>\int_gadd:Nn</code>	Adds the result of the <i><integer expression></i> to the current content of the <i><integer></i> .
<code>\int_gadd:cn</code>	
Updated: 2011-10-22	
<code>\int_decr:N</code>	<code>\int_decr:N <integer></code>
<code>\int_decr:c</code>	
<code>\int_gdecr:N</code>	Decreases the value stored in <i><integer></i> by 1.
<code>\int_gdecr:c</code>	
<code>\int_incr:N</code>	<code>\int_incr:N <integer></code>
<code>\int_incr:c</code>	
<code>\int_gincr:N</code>	Increases the value stored in <i><integer></i> by 1.
<code>\int_gincr:c</code>	
<code>\int_set:Nn</code>	<code>\int_set:Nn <integer> {<integer expression>}</code>
<code>\int_set:cn</code>	
<code>\int_gset:Nn</code>	Sets <i><integer></i> to the value of <i><integer expression></i> , which must evaluate to an integer (as described for <code>\int_eval:n</code>).
<code>\int_gset:cn</code>	
Updated: 2011-10-22	
<code>\int_sub:Nn</code>	<code>\int_sub:Nn <integer> {<integer expression>}</code>
<code>\int_sub:cn</code>	
<code>\int_gsub:Nn</code>	Subtracts the result of the <i><integer expression></i> to the current content of the <i><integer></i> .
<code>\int_gsub:cn</code>	
Updated: 2011-10-22	

47 Using integers

<code>\int_use:N</code>	★ <code>\int_use:N <integer></code>
<code>\int_use:c</code>	★
Updated: 2011-10-22	
Recovers the content of a <i><integer></i> and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Can be omitted in places where a <i><integer></i> is required (such as in the first and third arguments of <code>\int_compare:nNnTF</code>).	

T_EXhackers note: `\int_use:N` is the T_EX primitive `\the`: this is one of several L^AT_EX3 names for this primitive.

48 Integer expression conditionals

<code>\int_compare_p:nNn</code> ★	<code>\int_compare_p:nNn {⟨integer expression⟩} ⟨relation⟩ {⟨integer expression⟩}</code>
<code>\int_compare:nNnTF</code> ★	<code>\int_compare:nNnTF {⟨integer expression⟩} ⟨relation⟩ {⟨integer expression⟩}</code> <code>{⟨true code⟩} {⟨false code⟩}</code>

This function first evaluates each of the *⟨integer expressions⟩* as described for `\int_eval:n`. The two results are then compared using the *⟨relation⟩*:

Equal	=
Greater than	>
Less than	<

<code>\int_compare_p:n</code> ★	<code>\int_compare_p:n { ⟨integer expression⟩ ⟨relation⟩ ⟨integer expression⟩ }</code>
<code>\int_compare:nTF</code> ★	<code>\int_compare:nTF { ⟨integer expression⟩ ⟨relation⟩ ⟨integer expression⟩ }</code> <code>{⟨true code⟩} {⟨false code⟩}</code>

This function first evaluates each of the *⟨integer expressions⟩* as described for `\int_eval:n`. The two results are then compared using the *⟨relation⟩*:

Equal	= or ==
Greater than or equal to	=>
Greater than	>
Less than or equal to	=<
Less than	<
Not equal	!=

<code>\int_if_even_p:n</code> ★	<code>\int_if_odd_p:n {⟨integer expression⟩}</code>
<code>\int_if_even:nTF</code> ★	<code>\int_if_odd:nTF {⟨integer expression⟩}</code>
<code>\int_if_odd_p:n</code> ★	<code>{⟨true code⟩} {⟨false code⟩}</code>
<code>\int_if_odd:nTF</code> ★	

This function first evaluates the *⟨integer expression⟩* as described for `\int_eval:n`. It then evaluates if this is odd or even, as appropriate.

49 Integer expression loops

<code>\int_do_while:nNnn</code> ☆	<code>\int_do_while:nNnn {⟨integer expression⟩} ⟨relation⟩ {⟨integer expression⟩} {⟨code⟩}</code>
-----------------------------------	---

Evaluates the relationship between the two *⟨integer expressions⟩* as described for `\int_compare:nNnTF`, and then places the *⟨code⟩* in the input stream if the *⟨relation⟩* is **true**. After the *⟨code⟩* has been processed by \TeX the test will be repeated, and a loop will occur until the test is **false**.

<hr/> <code>\int_do_until:nNnn</code> ☆ <hr/>	<code>\int_do_until:nNnn</code> <code>{\langle intexpr_1 \rangle \langle relation \rangle {\langle intexpr_2 \rangle} {\langle code \rangle}}</code> <p>Evaluates the relationship between the two <i>integer expressions</i> as described for <code>\int_compare:nNnTF</code>, and then places the <i>code</i> in the input stream if the <i>relation</i> is false. After the <i>code</i> has been processed by T_EX the test will be repeated, and a loop will occur until the test is true.</p>
<hr/> <code>\int_until_do:nNnn</code> ☆ <hr/>	<code>\int_until_do:nNnn</code> <code>{\langle intexpr_1 \rangle \langle relation \rangle {\langle intexpr_2 \rangle} {\langle code \rangle}}</code> <p>Places the <i>code</i> in the input stream for T_EX to process, and then evaluates the relationship between the two <i>integer expressions</i> as described for <code>\int_compare:nNnTF</code>. If the test is false then the <i>code</i> will be inserted into the input stream again and a loop will occur until the <i>relation</i> is true.</p>
<hr/> <code>\int_while_do:nNnn</code> ☆ <hr/>	<code>\int_while_do:nNnn</code> <code>{\langle intexpr_1 \rangle \langle relation \rangle {\langle intexpr_2 \rangle} {\langle code \rangle}}</code> <p>Places the <i>code</i> in the input stream for T_EX to process, and then evaluates the relationship between the two <i>integer expressions</i> as described for <code>\int_compare:nNnTF</code>. If the test is true then the <i>code</i> will be inserted into the input stream again and a loop will occur until the <i>relation</i> is false.</p>
<hr/> <code>\int_do_while:nn</code> ☆ <hr/>	<code>\int_do_while:nn</code> <code>{ \langle intexpr_1 \rangle \langle relation \rangle \langle intexpr_2 \rangle } {\langle code \rangle}</code> <p>Evaluates the relationship between the two <i>integer expressions</i> as described for <code>\int_compare:nTF</code>, and then places the <i>code</i> in the input stream if the <i>relation</i> is true. After the <i>code</i> has been processed by T_EX the test will be repeated, and a loop will occur until the test is false.</p>
<hr/> <code>\int_do_until:nn</code> ☆ <hr/>	<code>\int_do_until:nn</code> <code>{ \langle intexpr_1 \rangle \langle relation \rangle \langle intexpr_2 \rangle } {\langle code \rangle}</code> <p>Evaluates the relationship between the two <i>integer expressions</i> as described for <code>\int_compare:nTF</code>, and then places the <i>code</i> in the input stream if the <i>relation</i> is false. After the <i>code</i> has been processed by T_EX the test will be repeated, and a loop will occur until the test is true.</p>
<hr/> <code>\int_until_do:nn</code> ☆ <hr/>	<code>\int_until_do:nn</code> <code>{ \langle intexpr_1 \rangle \langle relation \rangle \langle intexpr_2 \rangle } {\langle code \rangle}</code> <p>Places the <i>code</i> in the input stream for T_EX to process, and then evaluates the relationship between the two <i>integer expressions</i> as described for <code>\int_compare:nTF</code>. If the test is false then the <i>code</i> will be inserted into the input stream again and a loop will occur until the <i>relation</i> is true.</p>

<code>\int_while_do:nn</code> ☆	<code>\int_while_do:nn { <intexpr1> <relation> <intexpr2> } {<code>}</code>
---------------------------------	---

Places the `<code>` in the input stream for T_EX to process, and then evaluates the relationship between the two *<integer expressions>* as described for `\int_compare:nTF`. If the test is `true` then the `<code>` will be inserted into the input stream again and a loop will occur until the *<relation>* is `false`.

50 Formatting integers

Integers can be placed into the output stream with formatting. These conversions apply to any integer expressions.

<code>\int_to_arabic:n</code> ☆	<code>\int_to_arabic:n {<integer expression>}</code>
---------------------------------	--

Updated: 2011-10-22

Places the value of the *<integer expression>* in the input stream as digits, with category code 12 (other).

<code>\int_to_alph:n</code> ☆	<code>\int_to_alph:n {<integer expression>}</code>
<code>\int_to_Alph:n</code> ☆	Evaluates the <i><integer expression></i> and converts the result into a series of letters, which are then left in the input stream. The conversion rule uses the 26 letters of the English alphabet, in order, adding letters when necessary to increase the total possible range of representable numbers. Thus

Updated: 2011-09-17

`\int_to_alph:n { 1 }`

places `a` in the input stream,

`\int_to_alph:n { 26 }`

is represented as `z` and

`\int_to_alph:n { 27 }`

is converted to `aa`. For conversions using other alphabets, use `\int_convert_to_symbols:nnn` to define an alphabet-specific function. The basic `\int_to_alph:n` and `\int_to_Alph:n` functions should not be modified.

`\int_to_symbols:nnn` ★
Updated: 2011-09-17

`\int_to_symbols:nnn`
 $\{\langle integer\ expression \rangle\} \{\langle total\ symbols \rangle\}$
 $\langle value\ to\ symbol\ mapping \rangle$

This is the low-level function for conversion of an $\langle integer\ expression \rangle$ into a symbolic form (which will often be letters). The $\langle total\ symbols \rangle$ available should be given as an integer expression. Values are actually converted to symbols according to the $\langle value\ to\ symbol\ mapping \rangle$. This should be given as $\langle total\ symbols \rangle$ pairs of entries, a number and the appropriate symbol. Thus the `\int_to_alph:n` function is defined as

```
\cs_new:Npn \int_to_alph:n #1
{
  \int_convert_to_symbols:nnn {#1} { 26 }
  {
    { 1 } { a }
    { 2 } { b }
    ...
    { 26 } { z }
  }
}
```

`\int_to_binary:n` ★
Updated: 2011-09-17

`\int_to_binary:n` $\{\langle integer\ expression \rangle\}$

Calculates the value of the $\langle integer\ expression \rangle$ and places the binary representation of the result in the input stream.

`\int_to_hexadecimal:n` ★
Updated: 2011-09-17

`\int_to_hexadecimal:n` $\{\langle integer\ expression \rangle\}$

Calculates the value of the $\langle integer\ expression \rangle$ and places the hexadecimal (base 16) representation of the result in the input stream. Upper case letters are used for digits beyond 9.

`\int_to_octal:n` ★
Updated: 2011-09-17

`\int_to_octal:n` $\{\langle integer\ expression \rangle\}$

Calculates the value of the $\langle integer\ expression \rangle$ and places the octal (base 8) representation of the result in the input stream.

`\int_to_base:nn` ★
Updated: 2011-09-17

`\int_to_base:nn` $\{\langle integer\ expression \rangle\} \{\langle base \rangle\}$

Calculates the value of the $\langle integer\ expression \rangle$ and converts it into the appropriate representation in the $\langle base \rangle$; the later may be given as an integer expression. For bases greater than 10 the higher “digits” are represented by the upper case letters from the English alphabet. The maximum $\langle base \rangle$ value is 36.

T_EXhackers note: This is a generic version of `\int_to_binary:n`, etc.

<hr/> <code>\int_to_roman:n</code> ☆	<code>\int_to_roman:n {\langle integer expression \rangle}</code>
<code>\int_to_Roman:n</code> ☆	
<hr/> Updated: 2011-10-22 <hr/>	Places the value of the $\langle integer expression \rangle$ in the input stream as Roman numerals, either lower case (<code>\int_to_roman:n</code>) or upper case (<code>\int_to_Roman:n</code>). The Roman numerals are letters with category code 11 (letter).

51 Converting from other formats to integers

<hr/> <code>\int_from_alph:n</code> ☆ <hr/>	<code>\int_from_alph:n {\langle letters \rangle}</code>
	Converts the $\langle letters \rangle$ into the integer (base 10) representation and leaves this in the input stream. The $\langle letters \rangle$ are treated using the English alphabet only, with “a” equal to 1 through to “z” equal to 26. Either lower or upper case letters may be used. This is the inverse function of <code>\int_to_alph:n</code> .

<hr/> <code>\int_from_binary:n</code> ☆ <hr/>	<code>\int_from_binary:n {\langle binary number \rangle}</code>
	Converts the $\langle binary number \rangle$ into the integer (base 10) representation and leaves this in the input stream.

<hr/> <code>\int_from_hexadecimal:n</code> ☆ <hr/>	<code>\int_from_hexadecimal:n {\langle hexadecimal number \rangle}</code>
	Converts the $\langle hexadecimal number \rangle$ into the integer (base 10) representation and leaves this in the input stream. Digits greater than 9 may be represented in the $\langle hexadecimal number \rangle$ by upper or lower case letters.

<hr/> <code>\int_from_octal:n</code> ☆ <hr/>	<code>\int_from_octal:n {\langle octal number \rangle}</code>
	Converts the $\langle octal number \rangle$ into the integer (base 10) representation and leaves this in the input stream.

<hr/> <code>\int_from_roman:n</code> ☆ <hr/>	<code>\int_from_roman:n {\langle roman numeral \rangle}</code>
	Converts the $\langle roman numeral \rangle$ into the integer (base 10) representation and leaves this in the input stream. The $\langle roman numeral \rangle$ may be in upper or lower case; if the numeral is not valid then the resulting value will be -1 .

<hr/> <code>\int_from_base:nn</code> ☆ <hr/>	<code>\int_from_base:nn {\langle number \rangle} {\langle base \rangle}</code>
	Converts the $\langle number \rangle$ in $\langle base \rangle$ into the appropriate value in base 10. The $\langle number \rangle$ should consist of digits and letters (either lower or upper case), plus optionally a leading sign. The maximum $\langle base \rangle$ value is 36.

52 Viewing integers

<hr/>	
<code>\int_show:N</code>	<code>\int_show:N</code> $\langle integer \rangle$
<code>\int_show:c</code>	Displays the value of the $\langle integer \rangle$ on the terminal.
<hr/>	
<code>\int_show:n</code>	<code>\int_show:n</code> $\langle integer expression \rangle$
New: 2011-11-22	Displays the result of evaluating the $\langle integer expression \rangle$ on the terminal.
<hr/>	

53 Constant integers

<hr/>	
<code>\c_minus_one</code>	Integer values used with primitive tests and assignments: self-terminating nature makes these more convenient and faster than literal numbers.
<code>\c_zero</code>	
<code>\c_one</code>	
<code>\c_two</code>	
<code>\c_three</code>	
<code>\c_four</code>	
<code>\c_five</code>	
<code>\c_six</code>	
<code>\c_seven</code>	
<code>\c_eight</code>	
<code>\c_nine</code>	
<code>\c_ten</code>	
<code>\c_eleven</code>	
<code>\c_twelve</code>	
<code>\c_thirteen</code>	
<code>\c_fourteen</code>	
<code>\c_fifteen</code>	
<code>\c_sixteen</code>	
<code>\c_thirty_two</code>	
<code>\c_one_hundred</code>	
<code>\c_two_hundred_fifty_five</code>	
<code>\c_two_hundred_fifty_six</code>	
<code>\c_one_thousand</code>	
<code>\c_ten_thousand</code>	
<hr/>	
<code>\c_max_int</code>	The maximum value that can be stored as an integer.
<hr/>	
<code>\c_max_register_int</code>	Maximum number of registers.
<hr/>	

54 Scratch integers

`\l_tmpa_int`
`\l_tmpb_int`
`\l_tmpc_int`

Scratch integer for local assignment. These are never used by the kernel code, and so are safe for use with any L^AT_EX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

`\g_tmpa_int`
`\g_tmpb_int`

Scratch integer for global assignment. These are never used by the kernel code, and so are safe for use with any L^AT_EX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

55 Internal functions

`\int_get_digits:n` ★

`\int_get_digits:n` $\langle value \rangle$

Parses the $\langle value \rangle$ to leave the absolute $\langle value \rangle$ in the input stream. This may therefore be used to remove multiple sign tokens from the $\langle value \rangle$ (which may be symbolic).

`\int_get_sign:n` ☆

`\int_get_sign:n` $\langle value \rangle$

Parses the $\langle value \rangle$ to leave a single sign token (either + or -) in the input stream. This may therefore be used to sanitise sign tokens from the $\langle value \rangle$ (which may be symbolic).

`\int_to_letter:n` ★

Updated: 2011-09-17

`\int_to_letter:n` $\langle integer\ value \rangle$

For $\langle integer\ values \rangle$ from 0 to 9, leaves the $\langle value \rangle$ in the input stream unchanged. For $\langle integer\ values \rangle$ from 10 to 35, leaves the appropriate upper case letter (from the standard English alphabet) in the input stream: for example, 10 is converted to A, 11 to B, *etc.*

`\int_to_roman:w` ★

`\int_to_roman:w` $\langle integer \rangle$ $\langle space \rangle$ or $\langle non-expandable\ token \rangle$

Converts $\langle integer \rangle$ to its lower case Roman representation. Expansion ends when a space or non-expandable token is found. Note that this function produces a string of letters with category code 12 and that protected functions *are* expanded by this process. Negative $\langle integer \rangle$ values result in no output, although the function does not terminate expansion until a suitable endpoint is found in the same way as for positive numbers.

T_EXhackers note: This is the T_EX primitive `\romannumeral` renamed.

<code>\if_num:w</code>	★	<code>\if_num:w <integer1> <relation> <integer2></code>
<code>\if_int_compare:w</code>	★	<code><true code></code>

`\else:`
`<false code>`
`\fi:`

Compare two integers using `<relation>`, which must be one of =, < or > with category code 12. The `\else:` branch is optional.

T_EXhackers note: These are both names for the T_EX primitive `\ifnum`.

<code>\if_case:w</code>	★	<code>\if_case:w <integer> <case0></code>
<code>\or</code>	★	<code>\or: <case1></code>
		<code>\or: ...</code>
		<code>\else: <default></code>

`\fi:`

Selects a case to execute based on the value of the `<integer>`. The first case (`<case0>`) is executed if `<integer>` is 0, the second (`<case1>`) if the `<integer>` is 1, *etc.* The `<integer>` may be a literal, a constant or an integer expression (*e.g.* using `\int_eval:n`).

T_EXhackers note: These are the T_EX primitives `\ifcase` and `\or`.

<code>\int_value:w</code>	★	<code>\int_value:w <integer></code>
		<code>\int_value:w <tokens> <optional space></code>

Expands `<tokens>` until an `<integer>` is formed. One space may be gobbled in the process.

T_EXhackers note: This is the T_EX primitive `\number`.

<code>\int_eval:w</code>	★	<code>\int_eval:w <intexpr> \int_eval_end:</code>
<code>\int_eval_end</code>	★	

Evaluates `<integer expression>` as described for `\int_eval:n`. The evaluation stops when an unexpandable token which is not a valid part of an integer is read or when `\int_eval_end:` is reached. The latter is gobbled by the scanner mechanism: `\int_eval_end:` itself is unexpandable but used correctly the entire construct is expandable.

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

<code>\if_int_odd:w</code>	★	<code>\if_int_odd:w <tokens> <optional space></code>
		<code><true code></code>
		<code>\else:</code>
		<code><true code></code>
		<code>\fi:</code>

Expands `<tokens>` until a non-numeric token or a space is found, and tests whether the resulting `<integer>` is odd. If so, `<true code>` is executed. The `\else:` branch is optional.

T_EXhackers note: This is the T_EX primitive `\ifodd`.

Part X

The l3skip package

Dimensions and skips

L^AT_EX3 provides two general length variables: `dim` and `skip`. Lengths stored as `dim` variables have a fixed length, whereas `skip` lengths have a rubber (stretch/shrink) component. In addition, the `muskip` type is available for use in math mode: this is a special form of `skip` where the lengths involved are determined by the current math font (in μ). There are common features in the creation and setting of length variables, but for clarity the functions are grouped by variable type.

56 Creating and initialising dim variables

<code>\dim_new:N</code>	<code>\dim_new:N <dimension></code>
<code>\dim_new:c</code>	Creates a new <i><dimension></i> or raises an error if the name is already taken. The declaration is global. The <i><dimension></i> will initially be equal to 0 pt.

<code>\dim_zero:N</code>	<code>\dim_zero:N <dimension></code>
<code>\dim_zero:c</code>	Sets <i><dimension></i> to 0 pt.
<code>\dim_gzero:N</code>	
<code>\dim_gzero:c</code>	

57 Setting dim variables

<code>\dim_add:Nn</code>	<code>\dim_add:Nn <dimension> {<dimension expression>}</code>
<code>\dim_add:cn</code>	Adds the result of the <i><dimension expression></i> to the current content of the <i><dimension></i> .
<code>\dim_gadd:Nn</code>	
<code>\dim_gadd:cn</code>	

Updated: 2011-10-22

<code>\dim_set:Nn</code>	<code>\dim_set:Nn <dimension> {<dimension expression>}</code>
<code>\dim_set:cn</code>	Sets <i><dimension></i> to the value of <i><dimension expression></i> , which must evaluate to a length with units.
<code>\dim_gset:Nn</code>	
<code>\dim_gset:cn</code>	

Updated: 2011-10-22

<code>\dim_set_eq:NN</code>	<code>\dim_set_eq:NN <dimension1> <dimension2></code>
<code>\dim_set_eq:(cN Nc cc)</code>	Sets the content of <i><dimension1></i> equal to that of <i><dimension2></i> .
<code>\dim_gset_eq:NN</code>	
<code>\dim_gset_eq:(cN Nc cc)</code>	

<hr/> <code>\dim_set_max:Nn</code> <code>\dim_set_max:cn</code> <code>\dim_gset_max:Nn</code> <code>\dim_gset_max:cn</code> <hr/> Updated: 2011-10-22 <hr/>	<code>\dim_set_max:Nn <dimension> {<dimension expression>}</code> Compares the current value of the $\langle dimension \rangle$ with that of the $\langle dimension expression \rangle$, and sets the $\langle dimension \rangle$ to the larger of these two value.
<hr/> <code>\dim_set_min:Nn</code> <code>\dim_set_min:cn</code> <code>\dim_gset_min:Nn</code> <code>\dim_gset_min:cn</code> <hr/> Updated: 2011-10-22 <hr/>	<code>\dim_set_min:Nn <dimension> {<dimension expression>}</code> Compares the current value of the $\langle dimension \rangle$ with that of the $\langle dimension expression \rangle$, and sets the $\langle dimension \rangle$ to the smaller of these two value.
<hr/> <code>\dim_sub:Nn</code> <code>\dim_sub:cn</code> <code>\dim_gsub:Nn</code> <code>\dim_gsub:cn</code> <hr/> Updated: 2011-10-22 <hr/>	<code>\dim_sub:Nn <dimension> {<dimension expression>}</code> Subtracts the result of the $\langle dimension expression \rangle$ to the current content of the $\langle dimension \rangle$.

58 Utilities for dimension calculations

<hr/> <code>\dim_abs:n</code> ★ <hr/> Updated: 2011-10-22 <hr/>	<code>\dim_abs:n {<dimexpr>}</code> Converts the $\langle dimexpr \rangle$ to its absolute value, leaving the result in the input stream as an $\langle dimension denotation \rangle$.
<hr/> <code>\dim_ratio:nn</code> ★ <hr/> Updated: 2011-10-22 <hr/>	<code>\dim_ratio:nn {<dimexpr₁>} {<dimexpr₂>}</code> Parses the two $\langle dimension expressions \rangle$ and converts the ratio of the two to a form suitable for use inside a $\langle dimension expression \rangle$. This ratio is then left in the input stream, allowing syntax such as

```
\dim_set:Nn \l_my_dim
{ 10 pt * \dim_ratio:nn { 5 pt } { 10 pt } }
```

The output of `\dim_ratio:nn` on full expansion is a ration expression between two integers, with all distances converted to scaled points. Thus

```
\tl_set:Nx \l_my_tl { \dim_ratio:nn { 5 pt } { 10 pt } }
\tl_show:N \l_my_tl
```

will display 327680/655360 on the terminal.

59 Dimension expression conditionals

<code>\dim_compare_p:nNn</code> ★	<code>\dim_compare_p:nNn {<dimexpr₁>} <relation> {<dimexpr₂>}</code>
<code>\dim_compare:nNnTF</code> ★	<code>\dim_compare:nNnTF</code> <code>{<dimexpr₁>} <relation> {<dimexpr₂>}</code> <code>{<true code>} {<false code>}</code>

This function first evaluates each of the *<dimension expressions>* as described for `\dim_eval:n`. The two results are then compared using the *<relation>*:

Equal	=
Greater than	>
Less than	<

<code>\dim_compare_p:n</code> ★	<code>\dim_compare_p:n { <dimexpr₁> <relation> <dimexpr₂> }</code>
<code>\dim_compare:nTF</code> ★	<code>\dim_compare:nTF</code> <code>{ <dimexpr₁> <relation> <dimexpr₂> }</code> <code>{<true code>} {<false code>}</code>

This function first evaluates each of the *<dimension expressions>* as described for `\dim_eval:n`. The two results are then compared using the *<relation>*:

Equal	= or ==
Greater than or equal to	=>
Greater than	>
Less than or equal to	=<
Less than	<
Not equal	!=

60 Dimension expression loops

<code>\dim_do_while:nNnn</code> ☆	<code>\dim_do_while:nNnn {<dimexpr₁>} <relation> {<dimexpr₂>} {<code>}</code>
-----------------------------------	---

Evaluates the relationship between the two *<dimension expressions>* as described for `\dim_compare:nNnTF`, and then places the *<code>* in the input stream if the *<relation>* is **true**. After the *<code>* has been processed by T_EX the test will be repeated, and a loop will occur until the test is **false**.

<code>\dim_do_until:nNnn</code> ☆	<code>\dim_do_until:nNnn {<dimexpr₁>} <relation> {<dimexpr₂>} {<code>}</code>
-----------------------------------	---

Evaluates the relationship between the two *<dimension expressions>* as described for `\dim_compare:nNnTF`, and then places the *<code>* in the input stream if the *<relation>* is **false**. After the *<code>* has been processed by T_EX the test will be repeated, and a loop will occur until the test is **true**.

<hr/> <code>\dim_until_do:nNnn</code> ☆ <hr/>	$\backslash\mathrm{dim_until_do:nNnn}\{\langle\mathrm{dimexpr}_1\rangle\}\langle\mathrm{relation}\rangle\{\langle\mathrm{dimexpr}_2\rangle\}\{\langle\mathrm{code}\rangle\}$ <p>Places the $\langle\mathrm{code}\rangle$ in the input stream for T_EX to process, and then evaluates the relationship between the two $\langle\mathrm{dimension\ expressions}\rangle$ as described for <code>\dim_compare:nNnTF</code>. If the test is <code>false</code> then the $\langle\mathrm{code}\rangle$ will be inserted into the input stream again and a loop will occur until the $\langle\mathrm{relation}\rangle$ is <code>true</code>.</p>
<hr/> <code>\dim_while_do:nNnn</code> ☆ <hr/>	$\backslash\mathrm{dim_while_do:nNnn}\{\langle\mathrm{dimexpr}_1\rangle\}\langle\mathrm{relation}\rangle\{\langle\mathrm{dimexpr}_2\rangle\}\{\langle\mathrm{code}\rangle\}$ <p>Places the $\langle\mathrm{code}\rangle$ in the input stream for T_EX to process, and then evaluates the relationship between the two $\langle\mathrm{dimension\ expressions}\rangle$ as described for <code>\dim_compare:nNnTF</code>. If the test is <code>true</code> then the $\langle\mathrm{code}\rangle$ will be inserted into the input stream again and a loop will occur until the $\langle\mathrm{relation}\rangle$ is <code>false</code>.</p>
<hr/> <code>\dim_do_while:nn</code> ☆ <hr/>	$\backslash\mathrm{dim_do_while:nNnn}\{\langle\mathrm{dimexpr}_1\rangle\langle\mathrm{relation}\rangle\langle\mathrm{dimexpr}_2\rangle\}\{\langle\mathrm{code}\rangle\}$ <p>Evaluates the relationship between the two $\langle\mathrm{dimension\ expressions}\rangle$ as described for <code>\dim_compare:nTF</code>, and then places the $\langle\mathrm{code}\rangle$ in the input stream if the $\langle\mathrm{relation}\rangle$ is <code>true</code>. After the $\langle\mathrm{code}\rangle$ has been processed by T_EX the test will be repeated, and a loop will occur until the test is <code>false</code>.</p>
<hr/> <code>\dim_do_until:nn</code> ☆ <hr/>	$\backslash\mathrm{dim_do_until:nn}\{\langle\mathrm{dimexpr}_1\rangle\langle\mathrm{relation}\rangle\langle\mathrm{dimexpr}_2\rangle\}\{\langle\mathrm{code}\rangle\}$ <p>Evaluates the relationship between the two $\langle\mathrm{dimension\ expressions}\rangle$ as described for <code>\dim_compare:nTF</code>, and then places the $\langle\mathrm{code}\rangle$ in the input stream if the $\langle\mathrm{relation}\rangle$ is <code>false</code>. After the $\langle\mathrm{code}\rangle$ has been processed by T_EX the test will be repeated, and a loop will occur until the test is <code>true</code>.</p>
<hr/> <code>\dim_until_do:nn</code> ☆ <hr/>	$\backslash\mathrm{dim_until_do:nn}\{\langle\mathrm{dimexpr}_1\rangle\langle\mathrm{relation}\rangle\langle\mathrm{dimexpr}_2\rangle\}\{\langle\mathrm{code}\rangle\}$ <p>Places the $\langle\mathrm{code}\rangle$ in the input stream for T_EX to process, and then evaluates the relationship between the two $\langle\mathrm{dimension\ expressions}\rangle$ as described for <code>\dim_compare:nTF</code>. If the test is <code>false</code> then the $\langle\mathrm{code}\rangle$ will be inserted into the input stream again and a loop will occur until the $\langle\mathrm{relation}\rangle$ is <code>true</code>.</p>
<hr/> <code>\dim_while_do:nn</code> ☆ <hr/>	$\backslash\mathrm{dim_while_do:nn}\{\langle\mathrm{dimexpr}_1\rangle\langle\mathrm{relation}\rangle\langle\mathrm{dimexpr}_2\rangle\}\{\langle\mathrm{code}\rangle\}$ <p>Places the $\langle\mathrm{code}\rangle$ in the input stream for T_EX to process, and then evaluates the relationship between the two $\langle\mathrm{dimension\ expressions}\rangle$ as described for <code>\dim_compare:nTF</code>. If the test is <code>true</code> then the $\langle\mathrm{code}\rangle$ will be inserted into the input stream again and a loop will occur until the $\langle\mathrm{relation}\rangle$ is <code>false</code>.</p>

61 Using dim expressions and variables

<code>\dim_eval:n</code> ★	<code>\dim_eval:n {⟨dimension expression⟩}</code>
Updated: 2011-10-22	Evaluates the <i>⟨dimension expression⟩</i> , expanding any dimensions and token list variables within the <i>⟨expression⟩</i> to their content (without requiring <code>\dim_use:N/\tl_use:N</code>) and applying the standard mathematical rules. The result of the calculation is left in the input stream as a <i>⟨dimension denotation⟩</i> after two expansions. This will be expressed in points (<code>\pt</code>), and will require suitable termination if used in a TeX-style assignment as it is <i>not</i> an <i>⟨internal dimension⟩</i> .

<code>\dim_use:N</code> ★	<code>\dim_use:N ⟨dimension⟩</code>
<code>\dim_use:c</code> ★	Recovers the content of a <i>⟨dimension⟩</i> and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Can be omitted in places where a <i>⟨dimension⟩</i> is required (such as in the argument of <code>\dim_eval:n</code>).

TeXhackers note: `\dim_use:N` is the TeX primitive `\the`: this is one of several L^AT_EX3 names for this primitive.

62 Viewing dim variables

<code>\dim_show:N</code>	<code>\dim_show:N ⟨dimension⟩</code>
<code>\dim_show:c</code>	Displays the value of the <i>⟨dimension⟩</i> on the terminal.

<code>\dim_show:n</code>	<code>\dim_show:n ⟨dimension expression⟩</code>
New: 2011-11-22	Displays the result of evaluating the <i>⟨dimension expression⟩</i> on the terminal.

63 Constant dimensions

<code>\c_max_dim</code>	The maximum value that can be stored as a dimension or skip (these are equivalent).
<code>\c_zero_dim</code>	A zero length as a dimension or a skip (these are equivalent).

64 Scratch dimensions

`\l_tmpa_dim`
`\l_tmpb_dim`
`\l_tmpc_dim`

Scratch dimension for local assignment. These are never used by the kernel code, and so are safe for use with any L^AT_EX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

`\g_tmpa_dim`
`\g_tmpb_dim`

Scratch dimension for global assignment. These are never used by the kernel code, and so are safe for use with any L^AT_EX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

65 Creating and initialising skip variables

`\skip_new:N`
`\skip_new:c`

`\skip_new:N` $\langle skip \rangle$

Creates a new $\langle skip \rangle$ or raises an error if the name is already taken. The declaration is global. The $\langle skip \rangle$ will initially be equal to 0 pt.

`\skip_zero:N`
`\skip_zero:c`
`\skip_gzero:N`
`\skip_gzero:c`

`\skip_zero:N` $\langle skip \rangle$

Sets $\langle skip \rangle$ to 0 pt.

66 Setting skip variables

`\skip_add:Nn`
`\skip_add:cn`
`\skip_gadd:Nn`
`\skip_gadd:cn`

`\skip_add:Nn` $\langle skip \rangle$ $\{ \langle skip \text{ expression} \rangle \}$

Adds the result of the $\langle skip \text{ expression} \rangle$ to the current content of the $\langle skip \rangle$.

Updated: 2011-10-22

`\skip_set:Nn`
`\skip_set:cn`
`\skip_gset:Nn`
`\skip_gset:cn`

`\skip_set:Nn` $\langle skip \rangle$ $\{ \langle skip \text{ expression} \rangle \}$

Sets $\langle skip \rangle$ to the value of $\langle skip \text{ expression} \rangle$, which must evaluate to a length with units and may include a rubber component (for example 1 cm plus 0.5 cm).

Updated: 2011-10-22

`\skip_set_eq:NN`
`\skip_set_eq:(cN|Nc|cc)`
`\skip_gset_eq:NN`
`\skip_gset_eq:(cN|Nc|cc)`

`\skip_set_eq:NN` $\langle skip1 \rangle$ $\langle skip2 \rangle$

Sets the content of $\langle skip1 \rangle$ equal to that of $\langle skip2 \rangle$.

<code>\skip_sub:Nn</code>	<code>\skip_sub:Nn <skip> {<skip expression>}</code>
<code>\skip_sub:cn</code>	
<code>\skip_gsub:Nn</code>	Subtracts the result of the <i><skip expression></i> to the current content of the <i><skip></i> .
<code>\skip_gsub:cn</code>	

Updated: 2011-10-22

67 Skip expression conditionals

<code>\skip_if_eq_p:n</code> ★	<code>\skip_if_eq_p:n {<skipexpr₁>} {<skipexpr₂>}</code>
<code>\skip_if_eq:nnTF</code> ★	<code>\dim_compare:nTF</code> <code>{<skip expr₁>} {<skip expr₂>}</code> <code>{<true code>} {<false code>}</code>

This function first evaluates each of the *<skip expressions>* as described for `\skip_eval:n`. The two results are then compared for exact equality, *i.e.* both the fixed and rubber components must be the same for the test to be true.

<code>\skip_if_infinite_glue_p:n</code> ★	<code>\skip_if_infinite_glue_p:n {<skipexpr>}</code>
<code>\skip_if_infinite_glue:nnTF</code> ★	<code>\skip_if_infinite_glue:nnTF {<skipexpr>} {<true code>} {<false code>}</code>

Evaluates the *<skip expression>* as described for `\skip_eval:n`, and then tests if this contains an infinite stretch or shrink component (or both).

68 Using skip expressions and variables

<code>\skip_eval:n</code> ★	<code>\skip_eval:n {<skip expression>}</code>
-----------------------------	---

Updated: 2011-10-22

Evaluates the *<skip expression>*, expanding any skips and token list variables within the *<expression>* to their content (without requiring `\skip_use:N/\tl_use:N`) and applying the standard mathematical rules. The result of the calculation is left in the input stream as a *<glue denotation>* after two expansions. This will be expressed in points (`\pt`), and will require suitable termination if used in a TeX-style assignment as it is *not* an *<internal glue>*.

<code>\skip_use:N</code> ★	<code>\skip_use:N <skip></code>
<code>\skip_use:c</code> ★	

Recovers the content of a *<skip>* and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Can be omitted in places where a *<dimension>* is required (such as in the argument of `\skip_eval:n`).

TeXhackers note: `\skip_use:N` is the TeX primitive `\the`: this is one of several L^AT_EX3 names for this primitive.

69 Viewing skip variables

<hr/> <code>\skip_show:N</code> <hr/>	<code>\skip_show:N</code> $\langle skip \rangle$
<code>\skip_show:c</code> <hr/>	Displays the value of the $\langle skip \rangle$ on the terminal.
<hr/> <code>\skip_show:n</code> <hr/>	<code>\skip_show:n</code> $\langle skip\ expression \rangle$
<code>New: 2011-11-22</code> <hr/>	Displays the result of evaluating the $\langle skip\ expression \rangle$ on the terminal.

70 Constant skips

<hr/> <code>\c_max_skip</code> <hr/>	The maximum value that can be stored as a dimension or skip (these are equivalent).
<hr/> <code>\c_zero_skip</code> <hr/>	A zero length as a dimension or a skip (these are equivalent).

71 Scratch skips

<hr/> <code>\l_tmpa_skip</code> <code>\l_tmpb_skip</code> <code>\l_tmpc_skip</code> <hr/>	Scratch skip for local assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
<hr/> <code>\g_tmpa_skip</code> <code>\g_tmpb_skip</code> <hr/>	Scratch skip for global assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

72 Creating and initialising muskip variables

<hr/> <code>\muskip_new:N</code> <hr/>	<code>\muskip_new:N</code> $\langle muskip \rangle$
<code>\muskip_new:c</code> <hr/>	Creates a new $\langle muskip \rangle$ or raises an error if the name is already taken. The declaration is global. The $\langle muskip \rangle$ will initially be equal to 0 mu.
<hr/> <code>\muskip_zero:N</code> <code>\muskip_zero:c</code> <code>\muskip_gzero:N</code> <code>\muskip_gzero:c</code> <hr/>	<code>\skip_zero:N</code> $\langle muskip \rangle$ Sets $\langle muskip \rangle$ to 0 mu.

73 Setting muskip variables

<code>\muskip_add:Nn</code>	<code>\muskip_add:Nn <muskip> {<muskip expression>}</code>
<code>\muskip_add:cn</code>	Adds the result of the $\langle muskip expression \rangle$ to the current content of the $\langle muskip \rangle$.
<code>\muskip_gadd:Nn</code>	
<code>\muskip_gadd:cn</code>	
Updated: 2011-10-22	
<code>\muskip_set:Nn</code>	<code>\muskip_set:Nn <muskip> {<muskip expression>}</code>
<code>\muskip_set:cn</code>	Sets $\langle muskip \rangle$ to the value of $\langle muskip expression \rangle$, which must evaluate to a math length with units and may include a rubber component (for example 1 mu plus 0.5 mu).
<code>\muskip_gset:Nn</code>	
<code>\muskip_gset:cn</code>	
Updated: 2011-10-22	

<code>\muskip_set_eq:NN</code>	<code>\muskip_set_eq:NN <muskip1> <muskip2></code>
<code>\muskip_set_eq:(cN Nc cc)</code>	Sets the content of $\langle muskip1 \rangle$ equal to that of $\langle muskip2 \rangle$.
<code>\muskip_gset_eq:NN</code>	
<code>\muskip_gset_eq:(cN Nc cc)</code>	

<code>\muskip_sub:Nn</code>	<code>\muskip_sub:Nn <muskip> {<muskip expression>}</code>
<code>\muskip_sub:cn</code>	Subtracts the result of the $\langle muskip expression \rangle$ to the current content of the $\langle skip \rangle$.
<code>\muskip_gsub:Nn</code>	
<code>\muskip_gsub:cn</code>	
Updated: 2011-10-22	

74 Using muskip expressions and variables

<code>\muskip_eval:n</code> ★	<code>\muskip_eval:n {<muskip expression>}</code>
Updated: 2011-10-22	Evaluates the $\langle muskip expression \rangle$, expanding any skips and token list variables within the $\langle expression \rangle$ to their content (without requiring <code>\muskip_use:N/\tl_use:N</code>) and applying the standard mathematical rules. The result of the calculation is left in the input stream as a $\langle muglue denotation \rangle$ after two expansions. This will be expressed in mu, and will require suitable termination if used in a TeX-style assignment as it is <i>not</i> an $\langle internal muglue \rangle$.
<code>\muskip_use:N</code> ★	<code>\muskip_use:N <muskip></code>
<code>\muskip_use:c</code> ★	Recovers the content of a $\langle skip \rangle$ and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Can be omitted in places where a $\langle dimension \rangle$ is required (such as in the argument of <code>\muskip_eval:n</code>).

TeXhackers note: `\muskip_use:N` is the TeX primitive `\the`: this is one of several L^AT_EX3 names for this primitive.

75 Inserting skips into the output

<code>\skip_horizontal:N</code>	<code>\skip_horizontal:N <skip></code>
<code>\skip_horizontal:(c n)</code>	<code>\skip_horizontal:n {<skipexpr>}</code>
Updated: 2011-10-22	Inserts a horizontal <i><skip></i> into the current list.
T_EXhackers note: <code>\skip_horizontal:N</code> is the T _E X primitive <code>\hskip</code> renamed.	

<code>\skip_vertical:N</code>	<code>\skip_vertical:N <skip></code>
<code>\skip_vertical:(c n)</code>	<code>\skip_vertical:n {<skipexpr>}</code>
Updated: 2011-10-22	Inserts a vertical <i><skip></i> into the current list.

T_EXhackers note: `\skip_vertical:N` is the T_EX primitive `\vskip` renamed.

76 Viewing muskip variables

<code>\muskip_show:N</code>	<code>\muskip_show:N <muskip></code>
<code>\muskip_show:c</code>	Displays the value of the <i><muskip></i> on the terminal.
<code>\muskip_show:n</code>	<code>\muskip_show:n <muskip expression></code>
New: 2011-11-22	Displays the result of evaluating the <i><muskip expression></i> on the terminal.

77 Internal functions

<code>\if_dim:w</code>	<code>\if_dim:w <dimen1> <relation> <dimen1></code>
	<code><true code></code>
	<code>\else:</code>
	<code><false></code>
	<code>\fi:</code>
	Compare two dimensions. The <i><relation></i> is one of <, = or > with category code 12.

T_EXhackers note: This is the T_EX primitive `\ifdim`.

<code>\dim_eval:w</code> ★	<code>\dim_eval:w <dimexpr> \dim_eval_end:</code>
<code>\dim_eval_end</code> ★	Evaluates <i><dimension expression></i> as described for <code>\dim_eval:n</code> . The evaluation stops when an unexpandable token which is not a valid part of a dimension is read or when <code>\dim_eval_end:</code> is reached. The latter is gobbled by the scanner mechanism: <code>\dim_eval_end:</code> itself is unexpandable but used correctly the entire construct is expandable.

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

78 Experimental skip functions

<code>\skip_split_finite_else_action:nnNN</code>	<code>\skip_split_finite_else_action:nnNN {<skipexpr>} {<action>}</code> <code> <dimen1> <dimen2></code>
--	--

Updated: 2011-10-22

Checks if the $\langle skipexpr \rangle$ contains finite glue. If it does then it assigns $\langle dimen1 \rangle$ the stretch component and $\langle dimen2 \rangle$ the shrink component. If it contains infinite glue set $\langle dimen1 \rangle$ and $\langle dimen2 \rangle$ to 0pt and place #2 into the input stream: this is usually an error or warning message of some sort.

79 Internal functions

<code>\dim_strip_bp:n</code> ★ <code>\dim_strip_pt:n</code> ★	<code>\dim_strip_bp:n {<dimension expression>}</code> <code>\dim_strip_pt:n {<dimension expression>}</code>
--	--

New: 2011-11-11

Evaluates the $\langle dimension expression \rangle$, expanding any dimensions and token list variables within the $\langle expression \rangle$ to their content (without requiring `\dim_use:N/\tl_use:N`) and applying the standard mathematical rules. The magnitude of the result, expressed in big points (**bp**) or points (**pt**), will be left in the input stream with *no units*. If the decimal part of the magnitude is zero, this will be omitted.

If the $\{ \langle dimension expression \rangle \}$ contains additional units, these will be ignored, so for example

```
\dim_strip_pt:n { 1 bp pt }
```

will leave 1.00374 in the input stream (*i.e.* the magnitude of one “big point” when converted to points).

Part XI

The l3tl package

Token lists

T_EX works with tokens, and L^AT_EX3 therefore provides a number of functions to deal with token lists. Token lists may be present directly in the argument to a function:

```
\foo:n { a collection of \tokens }
```

or may be stored for processing in a so-called “token list variable”, which have the suffix `tl`: the argument to a function:

```
\foo:N \l_some_tl
```

In both cases, functions are available to test and manipulate the lists of tokens, and these have the module prefix `tl`. In many cases, function which can be applied to token list variables are paired with similar functions for application to explicit lists of tokens: the two “views” of a token list are therefore collected together here.

A token list can be seen either as a list of “items”, or a list of “tokens”. An item is whatever `\use_none:n` grabs as its argument: either a single token or a brace group, with optional leading explicit space characters (each item is thus itself a token list). A token is either a normal `N` argument, or `{`, `{`, or `}` (assuming normal T_EX category codes). Thus for example

```
{ Hello } ~ world
```

contains six items (Hello, w, o, r, l and d), but thirteen tokens (`{`, H, e, l, l, o, `}`, `~`, w, o, r, l and d). Functions which act on items are often faster than their analogue acting directly on tokens.

80 Creating and initialising token list variables

```
\tl_new:N
\tl_new:c
```

```
\tl_new:N <tl var>
```

Creates a new *<tl var>* or raises an error if the name is already taken. The declaration is global. The *<tl var>* will initially be empty.

```
\tl_const:Nn
\tl_const:(Nx|cn|cx)
```

```
\tl_const:Nn <tl var> {<token list>}
```

Creates a new constant *<tl var>* or raises an error if the name is already taken. The value of the *<tl var>* will be set globally to the *<token list>*.

```
\tl_clear:N
\tl_clear:c
\tl_gclear:N
\tl_gclear:c
```

```
\tl_clear:N <tl var>
```

Clears all entries from the *<tl var>* within the scope of the current T_EX group.

<code>\tl_clear_new:N</code>	<code>\tl_clear_new:N <tl var></code>
<code>\tl_clear_new:c</code>	
<code>\tl_gclear_new:N</code>	Ensures that the <code><tl var></code> exists globally by applying <code>\tl_new:N</code> if necessary, then applies
<code>\tl_gclear_new:c</code>	<code>\tl_(g)clear:N</code> to leave the <code><tl var></code> empty.

<code>\tl_set_eq:NN</code>	<code>\tl_set_eq:NN <tl var1> <tl var2></code>
<code>\tl_set_eq:(cN Nc cc)</code>	Sets the content of <code><tl var1></code> equal to that of <code><tl var2></code> .
<code>\tl_gset_eq:NN</code>	
<code>\tl_gset_eq:(cN Nc cc)</code>	

81 Adding data to token list variables

<code>\tl_set:Nn</code>	<code>\tl_set:Nn <tl var> {<tokens>}</code>
<code>\tl_set:(NV Nv No Nf Nx cn NV Nv co cf cx)</code>	
<code>\tl_gset:Nn</code>	
<code>\tl_gset:(NV Nv No Nf Nx cn cV cV co cf cx)</code>	

Sets `<tl var>` to contain `<tokens>`, removing any previous content from the variable.

<code>\tl_put_left:Nn</code>	<code>\tl_put_left:Nn <tl var> {<tokens>}</code>
<code>\tl_put_left:(NV No Nx cn cV co cx)</code>	
<code>\tl_gput_left:Nn</code>	
<code>\tl_gput_left:(NV No Nx cn cV co cx)</code>	

Appends `<tokens>` to the left side of the current content of `<tl var>`.

<code>\tl_put_right:Nn</code>	<code>\tl_put_right:Nn <tl var> {<tokens>}</code>
<code>\tl_put_right:(NV No Nx cn cV co cx)</code>	
<code>\tl_gput_right:Nn</code>	
<code>\tl_gput_right:(NV No Nx cn cV co cx)</code>	

Appends `<tokens>` to the right side of the current content of `<tl var>`.

82 Modifying token list variables

<code>\tl_replace_once:Nnn</code>	<code>\tl_replace_once:Nnn <tl var> {<old tokens>} {<new tokens>}</code>
<code>\tl_replace_once:cnn</code>	
<code>\tl_greplace_once:Nnn</code>	Replaces the first (leftmost) occurrence of <code><old tokens></code> in the <code><tl var></code> with <code><new tokens></code> .
<code>\tl_greplace_once:cnn</code>	<code><Old tokens></code> cannot contain <code>{</code> , <code>}</code> or <code>#</code> (assuming normal T _E X category codes).

Updated: 2011-08-11

<code>\tl_replace_all:Nnn</code>	<code>\tl_replace_all:Nnn <tl var> {<old tokens>} {<new tokens>}</code>
<code>\tl_replace_all:cnn</code>	
<code>\tl_greplace_all:Nnn</code>	Replaces all occurrences of <i><old tokens></i> in the <i><tl var></i> with <i><new tokens></i> . <i><Old tokens></i> cannot contain <code>{</code> , <code>}</code> or <code>#</code> (assuming normal T _E X category codes). As this function operates from left to right, the pattern <i><old tokens></i> may remain after the replacement (see <code>\tl_remove_all:Nn</code> for an example). The assignment is restricted to the current T _E X group.
<code>\tl_greplace_all:cnn</code>	
Updated: 2011-08-11	

<code>\tl_remove_once:Nn</code>	<code>\tl_remove_once:Nn <tl var> {<tokens>}</code>
<code>\tl_remove_once:cn</code>	
<code>\tl_gremove_once:Nn</code>	Removes the first (leftmost) occurrence of <i><tokens></i> from the <i><tl var></i> . <i><Tokens></i> cannot contain <code>{</code> , <code>}</code> or <code>#</code> (assuming normal T _E X category codes).
<code>\tl_gremove_once:cn</code>	
Updated: 2011-08-11	

<code>\tl_remove_all:Nn</code>	<code>\tl_remove_all:Nn <tl var> {<tokens>}</code>
<code>\tl_remove_all:cn</code>	
<code>\tl_gremove_all:Nn</code>	Removes all occurrences of <i><tokens></i> from the <i><tl var></i> . <i><Tokens></i> cannot contain <code>{</code> , <code>}</code> or <code>#</code> (assuming normal T _E X category codes). As this function operates from left to right, the pattern <i><tokens></i> may remain after the removal, for instance,
<code>\tl_gremove_all:cn</code>	
Updated: 2011-08-11	
<code>\tl_set:Nn \l_tmpa_tl {abbccd} \tl_remove_all:Nn \l_tmpa_tl {bc}</code> will result in <code>\l_tmpa_tl</code> containing <code>abcd</code> .	

83 Reassigning token list category codes

<code>\tl_set_rescan:Nnn</code>	<code>\tl_set_rescan:Nnn <tl var> {<setup>} {<tokens>}</code>
<code>\tl_set_rescan:(Nno Nnx cnn cno cnx)</code>	
<code>\tl_gset_rescan:Nnn</code>	
<code>\tl_gset_rescan:(Nno Nnx cnn cno cnx)</code>	
Updated: 2011-12-18	
Sets <i><tl var></i> to contain <i><tokens></i> , applying the category code régime specified in the <i><setup></i> before carrying out the assignment. This allows the <i><tl var></i> to contain material with category codes other than those that apply when <i><tokens></i> are absorbed. See also <code>\tl_rescan:nn</code> .	

<code>\tl_rescan:nn</code>	<code>\tl_rescan:nn {<setup>} {<tokens>}</code>
Updated: 2011-12-18	Rescans <i><tokens></i> applying the category code régime specified in the <i><setup></i> , and leaves the resulting tokens in the input stream. See also <code>\tl_set_rescan:Nnn</code> .

84 Reassigning token list character codes

`\tl_to_lowercase:n`

`\tl_to_lowercase:n {⟨tokens⟩}`

Works through all of the *⟨tokens⟩*, replacing each character with the lower case equivalent as defined by `\char_set_lccode:nn`. Characters with no defined lower case character code are left unchanged. This process does not alter the category code assigned to the *⟨tokens⟩*.

T_EXhackers note: This is the T_EX primitive `\lowercase` renamed. As a result, this function takes place on execution and not on expansion.

`\tl_to_uppercase:n`

`\tl_to_uppercase:n {⟨tokens⟩}`

Works through all of the *⟨tokens⟩*, replacing each character with the upper case equivalent as defined by `\char_set_uccode:nn`. Characters with no defined lower case character code are left unchanged. This process does not alter the category code assigned to the *⟨tokens⟩*.

T_EXhackers note: This is the T_EX primitive `\uppercase` renamed. As a result, this function takes place on execution and not on expansion.

85 Token list conditionals

`\tl_if_blank_p:n` ★

`\tl_if_blank_p:n {⟨token list⟩}`

`\tl_if_blank_p:(V|o)` ★

`\tl_if_blank:nTF {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}`

`\tl_if_blank:nTF` ★

`\tl_if_blank:(V|o)TF` ★

Tests if the *⟨token list⟩* consists only of blank spaces (*i.e.* contains no item). The test is **true** if *⟨token list⟩* is zero or more explicit tokens of character code 32 and category code 10, and is **false** otherwise.

`\tl_if_empty_p:N` ★

`\tl_if_empty_p:N ⟨tl var⟩`

`\tl_if_empty_p:c` ★

`\tl_if_empty:NTF ⟨tl var⟩ {⟨true code⟩} {⟨false code⟩}`

`\tl_if_empty:NTF` ★

`\tl_if_empty:cTF` ★

Tests if the *⟨token list variable⟩* is entirely empty (*i.e.* contains no tokens at all).

`\tl_if_empty_p:n` ★

`\tl_if_empty_p:n {⟨token list⟩}`

`\tl_if_empty_p:(V|o|x)` ★

`\tl_if_empty:nTF {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}`

`\tl_if_empty:nTF` ★

`\tl_if_empty:(V|o|x)TF` ★

Tests if the *⟨token list⟩* is entirely empty (*i.e.* contains no tokens at all). All versions of these functions are fully expandable (including those involving an **x**-type expansion).

<code>\tl_if_eq_p:NN</code> ★ <code>\tl_if_eq_p:(Nc cN cc)</code> ★ <code>\tl_if_eq:NNTF</code> ★ <code>\tl_if_eq:(Nc cN cc)TF</code> ★	<code>\tl_if_eq_p:NN</code> $\{\langle tl\ var_1 \rangle\} \{\langle tl\ var_2 \rangle\}$ <code>\tl_if_eq:NNTF</code> $\{\langle tl\ var_1 \rangle\} \{\langle tl\ var_2 \rangle\} \{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$ <p>Compares the content of two $\langle token\ list\ variables \rangle$ and is logically true if the two contain the same list of tokens (<i>i.e.</i> identical in both the list of characters they contain and the category codes of those characters). Thus for example</p>
--	--

```

\tl_set:Nn \l_tmpa_tl { abc }
\tl_set:Nx \l_tmpb_tl { \tl_to_str:n { abc } }
\tl_if_eq_p:NN \l_tmpa_tl \l_tmpb_tl

```

is logically **false**.

<code>\tl_if_eq:nnTF</code>	<code>\tl_if_eq:nnTF</code> $\langle token\ list_1 \rangle \{\langle token\ list_2 \rangle\} \{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$ <p>Tests if $\langle token\ list_1 \rangle$ and $\langle token\ list_2 \rangle$ are equal, both in respect of character codes and category codes.</p>
-----------------------------	---

<code>\tl_if_in:NnTF</code> <code>\tl_if_in:cnTF</code>	<code>\tl_if_in:NnTF</code> $\langle tl\ var \rangle \{\langle token\ list \rangle\} \{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$ <p>Tests if the $\langle token\ list \rangle$ is found in the content of the $\langle token\ list\ variable \rangle$. The $\langle token\ list \rangle$ cannot contain the tokens <code>{</code>, <code>}</code> or <code>#</code> (assuming the usual T_EX category codes apply).</p>
--	---

<code>\tl_if_in:nnTF</code> <code>\tl_if_in:(Vn on no)TF</code>	<code>\tl_if_in:nnTF</code> $\{\langle token\ list_1 \rangle\} \{\langle token\ list_2 \rangle\} \{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$ <p>Tests if $\langle token\ list_2 \rangle$ is found inside $\langle token\ list_1 \rangle$. The $\langle token\ list \rangle$ cannot contain the tokens <code>{</code>, <code>}</code> or <code>#</code> (assuming the usual T_EX category codes apply).</p>
--	--

<code>\tl_if_single_p:N</code> ★ <code>\tl_if_single_p:c</code> ★ <code>\tl_if_single:NTF</code> ★ <code>\tl_if_single:cTF</code> ★	<code>\tl_if_single_p:N</code> $\{\langle tl\ var \rangle\}$ <code>\tl_if_single:NNTF</code> $\{\langle tl\ var \rangle\} \{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$ <p>Tests if the content of the $\langle tl\ var \rangle$ consists of a single item, <i>i.e.</i> is either a single normal token (excluding spaces, and brace tokens) or a single brace group, surrounded by optional spaces on both sides. In other words, such a token list has length 1 according to <code>\tl_length:N</code>.</p>
--	--

Updated: 2011-08-13

<code>\tl_if_single_p:n</code> ★ <code>\tl_if_single:nTF</code> ★	<code>\tl_if_single_p:n</code> $\{\langle token\ list \rangle\}$ <code>\tl_if_single:nNTF</code> $\{\langle token\ list \rangle\} \{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$ <p>Tests if the token list has exactly one item, <i>i.e.</i> is either a single normal token or a single brace group, surrounded by optional spaces on both sides. In other words, such a token list has length 1 according to <code>\tl_length:n</code>.</p>
--	---

Updated: 2011-08-13

<code>\tl_if_single_token_p:n</code> ★ <code>\tl_if_single_token:nTF</code> ★	<code>\tl_if_single_token_p:n</code> $\{\langle token\ list \rangle\}$ <code>\tl_if_single_token:nNTF</code> $\{\langle token\ list \rangle\} \{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$ <p>Tests if the token list consists of exactly one token, <i>i.e.</i> is either a single space character or a single “normal” token. Token groups (<code>{...}</code>) are not single tokens.</p>
--	---

New: 2011-08-11

86 Mapping to token lists

<hr/>	
<code>\tl_map_function:NN</code> ☆	<code>\tl_map_function:NN <tl var> <function></code>
<code>\tl_map_function:cN</code> ☆	
<hr/>	Applies <i><function></i> to every <i><item></i> in the <i><tl var></i> . The <i><function></i> will receive one argument for each iteration. This may be a number of tokens if the <i><item></i> was stored within braces. Hence the <i><function></i> should anticipate receiving n-type arguments. See also <code>\tl_map_function:nN</code> .
<hr/>	
<code>\tl_map_function:nN</code> ☆	<code>\tl_map_function:nN <token list> <function></code>
<hr/>	Applies <i><function></i> to every <i><item></i> in the <i><token list></i> . The <i><function></i> will receive one argument for each iteration. This may be a number of tokens if the <i><item></i> was stored within braces. Hence the <i><function></i> should anticipate receiving n-type arguments. See also <code>\tl_map_function:nN</code> .
<hr/>	
<code>\tl_map_inline:Nn</code>	<code>\tl_map_inline:Nn <tl var> {<inline function>}</code>
<code>\tl_map_inline:cN</code>	
<hr/>	Applies the <i><inline function></i> to every <i><item></i> stored within the <i><tl var></i> . The <i><inline function></i> should consist of code which will receive the <i><item></i> as #1. One in line mapping can be nested inside another. See also <code>\tl_map_function:Nn</code> .
<hr/>	
<code>\tl_map_inline:nn</code>	<code>\tl_map_inline:nn <token list> {<inline function>}</code>
<hr/>	Applies the <i><inline function></i> to every <i><item></i> stored within the <i><token list></i> . The <i><inline function></i> should consist of code which will receive the <i><item></i> as #1. One in line mapping can be nested inside another. See also <code>\tl_map_function:nn</code> .
<hr/>	
<code>\tl_map_variable:NNn</code>	<code>\tl_map_variable:NNn <tl var> <variable> {<function>}</code>
<code>\tl_map_variable:cNn</code>	
<hr/>	Applies the <i><function></i> to every <i><item></i> stored within the <i><tl var></i> . The <i><function></i> should consist of code which will receive the <i><item></i> stored in the <i><variable></i> . One variable mapping can be nested inside another. See also <code>\tl_map_inline:Nn</code> .
<hr/>	
<code>\tl_map_variable:nNn</code>	<code>\tl_map_variable:nNn <token list> <variable> {<function>}</code>
<hr/>	Applies the <i><function></i> to every <i><item></i> stored within the <i><token list></i> . The <i><function></i> should consist of code which will receive the <i><item></i> stored in the <i><variable></i> . One variable mapping can be nested inside another. See also <code>\tl_map_inline:nn</code> .

<code>\tl_map_break</code> ☆	<code>\tl_map_break:</code> Used to terminate a <code>\tl_map...</code> function before all entries in the <i>token list variable</i> have been processed. This will normally take place within a conditional statement, for example
------------------------------	---

```

\tl_map_inline:Nn \l_my_tl
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \tl_map_break: }
  {
    % Do something useful
  }
}

```

Use outside of a `\tl_map...` scenario will lead low level T_EX errors.

87 Using token lists

<code>\tl_to_str:N</code> ★	<code>\tl_to_str:N</code> <i>tl var</i>
<code>\tl_to_str:c</code> ★	Converts the content of the <i>tl var</i> into a series of characters with category code 12 (other) with the exception of spaces, which retain category code 10 (space). This <i>string</i> is then left in the input stream.

<code>\tl_to_str:n</code> ★	<code>\tl_to_str:n</code> <i>{tokens}</i> Converts the given <i>tokens</i> into a series of characters with category code 12 (other) with the exception of spaces, which retain category code 10 (space). This <i>string</i> is then left in the input stream. Note that this function requires only a single expansion.
-----------------------------	---

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

<code>\tl_use:N</code> ★	<code>\tl_use:N</code> <i>tl var</i>
<code>\tl_use:c</code> ★	Recovers the content of a <i>tl var</i> and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Note that it is possible to use a <i>tl var</i> directly without an accessor function.

88 Working with the content of token lists

<code>\tl_length:n</code> ★	<code>\tl_length:n</code> <i>{tokens}</i>
<code>\tl_length:(V o)</code> ★	Counts the number of <i>items</i> in <i>tokens</i> and leaves this information in the input stream. Unbraced tokens count as one element as do each token group <i>{...}</i> . This process will ignore any unprotected spaces within <i>tokens</i> . See also <code>\tl_length:N</code> . This function requires three expansions, giving an <i>integer denotation</i> .

Updated: 2011-08-13

<hr/> <code>\tl_length:N</code> ★ <code>\tl_length:c</code> ★ <hr/> Updated: 2011-08-13 <hr/>	<code>\tl_length:N</code> { <i>tl var</i> } Counts the number of token groups in the <i>tl var</i> and leaves this information in the input stream. Unbraced tokens count as one element as do each token group { <i>...</i> }. This process will ignore any unprotected spaces within <i>tokens</i> . See also <code>\tl_length:n</code> . This function requires three expansions, giving an <i>integer denotation</i> .
<hr/> <code>\tl_reverse:n</code> ★ <code>\tl_reverse:(V o)</code> ★ <hr/> Updated: 2011-08-13 <hr/>	<code>\tl_reverse:n</code> { <i>token list</i> } Reverses the order of the <i>items</i> in the <i>token list</i> , so that <i>item1</i> <i>item2</i> <i>item3</i> ... <i>item_n</i> becomes <i>item_n</i> ... <i>item3</i> <i>item2</i> <i>item1</i> . This process will preserve unprotected space within the <i>token list</i> . Tokens are not reversed within braced token groups, which keep their outer set of braces. In situations where performance is important, consider <code>\tl_reverse_items:n</code> . See also <code>\tl_reverse:N</code> .
<hr/> <code>\tl_reverse:N</code> <code>\tl_reverse:c</code> <code>\tl_greverse:N</code> <code>\tl_greverse:c</code> <hr/> Updated: 2011-11-22 <hr/>	<code>\tl_reverse:N</code> { <i>tl var</i> } Reverses the order of the <i>items</i> stored in <i>tl var</i> , so that <i>item1</i> <i>item2</i> <i>item3</i> ... <i>item_n</i> becomes <i>item_n</i> ... <i>item3</i> <i>item2</i> <i>item1</i> . This process will preserve unprotected spaces within the <i>token list variable</i> . Braced token groups are copied without reversing the order of tokens, but keep the outer set of braces. See also <code>\tl_reverse:n</code> .
<hr/> <code>\tl_reverse_items:n</code> ★ <hr/> New: 2011-08-13 <hr/>	<code>\tl_reverse_items:n</code> { <i>token list</i> } Reverses the order of the <i>items</i> stored in <i>tl var</i> , so that { <i>item1</i> }{ <i>item2</i> }{ <i>item3</i> }...{ <i>item_n</i> } becomes { <i>item_n</i> }... { <i>item3</i> }{ <i>item2</i> }{ <i>item1</i> }. This process will remove any unprotected space within the <i>token list</i> . Braced token groups are copied without reversing the order of tokens, and keep the outer set of braces. Items which are initially not braced are copied with braces in the result. In cases where preserving spaces is important, consider <code>\tl_reverse:n</code> or <code>\tl_reverse_tokens:n</code> .
<hr/> <code>\tl_trim_spaces:n</code> ★ <hr/> New: 2011-07-09 Updated: 2011-08-13 <hr/>	<code>\tl_trim_spaces:n</code> <i>token list</i> Removes any leading and trailing explicit space characters from the <i>token list</i> and leaves the result in the input stream. This process requires two expansions. <p>T_EXhackers note: The result is return within the <code>\unexpanded</code> primitive (<code>\exp_not:n</code>), which means that the token list will not expand further when appearing in an x-type argument expansion.</p>
<hr/> <code>\tl_trim_spaces:N</code> <code>\tl_trim_spaces:c</code> <code>\tl_gtrim_spaces:N</code> <code>\tl_gtrim_spaces:c</code> <hr/> New: 2011-07-09 <hr/>	<code>\tl_trim_spaces:N</code> <i>tl var</i> Removes any leading and trailing explicit space characters from the content of the <i>tl var</i> .

89 The first token from a token list

Functions which deal with either only the very first token of a token list or everything except the first token.

`\tl_head:n` ★
`\tl_head:(V|v|f)` ★

Updated: 2011-08-09

`\tl_head:n` { $\langle tokens \rangle$ }

Leaves in the input stream the first non-space token from the $\langle tokens \rangle$. Any leading space tokens will be discarded, and thus for example

`\tl_head:n` { abc }

and

`\tl_head:n` { ~ abc }

will both leave a in the input stream. An empty list of $\langle tokens \rangle$ or one which consists only of space (category code 10) tokens will result in `\tl_head:n` leaving nothing in the input stream.

`\tl_head:w` ★

`\tl_head:w` $\langle tokens \rangle$ \q_stop

Leaves in the input stream the first non-space token from the $\langle tokens \rangle$. An empty list of $\langle tokens \rangle$ or one which consists only of space (category code 10) tokens will result in an error, and thus $\langle tokens \rangle$ must *not* be “blank” as determined by `\tl_if_blank:n(TF)`. This function requires only a single expansion, and thus is suitable for use within an o-type expansion. In general, `\tl_head:n` should be preferred if the number of expansions is not critical.

`\tl_tail:n` ★
`\tl_tail:(V|v|f)` ★

Updated: 2011-08-09

`\tl_tail:n` { $\langle tokens \rangle$ }

Discards the all leading space tokens and the first non-space token in the $\langle tokens \rangle$, and leaves the remaining tokens in the input stream. Thus for example

`\tl_tail:n` { abc }

and

`\tl_tail:n` { ~ abc }

will both leave bc in the input stream. An empty list of $\langle tokens \rangle$ or one which consists only of space (category code 10) tokens will result in `\tl_tail:n` leaving nothing in the input stream.

`\tl_tail:w` ★

`\tl_tail:w` { $\langle tokens \rangle$ } \q_stop

Discards the all leading space tokens and the first non-space token in the $\langle tokens \rangle$, and leaves the remaining tokens in the input stream. An empty list of $\langle tokens \rangle$ or one which consists only of space (category code 10) tokens will result in an error, and thus $\langle tokens \rangle$ must *not* be “blank” as determined by `\tl_if_blank:n(TF)`. This function requires only a single expansion, and thus is suitable for use within an o-type expansion. In general, `\tl_tail:n` should be preferred if the number of expansions is not critical.

<code>\str_head:n</code> ★	<code>\str_head:n {⟨tokens⟩}</code>
<code>\str_tail:n</code> ★	<code>\str_tail:n {⟨tokens⟩}</code>

New: 2011-08-10

Converts the $\langle tokens \rangle$ into a string, as described for `\tl_to_str:n`. The `\str_head:n` function then leaves the first character of this string in the input stream. The `\str_tail:n` function leaves all characters except the first in the input stream. The first character may be a space. If the $\langle tokens \rangle$ argument is entirely empty, nothing is left in the input stream.

<code>\tl_if_head_eq_catcode_p:nN</code> ★	<code>\tl_if_head_eq_catcode_p:nN {⟨token list⟩} ⟨test token⟩</code>
<code>\tl_if_head_eq_catcode:nNTF</code> ★	<code>\tl_if_head_eq_catcode:nNTF {⟨token list⟩} ⟨test token⟩</code>
	<code>{⟨true code⟩} {⟨false code⟩}</code>

Updated: 2011-08-10

Tests if the first $\langle token \rangle$ in the $\langle token list \rangle$ has the same category code as the $\langle test token \rangle$. In the case where $\langle token list \rangle$ is empty, its head is considered to be `\q_nil`, and the test will be true if $\langle test token \rangle$ is a control sequence.

<code>\tl_if_head_eq_charcode_p:nN</code> ★	<code>\tl_if_head_eq_charcode_p:nN {⟨token list⟩} ⟨test token⟩</code>
<code>\tl_if_head_eq_charcode_p:fN</code> ★	<code>\tl_if_head_eq_charcode:nNTF {⟨token list⟩} ⟨test token⟩</code>
<code>\tl_if_head_eq_charcode:nNTF</code> ★	<code>{⟨true code⟩} {⟨false code⟩}</code>
<code>\tl_if_head_eq_charcode:fNTF</code> ★	

Updated: 2011-08-10

Tests if the first $\langle token \rangle$ in the $\langle token list \rangle$ has the same character code as the $\langle test token \rangle$. In the case where $\langle token list \rangle$ is empty, its head is considered to be `\q_nil`, and the test will be true if $\langle test token \rangle$ is a control sequence.

<code>\tl_if_head_eq_meaning_p:nN</code> ★	<code>\tl_if_head_eq_meaning_p:nN {⟨token list⟩} ⟨test token⟩</code>
<code>\tl_if_head_eq_meaning:nNTF</code> ★	<code>\tl_if_head_eq_meaning:nNTF {⟨token list⟩} ⟨test token⟩</code>
	<code>{⟨true code⟩} {⟨false code⟩}</code>

Updated: 2011-08-10

Tests if the first $\langle token \rangle$ in the $\langle token list \rangle$ has the same meaning as the $\langle test token \rangle$. In the case where $\langle token list \rangle$ is empty, its head is considered to be `\q_nil`, and the test will be true if $\langle test token \rangle$ has the same meaning as `\q_nil`.

<code>\tl_if_head_group_p:n</code> ★	<code>\tl_if_head_group_p:n {⟨token list⟩}</code>
<code>\tl_if_head_group:nTF</code> ★	<code>\tl_if_head_group:nTF {⟨token list⟩} {⟨true code⟩} {⟨false code⟩}</code>

Updated: 2011-08-11

Tests if the first $\langle token \rangle$ in the $\langle token list \rangle$ is an explicit begin-group character (with category code 1 and any character code), in other words, if the $\langle token list \rangle$ starts with a brace group. In particular, the test is false if the $\langle token list \rangle$ starts with an implicit token such as `\c_group_begin_token`, or if it empty. This function is useful to implement actions on token lists on a token by token basis.

<hr/>	
<code>\tl_if_head_N_type_p:n</code> ★	<code>\tl_if_head_N_type_p:n {\token list}</code>
<code>\tl_if_head_N_type:nTF</code> ★	<code>\tl_if_head_N_type:nTF {\token list} {\true code} {\false code}</code>
<hr/>	
New: 2011-08-11	Tests if the first $\langle token \rangle$ in the $\langle token list \rangle$ is a normal N-type argument. In other words, it is neither an explicit space character (with category code 10 and character code 32) nor an explicit begin-group character (with category code 1 and any character code). An empty argument yields false, as it does not have a “normal” first token. This function is useful to implement actions on token lists on a token by token basis.

<hr/>	
<code>\tl_if_head_space_p:n</code> ★	<code>\tl_if_head_space_p:n {\token list}</code>
<code>\tl_if_head_space:nTF</code> ★	<code>\tl_if_head_space:nTF {\token list} {\true code} {\false code}</code>
<hr/>	
Updated: 2011-08-11	Tests if the first $\langle token \rangle$ in the $\langle token list \rangle$ is an explicit space character (with category code 10 and character code 32). If $\langle token list \rangle$ starts with an implicit token such as <code>\c_space_token</code> , the test will yield false, as well as if the argument is empty. This function is useful to implement actions on token lists on a token by token basis.

TeXhackers note: When TeX reads a character of category code 10 for the first time, it is converted to an explicit space token, with character code 32, regardless of the initial character code. “Funny” spaces with a different category code, can be produced using `\lowercase`. Explicit spaces are also produced as a result of `\token_to_str:N`, `\tl_to_str:n`, etc.

90 Viewing token lists

<hr/>	
<code>\tl_show:N</code>	<code>\tl_show:N \tl var</code>
<code>\tl_show:c</code>	Displays the content of the $\langle tl var \rangle$ on the terminal.

TeXhackers note: `\tl_show:N` is the TeX primitive `\show`.

<hr/>	
<code>\tl_show:n</code>	<code>\tl_show:n \token list</code>
	Displays the $\langle token list \rangle$ on the terminal.

TeXhackers note: `\tl_show:n` is the ε -TeX primitive `\showtokens`.

91 Constant token lists

<hr/>	
<code>\c_job_name_tl</code>	Constant that gets the “job name” assigned when TeX starts.
<hr/>	
Updated: 2011-08-18	TeXhackers note: This is the new name for the primitive <code>\jobname</code> . It is a constant that is set by TeX and should not be overwritten by the package.

<code>\c_empty_tl</code>	Constant that is always empty.
--------------------------	--------------------------------

<code>\c_space_tl</code>	A space token contained in a token list (compare this with <code>\c_space_token</code>). For use where an explicit space is required.
--------------------------	--

92 Scratch token lists

<code>\l_tmpa_tl</code> <code>\l_tmpb_tl</code>	Scratch token lists for local assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
--	---

<code>\g_tmpa_tl</code> <code>\g_tmpb_tl</code>	Scratch token lists for global assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
--	--

93 Experimental token list functions

<code>\tl_reverse_tokens:n</code> ★	<code>\tl_reverse_tokens:n {\tokens}</code>
-------------------------------------	---

New: 2011-08-11

This function, which works directly on T_EX tokens, reverses the order of the *tokens*: the first will be the last and the last will become first. Spaces are preserved. The reversal also operates within brace groups, but the braces themselves are not exchanged, as this would lead to an unbalanced token list. For instance, `\tl_reverse_tokens:n {a~{b()}}` leaves `{() (b)~a` in the input stream. This function requires two steps of expansion.

<code>\tl_length_tokens:n</code> ★	<code>\tl_length_tokens:n {\tokens}</code>
------------------------------------	--

New: 2011-08-11

Counts the number of T_EX tokens in the *tokens* and leaves this information in the input stream. Every token, including spaces and braces, contributes one to the total; thus for instance, the length of `a~{bc}` is 6. This function requires three expansions, giving an *integer denotation*.

<code>\tl_expandable_uppercase:n</code>	★	<code>\tl_expandable_uppercase:n {⟨tokens⟩}</code>
<code>\tl_expandable_lowercase:n</code>	★	<code>\tl_expandable_lowercase:n {⟨tokens⟩}</code>

New: 2011-08-13

The `\tl_expandable_uppercase:n` function works through all of the *⟨tokens⟩*, replacing characters in the range **a–z** (with arbitrary category code) by the corresponding letter in the range **A–Z**, with category code 11 (letter). Similarly, `\tl_expandable_lowercase:n` replaces characters in the range **A–Z** by letters in the range **a–z**, and leaves other tokens unchanged. This function requires two steps of expansion.

TeXhackers note: Begin-group and end-group characters are normalized and become { and }, respectively.

<code>\tl_item:nn</code>	★	<code>\tl_item:nn {⟨token list⟩} {⟨integer expression⟩}</code>
<code>\tl_item:(Nn cn)</code>	★	

New: 2011-11-21

Indexing items in the *⟨token list⟩* from 0 on the left, this function will evaluate the *⟨integer expression⟩* and leave the appropriate item from the *⟨token list⟩* in the input stream. If the *⟨integer expression⟩* is negative, indexing occurs from the right of the sequence, starting at -1 for the right-most item. If the index is out of bounds, then the function expands to nothing.

94 Internal functions

<code>\q_tl_act_mark</code>
<code>\q_tl_act_stop</code>

Quarks which are only used for the particular purposes of `\tl_act_...` functions.

Part XII

The l3seq package

Sequences and stacks

L^AT_EX3 implements a “sequence” data type, which contain an ordered list of entries which may contain any *⟨balanced text⟩*. It is possible to map functions to sequences such that the function is applied to every item in the sequence.

Sequences are also used to implement stack functions in L^AT_EX3. This is achieved using a number of dedicated stack functions.

95 Creating and initialising sequences

`\seq_new:N`
`\seq_new:c`

`\seq_new:N` *⟨sequence⟩*

Creates a new *⟨sequence⟩* or raises an error if the name is already taken. The declaration is global. The *⟨sequence⟩* will initially contain no items.

`\seq_clear:N`
`\seq_clear:c`
`\seq_gclear:N`
`\seq_gclear:c`

`\seq_clear:N` *⟨sequence⟩*

Clears all items from the *⟨sequence⟩*.

`\seq_clear_new:N`
`\seq_clear_new:c`
`\seq_gclear_new:N`
`\seq_gclear_new:c`

`\seq_clear_new:N` *⟨sequence⟩*

Ensures that the *⟨sequence⟩* exists globally by applying `\seq_new:N` if necessary, then applies `\seq_(g)clear:N` to leave the *⟨sequence⟩* empty.

`\seq_set_eq:NN`
`\seq_set_eq:(cN|Nc|cc)`
`\seq_gset_eq:NN`
`\seq_gset_eq:(cN|Nc|cc)`

`\seq_set_eq:NN` *⟨sequence1⟩* *⟨sequence2⟩*

Sets the content of *⟨sequence1⟩* equal to that of *⟨sequence2⟩*.

`\seq_set_split:Nnn`
`\seq_gset_split:Nnn`

`\seq_set_split:Nnn` *⟨sequence⟩* *{⟨delimiter⟩}* *{⟨token list⟩}*

Splits the *⟨token list⟩* into *⟨items⟩* separated by *⟨delimiter⟩*, and assigns the result to the *⟨sequence⟩*. Spaces on both sides of each *⟨item⟩* are ignored, then one set of outer braces is removed (if any); this space trimming behaviour is identical to that of l3clist functions. Empty *⟨items⟩* are preserved by `\seq_set_split:Nnn`, and can be removed afterwards using `\seq_remove_all:Nn` *⟨sequence⟩* *{⟨⟩}*. The *⟨delimiter⟩* may not contain `{`, `}` or `#` (assuming T_EX’s normal category code régime). If the *⟨delimiter⟩* is empty, the *⟨token list⟩* is split into *⟨items⟩* as a *⟨token list⟩*.

New: 2011-08-15
Updated: 2011-12-07

<hr/>	<code>\seq_concat:NNN</code>	<code>\seq_concat:NNN</code> $\langle sequence1 \rangle$ $\langle sequence2 \rangle$ $\langle sequence3 \rangle$
<code>\seq_concat:ccc</code>		
<code>\seq_gconcat:NNN</code>		Concatenates the content of $\langle sequence2 \rangle$ and $\langle sequence3 \rangle$ together and saves the result in $\langle sequence1 \rangle$. The items in $\langle sequence2 \rangle$ will be placed at the left side of the new sequence.
<code>\seq_gconcat:ccc</code>		

96 Appending data to sequences

<hr/>	<code>\seq_put_left:Nn</code>	<code>\seq_put_left:Nn</code> $\langle sequence \rangle$ $\{ \langle item \rangle \}$
<code>\seq_put_left:(NV Nv No Nx cn cV cv co cx)</code>		
<code>\seq_gput_left:Nn</code>		
<code>\seq_gput_left:(NV Nv No Nx cn cV cv co cx)</code>		

Appends the $\langle item \rangle$ to the left of the $\langle sequence \rangle$.

<hr/>	<code>\seq_put_right:Nn</code>	<code>\seq_put_right:Nn</code> $\langle sequence \rangle$ $\{ \langle item \rangle \}$
<code>\seq_put_right:(NV Nv No Nx cn cV cv co cx)</code>		
<code>\seq_gput_right:Nn</code>		
<code>\seq_gput_right:(NV Nv No Nx cn cV cv co cx)</code>		

Appends the $\langle item \rangle$ to the right of the $\langle sequence \rangle$.

97 Recovering items from sequences

Items can be recovered from either the left or the right of sequences. For implementation reasons, the actions at the left of the sequence are faster than those acting on the right. These functions all assign the recovered material locally, *i.e.* setting the $\langle token list variable \rangle$ used with `\tl_set:Nn` and *never* `\tl_gset:Nn`.

<hr/>	<code>\seq_get_left:NN</code>	<code>\seq_get_left:NN</code> $\langle sequence \rangle$ $\langle token list variable \rangle$
<code>\seq_get_left:cN</code>		
		Stores the left-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$ without removing it from the $\langle sequence \rangle$. The $\langle token list variable \rangle$ is assigned locally. If $\langle sequence \rangle$ is empty an error will be raised.

<hr/>	<code>\seq_get_right:NN</code>	<code>\seq_get_right:NN</code> $\langle sequence \rangle$ $\langle token list variable \rangle$
<code>\seq_get_right:cN</code>		
		Stores the right-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$ without removing it from the $\langle sequence \rangle$. The $\langle token list variable \rangle$ is assigned locally. If $\langle sequence \rangle$ is empty an error will be raised.

<hr/>	<code>\seq_pop_left:NN</code>	<code>\seq_pop_left:NN</code> $\langle sequence \rangle$ $\langle token list variable \rangle$
<code>\seq_pop_left:cN</code>		
		Pops the left-most item from a $\langle sequence \rangle$ into the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from the sequence and stores it in the $\langle token list variable \rangle$. Both of the variables are assigned locally. If $\langle sequence \rangle$ is empty an error will be raised.

<hr/>	
<code>\seq_gpop_left:NN</code>	<code>\seq_gpop_left:NN</code> $\langle sequence \rangle$ $\langle token list variable \rangle$
<code>\seq_gpop_left:cN</code>	
<hr/>	
	Pops the left-most item from a $\langle sequence \rangle$ into the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from the sequence and stores it in the $\langle token list variable \rangle$. The $\langle sequence \rangle$ is modified globally, while the assignment of the $\langle token list variable \rangle$ is local. If $\langle sequence \rangle$ is empty an error will be raised.
<hr/>	
<code>\seq_pop_right:NN</code>	<code>\seq_pop_right:NN</code> $\langle sequence \rangle$ $\langle token list variable \rangle$
<code>\seq_pop_right:cN</code>	
<hr/>	
	Pops the right-most item from a $\langle sequence \rangle$ into the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from the sequence and stores it in the $\langle token list variable \rangle$. Both of the variables are assigned locally. If $\langle sequence \rangle$ is empty an error will be raised.
<hr/>	
<code>\seq_gpop_right:NN</code>	<code>\seq_gpop_right:NN</code> $\langle sequence \rangle$ $\langle token list variable \rangle$
<code>\seq_gpop_right:cN</code>	
<hr/>	
	Pops the right-most item from a $\langle sequence \rangle$ into the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from the sequence and stores it in the $\langle token list variable \rangle$. The $\langle sequence \rangle$ is modified globally, while the assignment of the $\langle token list variable \rangle$ is local. If $\langle sequence \rangle$ is empty an error will be raised.

98 Modifying sequences

While sequences are normally used as ordered lists, it may be necessary to modify the content. The functions here may be used to update sequences, while retaining the order of the unaffected entries.

<hr/>	
<code>\seq_remove_duplicates:N</code>	<code>\seq_remove_duplicates:N</code> $\langle sequence \rangle$
<code>\seq_remove_duplicates:c</code>	
<code>\seq_gremove_duplicates:N</code>	Removes duplicate items from the $\langle sequence \rangle$, leaving the left most copy of each item in the $\langle sequence \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for <code>\tl_if_eq:nn(TF)</code> .
<code>\seq_gremove_duplicates:c</code>	
<hr/>	

T_EXhackers note: This function iterates through every item in the $\langle sequence \rangle$ and does a comparison with the $\langle items \rangle$ already checked. It is therefore relatively slow with large sequences.

<hr/>	
<code>\seq_remove_all:Nn</code>	<code>\seq_remove_all:Nn</code> $\langle sequence \rangle$ $\{\langle item \rangle\}$
<code>\seq_remove_all:cn</code>	
<code>\seq_gremove_all:Nn</code>	Removes every occurrence of $\langle item \rangle$ from the $\langle sequence \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for <code>\tl_if_eq:nn(TF)</code> .
<code>\seq_gremove_all:cn</code>	
<hr/>	

99 Sequence conditionals

<hr/>	
<code>\seq_if_empty_p:N</code> ★	<code>\seq_if_empty_p:N</code> $\langle sequence \rangle$
<code>\seq_if_empty_p:c</code> ★	<code>\seq_if_empty:N</code> $\langle sequence \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$
<code>\seq_if_empty:N</code> ★	
<code>\seq_if_empty:c</code> ★	Tests if the $\langle sequence \rangle$ is empty (containing no items).
<hr/>	

<code>\seq_if_in:NnTF</code>	<code>\seq_if_in:NnTF <sequence> {<item>} {<true code>} {<false code>}</code>
<code>\seq_if_in:(NV Nv No Nx cn cV cv co cx)TF</code>	

Tests if the *<item>* is present in the *<sequence>*.

100 Mapping to sequences

<code>\seq_map_function:NN</code> ☆	<code>\seq_map_function:NN <sequence> <function></code>
<code>\seq_map_function:cN</code> ☆	

Applies *<function>* to every *<item>* stored in the *<sequence>*. The *<function>* will receive one argument for each iteration. The *<items>* are returned from left to right. The function `\seq_map_inline:Nn` is in general more efficient than `\seq_map_function:NN`. One mapping may be nested inside another.

<code>\seq_map_inline:Nn</code>	<code>\seq_map_inline:Nn <sequence> {<inline function>}</code>
<code>\seq_map_inline:cn</code>	

Applies *<inline function>* to every *<item>* stored within the *<sequence>*. The *<inline function>* should consist of code which will receive the *<item>* as #1. One in line mapping can be nested inside another. The *<items>* are returned from left to right.

<code>\seq_map_variable:NNn</code>	<code>\seq_map_variable:NNn <sequence> <tl var.> {<function using tl var.>}</code>
<code>\seq_map_variable:(Ncn cNn ccn)</code>	

Stores each entry in the *<sequence>* in turn in the *<tl var.>* and applies the *<function using tl var.>* The *<function>* will usually consist of code making use of the *<tl var.>*, but this is not enforced. One variable mapping can be nested inside another. The *<items>* are returned from left to right.

<code>\seq_map_break</code> ☆	<code>\seq_map_break:</code>
-------------------------------	------------------------------

Used to terminate a `\seq_map...` function before all entries in the *<sequence>* have been processed. This will normally take place within a conditional statement, for example

```
\seq_map_inline:Nn \l_my_seq
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \seq_map_break: }
  {
    % Do something useful
  }
}
```

Use outside of a `\seq_map...` scenario will lead to low level T_EX errors.

T_EXhackers note: When the mapping is broken, additional tokens may be inserted by the internal macro `\prg_break_point:n` before further items are taken from the input stream. This will depend on the design of the mapping function.

`\seq_map_break:n` ☆

`\seq_map_break:n {⟨tokens⟩}`

Used to terminate a `\seq_map...` function before all entries in the *⟨sequence⟩* have been processed, inserting the *⟨tokens⟩* after the mapping has ended. This will normally take place within a conditional statement, for example

```
\seq_map_inline:Nn \l_my_seq
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \seq_map_break:n { <tokens> } }
  {
    % Do something useful
  }
}
```

Use outside of a `\seq_map...` scenario will lead to low level T_EX errors.

T_EXhackers note: When the mapping is broken, additional tokens may be inserted by the internal macro `\prg_break_point:n` before the *⟨tokens⟩* are inserted into the input stream. This will depend on the design of the mapping function.

101 Sequences as stacks

Sequences can be used as stacks, where data is pushed to and popped from the top of the sequence. (The left of a sequence is the top, for performance reasons.) The stack functions for sequences are not intended to be mixed with the general ordered data functions detailed in the previous section: a sequence should either be used as an ordered data type or as a stack, but not in both ways.

`\seq_get:NN`
`\seq_get:cN`

`\seq_get:NN ⟨sequence⟩ ⟨token list variable⟩`

Reads the top item from a *⟨sequence⟩* into the *⟨token list variable⟩* without removing it from the *⟨sequence⟩*. The *⟨token list variable⟩* is assigned locally. If *⟨sequence⟩* is empty an error will be raised.

`\seq_pop:NN`
`\seq_pop:cN`

`\seq_pop:NN ⟨sequence⟩ ⟨token list variable⟩`

Pops the top item from a *⟨sequence⟩* into the *⟨token list variable⟩*. Both of the variables are assigned locally. If *⟨sequence⟩* is empty an error will be raised.

`\seq_gpop:NN`
`\seq_gpop:cN`

`\seq_gpop:NN ⟨sequence⟩ ⟨token list variable⟩`

Pops the top item from a *⟨sequence⟩* into the *⟨token list variable⟩*. The *⟨sequence⟩* is modified globally, while the *⟨token list variable⟩* is assigned locally. If *⟨sequence⟩* is empty an error will be raised.

<code>\seq_push:Nn</code>	<code>\seq_push:Nn <sequence> {(item)}</code>
<code>\seq_push:(NV Nv No Nx cn cV cv co cx)</code>	
<code>\seq_gpush:Nn</code>	
<code>\seq_gpush:(NV Nv No Nx cn cV cv co cx)</code>	

Adds the $\{(item)\}$ to the top of the $\langle sequence \rangle$.

102 Viewing sequences

<code>\seq_show:N</code>	<code>\seq_show:N <sequence></code>
<code>\seq_show:c</code>	Displays the entries in the $\langle sequence \rangle$ in the terminal.

103 Experimental sequence functions

This section contains functions which may or may not be retained, depending on how useful they are found to be.

<code>\seq_get_left:NNTF</code>	<code>\seq_get_left:NNTF <sequence> <token list variable> {(true code)} {(false code)}</code>
<code>\seq_get_left:cNTF</code>	If the $\langle sequence \rangle$ is empty, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle sequence \rangle$ is non-empty, stores the left-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$ without removing it from a $\langle sequence \rangle$. The $\langle token list variable \rangle$ is assigned locally.

<code>\seq_get_right:NNTF</code>	<code>\seq_get_right:NNTF <sequence> <token list variable> {(true code)} {(false code)}</code>
<code>\seq_get_right:cNTF</code>	If the $\langle sequence \rangle$ is empty, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle sequence \rangle$ is non-empty, stores the right-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$ without removing it from a $\langle sequence \rangle$. The $\langle token list variable \rangle$ is assigned locally.

<code>\seq_pop_left:NNTF</code>	<code>\seq_pop_left:NNTF <sequence> <token list variable> {(true code)} {(false code)}</code>
<code>\seq_pop_left:cNTF</code>	If the $\langle sequence \rangle$ is empty, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle sequence \rangle$ is non-empty, pops the left-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from a $\langle sequence \rangle$. Both the $\langle sequence \rangle$ and the $\langle token list variable \rangle$ are assigned locally.

<code>\seq_gpop_left:NNTF</code>	<code>\seq_gpop_left:NNTF <sequence> <token list variable> {(true code)} {(false code)}</code>
<code>\seq_gpop_left:cNTF</code>	If the $\langle sequence \rangle$ is empty, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle sequence \rangle$ is non-empty, pops the left-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from a $\langle sequence \rangle$. The $\langle sequence \rangle$ is modified globally, while the $\langle token list variable \rangle$ is assigned locally.

<hr/> <u>\seq_pop_right:NNTF</u> <u>\seq_pop_right:cNTF</u> <hr/>	<u>\seq_pop_right:NNTF</u> $\langle sequence \rangle$ $\langle token list variable \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$ If the $\langle sequence \rangle$ is empty, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle sequence \rangle$ is non-empty, pops the right-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from a $\langle sequence \rangle$. Both the $\langle sequence \rangle$ and the $\langle token list variable \rangle$ are assigned locally.
<hr/> <u>\seq_gpop_right:NNTF</u> <u>\seq_gpop_right:cNTF</u> <hr/>	<u>\seq_gpop_right:NNTF</u> $\langle sequence \rangle$ $\langle token list variable \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$ If the $\langle sequence \rangle$ is empty, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle sequence \rangle$ is non-empty, pops the right-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from a $\langle sequence \rangle$. The $\langle sequence \rangle$ is modified globally, while the $\langle token list variable \rangle$ is assigned locally.
<hr/> <u>\seq_length:N</u> ☆ <u>\seq_length:c</u> ☆ <hr/>	<u>\seq_length:N</u> $\langle sequence \rangle$ Leaves the number of items in the $\langle sequence \rangle$ in the input stream as an $\langle integer denotation \rangle$. The total number of items in a $\langle sequence \rangle$ will include those which are empty and duplicates, <i>i.e.</i> every item in a $\langle sequence \rangle$ is unique.
<hr/> <u>\seq_item:Nn</u> ☆ <u>\seq_item:cn</u> ☆ <hr/>	<u>\seq_item:Nn</u> $\langle sequence \rangle$ $\{\langle integer expression \rangle\}$ Indexing items in the $\langle sequence \rangle$ from 0 at the top (left), this function will evaluate the $\langle integer expression \rangle$ and leave the appropriate item from the sequence in the input stream. If the $\langle integer expression \rangle$ is negative, indexing occurs from the bottom (right) of the sequence. When the $\langle integer expression \rangle$ is larger than the number of items in the $\langle sequence \rangle$ (as calculated by <u>\seq_length:N</u>) then the function will expand to nothing.
<hr/> <u>\seq_use:N</u> ☆ <u>\seq_use:c</u> ☆ <hr/>	<u>\seq_use:N</u> $\langle sequence \rangle$ Places each $\langle item \rangle$ in the $\langle sequence \rangle$ in turn in the input stream. This occurs in an expandable fashion, and is implemented as a mapping. This means that the process may be prematurely terminated using <u>\seq_map_break:</u> or <u>\seq_map_break:n</u> . The $\langle items \rangle$ in the $\langle sequence \rangle$ will be used from left (top) to right (bottom).
<hr/> <u>\seq_mapthread_function:NNN</u> ☆ <u>\seq_mapthread_function:(NcN cNN ccN)</u> ☆ <hr/>	<u>\seq_mapthread_function:NNN</u> $\langle seq1 \rangle$ $\langle seq2 \rangle$ $\langle function \rangle$ Applies $\langle function \rangle$ to every pair of items $\langle seq1-item \rangle$ – $\langle seq2-item \rangle$ from the two sequences, returning items from both sequences from left to right. The $\langle function \rangle$ will receive two n -type arguments for each iteration. The mapping will terminate when the end of either sequence is reached (<i>i.e.</i> whichever sequence has fewer items determines how many iterations occur).

<code>\seq_set_from_clist:NN</code>	<code>\seq_set_from_clist:NN <sequence> <comma-list></code>
<code>\seq_set_from_clist:(cN Nc cc Nn cn)</code>	
<code>\seq_gset_from_clist:NN</code>	
<code>\seq_gset_from_clist:(cN Nc cc Nn cn)</code>	

Sets the *<sequence>* within the current \TeX group to be equal to the content of the *<comma-list>*.

<code>\seq_reverse:N</code>	<code>\seq_reverse:N <sequence></code>
<code>\seq_greverse:N</code>	

New: 2011-11-22
Updated: 2011-11-24

Reverses the order of items in the *<sequence>*, and assigns the result to *<sequence>*, locally or globally according to the variant chosen.

<code>\seq_set_filter:NNn</code>	<code>\seq_set_filter:NNn <sequence1> <sequence2> {\inline boolexpr}</code>
<code>\seq_gset_filter:NNn</code>	

New: 2011-12-22

Evaluates the *<inline boolexpr>* for every *<item>* stored within the *<sequence2>*. The *<inline boolexpr>* will receive the *<item>* as **#1**. The sequence of all *<items>* for which the *<inline boolexpr>* evaluated to **true** is assigned to *<sequence1>*.

\TeX hackers note: Contrarily to other mapping functions, `\seq_map_break:` cannot be used in this function, and will lead to low-level \TeX errors.

<code>\seq_set_map:NNn</code>	<code>\seq_set_map:NNn <sequence1> <sequence2> {\inline function}</code>
<code>\seq_gset_map:NNn</code>	

New: 2011-12-22

Applies *<inline function>* to every *<item>* stored within the *<sequence2>*. The *<inline function>* should consist of code which will receive the *<item>* as **#1**. The sequence resulting from x-expanding *<inline function>* applied to each *<item>* is assigned to *<sequence1>*. As such, the code in *<inline function>* should be expandable.

\TeX hackers note: Contrarily to other mapping functions, `\seq_map_break:` cannot be used in this function, and will lead to low-level \TeX errors.

104 Internal sequence functions

<code>\seq_if_empty_err_break:N</code>	<code>\seq_if_empty_err_break:N <sequence></code>
--	---

Tests if the *<sequence>* is empty, and if so issues an error message before skipping over any tokens up to `\prg_break_point:n`. This function is used to avoid more serious errors which would otherwise occur if some internal functions were applied to an empty *<sequence>*.

<code>\seq_item:n</code> ★	<code>\seq_item:n <item></code>
----------------------------	---------------------------------------

The internal token used to begin each sequence entry. If expanded outside of a mapping or manipulation function, an error will be raised. The definition should always be set globally.

<u>\seq_push_item_def:n</u>	\seq_push_item_def:n {<code>}
<u>\seq_push_item_def:x</u>	Saves the definition of \seq_item:n and redefines it to accept one parameter and expand to <code>. This function should always be balanced by use of \seq_pop_item_def:.
<u>\seq_pop_item_def</u>	\seq_pop_item_def: Restores the definition of \seq_item:n most recently saved by \seq_push_item_def:n. This function should always be used in a balanced pair with \seq_push_item_def:n.
<u>\seq_break *</u>	\seq_break: Used to terminate sequence functions by gobbling all tokens up to \prg_break_point:n. This function is a copy of \seq_map_break:, but is used in situations which are not mappings.
<u>\seq_break:n *</u>	\seq_break:n {<tokens>} Used to terminate sequence functions by gobbling all tokens up to \prg_break_point:n, then inserting the <tokens> before continuing reading the input stream. This function is a copy of \seq_map_break:n, but is used in situations which are not mappings.

Part XIII

The l3clist package

Comma separated lists

Comma lists contain ordered data where items can be added to the left or right end of the list. The resulting ordered list can then be mapped over using `\clist_map_function:NN`. Several items can be added at once, and spaces are removed from both sides of each item on input. Hence,

```
\clist_new:N \l_my_clist
\clist_put_left:Nn \l_my_clist { ~ a ~ , ~ {b} ~ }
\clist_put_right:Nn \l_my_clist { ~ { c ~ } , d }
```

results in `\l_my_clist` containing `a,{b},{c~},d`. Comma lists cannot contain empty items, thus

```
\clist_clear_new:N \l_my_clist
\clist_put_right:Nn \l_my_clist { , ~ , , }
\clist_if_empty:NTF \l_my_clist { true } { false }
```

will leave `true` in the input stream. To include an item which contains a comma, or starts or ends with a space, surround it with braces.

105 Creating and initialising comma lists

<code>\clist_new:N</code>	<code>\clist_new:N <comma list></code>
---------------------------	--

<code>\clist_new:c</code>	Creates a new <i><comma list></i> or raises an error if the name is already taken. The declaration is global. The <i><comma list></i> will initially contain no items.
---------------------------	--

<code>\clist_clear:N</code>	<code>\clist_clear:N <comma list></code>
-----------------------------	--

<code>\clist_clear:c</code>	Clears all items from the <i><comma list></i> .
<code>\clist_gclear:N</code>	

<code>\clist_gclear:c</code>

<code>\clist_clear_new:N</code>	<code>\clist_clear_new:N <comma list></code>
---------------------------------	--

<code>\clist_clear_new:c</code>	Ensures that the <i><comma list></i> exists globally by applying <code>\clsit_new:N</code> if necessary, then applies <code>\clist_(g)clear:N</code> to leave the list empty.
<code>\clist_gclear_new:N</code>	
<code>\clist_gclear_new:c</code>	

<code>\clist_set_eq:NN</code>	<code>\clist_set_eq:NN <comma list1> <comma list2></code>
-------------------------------	---

<code>\clist_set_eq:(cN Nc cc)</code>	Sets the content of <i><comma list1></i> equal to that of <i><comma list2></i> .
<code>\clist_gset_eq:NN</code>	
<code>\clist_gset_eq:(cN Nc cc)</code>	

<hr/>	<code>\clist_concat:NNN</code>	<code>\clist_concat:NNN</code>	$\langle comma list1 \rangle \langle comma list2 \rangle \langle comma list3 \rangle$
<code>\clist_concat:ccc</code>			
<code>\clist_gconcat:NNN</code>			Concatenates the content of $\langle comma list2 \rangle$ and $\langle comma list3 \rangle$ together and saves the result in $\langle comma list1 \rangle$. The items in $\langle comma list2 \rangle$ will be placed at the left side of the new comma list.
<code>\clist_gconcat:ccc</code>			

106 Adding data to comma lists

<hr/>	<code>\clist_set:Nn</code>	<code>\clist_set:Nn</code>	$\langle comma list \rangle \{ \langle item1 \rangle, \dots, \langle item_n \rangle \}$
<code>\clist_set:(NV No Nx cn cV co cx)</code>			
<code>\clist_gset:Nn</code>			
<code>\clist_gset:(NV No Nx cn cV co cx)</code>			

New: 2011-09-06

Sets $\langle comma list \rangle$ to contain the $\langle items \rangle$, removing any previous content from the variable. Spaces are removed from both sides of each item.

<hr/>	<code>\clist_put_left:Nn</code>	<code>\clist_put_left:Nn</code>	$\langle comma list \rangle \{ \langle item1 \rangle, \dots, \langle item_n \rangle \}$
<code>\clist_put_left:(NV No Nx cn cV co cx)</code>			
<code>\clist_gput_left:Nn</code>			
<code>\clist_gput_left:(NV No Nx cn cV co cx)</code>			

Updated: 2011-09-05

Appends the $\langle items \rangle$ to the left of the $\langle comma list \rangle$. Spaces are removed from both sides of each item.

<hr/>	<code>\clist_put_right:Nn</code>	<code>\clist_put_right:Nn</code>	$\langle comma list \rangle \{ \langle item1 \rangle, \dots, \langle item_n \rangle \}$
<code>\clist_put_right:(NV No Nx cn cV co cx)</code>			
<code>\clist_gput_right:Nn</code>			
<code>\clist_gput_right:(NV No Nx cn cV co cx)</code>			

Updated: 2011-09-05

Appends the $\langle items \rangle$ to the right of the $\langle comma list \rangle$. Spaces are removed from both sides of each item.

107 Using comma lists

<hr/>	<code>\clist_use:N</code>	★	<code>\clist_use:N</code>	$\langle comma list \rangle$
<code>\clist_use:c</code>	★			Places the $\langle comma list \rangle$ directly into the input stream, thus treating it as a $\langle token list \rangle$.

108 Modifying comma lists

While comma lists are normally used as ordered lists, it may be necessary to modify the content. The functions here may be used to update comma lists, while retaining the order of the unaffected entries.

<code>\clist_remove_duplicates:N</code>	<code>\clist_remove_duplicates:N <comma list></code>
<code>\clist_remove_duplicates:c</code>	
<code>\clist_gremove_duplicates:N</code>	
<code>\clist_gremove_duplicates:c</code>	

Removes duplicate items from the $\langle comma list \rangle$, leaving the left most copy of each item in the $\langle comma list \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`.

T_EXhackers note: This function iterates through every item in the $\langle comma list \rangle$ and does a comparison with the $\langle items \rangle$ already checked. It is therefore relatively slow with large comma lists. Furthermore, it will not work if any of the items in the $\langle comma list \rangle$ contains `{`, `}`, or `#` (assuming the usual T_EX category codes apply).

<code>\clist_remove_all:Nn</code>	<code>\clist_remove_all:Nn <comma list> {<item>}</code>
<code>\clist_remove_all:cn</code>	
<code>\clist_gremove_all:Nn</code>	
<code>\clist_gremove_all:cn</code>	

Removes every occurrence of $\langle item \rangle$ from the $\langle comma list \rangle$. The $\langle item \rangle$ comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`.

Updated: 2011-09-06

T_EXhackers note: The $\langle item \rangle$ may not contain `{`, `}`, or `#` (assuming the usual T_EX category codes apply).

109 Comma list conditionals

<code>\clist_if_empty_p:N</code> ★	<code>\clist_if_empty_p:N <comma list></code>
<code>\clist_if_empty_p:c</code> ★	<code>\clist_if_empty:NNTF <comma list> {<true code>} {<false code>}</code>
<code>\clist_if_empty:NNTF</code> ★	
<code>\clist_if_empty:cTF</code> ★	

Tests if the $\langle comma list \rangle$ is empty (containing no items).

<code>\clist_if_eq_p:NN</code> ★	<code>\clist_if_eq_p:NN {<clist₁>} {<clist₂>}</code>
<code>\clist_if_eq_p:(Nc cN cc)</code> ★	<code>\clist_if_eq:NNTF {<clist₁>} {<clist₂>} {<true code>} {<false code>}</code>
<code>\clist_if_eq:NNTF</code> ★	
<code>\clist_if_eq:(Nc cN cc)TF</code> ★	

Compares the content of two $\langle comma lists \rangle$ and is logically **true** if the two contain the same list of entries in the same order.

<code>\clist_if_in:NnTF</code> <code>\clist_if_in:(NV No cn cV co nn nV no)TF</code>	<code>\clist_if_in:NnTF <comma list> {<item>} {<true code>} {<false code>}</code>
---	---

Updated: 2011-09-06

Tests if the $\langle item \rangle$ is present in the $\langle comma list \rangle$. In the case of an **n**-type $\langle comma list \rangle$, spaces are stripped from each item, but braces are not removed. Hence,

`\clist_if_in:nnTF { a , {b}~ , {b} , c } { b } {true} {false}`

yields **false**.

T_EXhackers note: The $\langle item \rangle$ may not contain `{`, `}`, or `#` (assuming the usual T_EX category codes apply), and should not contain `,` nor start or end with a space.

110 Mapping to comma lists

The functions described in this section apply a specified function to each item of a comma list.

When the comma list is given explicitly, as an **n**-type argument, spaces are trimmed around each item. If the result of trimming spaces is empty, the item is ignored. Otherwise, if the item is surrounded by braces, one set is removed, and the result is passed to the mapped function. Thus, if your comma list that is being mapped is `{a_ , {b_} , { } , {c} , }` then the arguments passed to the mapped function are ‘a’, ‘{b_}’, an empty argument, and ‘c’.

When the comma list is given as an **N**-type argument, spaces have already been trimmed on input, and items are simply stripped of one set of braces if any. This case is more efficient than using **n**-type comma lists.

<code>\clist_map_function:NN</code> <code>\clist_map_function:(cn nn)</code>	<code>\clist_map_function:NN <comma list> <function></code> ☆
---	---

Applies $\langle function \rangle$ to every $\langle item \rangle$ stored in the $\langle comma list \rangle$. The $\langle function \rangle$ will receive one argument for each iteration. The $\langle items \rangle$ are returned from left to right. The function `\clist_map_inline:Nn` is in general more efficient than `\clist_map_function:NN`. One mapping may be nested inside another.

<code>\clist_map_inline:Nn</code> <code>\clist_map_inline:(cn nn)</code>	<code>\clist_map_inline:Nn <comma list> {<inline function>}</code>
---	--

Applies $\langle inline function \rangle$ to every $\langle item \rangle$ stored within the $\langle comma list \rangle$. The $\langle inline function \rangle$ should consist of code which will receive the $\langle item \rangle$ as **#1**. One in line mapping can be nested inside another. The $\langle items \rangle$ are returned from left to right.

<code>\clist_map_variable:Nn</code>	<code>\clist_map_variable:Nn <comma list> <tl var.> {<function using tl var.>}</code>
<code>\clist_map_variable:(cNn nNn)</code>	

Stores each entry in the *<comma list>* in turn in the *<tl var.>* and applies the *<function using tl var.>* The *<function>* will usually consist of code making use of the *<tl var.>*, but this is not enforced. One variable mapping can be nested inside another. The *<items>* are returned from left to right.

<code>\clist_map_break</code> ☆	<code>\clist_map_break:</code>
---------------------------------	--------------------------------

Used to terminate a `\clist_map...` function before all entries in the *<comma list>* have been processed. This will normally take place within a conditional statement, for example

```
\clist_map_inline:Nn \l_my_clist
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \clist_map_break: }
  {
    % Do something useful
  }
}
```

Use outside of a `\clist_map...` scenario will lead to low level TeX errors.

TeXhackers note: When the mapping is broken, additional tokens may be inserted by the internal macro `\prg_break_point:n` before further items are taken from the input stream. This will depend on the design of the mapping function.

<code>\clist_map_break:n</code>	☆	<code>\clist_map_break:n {⟨tokens⟩}</code>
---------------------------------	---	--

Used to terminate a `\clist_map_...` function before all entries in the *⟨comma list⟩* have been processed, inserting the *⟨tokens⟩* after the mapping has ended. This will normally take place within a conditional statement, for example

```
\clist_map_inline:Nn \l_my_clist
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \clist_map_break:n { <tokens> } }
  {
    % Do something useful
  }
}
```

Use outside of a `\clist_map_...` scenario will lead to low level TeX errors.

TeXhackers note: When the mapping is broken, additional tokens may be inserted by the internal macro `\prg_break_point:n` before the *⟨tokens⟩* are inserted into the input stream. This will depend on the design of the mapping function.

111 Comma lists as stacks

Comma lists can be used as stacks, where data is pushed to and popped from the top of the comma list. (The left of a comma list is the top, for performance reasons.) The stack functions for comma lists are not intended to be mixed with the general ordered data functions detailed in the previous section: a comma list should either be used as an ordered data type or as a stack, but not in both ways.

<code>\clist_get:NN</code>	<code>\clist_get:NN ⟨comma list⟩ ⟨token list variable⟩</code>
----------------------------	---

<code>\clist_get:cN</code>	Stores the left-most item from a <i>⟨comma list⟩</i> in the <i>⟨token list variable⟩</i> without removing it from the <i>⟨comma list⟩</i> . The <i>⟨token list variable⟩</i> is assigned locally.
----------------------------	---

<code>\clist_get:NN</code>	<code>\clist_get:NN ⟨comma list⟩ ⟨token list variable⟩</code>
----------------------------	---

<code>\clist_get:cN</code>	Stores the right-most item from a <i>⟨comma list⟩</i> in the <i>⟨token list variable⟩</i> without removing it from the <i>⟨comma list⟩</i> . The <i>⟨token list variable⟩</i> is assigned locally.
----------------------------	--

<code>\clist_pop:NN</code>	<code>\clist_pop:NN ⟨comma list⟩ ⟨token list variable⟩</code>
----------------------------	---

<code>\clist_pop:cN</code>	Pops the left-most item from a <i>⟨comma list⟩</i> into the <i>⟨token list variable⟩</i> , <i>i.e.</i> removes the item from the comma list and stores it in the <i>⟨token list variable⟩</i> . Both of the variables are assigned locally.
----------------------------	---

Updated: 2011-09-06

<hr/> <code>\clist_gpop:Nn</code> <hr/>	<code>\clist_gpop:NN</code> $\langle comma list \rangle$ $\langle token list variable \rangle$
<code>\clist_gpop:cN</code> <hr/>	Pops the left-most item from a $\langle comma list \rangle$ into the $\langle token list variable \rangle$, <i>i.e.</i> removes the item from the comma list and stores it in the $\langle token list variable \rangle$. The $\langle comma list \rangle$ is modified globally, while the assignment of the $\langle token list variable \rangle$ is local.

<hr/> <code>\clist_push:Nn</code> <hr/>	<code>\clist_push:Nn</code> $\langle comma list \rangle$ $\{ \langle items \rangle \}$
<code>\clist_push:(NV No Nx cn cV co cx)</code>	
<code>\clist_gpush:Nn</code>	
<code>\clist_gpush:(NV No Nx cn cV co cx)</code> <hr/>	

Adds the $\{ \langle items \rangle \}$ to the top of the $\langle comma list \rangle$. Spaces are removed from both sides of each item.

112 Viewing comma lists

<hr/> <code>\clist_show:N</code> <hr/>	<code>\clist_show:N</code> $\langle comma list \rangle$
<code>\clist_show:c</code>	Displays the entries in the $\langle comma list \rangle$ in the terminal.
<hr/> <code>\clist_show:n</code> <hr/>	<code>\clist_show:n</code> $\{ \langle tokens \rangle \}$
	Displays the entries in the comma list in the terminal.

113 Scratch comma lists

<hr/> <code>\l_tmpa_clist</code> <hr/>	Scratch comma lists for local assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
<code>\l_tmpb_clist</code>	
<hr/> New: 2011-09-06 <hr/>	
<hr/> <code>\g_tmpa_clist</code> <hr/>	Scratch comma lists for global assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
<code>\g_tmpb_clist</code>	
<hr/> New: 2011-09-06 <hr/>	

114 Experimental comma list functions

This section contains functions which may or may not be retained, depending on how useful they are found to be.

<hr/> <code>\clist_length:N</code> ★	<code>\clist_length:N</code> $\langle comma list \rangle$
<code>\clist_length:(c n)</code> ★	Leaves the number of items in the $\langle comma list \rangle$ in the input stream as an $\langle integer denotation \rangle$. The total number of items in a $\langle comma list \rangle$ will include those which are duplicates, <i>i.e.</i> every item in a $\langle comma list \rangle$ is unique.
<hr/> New: 2011-06-25 <hr/>	
Updated: 2011-09-06 <hr/>	

<code>\clist_item:Nn</code> ★	<code>\clist_item:Nn <comma list> {<integer expression>}</code>
<code>\clist_item:(cn nn)</code> ★	
Updated: 2011-09-06	Indexing items in the <i><comma list></i> from 0 at the top (left), this function will evaluate the <i><integer expression></i> and leave the appropriate item from the comma list in the input stream. If the <i><integer expression></i> is negative, indexing occurs from the bottom (right) of the comma list. When the <i><integer expression></i> is larger than the number of items in the <i><comma list></i> (as calculated by <code>\clist_length:N</code>) then the function will expand to nothing.

<code>\clist_set_from_seq:NN</code>	<code>\clist_set_from_seq:NN <comma list> <sequence></code>
<code>\clist_set_from_seq:(cn Nc cc)</code>	
<code>\clist_gset_from_seq:NN</code>	
<code>\clist_gset_from_seq:(cn Nc cc)</code>	
Updated: 2011-08-31	Sets the <i><comma list></i> to be equal to the content of the <i><sequence></i> . Items which contain either spaces or commas are surrounded by braces.

<code>\clist_const:Nn</code>	<code>\clist_const:Nn <clist var> {<comma list>}</code>
<code>\clist_const:(Nx cn cx)</code>	
New: 2011-11-26	Creates a new constant <i><clist var></i> or raises an error if the name is already taken. The value of the <i><clist var></i> will be set globally to the <i><comma list></i> .

<code>\clist_if_empty_p:n</code> ★	<code>\clist_if_empty_p:n {<comma list>}</code>
<code>\clist_if_empty:nTF</code> ★	<code>\clist_if_empty:nTF {<comma list>} {<true code>} {<false code>}</code>
New: 2011-12-07	Tests if the <i><comma list></i> is empty (containing no items). The rules for space trimming are as for other n-type comma-list functions, hence the comma list <i>{~,~,~}</i> (without outer braces) is empty, while <i>{~,{~,~}}</i> (without outer braces) contains one element, which happens to be empty: the comma-list is not empty.

115 Internal comma-list functions

<code>\clist_trim_spaces:n</code> ☆	<code>\clist_trim_spaces:n {<comma list>}</code>
New: 2011-07-09	Removes leading and trailing spaces from each <i><item></i> in the <i><comma list></i> , leaving the resulting modified list in the input stream. This is used by the functions which add data into a comma list.

Part XIV

The l3prop package

Property lists

L^AT_EX3 implements a “property list” data type, which contain an unordered list of entries each of which consists of a $\langle key \rangle$ and an associated $\langle value \rangle$. The $\langle key \rangle$ and $\langle value \rangle$ may both be any $\langle balanced\ text \rangle$. It is possible to map functions to property lists such that the function is applied to every key–value pair within the list.

Each entry in a property list must have a unique $\langle key \rangle$: if an entry is added to a property list which already contains the $\langle key \rangle$ then the new entry will overwrite the existing one. The $\langle keys \rangle$ are compared on a string basis, using the same method as `\str_if_eq:nn`.

Property lists are intended for storing key-based information for use within code. This is in contrast to key–value lists, which are a form of *input* parsed by the `keys` module.

116 Creating and initialising property lists

```
\prop_new:N
\prop_new:c
```

```
\prop_new:N <property list>
```

Creates a new $\langle property\ list \rangle$ or raises an error if the name is already taken. The declaration is global. The $\langle property\ lists \rangle$ will initially contain no entries.

```
\prop_clear:N
\prop_clear:c
\prop_gclear:N
\prop_gclear:c
```

```
\prop_clear:N <property list>
```

Clears all entries from the $\langle property\ list \rangle$.

```
\prop_clear_new:N
\prop_clear_new:c
\prop_gclear_new:N
\prop_gclear_new:c
```

```
\prop_clear_new:N <property list>
```

Ensures that the $\langle property\ list \rangle$ exists globally by applying `\prop_new:N` if necessary, then applies `\prop_(g)clear:N` to leave the list empty.

```
\prop_set_eq:NN
\prop_set_eq:(cN|Nc|cc)
\prop_gset_eq:NN
\prop_gset_eq:(cN|Nc|cc)
```

```
\prop_set_eq:NN <property list1> <property list2>
```

Sets the content of $\langle property\ list1 \rangle$ equal to that of $\langle property\ list2 \rangle$.

117 Adding entries to property lists

<code>\prop_put:Nnn</code>	<code>\prop_put:Nnn <property list></code>
<code>\prop_put:(NnV Nno Nnx NVn NVV Non Noo cnn cnV cno cnx cVn cVV con coo)</code>	<code>{<key>} {<value>}</code>
<code>\prop_gput:Nnn</code>	
<code>\prop_gput:(NnV Nno Nnx NVn NVV Non Noo cnn cnV cno cnx cVn cVV con coo)</code>	

Adds an entry to the *<property list>* which may be accessed using the *<key>* and which has *<value>*. Both the *<key>* and *<value>* may contain any *<balanced text>*. The *<key>* is stored after processing with `\tl_to_str:n`, meaning that category codes are ignored. If the *<key>* is already present in the *<property list>*, the existing entry is overwritten by the new *<value>*.

<code>\prop_put_if_new:Nnn</code>	<code>\prop_put_if_new:Nnn <property list> {<key>} {<value>}</code>
<code>\prop_put_if_new:cnn</code>	
<code>\prop_gput_if_new:Nnn</code>	If the <i><key></i> is present in the <i><property list></i> then no action is taken. If the <i><key></i> is not present in the <i><property list></i> then a new entry is added. Both the <i><key></i> and <i><value></i> may contain any <i><balanced text></i> . The <i><key></i> is stored after processing with <code>\tl_to_str:n</code> , meaning that category codes are ignored.
<code>\prop_gput_if_new:cnn</code>	

118 Recovering values from property lists

<code>\prop_get:NnN</code>	<code>\prop_get:NnN <property list> {<key>} <tl var></code>
<code>\prop_get:(NVN NoN cnN cVN coN)</code>	

Updated: 2011-08-28

Recovers the *<value>* stored with *<key>* from the *<property list>*, and places this in the *<token list variable>*. If the *<key>* is not found in the *<property list>* then the *<token list variable>* will contain the special marker `\q_no_value`. The *<token list variable>* is set within the current \TeX group. See also `\prop_get:NnNTF`.

<code>\prop_pop:NnN</code>	<code>\prop_pop:NnN <property list> {<key>} <tl var></code>
<code>\prop_pop:(NoN cnN coN)</code>	
Updated: 2011-08-18	Recovers the <i><value></i> stored with <i><key></i> from the <i><property list></i> , and places this in the <i><token list variable></i> . If the <i><key></i> is not found in the <i><property list></i> then the <i><token list variable></i> will contain the special marker <code>\q_no_value</code> . The <i><key></i> and <i><value></i> are then deleted from the property list. Both assignments are local.

<code>\prop_gpop:NnN</code>	<code>\prop_gpop:NnN <property list> {<key>} <tl var></code>
<code>\prop_gpop:(NoN cnN coN)</code>	
Updated: 2011-08-18	Recovers the <i><value></i> stored with <i><key></i> from the <i><property list></i> , and places this in the <i><token list variable></i> . If the <i><key></i> is not found in the <i><property list></i> then the <i><token list variable></i> will contain the special marker <code>\q_no_value</code> . The <i><key></i> and <i><value></i> are then deleted from the property list. The <i><property list></i> is modified globally, while the assignment of the <i><token list variable></i> is local.

119 Modifying property lists

<code>\prop_del:Nn</code>	<code>\prop_del:Nn <property list> {<key>}</code>
<code>\prop_del:(NV cn cV)</code>	
<code>\prop_gdel:Nn</code>	Deletes the entry listed under <i><key></i> from the <i><property list></i> which may be accessed. If
<code>\prop_gdel:(NV cn cV)</code>	the <i><key></i> is not found in the <i><property list></i> no change occurs, <i>i.e</i> there is no need to test
	for the existence of a key before deleting it. The deletion is restricted to the current $\mathrm{T}_{\mathrm{E}}\mathrm{X}$
	group.

120 Property list conditionals

<code>\prop_if_empty_p:N</code> *	<code>\prop_if_empty_p:N <property list></code>
<code>\prop_if_empty_p:c</code> *	<code>\prop_if_empty:NNTF <property list> {<true code>} {<false code>}</code>
<code>\prop_if_empty:NNTF</code> *	Tests if the <i><property list></i> is empty (containing no entries).
<code>\prop_if_empty:cTF</code> *	
<hr/>	
<code>\prop_if_in_p:Nn</code> *	<code>\prop_if_in:NnTF <property list> {<key>} {<true code>} {<false code>}</code>
<code>\prop_if_in_p:(NV No cn cV co)</code> *	
<code>\prop_if_in:NnTF</code> *	
<code>\prop_if_in:(NV No cn cV co)TF</code> *	

Updated: 2011-09-15

Tests if the *<key>* is present in the *<property list>*, making the comparison using the method described by `\str_if_eq:nnTF`.

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ hackers note: This function iterates through every key–value pair in the *<property list>* and is therefore slower than using the non-expandable `\prop_get:NnNTF`.

121 Recovering values from property lists with branching

The functions in this section combine tests for the presence of a key in a property list with recovery of the associated valued. This makes them useful for cases where different cases follow dependent on the presence or absence of a key in a property list. They offer increased readability and performance over separate testing and recovery phases.

<code>\prop_get:NnNTF</code>	<code>\prop_get:NnNTF <property list> {<key>} <token list variable></code>
<code>\prop_get:(NVN NoN cnN cVN coN)TF</code>	<code>{<true code>} {<false code>}</code>

Updated: 2011-08-28

If the *<key>* is not present in the *<property list>*, leaves the *<false code>* in the input stream and leaves the *<token list variable>* unchanged. If the *<key>* is present in the *<property list>*, stores the corresponding *<value>* in the *<token list variable>* without removing it from the *<property list>*. The *<token list variable>* is assigned locally.

`\prop_pop:NnNTF`
`\prop_pop:cnNTF`

New: 2011-08-18

`\prop_pop:NnNTF` $\langle property\ list \rangle$ $\{\langle key \rangle\}$ $\langle token\ list\ variable \rangle$
 $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

If the $\langle key \rangle$ is not present in the $\langle property\ list \rangle$, leaves the $\langle false\ code \rangle$ in the input stream and leaves the $\langle token\ list\ variable \rangle$ unchanged. If the $\langle key \rangle$ is present in the $\langle property\ list \rangle$, pops the corresponding $\langle value \rangle$ in the $\langle token\ list\ variable \rangle$, *i.e.* removes the item from the $\langle property\ list \rangle$. Both the $\langle property\ list \rangle$ and the $\langle token\ list\ variable \rangle$ are assigned locally.

122 Mapping to property lists

`\prop_map_function:Nn` ☆
`\prop_map_function:cn` ☆

`\prop_map_function:Nn` $\langle property\ list \rangle$ $\langle function \rangle$

Applies $\langle function \rangle$ to every $\langle entry \rangle$ stored in the $\langle property\ list \rangle$. The $\langle function \rangle$ will receive two argument for each iteration: the $\langle key \rangle$ and associated $\langle value \rangle$. The order in which $\langle entries \rangle$ are returned is not defined and should not be relied upon.

`\prop_map_inline:Nn`
`\prop_map_inline:cn`

`\prop_map_inline:Nn` $\langle property\ list \rangle$ $\{\langle inline\ function \rangle\}$

Applies $\langle inline\ function \rangle$ to every $\langle entry \rangle$ stored within the $\langle property\ list \rangle$. The $\langle inline\ function \rangle$ should consist of code which will receive the $\langle key \rangle$ as #1 and the $\langle value \rangle$ as #2. The order in which $\langle entries \rangle$ are returned is not defined and should not be relied upon.

`\prop_map_break` ☆

`\prop_map_break:`

Used to terminate a `\prop_map...` function before all entries in the $\langle property\ list \rangle$ have been processed. This will normally take place within a conditional statement, for example

```
\prop_map_inline:Nn \l_my_prop
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \prop_map_break: }
  {
    % Do something useful
  }
}
```

Use outside of a `\prop_map...` scenario will lead low level TeX errors.

`\prop_map_break:n` ☆

`\prop_map_break:n` $\{(tokens)\}$

Used to terminate a `\prop_map...` function before all entries in the $\langle property list \rangle$ have been processed, inserting the $\langle tokens \rangle$ after the mapping has ended. This will normally take place within a conditional statement, for example

```
\prop_map_inline:Nn \l_my_prop
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \prop_map_break:n { <tokens> } }
  {
    % Do something useful
  }
}
```

Use outside of a `\prop_map...` scenario will lead low level T_EX errors.

123 Viewing property lists

`\prop_show:N`

`\prop_show:N` $\langle property list \rangle$

`\prop_show:c`

Displays the entries in the $\langle property list \rangle$ in the terminal.

124 Experimental property list functions

This section contains functions which may or may not be retained, depending on how useful they are found to be.

`\prop_gpop:NnNTF`

`\prop_gpop:cnNTF`

New: 2011-08-18

`\prop_gpop:NnNTF` $\langle property list \rangle$ $\{\langle key \rangle\}$ $\langle token list variable \rangle$
 $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

If the $\langle key \rangle$ is not present in the $\langle property list \rangle$, leaves the $\langle false code \rangle$ in the input stream and leaves the $\langle token list variable \rangle$ unchanged. If the $\langle key \rangle$ is present in the $\langle property list \rangle$, pops the corresponding $\langle value \rangle$ in the $\langle token list variable \rangle$, *i.e.* removes the item from the $\langle property list \rangle$. The $\langle property list \rangle$ is modified globally, while the $\langle token list variable \rangle$ is assigned locally.

`\prop_map_tokens:Nn` ☆

`\prop_map_tokens:cn` ☆

New: 2011-08-18

`\prop_map_tokens:Nn` $\langle property list \rangle$ $\{\langle code \rangle\}$

Analogue of `\prop_map_function:NN` which maps several tokens instead of a single function. Useful in particular when mapping through a property list while keeping track of a given key.

<code>\prop_get:Nn</code> ★	<code>\prop_get:Nn <property list> {<key>}</code>
<code>\prop_get:cn</code> ★	Expands to the <i><value></i> corresponding to the <i><key></i> in the <i><property list></i> . If the <i><key></i> is missing, this has an empty expansion.

Updated: 2011-09-15

T_EXhackers note: This function is slower than the non-expandable analogue `\prop_get:NnN`.

125 Internal property list functions

<code>\q_prop</code>	The internal token used to separate out property list entries, separating both the <i><key></i> from the <i><value></i> and also one entry from another.
----------------------	--

<code>\c_empty_prop</code>	A permanently-empty property list used for internal comparisons.
----------------------------	--

<code>\prop_split:Nnn</code>	<code>\prop_split:Nnn <property list> {<key>} {<code>}</code> Splits the <i><property list></i> at the <i><key></i> , giving three groups: the <i><extract></i> of <i><property list></i> before the <i><key></i> , the <i><value></i> associated with the <i><key></i> and the <i><extract></i> of the <i><property list></i> after the <i><value></i> . The first <i><extract></i> retains the internal structure of a property list. The second is only missing the leading separator <code>\q_prop</code> . This ensures that the concatenation of the two <i><extracts></i> is a property list. If the <i><key></i> is not present in the <i><property list></i> then the second group will contain the marker <code>\q_no_value</code> and the third is empty. Once the split has occurred, the <i><code></i> is inserted followed by the three groups: thus the <i><code></i> should properly absorb three arguments. The <i><key></i> comparison takes place as described for <code>\str_if_eq:nn</code> .
------------------------------	---

<code>\prop_split:NnTF</code>	<code>\prop_split:NnTF <property list> {<key>} {<true code>} {<false code>}</code> Splits the <i><property list></i> at the <i><key></i> , giving three groups: the <i><extract></i> of <i><property list></i> before the <i><key></i> , the <i><value></i> associated with the <i><key></i> and the <i><extract></i> of the <i><property list></i> after the <i><value></i> . The first <i><extract></i> retains the internal structure of a property list. The second is only missing the leading separator <code>\q_prop</code> . This ensures that the concatenation of the two <i><extracts></i> is a property list. If the <i><key></i> is present in the <i><property list></i> then the <i><true code></i> is left in the input stream, followed by the three groups: thus the <i><true code></i> should properly absorb three arguments. If the <i><key></i> is not present in the <i><property list></i> then the <i><false code></i> is left in the input stream, with no trailing material. The <i><key></i> comparison takes place as described for <code>\str_if_eq:nn</code> .
-------------------------------	--

Part XV

The l3box package

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_`.

126 Creating and initialising boxes

<code>\box_new:N</code>	<code>\box_new:N <box></code>
<code>\box_new:c</code>	Creates a new $\langle box \rangle$ or raises an error if the name is already taken. The declaration is global. The $\langle box \rangle$ will initially be void.

<code>\box_clear:N</code>	<code>\box_clear:N <box></code>
<code>\box_clear:c</code>	Clears the content of the $\langle box \rangle$ by setting the box equal to <code>\c_void_box</code> .
<code>\box_gclear:N</code>	
<code>\box_gclear:c</code>	

<code>\box_clear_new:N</code>	<code>\box_clear_new:N <box></code>
<code>\box_clear_new:c</code>	Ensures that the $\langle box \rangle$ exists globally by applying <code>\box_new:N</code> if necessary, then applies <code>\box_(g)clear:N</code> to leave the $\langle box \rangle$ empty.
<code>\box_gclear_new:N</code>	
<code>\box_gclear_new:c</code>	

<code>\box_set_eq:NN</code>	<code>\box_set_eq:NN <box1> <box2></code>
<code>\box_set_eq:(cN Nc cc)</code>	Sets the content of $\langle box1 \rangle$ equal to that of $\langle box2 \rangle$.
<code>\box_gset_eq:NN</code>	
<code>\box_gset_eq:(cN Nc cc)</code>	

<code>\box_set_eq_clear:NN</code>	<code>\box_set_eq_clear:NN <box1> <box2></code>
<code>\box_set_eq_clear:(cN Nc cc)</code>	Sets the content of $\langle box1 \rangle$ within the current TeX group equal to that of $\langle box2 \rangle$, then clears $\langle box2 \rangle$ globally.

<code>\box_gset_eq_clear:NN</code>	<code>\box_gset_eq_clear:NN <box1> <box2></code>
<code>\box_gset_eq_clear:(cN Nc cc)</code>	Sets the content of $\langle box1 \rangle$ equal to that of $\langle box2 \rangle$, then clears $\langle box2 \rangle$. These assignments are global.

127 Using boxes

<code>\box_use:N</code>	<code>\box_use:N <box></code>
<code>\box_use:c</code>	Inserts the current content of the $\langle box \rangle$ onto the current list for typesetting.

T_EXhackers note: This is the T_EX primitive `\copy`.

<code>\box_use_clear:N</code>	<code>\box_use_clear:N <box></code>
<code>\box_use_clear:c</code>	Inserts the current content of the $\langle box \rangle$ onto the current list for typesetting, then globally clears the content of the $\langle box \rangle$.

T_EXhackers note: This is the T_EX primitive `\box`.

<code>\box_move_right:nn</code>	<code>\box_move_right:nn {<dimexpr>} {<box function>}</code>
<code>\box_move_left:nn</code>	This function operates in vertical mode, and inserts the material specified by the $\langle box \text{ function} \rangle$ such that its reference point is displaced horizontally by the given $\langle dimexpr \rangle$ from the reference point for typesetting, to the right or left as appropriate. The $\langle box \text{ function} \rangle$ should be a box operation such as <code>\box_use:N <box></code> or a “raw” box specification such as <code>\vbox:n { xyz }</code> .

<code>\box_move_up:nn</code>	<code>\box_move_up:nn {<dimexpr>} {<box function>}</code>
<code>\box_move_down:nn</code>	This function operates in horizontal mode, and inserts the material specified by the $\langle box \text{ function} \rangle$ such that its reference point is displaced vertical by the given $\langle dimexpr \rangle$ from the reference point for typesetting, up or down as appropriate. The $\langle box \text{ function} \rangle$ should be a box operation such as <code>\box_use:N <box></code> or a “raw” box specification such as <code>\vbox:n { xyz }</code> .

128 Measuring and setting box dimensions

<code>\box_dp:N</code>	<code>\box_dp:N <box></code>
<code>\box_dp:c</code>	Calculates the depth (below the baseline) of the $\langle box \rangle$ in a form suitable for use in a $\langle dimension \text{ expression} \rangle$.

T_EXhackers note: This is the T_EX primitive `\dp`.

<code>\box_ht:N</code>	<code>\box_ht:N <box></code>
<code>\box_ht:c</code>	Calculates the height (above the baseline) of the $\langle box \rangle$ in a form suitable for use in a $\langle dimension \text{ expression} \rangle$.

T_EXhackers note: This is the T_EX primitive `\ht`.

<hr/> <code>\box_wd:N</code> <hr/>	<code>\box_wd:N <box></code>
<code>\box_wd:c</code> <hr/>	Calculates the width of the <code><box></code> in a form suitable for use in a <i><dimension expression></i> .
TeXhackers note: This is the TeX primitive <code>\wd</code> .	
<hr/> <code>\box_set_dp:Nn</code> <hr/>	<code>\box_set_dp:Nn <box> {<dimension expression>}</code>
<code>\box_set_dp:cn</code> <hr/>	Set the depth (below the baseline) of the <code><box></code> to the value of the <i>{<dimension expression>}</i> . This is a global assignment.
Updated: 2011-10-22 <hr/>	
<hr/> <code>\box_set_ht:Nn</code> <hr/>	<code>\box_set_ht:Nn <box> {<dimension expression>}</code>
<code>\box_set_ht:cn</code> <hr/>	Set the height (above the baseline) of the <code><box></code> to the value of the <i>{<dimension expression>}</i> . This is a global assignment.
Updated: 2011-10-22 <hr/>	
<hr/> <code>\box_set_wd:Nn</code> <hr/>	<code>\box_set_wd:Nn <box> {<dimension expression>}</code>
<code>\box_set_wd:cn</code> <hr/>	Set the width of the <code><box></code> to the value of the <i>{<dimension expression>}</i> . This is a global assignment.
Updated: 2011-10-22 <hr/>	

129 Affine transformations

Affine transformations are changes which (informally) preserve straight lines. Simple translations are affine transformations, but are better handled in TeX by doing the translation first, then inserting an unmodified box. On the other hand, rotation and resizing of boxed material can best be handled by modifying boxes. These transformations are described here.

<hr/> <code>\box_resize:Nnn</code> <hr/>	<code>\box_resize:Nnn <box> {<x-size>} {<y-size>}</code>
<code>\box_resize:cnn</code> <hr/>	Resize the <code><box></code> to <code><x-size></code> horizontally and <code><y-size></code> vertically (both of the sizes are dimension expressions). The <code><y-size></code> is the vertical size (height plus depth) of the box. The updated <code><box></code> will be an hbox, irrespective of the nature of the <code><box></code> before the resizing is applied. Negative sizes will cause the material in the <code><box></code> to be reversed in direction, but the reference point of the <code><box></code> will be unchanged. The resizing applies within the current TeX group level.
New: 2011-09-02 <hr/>	This function is experimental

```
\box_resize_to_ht_plus_dp:Nn \box_resize_to_ht_plus_dp:Nn <box> {<y-size>}
\box_resize_to_ht_plus_dp:cn
```

New: 2011-09-02

Updated: 2011-10-22

Resize the $\langle box \rangle$ to $\langle y-size \rangle$ vertically, scaling the horizontal size by the same amount ($\langle y-size \rangle$ is a dimension expression). The $\langle y-size \rangle$ is the vertical size (height plus depth) of the box. The updated $\langle box \rangle$ will be an hbox, irrespective of the nature of the $\langle box \rangle$ before the resizing is applied. A negative size will cause the material in the $\langle box \rangle$ to be reversed in direction, but the reference point of the $\langle box \rangle$ will be unchanged. The resizing applies within the current \TeX group level.

This function is experimental

```
\box_resize_to_wd:Nnn \box_resize_to_wd:Nnn <box> {<x-size>}
\box_resize_to_wd:cn
```

New: 2011-09-02

Updated: 2011-10-22

Resize the $\langle box \rangle$ to $\langle x-size \rangle$ horizontally, scaling the vertical size by the same amount ($\langle x-size \rangle$ is a dimension expression). The updated $\langle box \rangle$ will be an hbox, irrespective of the nature of the $\langle box \rangle$ before the resizing is applied. A negative size will cause the material in the $\langle box \rangle$ to be reversed in direction, but the reference point of the $\langle box \rangle$ will be unchanged. The resizing applies within the current \TeX group level.

This function is experimental

```
\box_rotate:Nn \box_rotate:Nn <box> {<angle>}
\box_rotate:cn
```

New: 2011-09-02

Updated: 2011-10-22

Rotates the $\langle box \rangle$ by $\langle angle \rangle$ (in degrees) anti-clockwise about its reference point. The reference point of the updated box will be moved horizontally such that it is at the left side of the smallest rectangle enclosing the rotated material. The updated $\langle box \rangle$ will be an hbox, irrespective of the nature of the $\langle box \rangle$ before the rotation is applied. The rotation applies within the current \TeX group level.

This function is experimental

```
\box_scale:Nnn \box_scale:Nnn <box> {<x-scale>} {<y-scale>}
\box_scale:cnn
```

New: 2011-09-02

Updated: 2011-10-22

Scales the $\langle box \rangle$ by factors $\langle x-scale \rangle$ and $\langle y-scale \rangle$ in the horizontal and vertical directions, respectively (both scales are integer expressions). The updated $\langle box \rangle$ will be an hbox, irrespective of the nature of the $\langle box \rangle$ before the scaling is applied. Negative scalings will cause the material in the $\langle box \rangle$ to be reversed in direction, but the reference point of the $\langle box \rangle$ will be unchanged. The scaling applies within the current \TeX group level.

This function is experimental

130 Viewing part of a box

`\box_clip:N`
`\box_clip:c`

New: 2011-11-13

`\box_clip:N` $\langle box \rangle$

Clips the $\langle box \rangle$ in the output so that only material inside the bounding box is displayed in the output. The updated $\langle box \rangle$ will be an hbox, irrespective of the nature of the $\langle box \rangle$ before the clipping is applied. The clipping applies within the current T_EX group level.

This function is experimental

T_EXhackers note: Clipping is implemented by the driver, and as such the full content of the box is places in the output file. Thus clipping does not remove any information from the raw output, and hidden material can therefore be viewed by direct examination of the file.

`\box_trim:Nnnnn`
`\box_trim:cnnnn`

New: 2011-11-13

`\box_trim:Nnnnn` $\langle box \rangle$ $\{\langle left \rangle\}$ $\{\langle bottom \rangle\}$ $\{\langle right \rangle\}$ $\{\langle top \rangle\}$

Adjusts the bounding box of the $\langle box \rangle$ $\langle left \rangle$ is removed from the left-hand edge of the bounding box, $\langle right \rangle$ from the right-hand edge and so fourth. All adjustments are *dimension expressions*. Material output of the bounding box will still be displayed in the output unless `\box_clip:N` is subsequently applied. The updated $\langle box \rangle$ will be an hbox, irrespective of the nature of the $\langle box \rangle$ before the viewport operation is applied. The clipping applies within the current T_EX group level.

This function is experimental

`\box_viewport:Nnnnn`
`\box_viewport:cnnnn`

New: 2011-11-13

`\box_viewport:Nnnnn` $\langle box \rangle$ $\{\langle llx \rangle\}$ $\{\langle lly \rangle\}$ $\{\langle urx \rangle\}$ $\{\langle ury \rangle\}$

Adjusts the bounding box of the $\langle box \rangle$ such that it has lower-left co-ordinates ($\langle llx \rangle$, $\langle lly \rangle$) and upper-right co-ordinates ($\langle urx \rangle$, $\langle ury \rangle$). All four co-ordinate positions are *dimension expressions*. Material output of the bounding box will still be displayed in the output unless `\box_clip:N` is subsequently applied. The updated $\langle box \rangle$ will be an hbox, irrespective of the nature of the $\langle box \rangle$ before the viewport operation is applied. The clipping applies within the current T_EX group level.

This function is experimental

131 Box conditionals

`\box_if_empty_p:N` ★
`\box_if_empty_p:c` ★
`\box_if_empty:NTF` ★
`\box_if_empty:cTF` ★

`\box_if_empty_p:N` $\langle box \rangle$

`\box_if_empty:NTF` $\langle box \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if $\langle box \rangle$ is a empty (equal to `\c_empty_box`).

`\box_if_horizontal_p:N` ★
`\box_if_horizontal_p:c` ★
`\box_if_horizontal:NTF` ★
`\box_if_horizontal:cTF` ★

`\box_if_horizontal_p:N` $\langle box \rangle$

`\box_if_horizontal:NTF` $\langle box \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Tests if $\langle box \rangle$ is a horizontal box.

<code>\box_if_vertical_p:N</code> *	<code>\box_if_vertical_p:N</code> $\langle box \rangle$
<code>\box_if_vertical_p:c</code> *	<code>\box_if_vertical:NTF</code> $\langle box \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
<code>\box_if_vertical:NTF</code> *	Tests if $\langle box \rangle$ is a vertical box.
<code>\box_if_vertical:cTF</code> *	

132 The last box inserted

<code>\box_set_to_last:N</code>	<code>\box_set_to_last:N</code> $\langle box \rangle$
<code>\box_set_to_last:c</code>	Sets the $\langle box \rangle$ equal to the last item (box) added to the current partial list, removing the item from the list at the same time. When applied to the main vertical list, the $\langle box \rangle$ will always be void as it is not possible to recover the last added item.
<code>\box_gset_to_last:N</code>	
<code>\box_gset_to_last:c</code>	

133 Constant boxes

<code>\c_empty_box</code>	This is a permanently empty box, which is neither set as horizontal nor vertical.
---------------------------	---

134 Scratch boxes

<code>\l_tmpa_box</code>	Scratch boxes for local assignment. These are never used by the kernel code, and so are safe for use with any L ^A T _E X3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
<code>\l_tmpb_box</code>	

135 Viewing box contents

<code>\box_show:N</code>	<code>\box_show:N</code> $\langle box \rangle$
<code>\box_show:c</code>	Writes the contents of $\langle box \rangle$ to the log file.

T_EXhackers note: This is a wrapper around the T_EX primitive `\showbox`.

136 Horizontal mode boxes

<code>\hbox:n</code>	<code>\hbox:n</code> $\{\langle contents \rangle\}$
	Typesets the $\langle contents \rangle$ into a horizontal box of natural width and then includes this box in the current list for typesetting.

T_EXhackers note: This is the T_EX primitive `\hbox`.

<hr/> <hr/>	<code>\hbox_to_wd:nn</code>	<code>\hbox_to_wd:nn {<dimexpr>} {<contents>}</code>
		Typesets the $\langle contents \rangle$ into a horizontal box of width $\langle dimexpr \rangle$ and then includes this box in the current list for typesetting.
<hr/> <hr/>	<code>\hbox_to_zero:n</code>	<code>\hbox_to_zero:n {<contents>}</code>
		Typesets the $\langle contents \rangle$ into a horizontal box of zero width and then includes this box in the current list for typesetting.
<hr/> <hr/>	<code>\hbox_set:Nn</code> <code>\hbox_set:cn</code> <code>\hbox_gset:Nn</code> <code>\hbox_gset:cn</code>	<code>\hbox_set:Nn <box> {<contents>}</code> Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$.
<hr/> <hr/>	<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>	<code>\hbox_set_to_wd:Nnn <box> {<dimexpr>} {<contents>}</code> Typesets the $\langle contents \rangle$ to the width given by the $\langle dimexpr \rangle$ and then stores the result inside the $\langle box \rangle$.
<hr/> <hr/>	<code>\hbox_overlap_right:n</code>	<code>\hbox_overlap_right:n {<contents>}</code> Typesets the $\langle contents \rangle$ into a horizontal box of zero width such that material will protrude to the right of the insertion point.
<hr/> <hr/>	<code>\hbox_overlap_left:n</code>	<code>\hbox_overlap_left:n {<contents>}</code> Typesets the $\langle contents \rangle$ into a horizontal box of zero width such that material will protrude to the left of the insertion point.
<hr/> <hr/>	<code>\hbox_set:Nw</code> <code>\hbox_set:cw</code> <code>\hbox_set_end</code> <code>\hbox_gset:Nw</code> <code>\hbox_gset:cw</code> <code>\hbox_gset_end</code>	<code>\hbox_set:Nw <box> <contents> \hbox_set_end:</code> Typesets the $\langle contents \rangle$ at natural width and then stores the result inside the $\langle box \rangle$. In contrast to <code>\hbox_set:Nn</code> this function does not absorb the argument when finding the $\langle content \rangle$, and so can be used in circumstances where the $\langle content \rangle$ may not be a simple argument.
<hr/> <hr/>	<code>\hbox_unpack:N</code> <code>\hbox_unpack:c</code>	<code>\hbox_unpack:N <box></code> Unpacks the content of the horizontal $\langle box \rangle$, retaining any stretching or shrinking applied when the $\langle box \rangle$ was set.

T_EXhackers note: This is the T_EX primitive `\unhcopy`.

`\hbox_unpack_clear:N`
`\hbox_unpack_clear:c`

`\hbox_unpack_clear:N` $\langle box \rangle$

Unpacks the content of the horizontal $\langle box \rangle$, retaining any stretching or shrinking applied when the $\langle box \rangle$ was set. The $\langle box \rangle$ is then cleared globally.

T_EXhackers note: This is the T_EX primitive `\unhbox`.

137 Vertical mode boxes

Vertical boxes inherit their baseline from their contents. The standard case is that the baseline of the box is at the same position as that of the last item added to the box. This means that the box will have no depth unless the last item added to it had depth. As a result most vertical boxes have a large height value and small or zero depth. The exception are `_top` boxes, where the reference point is that of the first item added. These tend to have a large depth and small height, although the latter will typically be non-zero.

`\vbox:n`

`\vbox:n` $\{\langle contents \rangle\}$

Updated: 2011-12-18

Typesets the $\langle contents \rangle$ into a vertical box of natural height and includes this box in the current list for typesetting.

T_EXhackers note: This is the T_EX primitive `\vbox`.

`\vbox_top:n`

`\vbox_top:n` $\{\langle contents \rangle\}$

Updated: 2011-12-18

Typesets the $\langle contents \rangle$ into a vertical box of natural height and includes this box in the current list for typesetting. The baseline of the box will be equal to that of the *first* item added to the box.

T_EXhackers note: This is the T_EX primitive `\vtop`.

`\vbox_to_ht:nn`

`\vbox_to_ht:nn` $\{\langle dimexpr \rangle\} \{\langle contents \rangle\}$

Updated: 2011-12-18

Typesets the $\langle contents \rangle$ into a vertical box of height $\langle dimexpr \rangle$ and then includes this box in the current list for typesetting.

`\vbox_to_zero:n`

`\vbox_to_zero:n` $\{\langle contents \rangle\}$

Updated: 2011-12-18

Typesets the $\langle contents \rangle$ into a vertical box of zero height and then includes this box in the current list for typesetting.

`\vbox_set:Nn`

`\vbox_set:Nn` $\langle box \rangle \{\langle contents \rangle\}$

`\vbox_set:cn`

Typesets the $\langle contents \rangle$ at natural height and then stores the result inside the $\langle box \rangle$.

`\vbox_gset:Nn`

`\vbox_gset:cn`

Updated: 2011-12-18

`\vbox_set_top:Nn`
`\vbox_set_top:cn`
`\vbox_gset_top:Nn`
`\vbox_gset_top:cn`

Updated: 2011-12-18

`\vbox_set_top:Nn` $\langle box \rangle$ $\{\langle contents \rangle\}$

Typesets the $\langle contents \rangle$ at natural height and then stores the result inside the $\langle box \rangle$. The baseline of the box will be equal to that of the *first* item added to the box.

`\vbox_set_to_ht:Nnn`
`\vbox_set_to_ht:cnn`
`\vbox_gset_to_ht:Nnn`
`\vbox_gset_to_ht:cnn`

Updated: 2011-12-18

`\vbox_set_to_ht:Nnn` $\langle box \rangle$ $\{\langle dimexpr \rangle\}$ $\{\langle contents \rangle\}$

Typesets the $\langle contents \rangle$ to the height given by the $\langle dimexpr \rangle$ and then stores the result inside the $\langle box \rangle$.

`\vbox_set:Nw`
`\vbox_set:cw`
`\vbox_set_end`
`\vbox_gset:Nw`
`\vbox_gset:cw`
`\vbox_gset_end`

Updated: 2011-12-18

`\vbox_begin:Nw` $\langle box \rangle$ $\langle contents \rangle$ `\vbox_set_end:`

Typesets the $\langle contents \rangle$ at natural height and then stores the result inside the $\langle box \rangle$. In contrast to `\vbox_set:Nn` this function does not absorb the argument when finding the $\langle content \rangle$, and so can be used in circumstances where the $\langle content \rangle$ may not be a simple argument.

`\vbox_set_split_to_ht:Nnn`

Updated: 2011-10-22

`\vbox_set_split_to_ht:Nnn` $\langle box1 \rangle$ $\langle box2 \rangle$ $\{\langle dimexpr \rangle\}$

Sets $\langle box1 \rangle$ to contain material to the height given by the $\langle dimexpr \rangle$ by removing content from the top of $\langle box2 \rangle$ (which must be a vertical box).

T_EXhackers note: This is the T_EX primitive `\vsplit`.

`\vbox_unpack:N`
`\vbox_unpack:c`

`\vbox_unpack:N` $\langle box \rangle$

Unpacks the content of the vertical $\langle box \rangle$, retaining any stretching or shrinking applied when the $\langle box \rangle$ was set.

T_EXhackers note: This is the T_EX primitive `\unvcopy`.

`\vbox_unpack_clear:N`
`\vbox_unpack_clear:c`

`\vbox_unpack:N` $\langle box \rangle$

Unpacks the content of the vertical $\langle box \rangle$, retaining any stretching or shrinking applied when the $\langle box \rangle$ was set. The $\langle box \rangle$ is then cleared globally.

T_EXhackers note: This is the T_EX primitive `\unvbox`.

138 Primitive box conditionals

<code>\if_hbox:N</code>	★
-------------------------	---

```
\if_hbox:N <box>
  <true code>
\else:
  <false code>
\fi:
```

Tests is $\langle box \rangle$ is a horizontal box.

T_EXhackers note: This is the T_EX primitive `\ifhbox`.

<code>\if_vbox:N</code>	★
-------------------------	---

```
\if_vbox:N <box>
  <true code>
\else:
  <false code>
\fi:
```

Tests is $\langle box \rangle$ is a vertical box.

T_EXhackers note: This is the T_EX primitive `\ifvbox`.

<code>\if_box_empty:N</code>	★
------------------------------	---

```
\if_box_empty:N <box>
  <true code>
\else:
  <false code>
\fi:
```

Tests is $\langle box \rangle$ is an empty (void) box.

T_EXhackers note: This is the T_EX primitive `\ifvoid`.

139 Experimental box functions

<code>\box_show:Nnn</code>	
----------------------------	--

```
\box_show:Nnn <box> <int 1> <int 2>
```

<code>\box_show:cnn</code>

New: 2011-11-21

Display the contents of $\langle box \rangle$ in the terminal, showing the first $\langle int\ 1 \rangle$ items of the box, and descending into $\langle int\ 1 \rangle$ levels of nesting.

T_EXhackers note: This is a wrapper around the T_EX primitives `\showbox`, `\showboxbreadth` and `\showboxdepth`.

<code>\box_show_full:N</code>	
-------------------------------	--

```
\box_show_full:N <box>
```

<code>\box_show_full:c</code>

New: 2011-11-22

Display the contents of $\langle box \rangle$ in the terminal, showing all items in the box.

Part XVI

The l3coffins package

Coffin code layer

The material in this module provides the low-level support system for coffins. For details about the design concept of a coffin, see the xcoffins module (in the l3experimental bundle).

140 Creating and initialising coffins

`\coffin_new:N``\coffin_new:c`

`New: 2011-08-17`

`\coffin_new:N` $\langle coffin \rangle$

Creates a new $\langle coffin \rangle$ or raises an error if the name is already taken. The declaration is global. The $\langle coffin \rangle$ will initially be empty.

`\coffin_clear:N``\coffin_clear:c`

`New: 2011-08-17`

`\coffin_clear:N` $\langle coffin \rangle$

Clears the content of the $\langle coffin \rangle$ within the current T_EX group level.

`\coffin_set_eq:NN``\coffin_set_eq:(Nc|cN|cc)`

`New: 2011-08-17`

`\coffin_set_eq:NN` $\langle coffin1 \rangle$ $\langle coffin2 \rangle$

Sets both the content and poles of $\langle coffin1 \rangle$ equal to those of $\langle coffin2 \rangle$ within the current T_EX group level.

141 Setting coffin content and poles

All coffin functions create and manipulate coffins locally within the current T_EX group level.

`\hcoffin_set:Nn``\hcoffin_set:cn`

`New: 2011-08-17``Updated: 2011-09-03`

`\hcoffin_set:Nn` $\langle coffin \rangle$ $\{ \langle material \rangle \}$

Typesets the $\langle material \rangle$ in horizontal mode, storing the result in the $\langle coffin \rangle$. The standard poles for the $\langle coffin \rangle$ are then set up based on the size of the typeset material.

`\hcoffin_set:Nw``\hcoffin_set:cw``\hcoffin_set_end`

`New: 2011-09-10`

`\hcoffin_set:Nw` $\langle coffin \rangle$ $\langle material \rangle$ `\hcoffin_set_end:`

Typesets the $\langle material \rangle$ in horizontal mode, storing the result in the $\langle coffin \rangle$. The standard poles for the $\langle coffin \rangle$ are then set up based on the size of the typeset material. These functions are useful for setting the entire contents of an environment in a coffin.

<code>\vcoffin_set:Nnn</code>	<code>\vcoffin_set:Nnn <coffin> {<width>} {<material>}</code>
<code>\vcoffin_set:cnn</code>	

New: 2011-08-17
Updated: 2011-09-03

Typesets the $\langle material \rangle$ in vertical mode constrained to the given $\langle width \rangle$ and stores the result in the $\langle coffin \rangle$. The standard poles for the $\langle coffin \rangle$ are then set up based on the size of the typeset material.

<code>\vcoffin_set:Nnw</code>	<code>\vcoffin_set:Nnw <coffin> {<width>} <material> \vcoffin_set_end:</code>
<code>\vcoffin_set:cnnw</code>	
<code>\vcoffin_set_end</code>	

New: 2011-09-10

Typesets the $\langle material \rangle$ in vertical mode constrained to the given $\langle width \rangle$ and stores the result in the $\langle coffin \rangle$. The standard poles for the $\langle coffin \rangle$ are then set up based on the size of the typeset material. These functions are useful for setting the entire contents of an environment in a coffin.

<code>\coffin_set_horizontal_pole:Nnn</code>	<code>\coffin_set_horizontal_pole:Nnn <coffin></code>
<code>\coffin_set_horizontal_pole:cnn</code>	<code>{<pole>} {<offset>}</code>

New: 2011-08-17

Sets the $\langle pole \rangle$ to run horizontally through the $\langle coffin \rangle$. The $\langle pole \rangle$ will be located at the $\langle offset \rangle$ from the bottom edge of the bounding box of the $\langle coffin \rangle$. The $\langle offset \rangle$ should be given as a dimension expression; this may include the terms `\TotalHeight`, `\Height`, `\Depth` and `\Width`, which will evaluate to the appropriate dimensions of the $\langle coffin \rangle$.

<code>\coffin_set_vertical_pole:Nnn</code>	<code>\coffin_set_vertical_pole:Nnn <coffin> {<pole>} {<offset>}</code>
<code>\coffin_set_vertical_pole:cnn</code>	

New: 2011-08-17

Sets the $\langle pole \rangle$ to run vertically through the $\langle coffin \rangle$. The $\langle pole \rangle$ will be located at the $\langle offset \rangle$ from the left-hand edge of the bounding box of the $\langle coffin \rangle$. The $\langle offset \rangle$ should be given as a dimension expression; this may include the terms `\TotalHeight`, `\Height`, `\Depth` and `\Width`, which will evaluate to the appropriate dimensions of the $\langle coffin \rangle$.

142 Coffin transformations

<code>\coffin_resize:Nnn</code>	<code>\coffin_resize:Nnn <coffin> {<width>} {<total-height>}</code>
<code>\coffin_resize:cnn</code>	

New: 2011-09-02

Resized the $\langle coffin \rangle$ to $\langle width \rangle$ and $\langle total-height \rangle$, both of which should be given as dimension expressions. These may include the terms `\TotalHeight`, `\Height`, `\Depth` and `\Width`, which will evaluate to the appropriate dimensions of the $\langle coffin \rangle$.

This function is experimental.

<code>\coffin_rotate:Nn</code>	<code>\coffin_rotate:Nn <coffin> {<angle>}</code>
<code>\coffin_rotate:cnn</code>	

New: 2011-09-02

Rotates the $\langle coffin \rangle$ by the given $\langle angle \rangle$ (given in degrees counter-clockwise). This process will rotate both the coffin content and poles. Multiple rotations will not result in the bounding box of the coffin growing unnecessarily.

```
\coffin_scale:Nnn
\coffin_scale:cnn
```

New: 2011-09-02

```
\coffin_scale:Nnn <coffin> {\<x-scale>} {\<y-scale>}
```

Scales the $\langle coffin \rangle$ by a factors $\langle x-scale \rangle$ and $\langle y-scale \rangle$ in the horizontal and vertical directions, respectively. The two scale factors should be given as real numbers.

This function is experimental.

143 Joining and using coffins

```
\coffin_attach:NnnNnnnn
```

```
\coffin_attach:(cnnNnnnn|Nnnncnnnn|cnnncnnnn)
```

```
\coffin_attach:NnnNnnnn
```

```
<coffin1> {\<coffin1-pole1>} {\<coffin1-pole2>}
<coffin2> {\<coffin2-pole1>} {\<coffin2-pole2>}
{\<x-offset>} {\<y-offset>}
```

This function attaches $\langle coffin2 \rangle$ to $\langle coffin1 \rangle$ such that the bounding box of $\langle coffin1 \rangle$ is not altered, *i.e.* $\langle coffin2 \rangle$ can protrude outside of the bounding box of the coffin. The alignment is carried out by first calculating $\langle handle1 \rangle$, the point of intersection of $\langle coffin1-pole1 \rangle$ and $\langle coffin1-pole2 \rangle$, and $\langle handle2 \rangle$, the point of intersection of $\langle coffin2-pole1 \rangle$ and $\langle coffin2-pole2 \rangle$. $\langle coffin2 \rangle$ is then attached to $\langle coffin1 \rangle$ such that the relationship between $\langle handle1 \rangle$ and $\langle handle2 \rangle$ is described by the $\langle x-offset \rangle$ and $\langle y-offset \rangle$. The two offsets should be given as dimension expressions.

```
\coffin_join:NnnNnnnn
```

```
\coffin_join:(cnnNnnnn|Nnnncnnnn|cnnncnnnn)
```

```
\coffin_join:NnnNnnnn
```

```
<coffin1> {\<coffin1-pole1>} {\<coffin1-pole2>}
<coffin2> {\<coffin2-pole1>} {\<coffin2-pole2>}
{\<x-offset>} {\<y-offset>}
```

This function joins $\langle coffin2 \rangle$ to $\langle coffin1 \rangle$ such that the bounding box of $\langle coffin1 \rangle$ may expand. The new bounding box will cover the area containing the bounding boxes of the two original coffins. The alignment is carried out by first calculating $\langle handle1 \rangle$, the point of intersection of $\langle coffin1-pole1 \rangle$ and $\langle coffin1-pole2 \rangle$, and $\langle handle2 \rangle$, the point of intersection of $\langle coffin2-pole1 \rangle$ and $\langle coffin2-pole2 \rangle$. $\langle coffin2 \rangle$ is then attached to $\langle coffin1 \rangle$ such that the relationship between $\langle handle1 \rangle$ and $\langle handle2 \rangle$ is described by the $\langle x-offset \rangle$ and $\langle y-offset \rangle$. The two offsets should be given as dimension expressions.

```
\coffin_typeset:Nnnnn
```

```
\coffin_typeset:cnnnn
```

```
\coffin_typeset:Nnnnn <coffin> {\<pole1>} {\<pole2>}
{\<x-offset>} {\<y-offset>}
```

Typesetting is carried out by first calculating $\langle handle \rangle$, the point of intersection of $\langle pole1 \rangle$ and $\langle pole2 \rangle$. The coffin is then typeset such that the relationship between the current reference point in the document and the $\langle handle \rangle$ is described by the $\langle x-offset \rangle$ and $\langle y-offset \rangle$. The two offsets should be given as dimension expressions. Typesetting a coffin is therefore analogous to carrying out an alignment where the “parent” coffin is the current insertion point.

144 Measuring coffins

<hr/> <code>\coffin_dp:N</code> <hr/>	<code>\coffin_dp:N <coffin></code>
<code>\coffin_dp:c</code> <hr/>	Calculates the depth (below the baseline) of the <code><coffin></code> in a form suitable for use in a <code><dimension expression></code> .
<hr/> <code>\coffin_ht:N</code> <hr/>	<code>\coffin_ht:N <coffin></code>
<code>\coffin_ht:c</code> <hr/>	Calculates the height (above the baseline) of the <code><coffin></code> in a form suitable for use in a <code><dimension expression></code> .
<hr/> <code>\coffin_wd:N</code> <hr/>	<code>\coffin_wd:N <coffin></code>
<code>\coffin_wd:c</code> <hr/>	Calculates the width of the <code><coffin></code> in a form suitable for use in a <code><dimension expression></code> .

145 Coffin diagnostics

<hr/> <code>\coffin_display_handles:Nn</code> <hr/>	<code>\coffin_display_handles:Nn <coffin> {<colour>}</code>
<code>\coffin_display_handles:cn</code> <hr/>	This function first calculates the intersections between all of the <code><poles></code> of the <code><coffin></code> to give a set of <code><handles></code> . It then prints the <code><coffin></code> at the current location in the source, with the position of the <code><handles></code> marked on the coffin. The <code><handles></code> will be labelled as part of this process: the locations of the <code><handles></code> and the labels are both printed in the <code><colour></code> specified.
Updated: 2011-09-02 <hr/>	
<hr/> <code>\coffin_mark_handle:Nnnn</code> <hr/>	<code>\coffin_mark_handle:Nnnn <coffin> {<pole₁>} {<pole₂>} {<colour>}</code>
<code>\coffin_mark_handle:cnnn</code> <hr/>	This function first calculates the <code><handle></code> for the <code><coffin></code> as defined by the intersection of <code><pole₁></code> and <code><pole₂></code> . It then marks the position of the <code><handle></code> on the <code><coffin></code> . The <code><handle></code> will be labelled as part of this process: the location of the <code><handle></code> and the label are both printed in the <code><colour></code> specified.
Updated: 2011-09-02 <hr/>	
<hr/> <code>\coffin_show_structure:N</code> <hr/>	<code>\coffin_show_structure:N <coffin></code>
<code>\coffin_show_structure:c</code> <hr/>	This function shows the structural information about the <code><coffin></code> in the terminal. The width, height and depth of the typeset material are given, along with the location of all of the poles of the coffin.
	Notice that the poles of a coffin are defined by four values: the <i>x</i> and <i>y</i> co-ordinates of a point that the pole passes through and the <i>x</i> - and <i>y</i> -components of a vector denoting the direction of the pole. It is the ratio between the later, rather than the absolute values, which determines the direction of the pole.

Part XVII

The l3color package

Colour support

This module provides support for colour in L^AT_EX3. At present, the material here is mainly intended to support a small number of low-level requirements in other l3kernel modules.

146 Colour in boxes

Controlling the colour of text in boxes requires a small number of control functions, so that the boxed material uses the colour at the point where it is set, rather than where it is used.

<hr/> <code>\color_group_begin</code> <code>\color_group_end</code> <hr/> <div>New: 2011-09-03</div> <hr/>	<code>\color_group_begin:</code> ... <code>\color_group_end:</code> Creates a colour group: one used to “trap” colour settings.
<hr/> <code>\color_ensure_current</code> <hr/> <div>New: 2011-09-03</div> <hr/>	<code>\color_ensure_current:</code> Ensures that material inside a box will use the foreground colour at the point where the box is set, rather than that in force when the box is used. This function should usually be used within a <code>\color_group_begin: ... \color_group_end: group</code> .

Part XVIII

The l3io package

Input–output operations

Reading and writing from file streams is handled in L^AT_EX3 using functions with prefixes `\iow_...` (file reading) and `\ior_...` (file writing). Many of the basic functions are very similar, with reading and writing using the same syntax and function concepts. As a result, the reading and writing functions are documented together where this makes sense.

As T_EX is limited to 16 input streams and 16 output streams, direct use of the streams by the programmer is not supported in L^AT_EX3. Instead, an internal pool of streams is maintained, and these are allocated and deallocated as needed by other modules. As a result, the programmer should close streams when they are no longer needed, to release them for other processes.

Reading from or writing to a file requires a $\langle stream \rangle$ to be used. This is a csname which refers to the file being processed, and is independent of the name of the file (except of course that the file name is needed when the file is opened).

147 Managing streams

```
\ior_new:N
\ior_new:c
\iow_new:N
\io_new:c
```

New: 2011-09-26

Updated: 2011-12-27

```
\ior_open:Nn
\ior_open:cn
```

Updated: 2011-12-27

```
\iow_open:Nn
\iow_open:cn
```

Updated: 2011-12-27

```
\ior_new:Nn  $\langle stream \rangle$ 
```

Globally reserves the name of the $\langle stream \rangle$, either for reading or for writing as appropriate. The $\langle stream \rangle$ is not opened until the appropriate `\..._open:Nn` function is used. Attempting to use a $\langle stream \rangle$ which has not been opened will result in a T_EX error.

```
\ior_open:Nn  $\langle stream \rangle$  { $\langle file name \rangle$ }
```

Opens $\langle file name \rangle$ for reading using $\langle stream \rangle$ as the control sequence for file access. If the $\langle stream \rangle$ was already open it is closed before the new operation begins. The $\langle stream \rangle$ is available for access immediately and will remain allocated to $\langle file name \rangle$ until a `\ior_close:N` instruction is given or the file ends.

```
\iow_open:Nn  $\langle stream \rangle$  { $\langle file name \rangle$ }
```

Opens $\langle file name \rangle$ for writing using $\langle stream \rangle$ as the control sequence for file access. If the $\langle stream \rangle$ was already open it is closed before the new operation begins. The $\langle stream \rangle$ is available for access immediately and will remain allocated to $\langle file name \rangle$ until a `\iow_close:N` instruction is given or the file ends. Opening a file for writing will clear any existing content in the file (*i.e.* writing is *not* additive).

<code>\ior_close:N</code>	<code>\ior_close:N <stream></code>
<code>\ior_close:c</code>	Closes the <code><stream></code> . Streams should always be closed when they are finished with as this ensures that they remain available to other programmer.
Updated: 2011-12-27	

<code>\iow_close:N</code>	<code>\iow_close:N <stream></code>
<code>\iow_close:c</code>	Closes the <code><stream></code> . Streams should always be closed when they are finished with as this ensures that they remain available to other programmer.
Updated: 2011-12-27	

<code>\ior_list_streams</code>	<code>\ior_list_streams:</code>
<code>\iow_list_streams</code>	<code>\iow_list_streams:</code>
Displays a list of the file names associated with each open stream: intended for tracking down problems.	

148 Writing to files

<code>\iow_now:Nn</code>	<code>\iow_now:Nn <stream> {<tokens>}</code>
<code>\iow_now:Nx</code>	This functions writes <code><tokens></code> to the specified <code><stream></code> immediately (<i>i.e.</i> the write operation is called on expansion of <code>\iow_now:Nn</code>).

T_EXhackers note: `\iow_now:Nx` is a protected macro which expands to the two T_EX primitives `\immediate\write`.

<code>\iow_log:n</code>	<code>\iow_log:n {<tokens>}</code>
<code>\iow_log:x</code>	This function writes the given <code><tokens></code> to the log (transcript) file immediately: it is a dedicated version of <code>\iow_now:Nn</code> .

<code>\iow_term:n</code>	<code>\iow_term:n {<tokens>}</code>
<code>\iow_term:x</code>	This function writes the given <code><tokens></code> to the terminal file immediately: it is a dedicated version of <code>\iow_now:Nn</code> .

<code>\iow_now_when_avail:Nn</code>	<code>\iow_now_when_avail:Nn <stream> {<tokens>}</code>
<code>\iow_now_when_avail:Nx</code>	If <code><stream></code> is open, writes the <code><tokens></code> to the <code><stream></code> in the same manner as <code>\iow_now:Nn</code> . If the <code><stream></code> is not open, the <code><tokens></code> are simply thrown away.

<code>\iow_shipout:Nn</code>	<code>\iow_shipout:Nn <stream> {<tokens>}</code>
<code>\iow_shipout:Nx</code>	This functions writes <code><tokens></code> to the specified <code><stream></code> when the current page is finalised (<i>i.e.</i> at shipout). The x-type variants expand the <code><tokens></code> at the point where the function is used but <i>not</i> when the resulting tokens are written to the <code><stream></code> (<i>cf.</i> <code>\iow_shipout_x:Nn</code>).

<code>\iow_shipout_x:Nn</code> <code>\iow_shipout_x:Nx</code>	<code>\iow_shipout_x:Nn <stream> {<tokens>}</code> <p>This function writes <i><tokens></i> to the specified <i><stream></i> when the current page is finalised (<i>i.e.</i> at shipout). The <i><tokens></i> are expanded at the time of writing in addition to any expansion when the function is used. This makes these functions suitable for including material finalised during the page building process (such as the page number integer).</p>
--	--

TeXhackers note: `\iow_shipout_x:Nn` is the TeX primitive `\write` renamed.

<code>\iow_char:N</code> ★	<code>\iow_char:N <token></code> <p>Inserts <i><token></i> into the output stream. Useful when trying to write difficult characters such as %, {, }, <i>etc.</i> in messages, for example:</p>
----------------------------	---

`\iow_now:Nx \g_my_iow { \iow_char:N \{ text \iow_char:N \} }`

The function has no effect if writing is taking place without expansion (*e.g.* in the second argument of `\iow_now:Nn`).

<code>\iow_newline</code> ★	<code>\iow_newline:</code> <p>Function to add a new line within the <i><tokens></i> written to a file. The function has no effect if writing is taking place without expansion (<i>e.g.</i> in the second argument of <code>\iow_now:Nn</code>).</p>
-----------------------------	---

149 Wrapping lines in output

<hr/> <code>\iow_wrap:xnnnN</code> <hr/>	<code>\iow_wrap:xnnnN</code> $\{\langle text \rangle\}$ $\{\langle run-on text \rangle\}$ $\{\langle run-on length \rangle\}$ $\{\langle set up \rangle\}$ $\langle function \rangle$
Updated: 2011-09-21	<p>This function will wrap the $\langle text \rangle$ to a fixed number of characters per line. At the start of each line which is wrapped, the $\langle run-on text \rangle$ will be inserted. The line length targeted will be the value of <code>\l_iow_line_length_int</code> minus the $\langle run-on length \rangle$. The later value should be the number of characters in the $\langle run-on text \rangle$. Additional functions may be added to the wrapping by using the $\langle set up \rangle$, which is executed before the wrapping takes place. The result of the wrapping operation is passed as a braced argument to the $\langle function \rangle$, which will typically be a wrapper around a writing operation. Within the $\langle text \rangle$,</p> <ul style="list-style-type: none"> • <code>\\</code> may be used to force a new line, • <code>\ </code> may be used to represent a forced space (for example after a control sequence), • <code>\#</code>, <code>\%</code>, <code>\{</code>, <code>\}</code>, <code>\~</code> may be used to represent the corresponding character, • <code>\iow_indent:n</code> may be used to indent a part of the message. <p>Both the wrapping process and the subsequent write operation will perform x-type expansion. For this reason, material which is to be written “as is” should be given as the argument to <code>\token_to_str:N</code> or <code>\tl_to_str:n</code> (as appropriate) within the $\langle text \rangle$. The output of <code>\iow_wrap:xnnnN</code> (<i>i.e.</i> the argument passed to the $\langle function \rangle$) will consist of characters of category code 12 (other) and 10 (space) only. This means that the output will <i>not</i> expand further when written to a file.</p>
<hr/> <code>\iow_indent:n</code> <hr/>	<code>\iow_indent:n</code> $\{\langle text \rangle\}$
New: 2011-09-21	<p>In the context of <code>\iow_wrap:xnnnN</code> (for instance in messages), indents $\langle text \rangle$ by four spaces. This function will not cause a line break, and only affects lines which start within the scope of the $\langle text \rangle$. In case the indented $\langle text \rangle$ should appear on separate lines from the surrounding text, use <code>\\</code> to force line breaks.</p>
<hr/> <code>\l_iow_line_length_int</code> <hr/>	<p>The maximum length of a line to be written by the <code>\iow_wrap:xnnnN</code> function. This value depends on the T_EX system in use: the standard value is 78, which is typically correct for unmodified T_EXlive and MiK_TTeX systems.</p>
<hr/> <code>\c_catcode_other_space_tl</code> <hr/>	Token list containing one character with category code 12, (“other”), and character code 32 (space).
New: 2011-09-05	

150 Reading from files

`\ior_to:NN` `\ior_to:NN` $\langle stream \rangle$ $\langle token list variable \rangle$

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input $\langle stream \rangle$ and stores the result locally in the $\langle token list \rangle$ variable. If the $\langle stream \rangle$ is not open, input is requested from the terminal. The material read from the $\langle stream \rangle$ will be tokenized by T_EX according to the category codes in force when the function is used.

T_EXhackers note: The is protected macro which expands to the T_EX primitive `\read` along with the `to` keyword.

`\ior_gto:NN` `\ior_gto:NN` $\langle stream \rangle$ $\langle token list variable \rangle$

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input $\langle stream \rangle$ and stores the result globally in the $\langle token list \rangle$ variable. If the $\langle stream \rangle$ is not open, input is requested from the terminal. The material read from the $\langle stream \rangle$ will be tokenized by T_EX according to the category codes in force when the function is used.

T_EXhackers note: The is protected macro which expands to the T_EX primitives `\global\read` along with the `to` keyword.

`\ior_str_to:NN` `\ior_str_to:NN` $\langle stream \rangle$ $\langle token list variable \rangle$

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input $\langle stream \rangle$ and stores the result locally in the $\langle token list \rangle$ variable. If the $\langle stream \rangle$ is not open, input is requested from the terminal. The material read from the $\langle stream \rangle$ as a series of tokens with category code 12 (other), with the exception of space characters which are given category code 10 (space).

T_EXhackers note: The is protected macro which expands to the ε -T_EX primitive `\readline` along with the `to` keyword.

`\ior_str_gto:NN` `\ior_str_gto:NN` $\langle stream \rangle$ $\langle token list variable \rangle$

Functions that reads one or more lines (until an equal number of left and right braces are found) from the input $\langle stream \rangle$ and stores the result globally in the $\langle token list \rangle$ variable. If the $\langle stream \rangle$ is not open, input is requested from the terminal. The material read from the $\langle stream \rangle$ as a series of tokens with category code 12 (other), with the exception of space characters which are given category code 10 (space).

T_EXhackers note: The is protected macro which expands to the primitives `\global\readline` along with the `to` keyword.

<code>\ior_if_eof_p:N</code> ★	<code>\ior_if_eof_p:N</code> $\langle stream \rangle$
<code>\ior_if_eof:NTF</code> ★	<code>\ior_if_eof:NTF</code> $\langle stream \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
Updated: 2011-09-26	Tests if the end of a $\langle stream \rangle$ has been reached during a reading operation. The test will also return a <code>true</code> value if the $\langle stream \rangle$ is not open or the $\langle file\ name \rangle$ associated with a $\langle stream \rangle$ does not exist at all.

151 Constants

<code>\c_ior_term_ior</code>	Constant input stream for reading from the terminal. Reading from this stream using <code>\ior_to:NN</code> or similar will result in a prompt from T _E X of the form
------------------------------	--

`<tl>=`

<code>\c_iow_log_iow</code>	Constant output streams for writing to the log and to the terminal (plus the log), respectively.
<code>\c_iow_term_iow</code>	

152 Internal input–output functions

<code>\if_eof:w</code> ★	<code>\if_eof:w</code> $\langle stream \rangle$ $\langle true\ code \rangle$ <code>\else:</code> $\langle false\ code \rangle$ <code>\fi:</code> Tests if the $\langle stream \rangle$ returns “end of file”, which is true for non-existent files. The <code>\else:</code> branch is optional.
--------------------------	--

T_EXhackers note: This is the T_EX primitive `\ifeof`.

<code>\ior_raw_new:N</code>	<code>\ior_raw_new:N</code> $\langle stream \rangle$
<code>\ior_raw_new:c</code>	Directly allocates a new stream for reading, bypassing the stack system. This is to be used only when a new stream is required at a T _E X level, when a new stream is requested by the stack itself.

<code>\iow_raw_new:N</code>	<code>\iow_raw_new:N</code> $\langle stream \rangle$
<code>\iow_raw_new:c</code>	Directly allocates a new stream for writing, bypassing the stack system. This is to be used only when a new stream is required at a T _E X level, when a new stream is requested by the stack itself.

Part XIX

The l3msg package

Messages

Messages need to be passed to the user by modules, either when errors occur or to indicate how the code is proceeding. The `l3msg` module provides a consistent method for doing this (as opposed to writing directly to the terminal or log).

The system used by `l3msg` to create messages divides the process into two distinct parts. Named messages are created in the first part of the process; at this stage, no decision is made about the type of output that the message will produce. The second part of the process is actually producing a message. At this stage a choice of message *class* has to be made, for example `error`, `warning` or `info`.

By separating out the creation and use of messages, several benefits are available. First, the messages can be altered later without needing details of where they are used in the code. This makes it possible to alter the language used, the detail level and so on. Secondly, the output which results from a given message can be altered. This can be done on a message class, module or message name basis. In this way, message behaviour can be altered and messages can be entirely suppressed.

153 Creating new messages

All messages have to be created before they can be used. All message setting is local, with the general assumption that messages will be managed as part of module set up outside of any \TeX grouping.

The text of messages will automatically be wrapped to the length available in the console. As a result, formatting is only needed where it will help to show meaning. In particular, `\` may be used to force a new line and `_` forces an explicit space.

```
\msg_new:nnnn
```

```
\msg_new:nnn
```

Updated: 2011-08-16

```
\msg_new:nnnn {<module>} {<message>} {<text>} {<more text>}
```

Creates a *<message>* for a given *<module>*. The message will be defined to first give *<text>* and then *<more text>* if the user requests it. If no *<more text>* is available then a standard text is given instead. Within *<text>* and *<more text>* four parameters (**#1** to **#4**) can be used: these will be supplied at the time the message is used. The parameters will be expanded when the message is used. Within the *<text>* and *<more text>* `\` can be used to start a new line. An error will be raised if the *<message>* already exists.

```
\msg_set:nnnn
```

```
\msg_set:nnn
```

```
\msg_gset:nnnn
```

```
\msg_gset:nnn
```

```
\msg_set:nnnn {<module>} {<message>} {<text>} {<more text>}
```

Sets up the text for a *<message>* for a given *<module>*. The message will be defined to first give *<text>* and then *<more text>* if the user requests it. If no *<more text>* is available then a standard text is given instead. Within *<text>* and *<more text>* four parameters (**#1** to **#4**) can be used: these will be supplied at the time the message is used. The parameters will be expanded when the message is used. Within the *<text>* and *<more text>* `\` can be used to start a new line.

154 Contextual information for messages

<hr/> <code>\msg_line_context</code> ☆ <hr/>	<code>\msg_line_context:</code> Prints the current line number when a message is given, and thus suitable for giving context to messages. The number itself is proceeded by the text <code>on line</code> .
<hr/> <code>\msg_line_number</code> ☆ <hr/>	<code>\msg_line_number:</code> Prints the current line number when a message is given.
<hr/> <code>\c_msg_return_text_tl</code> <hr/>	Standard text to indicate that the user should try pressing <code><return></code> to continue. The standard definition reads: Try typing <code><return></code> to proceed. If that doesn't work, type <code>X <return></code> to quit.
<hr/> <code>\c_msg_trouble_text_tl</code> <hr/>	Standard text to indicate that the more errors are likely and that aborting the run is advised. The standard definition reads: More errors will almost certainly follow: the LaTeX run should be aborted.
<hr/> <code>\msg_fatal_text:n</code> ☆ <hr/>	<code>\msg_fatal_text:n {<module>}</code> Produces the standard text: Fatal <code><module></code> error This function can be redefined to alter the language in which the message is give, using <code>#1</code> as the name of the <code><module></code> to be included.
<hr/> <code>\msg_critical_text:n</code> ☆ <hr/>	<code>\msg_critical_text:n {<module>}</code> Produces the standard text: Critical <code><module></code> error This function can be redefined to alter the language in which the message is give, using <code>#1</code> as the name of the <code><module></code> to be included.
<hr/> <code>\msg_error_text:n</code> ☆ <hr/>	<code>\msg_error_text:n {<module>}</code> Produces the standard text: <code><module></code> error This function can be redefined to alter the language in which the message is give, using <code>#1</code> as the name of the <code><module></code> to be included.

<code>\msg_warning_text:n</code>	★	<code>\msg_warning_text:n {<module>}</code>
----------------------------------	---	---

Produces the standard text:

`<module> warning`

This function can be redefined to alter the language in which the message is give, using #1 as the name of the `<module>` to be included.

<code>\msg_info_text:n</code>	★	<code>\msg_info_text:n {<module>}</code>
-------------------------------	---	--

Produces the standard text:

`<module> info`

This function can be redefined to alter the language in which the message is give, using #1 as the name of the `<module>` to be included.

155 Issuing messages

Messages behave differently depending on the message class. A number of standard message classes are supplied, but more can be created.

When issuing messages, any arguments passed should use `\tl_to_str:n` or `\token_to_str:N` to prevent unwanted expansion of the material.

<code>\msg_class_set:nn</code>	<code>\msg_class_set:nn {<class>} {<code>}</code>
--------------------------------	---

Sets a `<class>` to output a message, using `<code>` to process the message text. The `<class>` should be a text value, while the `<code>` may be any arbitrary material. The `<code>` will receive 6 arguments: the module name (#1), the message name (#2) and the four arguments taken by the message text (#3 to #6).

The kernel defines several common message classes. The following describes the standard behaviour of each class if no redirection of the class or message is active. In all cases, the message may be issued supplying 0 to 4 arguments. The code will ensure that there an no errors if the number of arguments supplied here does not match the number in the definition of the message (although of course the sense of the message may be impaired).

<code>\msg_fatal:nnxxxx</code>	<code>\msg_fatal:nnxxxx {<module>} {<message>} {<arg one>}</code>
<code>\msg_fatal:(nnxxx nnxx nnx nn)</code>	<code>{<arg two>} {<arg three>} {<arg four>}</code>

Issues `<module>` error `<message>`, passing `<arg one>` to `<arg four>` to the text-creating functions. After issuing a fatal error the T_EX run will halt.

<code>\msg_critical:nnxxxx</code>	<code>\msg_critical:nnxxxx {<module>} {<message>} {<arg one>}</code>
<code>\msg_critical:(nnxxx nnxx nnx nn)</code>	<code>{<arg two>} {<arg three>} {<arg four>}</code>

Issues `<module>` error `<message>`, passing `<arg one>` to `<arg four>` to the text-creating functions. After issuing the message reading the current input file will stop. This may halt the T_EX run (if the current file is the main file) or may abort reading a sub-file.

<code>\msg_error:nnxxxx</code>	<code>\msg_error:nnxxxx {<module>} {<message>} {<arg one>}</code>
<code>\msg_error:(nnxxx nnxx nnx nn)</code>	<code>{<arg two>} {<arg three>} {<arg four>}</code>

Issues *<module>* error *<message>*, passing *<arg one>* to *<arg four>* to the text-creating functions. The error will stop processing and issue the text at the terminal. After user input, the run will continue.

<code>\msg_warning:nnxxxx</code>	<code>\msg_warning:nnxxxx {<module>} {<message>} {<arg one>}</code>
<code>\msg_warning:(nnxxx nnxx nnx nn)</code>	<code>{<arg two>} {<arg three>} {<arg four>}</code>

Issues *<module>* warning *<message>*, passing *<arg one>* to *<arg four>* to the text-creating functions. The warning text will be added to the log file, but the T_EX run will not be interrupted.

<code>\msg_info:nnxxxx</code>	<code>\msg_info:nnxxxx {<module>} {<message>} {<arg one>}</code>
<code>\msg_info:(nnxxx nnxx nnx nn)</code>	<code>{<arg two>} {<arg three>} {<arg four>}</code>

Issues *<module>* information *<message>*, passing *<arg one>* to *<arg four>* to the text-creating functions. The information text will be added to the log file.

<code>\msg_log:nnxxxx</code>	<code>\msg_log:nnxxxx {<module>} {<message>} {<arg one>}</code>
<code>\msg_log:(nnxxx nnxx nnx nn)</code>	<code>{<arg two>} {<arg three>} {<arg four>}</code>

Issues *<module>* information *<message>*, passing *<arg one>* to *<arg four>* to the text-creating functions. The information text will be added to the log file: the output is briefer than `\msg_info:nnxxxx`.

<code>\msg_none:nnxxxx</code>	<code>\msg_none:nnxxxx {<module>} {<message>} {<arg one>}</code>
<code>\msg_none:(nnxxx nnxx nnx nn)</code>	<code>{<arg two>} {<arg three>} {<arg four>}</code>

Does nothing: used as a message class to prevent any output at all (see the discussion of message redirection).

156 Redirecting messages

Each message has a “name”, which can be used to alter the behaviour of the message when it is given. Thus we might have

```
\msg_new:nnnn { module } { my-message } { Some~text } { Some-more~text }
```

to define a message, with

```
\msg_error:nn { module } { my-message }
```

when it is used. With no filtering, this will raise an error. However, we could alter the behaviour with

```
\msg_redirect_class:nn { error } { warning }
```

to turn all errors into warnings, or with

```
\msg_redirect_module:nnn { module } { error } { warning }
```

to alter just those messages for module, or even

```
\msg_redirect_name:nnn { module } { my-message } { warning }
```

to target just one message.

```
\msg_redirect_class:nn
```

```
\msg_redirect_class:nn {<class one>} {<class two>}
```

Changes the behaviour of messages of *<class one>* so that they are processed using the code for those of *<class two>*. Multiple redirections are possible. Redirection to a missing class or infinite loops will raise errors when the messages are used, rather than at the point of redirection.

```
\msg_redirect_module:nnn
```

```
\msg_redirect_module:nnn {<module>} {<class one>} {<class two>}
```

Redirects message of *<class one>* for *<module>* to act as though they were from *<class two>*. Messages of *<class one>* from sources other than *<module>* are not affected by this redirection. This function can be used to make some messages “silent” by default. For example, all of the **trace** messages of *<module>* could be turned off with:

```
\msg_redirect_module:nnn { module } { trace } { none }
```

```
\msg_redirect_name:nnn
```

```
\msg_redirect_name:nn {<module>} {<message>} {<class>}
```

Redirects a specific *<message>* from a specific *<module>* to act as a member of *<class>* of messages. This function can be used to make a selected message “silent” without changing global parameters:

```
\msg_redirect_name:nnn { module } { annoying-message } { none }
```

157 Low-level message functions

The lower-level message functions should usually be accessed from the higher-level system. However, there are occasions where direct access to these functions is desirable.

```
\msg_newline
```

★

```
\msg_newline:
```

```
\msg_two_newlines
```

★

Forces a new line in a message. This is a low-level function, which will not include any additional printing information in the message: contrast with `\\` in messages. The **two** version adds two lines.

\msg_interrupt:xxx \msg_interrupt:xxx {<first line>} {<text>} {<extra text>}

Interrupts the T_EX run, issuing a formatted message comprising <first line> and <text> laid out in the format

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! <first line>
!
! <text>
!.....

```

where the <text> will be wrapped to fit within the current line length. The user may then request more information, at which stage the <extra text> will be shown in the terminal in the format

```

|,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
|  <extra text>
|.....

```

where the <extra text> will be wrapped to fit within the current line length.

\msg_log:x \msg_log:x {<text>}

Writes to the log file with the <text> laid out in the format

```

.....
. <text>
.....

```

where the <text> will be wrapped to fit within the current line length.

\msg_term:x \msg_term:x {<text>}

Writes to the terminal and log file with the <text> laid out in the format

```

*****
* <text>
*****

```

where the <text> will be wrapped to fit within the current line length.

158 Kernel-specific functions

Messages from L^AT_EX3 itself are handled by the general message system, but have their own functions. This allows some text to be pre-defined, and also ensures that serious errors can be handled properly.

```
\msg_kernel_new:nnnn
\msg_kernel_new:nnn
```

Updated: 2011-08-16

```
\msg_kernel_new:nnnn {\module} {\message} {\text} {\more text}
```

Creates a kernel $\langle message \rangle$ for a given $\langle module \rangle$. The message will be defined to first give $\langle text \rangle$ and then $\langle more text \rangle$ if the user requests it. If no $\langle more text \rangle$ is available then a standard text is given instead. Within $\langle text \rangle$ and $\langle more text \rangle$ four parameters (#1 to #4) can be used: these will be supplied at the time the message is used. The parameters will be expanded when the message is used. Within the $\langle text \rangle$ and $\langle more text \rangle$ $\backslash\backslash$ can be used to start a new line. An error will be raised if the $\langle message \rangle$ already exists.

```
\msg_kernel_set:nnnn
\msg_kernel_set:nnn
```

```
\msg_kernel_set:nnnn {\module} {\message} {\text} {\more text}
```

Sets up the text for a kernel $\langle message \rangle$ for a given $\langle module \rangle$. The message will be defined to first give $\langle text \rangle$ and then $\langle more text \rangle$ if the user requests it. If no $\langle more text \rangle$ is available then a standard text is given instead. Within $\langle text \rangle$ and $\langle more text \rangle$ four parameters (#1 to #4) can be used: these will be supplied at the time the message is used. The parameters will be expanded when the message is used. Within the $\langle text \rangle$ and $\langle more text \rangle$ $\backslash\backslash$ can be used to start a new line.

```
\msg_kernel_fatal:nnxxxx
\msg_kernel_fatal:(nnxxx|nnxx|nnx|nn)
```

```
\msg_kernel_fatal:nnxxxx {\module} {\message} {\arg one}
{\arg two} {\arg three} {\arg four}
```

Issues kernel $\langle module \rangle$ error $\langle message \rangle$, passing $\langle arg one \rangle$ to $\langle arg four \rangle$ to the text-creating functions. After issuing a fatal error the T_EX run will halt. Cannot be redirected.

```
\msg_kernel_error:nnxxxx
\msg_kernel_error:(nnxxx|nnxx|nnx|nn)
```

```
\msg_kernel_error:nnxxxx {\module} {\message} {\arg one}
{\arg two} {\arg three} {\arg four}
```

Issues kernel $\langle module \rangle$ error $\langle message \rangle$, passing $\langle arg one \rangle$ to $\langle arg four \rangle$ to the text-creating functions. The error will stop processing and issue the text at the terminal. After user input, the run will continue. Cannot be redirected.

```
\msg_kernel_warning:nnxxxx
\msg_kernel_warning:(nnxxx|nnxx|nnx|nn)
```

```
\msg_kernel_warning:nnxxxx {\module} {\message} {\arg one}
{\arg two} {\arg three} {\arg four}
```

Issues kernel $\langle module \rangle$ warning $\langle message \rangle$, passing $\langle arg one \rangle$ to $\langle arg four \rangle$ to the text-creating functions. The warning text will be added to the log file, but the T_EX run will not be interrupted.

```
\msg_kernel_info:nnxxxx
\msg_kernel_info:(nnxxx|nnxx|nnx|nn)
```

```
\msg_kernel_info:nnxxxx {\module} {\message} {\arg one}
{\arg two} {\arg three} {\arg four}
```

Issues kernel $\langle module \rangle$ information $\langle message \rangle$, passing $\langle arg one \rangle$ to $\langle arg four \rangle$ to the text-creating functions. The information text will be added to the log file.

159 Expandable errors

In a few places, the L^AT_EX3 kernel needs to produce errors in an expansion only context. This must be handled internally very differently from normal error messages, as none of

the tools to print to the terminal or the log file are expandable. However, the interface is similar.

<code>\msg_expandable_kernel_error:nnnnnn</code>	★	<code>\msg_expandable_kernel_error:nnnnnn {<module>}</code>
<code>\msg_expandable_kernel_error:(nnnnnn nnnn nnn nn)</code>	★	<code>{<message>}</code> <code>{<arg one>} {<arg two>} {<arg three>} {<arg four>}</code>
New: 2011-11-23		

Issues an error, passing *<arg one>* to *<arg four>* to the text-creating functions. The resulting string must be shorter than a line, otherwise it will be cropped.

<code>\msg_expandable_error:n</code>	★	<code>\msg_expandable_error:n {<error message>}</code>
New: 2011-08-11		Issues an “Undefined error” message from T _E X itself, and prints the <i><error message></i> .
Updated: 2011-08-13		The <i><error message></i> must be short: it is cropped at the end of one line.

T_EXhackers note: This function expands to an empty token list after two steps. Tokens inserted in response to T_EX’s prompt are read with the current category code setting, and inserted just after the place where the error message was issued.

160 Internal l3msg functions

The following functions are used in several kernel modules.

<code>\msg_aux_use:nn</code>	<code>\msg_aux_use:nnxx {<module>} {<message>}</code>
<code>\msg_aux_use:nnxxx</code>	<code>{<arg one>} {<arg two>} {<arg three>} {<arg four>}</code>

Prints the *<message>* from *<module>* in the terminal, without formatting.

<code>\msg_aux_show:x</code>	<code>\msg_aux_show:x {<formatted string>}</code>
------------------------------	---

Shows the *<formatted string>* on the terminal. The *<formatted string>* must start with `^^J>`, failure to do so causes low-level T_EX errors.

<code>\msg_aux_show:Nnx</code>	<code>\msg_aux_show:Nnx <variable> {<module>} {<token list>}</code>
--------------------------------	---

Auxiliary common to l3clist, l3prop and seq, which displays an appropriate message and the contents of the variable.

Part XX

The l3keys package

Key–value interfaces

The key–value method is a popular system for creating large numbers of settings for controlling function or package behaviour. For the user, the system normally results in input of the form

```
\PackageControlMacro{
  key-one = value one,
  key-two = value two
}
```

or

```
\PackageMacro[
  key-one = value one,
  key-two = value two
]{argument}.
```

The high level functions here are intended as a method to create key–value controls. Keys are themselves created using a key–value interface, minimising the number of functions and arguments required. Each key is created by setting one or more *properties* of the key:

```
\keys_define:nn { module }
{
  key-one .code:n    = code including parameter #1,
  key-two .tl_set:N = \l_module_store_tl
}
```

These values can then be set as with other key–value approaches:

```
\keys_set:nn { module }
{
  key-one = value one,
  key-two = value two
}
```

At a document level, `\keys_set:nn` will be used within a document function, for example

```
\DeclareDocumentCommand \SomePackageSetup { m }
{ \keys_set:nn { module } { #1 } }
\DeclareDocumentCommand \SomePackageMacro { o m }
{
  \group_begin:
```

```

\keys_set:nn { module } { #1 }
% Main code for \SomePackageMacro
\group_end:
}

```

Key names may contain any tokens, as they are handled internally using `\tl_to_str:n`. As will be discussed in section 162, it is suggested that the character `/` is reserved for sub-division of keys into logical groups. Functions and variables are *not* expanded when creating key names, and so

```

\tl_set:Nn \l_module_tmp_tl { key }
\keys_define:nn { module }
{
  \l_module_tmp_tl .code:n = code
}

```

will create a key called `\l_module_tmp_tl`, and not one called `key`.

161 Creating keys

```
\keys_define:nn {<module>} {<keyval list>}
```

Parses the *<keyval list>* and defines the keys listed there for *<module>*. The *<module>* name should be a text value, but there are no restrictions on the nature of the text. In practice the *<module>* should be chosen to be unique to the module in question (unless deliberately adding keys to an existing module).

The *<keyval list>* should consist of one or more key names along with an associated key *property*. The properties of a key determine how it acts. The individual properties are described in the following text; a typical use of `\keys_define:nn` might read

```

\keys_define:nn { mymodule }
{
  keyname .code:n = Some~code~using~#1,
  keyname .value_required:
}

```

where the properties of the key begin from the `.` after the key name.

The various properties available take either no arguments at all, or require exactly one argument. This is indicated in the name of the property using an argument specification. In the following discussion, each property is illustrated attached to an arbitrary *<key>*, which when used may be supplied with a *<value>*. All key *definitions* are local.

```
<key> .bool_set:N = <boolean>
```

Defines *<key>* to set *<boolean>* to *<value>* (which must be either `true` or `false`). If the variable does not exist, it will be created at the point that the key is set up. The *<boolean>* will be assigned locally.

<hr/> .bool_gset:N <hr/>	<p>$\langle key \rangle$.bool_gset:N = $\langle boolean \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle boolean \rangle$ to $\langle value \rangle$ (which must be either true or false). If the variable does not exist, it will be created at the point that the key is set up. The $\langle boolean \rangle$ will be assigned globally.</p>
<hr/> .bool_set_inverse:N <hr/> <hr/> New: 2011-08-28 <hr/>	<p>$\langle key \rangle$.bool_set_inverse:N = $\langle boolean \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle boolean \rangle$ to the logical inverse of $\langle value \rangle$ (which must be either true or false). If the $\langle boolean \rangle$ does not exist, it will be created at the point that the key is set up. The $\langle boolean \rangle$ will be assigned locally.</p> <p>This property is experimental.</p>
<hr/> .bool_gset_inverse:N <hr/>	<p>$\langle key \rangle$.bool_gset_inverse:N = $\langle boolean \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle boolean \rangle$ to the logical inverse of $\langle value \rangle$ (which must be either true or false). If the $\langle boolean \rangle$ does not exist, it will be created at the point that the key is set up. The $\langle boolean \rangle$ will be assigned globally.</p> <p>This property is experimental.</p>
<hr/> .choice: <hr/>	<p>$\langle key \rangle$.choice:</p> <p>Sets $\langle key \rangle$ to act as a choice key. Each valid choice for $\langle key \rangle$ must then be created, as discussed in section 163.</p>
<hr/> .choices:nn <hr/> <hr/> New: 2011-08-21 <hr/>	<p>$\langle key \rangle$.choices:nn $\langle choices \rangle$ $\langle code \rangle$</p> <p>Sets $\langle key \rangle$ to act as a choice key, and defines a series $\langle choices \rangle$ which are implemented using the $\langle code \rangle$. Inside $\langle code \rangle$, $\backslash l_keys_choice_tl$ will be the name of the choice made, and $\backslash l_keys_choice_int$ will be the position of the choice in the list of $\langle choices \rangle$ (indexed from 0). Choices are discussed in detail in section 163.</p> <p>This property is experimental.</p>
<hr/> .choice_code:n <hr/> <hr/> .choice_code:x <hr/>	<p>$\langle key \rangle$.choice_code:n = $\langle code \rangle$</p> <p>Stores $\langle code \rangle$ for use when .generate_choices:n creates one or more choice sub-keys of the current key. Inside $\langle code \rangle$, $\backslash l_keys_choice_tl$ will expand to the name of the choice made, and $\backslash l_keys_choice_int$ will be the position of the choice in the list given to .generate_choices:n. Choices are discussed in detail in section 163.</p>
<hr/> .clist_set:N <hr/> <hr/> .clist_set:c <hr/> <hr/> New: 2011/09/11 <hr/>	<p>$\langle key \rangle$.clist_set:N = $\langle comma\ list\ variable \rangle$</p> <p>Defines $\langle key \rangle$ to locally set $\langle comma\ list\ variable \rangle$ to $\langle value \rangle$. Spaces around commas and empty items will be stripped. If the variable does not exist, it will be created at the point that the key is set up.</p>
<hr/> .clist_gset:N <hr/> <hr/> .clist_gset:c <hr/> <hr/> New: 2011/09/11 <hr/>	<p>$\langle key \rangle$.clist_gset:N = $\langle comma\ list\ variable \rangle$</p> <p>Defines $\langle key \rangle$ to globally set $\langle comma\ list\ variable \rangle$ to $\langle value \rangle$. Spaces around commas and empty items will be stripped. If the variable does not exist, it will be created at the point that the key is set up.</p>

<u>.code:n</u> <u>.code:x</u>	<p>$\langle key \rangle$.code:n = $\langle code \rangle$</p> <p>Stores the $\langle code \rangle$ for execution when $\langle key \rangle$ is used. The $\langle code \rangle$ can include one parameter (#1), which will be the $\langle value \rangle$ given for the $\langle key \rangle$. The x-type variant will expand $\langle code \rangle$ at the point where the $\langle key \rangle$ is created.</p>
<u>.default:n</u> <u>.default:v</u>	<p>$\langle key \rangle$.default:n = $\langle default \rangle$</p> <p>Creates a $\langle default \rangle$ value for $\langle key \rangle$, which is used if no value is given. This will be used if only the key name is given, but not if a blank $\langle value \rangle$ is given:</p> <pre> \keys_define:nn { module } { key .code:n = Hello~#1, key .default:n = World } \keys_set:nn { module } { key = Fred, % Prints 'Hello Fred' key, % Prints 'Hello World' key = , % Prints 'Hello ' } </pre>
<u>.dim_set:N</u> <u>.dim_set:c</u>	<p>$\langle key \rangle$.dim_set:N = $\langle dimension \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle dimension \rangle$ to $\langle value \rangle$ (which must a dimension expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle dimension \rangle$ will be assigned locally.</p>
<u>.dim_gset:N</u> <u>.dim_gset:c</u>	<p>$\langle key \rangle$.dim_gset:N = $\langle dimension \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle dimension \rangle$ to $\langle value \rangle$ (which must a dimension expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle dimension \rangle$ will be assigned globally.</p>
<u>.fp_set:N</u> <u>.fp_set:c</u>	<p>$\langle key \rangle$.fp_set:N = $\langle floating point \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle floating point \rangle$ to $\langle value \rangle$ (which must a floating point number). If the variable does not exist, it will be created at the point that the key is set up. The $\langle integer \rangle$ will be assigned locally.</p>
<u>.fp_gset:N</u> <u>.fp_gset:c</u>	<p>$\langle key \rangle$.fp_gset:N = $\langle floating point \rangle$</p> <p>Defines $\langle key \rangle$ to set $\langle floating-point \rangle$ to $\langle value \rangle$ (which must a floating point number). If the variable does not exist, it will be created at the point that the key is set up. The $\langle integer \rangle$ will be assigned globally.</p>

<hr/> .generate_choices:n <hr/>	$\langle key \rangle$.generate_choices:n = { $\langle list \rangle$ }	This property will mark $\langle key \rangle$ as a multiple choice key, and will use the $\langle list \rangle$ to define the choices. The $\langle list \rangle$ should consist of a comma-separated list of choice names. Each choice will be set up to execute $\langle code \rangle$ as set using .choice_code:n (or .choice_code:x). Choices are discussed in detail in section 163.
<hr/> .int_set:N <hr/> .int_set:c <hr/>	$\langle key \rangle$.int_set:N = $\langle integer \rangle$	Defines $\langle key \rangle$ to set $\langle integer \rangle$ to $\langle value \rangle$ (which must be an integer expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle integer \rangle$ will be assigned locally.
<hr/> .int_gset:N <hr/> .int_gset:c <hr/>	$\langle key \rangle$.int_gset:N = $\langle integer \rangle$	Defines $\langle key \rangle$ to set $\langle integer \rangle$ to $\langle value \rangle$ (which must be an integer expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle integer \rangle$ will be assigned globally.
<hr/> .meta:n <hr/> .meta:x <hr/>	$\langle key \rangle$.meta:n = { $\langle keyval list \rangle$ }	Makes $\langle key \rangle$ a meta-key, which will set $\langle keyval list \rangle$ in one go. If $\langle key \rangle$ is given with a value at the time the key is used, then the value will be passed through to the subsidiary $\langle keys \rangle$ for processing (as #1).
<hr/> .multichoice: <hr/> <small>New: 2011-08-21</small> <hr/>	$\langle key \rangle$.multichoice:	Sets $\langle key \rangle$ to act as a multiple choice key. Each valid choice for $\langle key \rangle$ must then be created, as discussed in section 163. This property is experimental.
<hr/> .multichoice:nn <hr/> <small>New: 2011-08-21</small> <hr/>	$\langle key \rangle$.multichoice:nn $\langle choices \rangle$ $\langle code \rangle$	Sets $\langle key \rangle$ to act as a multiple choice key, and defines a series $\langle choices \rangle$ which are implemented using the $\langle code \rangle$. Inside $\langle code \rangle$, \l_keys_choice_tl will be the name of the choice made, and \l_keys_choice_int will be the position of the choice in the list of $\langle choices \rangle$ (indexed from 0). Choices are discussed in detail in section 163. This property is experimental.
<hr/> .skip_set:N <hr/> .skip_set:c <hr/>	$\langle key \rangle$.skip_set:N = $\langle skip \rangle$	Defines $\langle key \rangle$ to set $\langle skip \rangle$ to $\langle value \rangle$ (which must be a skip expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle skip \rangle$ will be assigned locally.
<hr/> .skip_gset:N <hr/> .skip_gset:c <hr/>	$\langle key \rangle$.skip_gset:N = $\langle skip \rangle$	Defines $\langle key \rangle$ to set $\langle skip \rangle$ to $\langle value \rangle$ (which must be a skip expression). If the variable does not exist, it will be created at the point that the key is set up. The $\langle skip \rangle$ will be assigned globally.

<hr/> <code>.tl_set:N</code> <hr/>	$\langle key \rangle$ <code>.tl_set:N = $\langle token list variable \rangle$</code>
<code>.tl_set:c</code>	Defines $\langle key \rangle$ to set $\langle token list variable \rangle$ to $\langle value \rangle$. If the variable does not exist, it will be created at the point that the key is set up. The $\langle token list variable \rangle$ will be assigned locally.
<hr/> <code>.tl_gset:N</code> <hr/>	$\langle key \rangle$ <code>.tl_gset:N = $\langle token list variable \rangle$</code>
<code>.tl_gset:c</code>	Defines $\langle key \rangle$ to set $\langle token list variable \rangle$ to $\langle value \rangle$. If the variable does not exist, it will be created at the point that the key is set up. The $\langle token list variable \rangle$ will be assigned globally.
<hr/> <code>.tl_set_x:N</code> <hr/>	$\langle key \rangle$ <code>.tl_set_x:N = $\langle token list variable \rangle$</code>
<code>.tl_set_x:c</code>	Defines $\langle key \rangle$ to set $\langle token list variable \rangle$ to $\langle value \rangle$, which will be subjected to an x-type expansion (<i>i.e.</i> using <code>\tl_set:Nx</code>). If the variable does not exist, it will be created at the point that the key is set up. The $\langle token list variable \rangle$ will be assigned locally.
<hr/> <code>.tl_gset_x:N</code> <hr/>	$\langle key \rangle$ <code>.tl_gset_x:N = $\langle token list variable \rangle$</code>
<code>.tl_gset_x:c</code>	Defines $\langle key \rangle$ to set $\langle token list variable \rangle$ to $\langle value \rangle$, which will be subjected to an x-type expansion (<i>i.e.</i> using <code>\tl_set:Nx</code>). If the variable does not exist, it will be created at the point that the key is set up. The $\langle token list variable \rangle$ will be assigned globally.
<hr/> <code>.value_forbidden:</code> <hr/>	$\langle key \rangle$ <code>.value_forbidden:</code>
	Specifies that $\langle key \rangle$ cannot receive a $\langle value \rangle$ when used. If a $\langle value \rangle$ is given then an error will be issued.
<hr/> <code>.value_required:</code> <hr/>	$\langle key \rangle$ <code>.value_required:</code>
	Specifies that $\langle key \rangle$ must receive a $\langle value \rangle$ when used. If a $\langle value \rangle$ is not given then an error will be issued.

162 Sub-dividing keys

When creating large numbers of keys, it may be desirable to divide them into several sub-groups for a given module. This can be achieved either by adding a sub-division to the module name:

```
\keys_define:nn { module / subgroup }
{ key .code:n = code }
```

or to the key name:

```
\keys_define:nn { module }
{ subgroup / key .code:n = code }
```

As illustrated, the best choice of token for sub-dividing keys in this way is `/`. This is because of the method that is used to represent keys internally. Both of the above code fragments set the same key, which has full name `module/subgroup/key`.

As will be illustrated in the next section, this subdivision is particularly relevant to making multiple choices.

163 Choice and multiple choice keys

The `l3keys` system supports two types of choice key, in which a series of pre-defined input values are linked to varying implementations. Choice keys are usually created so that the various values are mutually-exclusive: only one can apply at any one time. “Multiple” choice keys are also supported: these allow a selection of values to be chosen at the same time.

Mutually-exclusive choices are created by setting the `.choice:` property:

```
\keys_define:nn { module }
{ key .choice: }
```

For keys which are set up as choices, the valid choices are generated by creating sub-keys of the choice key. This can be carried out in two ways.

In many cases, choices execute similar code which is dependant only on the name of the choice or the position of the choice in the list of choices. Here, the keys can share the same code, and can be rapidly created using the `.choice_code:n` and `.generate_choices:n` properties:

```
\keys_define:nn { module }
{
  key .choice_code:n =
  {
    You~gave~choice~'\int_use:N \l_keys_choice_tl',~
    which~is~in~position~
    \int_use:N \l_keys_choice_int \c_space_tl
    in~the~list.
  },
  key .generate_choices:n =
  { choice-a, choice-b, choice-c }
}
```

Following common computing practice, `\l_keys_choice_int` is indexed from 0 (as an offset), so that the value of `\l_keys_choice_int` for the first choice in a list will be zero.

The same approach is also implemented by the *experimental* property `.choices:nn`. This combines the functionality of `.choice_code:n` and `.generate_choices:n` into one property:

```
\keys_define:nn { module }
{
  key .choices:nn =
  { choice-a, choice-b, choice-c }
  {
    You~gave~choice~'\int_use:N \l_keys_choice_tl',~
    which~is~in~position~
    \int_use:N \l_keys_choice_int \c_space_tl
    in~the~list.
  }
}
```

Note that the `.choices:nn` property should *not* be mixed with use of `.generate_choices:n`.

`\l_keys_choice_int`
`\l_keys_choice_tl`

Inside the code block for a choice generated using `.generate_choice:` or `.choices:nn`, the variables `\l_keys_choice_tl` and `\l_keys_choice_int` are available to indicate the name of the current choice, and its position in the comma list. The position is indexed from 0.

On the other hand, it is sometimes useful to create choices which use entirely different code from one another. This can be achieved by setting the `.choice:` property of a key, then manually defining sub-keys.

```
\keys_define:nn { module }
{
  key .choice:,
  key / choice-a .code:n = code-a,
  key / choice-b .code:n = code-b,
  key / choice-c .code:n = code-c,
}
```

It is possible to mix the two methods, but manually-created choices should *not* use `\l_keys_choice_tl` or `\l_keys_choice_int`. These variables do not have defined behaviour when used outside of code created using `.generate_choices:n` (*i.e.* anything might happen).

Multiple choices are created in a very similar manner to mutually-exclusive choices, using the properties `.multichoice:` and `.multichoices:nn`. As with mutually exclusive choices, multiple choices are define as sub-keys. Thus both

```
\keys_define:nn { module }
{
  key .multichoices:nn =
    { choice-a, choice-b, choice-c }
    {
      You~gave~choice~'\int_use:N \l_keys_choice_tl',~
      which~is~in~position~
      \int_use:N \l_keys_choice_int \c_space_tl
      in~the~list.
    }
}
```

and

```
\keys_define:nn { module }
{
  key .multichoice:,
  key / choice-a .code:n = code-a,
  key / choice-b .code:n = code-b,
  key / choice-c .code:n = code-c,
}
```

are valid. The `.multichoices:nn` property causes `\l_keys_choice_tl` and `\l_keys_choice_int` to be set in exactly the same way as described for `.choices:nn`.

When multiple choice keys are set, the value is treated as a comma-separated list:

```
\keys_set:nn { module }
{
  key = { a , b , c } % 'key' defined as a multiple choice
}
```

Each choice will be applied in turn, with the usual handling of unknown values.

164 Setting keys

`\keys_set:nn`
`\keys_set:(nV|nv|no)`

`\keys_set:nn {<module>} {<keyval list>}`

Parses the `<keyval list>`, and sets those keys which are defined for `<module>`. The behaviour on finding an unknown key can be set by defining a special `unknown` key: this will be illustrated later.

If a key is not known, `\keys_set:nn` will look for a special `unknown` key for the same module. This mechanism can be used to create new keys from user input.

```
\keys_define:nn { module }
{
  unknown .code:n =
    You~tried~to~set~key~'\l_keys_key_tl'~to~'#1'.
}
```

`\l_keys_key_tl`

When processing an unknown key, the name of the key is available as `\l_keys_key_tl`. Note that this will have been processed using `\tl_to_str:n`.

`\l_keys_path_tl`

When processing an unknown key, the path of the key used is available as `\l_keys_path_tl`. Note that this will have been processed using `\tl_to_str:n`.

`\l_keys_value_tl`

When processing an unknown key, the value of the key is available as `\l_keys_value_tl`. Note that this will be empty if no value was given for the key.

165 Setting known keys only

The functionality described in this section is experimental and may be altered or removed, depending on feedback.

<code>\keys_set_known:nnN</code>	<code>\keys_set_known:nn {<module>} {<keyval list>} <clist></code>
<code>\keys_set_known:(nVN nvN noN)</code>	

New: 2011-08-23

Parses the *<keyval list>*, and sets those keys which are defined for *<module>*. Any keys which are unknown are not processed further by the parser. The key–value pairs for each *unknown* key name will be stored in the *<clist>*.

166 Utility functions for keys

<code>\keys_if_exist_p:nn *</code>	<code>\keys_if_exist_p:nn <module> <key></code>
<code>\keys_if_exist:nnTF *</code>	<code>\keys_if_exist:nnTF <module> <key> {<true code>} {<false code>}</code>

Tests if the *<key>* exists for *<module>*, *i.e.* if any code has been defined for *<key>*.

<code>\keys_if_choice_exist_p:nn *</code>	<code>\keys_if_exist_p:nnn <module> <key> <choice></code>
<code>\keys_if_choice_exist:nnTF *</code>	<code>\keys_if_exist:nnnTF <module> <key> <choice> {<true code>} {<false code>}</code>

New: 2011-08-21

Tests if the *<choice>* is defined for the *<key>* within the *<module>*, *i.e.* if any code has been defined for *<key>/<choice>*. The test is **false** if the *<key>* itself is not defined.

<code>\keys_show:nn</code>	<code>\keys_show:nn {<module>} {<key>}</code>
----------------------------	---

Shows the function which is used to actually implement a *<key>* for a *<module>*.

167 Low-level interface for parsing key–val lists

To re-cap from earlier, a key–value list is input of the form

```
KeyOne = ValueOne ,
KeyTwo = ValueTwo ,
KeyThree
```

where each key–value pair is separated by a comma from the rest of the list, and each key–value pair does not necessarily contain an equals sign or a value! Processing this type of input correctly requires a number of careful steps, to correctly account for braces, spaces and the category codes of separators.

While the functions described earlier are used as a high-level interface for processing such input, in especial circumstances you may wish to use a lower-level approach. The low-level parsing system converts a *<key–value list>* into *<keys>* and associated *<values>*. After the parsing phase is completed, the resulting keys and values (or keys alone) are available for further processing. This processing is not carried out by the low-level parser itself, and so the parser requires the names of two functions along with the key–value list. One function is needed to process key–value pairs (*i.e.* two arguments), and a second function if required for keys given without arguments (*i.e.* a single argument).

The parser does not double # tokens or expand any input. The tokens = and , are corrected so that the parser does not “miss” any due to category code changes. Spaces are removed from the ends of the keys and values. Values which are given in braces will have exactly one set removed, thus

```
key = {value here},
```

and

```
key = value here,
```

are treated identically.

`\keyval_parse:NNn`

Updated: 2011-09-08

`\keyval_parse:NNn` $\langle function1 \rangle$ $\langle function2 \rangle$ $\{ \langle key-value list \rangle \}$

Parses the $\langle key-value list \rangle$ into a series of $\langle keys \rangle$ and associated $\langle values \rangle$, or keys alone (if no $\langle value \rangle$ was given). $\langle function1 \rangle$ should take one argument, while $\langle function2 \rangle$ should absorb two arguments. After `\keyval_parse:NNn` has parsed the $\langle key-value list \rangle$, $\langle function1 \rangle$ will be used to process keys given with no value and $\langle function2 \rangle$ will be used to process keys given with a value. The order of the $\langle keys \rangle$ in the $\langle key-value list \rangle$ will be preserved. Thus

```
\keyval_parse:NNn \function:n \function:nn
{ key1 = value1 , key2 = value2, key3 = , key4 }
```

will be converted into an input stream

```
\function:nn { key1 } { value1 }
\function:nn { key2 } { value2 }
\function:nn { key3 } { }
\function:n { key4 }
```

Note that there is a difference between an empty value (an equals sign followed by nothing) and a missing value (no equals sign at all). Spaces are trimmed from the ends of the $\langle key \rangle$ and $\langle value \rangle$, and any *outer* set of braces are removed from the $\langle value \rangle$ as part of the processing.

Part XXI

The l3file package

File operations

In contrast to the l3io module, which deals with the lowest level of file management, the l3file module provides a higher level interface for handling file contents. This involves providing convenient wrappers around many of the functions in l3io to make them more generally accessible.

It is important to remember that T_EX will attempt to locate files using both the operating system path and entries in the T_EX file database (most T_EX systems use such a database). Thus the “current path” for T_EX is somewhat broader than that for other programs.

168 File operation functions

`\g_file_current_name_tl`

Contains the name of the current L^AT_EX file. This variable should not be modified: it is intended for information only. It will be equal to `\c_job_name_tl` at the start of a L^AT_EX run and will be modified each time a file is read using `\file_input:n`.

`\file_if_exist:nTF`

`\file_if_exist:nTF {<file name>} {<true code>} {<false code>}`

Searches for `<file name>` using the current T_EX search path and the additional paths controlled by `\file_path_include:n`.

T_EXhackers note: The `<file name>` may contain both literal items and expandable content, which should on full expansion be the desired file name. The expansion occurs when T_EX searches for the file.

`\file_add_path:nN`

`\file_add_path:nN {<file name>} <tl var>`

Searches for `<file name>` in the path as detailed for `\file_if_exist:nTF`, and if found sets the `<tl var>` the fully-qualified name of the file, *i.e.* the path and file name. If the file is not found then the `<tl var>` will be empty.

T_EXhackers note: The `<file name>` may contain both literal items and expandable content, which should on full expansion be the desired file name. The expansion occurs when T_EX searches for the file.

<code>\file_input:n</code>	<code>\file_input:n {<file name>}</code>
----------------------------	--

Searches for *<file name>* in the path as detailed for `\file_if_exist:nTF`, and if found reads in the file as additional L^AT_EX source. All files read are recorded for information and the file name stack is updated by this function.

T_EXhackers note: The *<file name>* may contain both literal items and expandable content, which should on full expansion be the desired file name. The expansion occurs when T_EX searches for the file.

<code>\file_path_include:n</code>	<code>\file_path_include:n {<path>}</code>
-----------------------------------	--

Adds *<path>* to the list of those used to search for files by the `\file_input:n` and `\file_if_exist:n` function. The assignment is local.

<code>\file_path_remove:n</code>	<code>\file_path_remove:n {<path>}</code>
----------------------------------	---

Removes *<path>* from the list of those used to search for files by the `\file_input:n` and `\file_if_exist:n` function. The assignment is local.

<code>\file_list</code>	<code>\file_list:</code>
-------------------------	--------------------------

This function will list all files loaded using `\file_input:n` in the log file.

169 Internal file functions

<code>\g_file_stack_seq</code>	
--------------------------------	--

Stores the stack of nested files loaded using `\file_input:n`. This is needed to restore the appropriate file name to `\g_file_current_name_tl` at the end of each file.

<code>\g_file_record_seq</code>	
---------------------------------	--

Stores the name of every file loaded using `\file_input:n`. In contrast to `\g_file_stack_seq`, no items are ever removed from this sequence.

<code>\l_file_name_tl</code>	
------------------------------	--

Used to return the full name of a file for internal use.

<code>\l_file_search_path_seq</code>	
--------------------------------------	--

The sequence of file paths to search when loading a file.

<code>\l_file_search_path_saved_seq</code>	
--	--

When loaded on top of L^AT_EX 2_ε, there is a need to save the search path so that `\input@path` can be used as appropriate.

<code>\l_file_tmpa_seq</code>	
-------------------------------	--

When loaded on top of L^AT_EX 2_ε, there is a need to convert the comma lists `\input@path` and `\@filelist` to sequences.

New: 2011-09-06

Part XXII

The l3fp package

Floating-point operations

A floating point number is one which is stored as a mantissa and a separate exponent. This module implements arithmetic using radix 10 floating point numbers. This means that the mantissa should be a real number in the range $1 \leq |x| < 10$, with the exponent given as an integer between -99 and 99 . In the input, the exponent part is represented starting with an `e`. As this is a low-level module, error-checking is minimal. Numbers which are too large for the floating point unit to handle will result in errors, either from `TEX` or from `LATEX`. The `LATEX` code does not check that the input will not overflow, hence the possibility of a `TEX` error. On the other hand, numbers which are too small will be dropped, which will mean that extra decimal digits will simply be lost.

When parsing numbers, any missing parts will be interpreted as zero. So for example

```
\fp_set:Nn \l_my_fp { }  
\fp_set:Nn \l_my_fp { . }  
\fp_set:Nn \l_my_fp { - }
```

will all be interpreted as zero values without raising an error.

Operations which give an undefined result (such as division by 0) will not lead to errors. Instead special marker values are returned, which can be tested for using for example `\fp_if_undefined:N(TF)`. In this way it is possible to work with asymptotic functions without first checking the input. If these special values are carried forward in calculations they will be treated as 0.

Floating point numbers are stored in the `fp` floating point variable type. This has a standard range of functions for variable management.

170 Floating-point variables

<code>\fp_new:N</code>	<code>\fp_new:N</code> <i><floating point variable></i>
------------------------	---

<code>\fp_new:c</code>	Creates a new <i><floating point variable></i> or raises an error if the name is already taken. The declaration global. The <i><floating point></i> will initially be set to <code>+0.000000000e0</code> (the zero floating point).
------------------------	---

<code>\fp_const:Nn</code>	<code>\fp_const:Nn</code> <i><floating point variable></i> <i>{<value>}</i>
---------------------------	---

<code>\fp_const:cn</code>	Creates a new constant <i><floating point variable></i> or raises an error if the name is already taken. The value of the <i><floating point variable></i> will be set globally to the <i><value></i> .
---------------------------	---

<code>\fp_set_eq:NN</code>	<code>\fp_set_eq:NN</code> <i><fp var1></i> <i><fp var2></i>
----------------------------	--

<code>\fp_set_eq:(cN Nc cc)</code>	Sets the value of <i><floating point variable1></i> equal to that of <i><floating point variable2></i> .
------------------------------------	--

<code>\fp_gset_eq:NN</code>	
<code>\fp_gset_eq:(cN Nc cc)</code>	

<hr/>	
<code>\fp_zero:N</code>	<code>\fp_zero:N</code> <i><floating point variable></i>
<code>\fp_zero:c</code>	Sets the <i><floating point variable></i> to +0.000000000e0.
<code>\fp_gzero:N</code>	
<code>\fp_gzero:c</code>	
<hr/>	
<code>\fp_set:Nn</code>	<code>\fp_set:Nn</code> <i><floating point variable></i> { <i><value></i> }
<code>\fp_set:cn</code>	Sets the <i><floating point variable></i> variable to <i><value></i> .
<code>\fp_gset:Nn</code>	
<code>\fp_gset:cn</code>	
<hr/>	
<code>\fp_set_from_dim:Nn</code>	<code>\fp_set_from_dim:Nn</code> <i><floating point variable></i> { <i><dimexpr></i> }
<code>\fp_set_from_dim:cn</code>	Sets the <i><floating point variable></i> to the distance represented by the <i><dimension expression></i>
<code>\fp_gset_from_dim:Nn</code>	in the units points. This means that distances given in other units are first converted to
<code>\fp_gset_from_dim:cn</code>	points before being assigned to the <i><floating point variable></i> .
<hr/>	
<code>\fp_use:N</code> ☆	<code>\fp_use:N</code> <i><floating point variable></i>
<code>\fp_use:c</code> ☆	Inserts the value of the <i><floating point variable></i> into the input stream. The value will
<hr/>	be given as a real number without any exponent part, and will always include a decimal
	point. For example,
	 <code>\fp_new:Nn \test</code> <code>\fp_set:Nn \test { 1.234 e 5 }</code> <code>\fp_use:N \test</code>
	 will insert 12345.00000 into the input stream. As illustrated, a floating point will always
	be inserted with ten significant digits given. Very large and very small values will include
	additional zeros for place value.
<hr/>	
<code>\fp_show:N</code>	<code>\fp_show:N</code> <i><floating point variable></i>
<code>\fp_show:c</code>	Displays the content of the <i><floating point variable></i> on the terminal.
<hr/>	

171 Conversion of floating point values to other formats

It is useful to be able to convert floating point variables to other forms. These functions are expandable, so that the material can be used in a variety of contexts. The `\fp_use:N` function should also be consulted in this context, as it will insert the value of the floating point variable as a real number.

<hr/>	
<code>\fp_to_dim:N</code> ☆	<code>\fp_to_dim:N</code> <i><floating point variable></i>
<code>\fp_to_dim:c</code> ☆	Inserts the value of the <i><floating point variable></i> into the input stream converted into a
<hr/>	dimension in points.

<hr/>	
<code>\fp_to_int:N</code> ☆	<code>\fp_to_int:N</code> <i><floating point variable></i>
<code>\fp_to_int:c</code> ☆	Inserts the integer value of the <i><floating point variable></i> into the input stream. The decimal part of the number will not be included, but will be used to round the integer.
<hr/>	
<code>\fp_to_tl:N</code> ☆	<code>\fp_to_tl:N</code> <i><floating point variable></i>
<code>\fp_to_tl:c</code> ☆	Inserts a representation of the <i><floating point variable></i> into the input stream as a token list. The representation follows the conventions of a pocket calculator:

Floating point value	Representation
1.234000000000e0	1.234
-1.234000000000e0	-1.234
1.234000000000e3	1234
1.234000000000e13	1234e13
1.234000000000e-1	0.1234
1.234000000000e-2	0.01234
1.234000000000e-3	1.234e-3

Notice that trailing zeros are removed in this process, and that numbers which do not require a decimal part do *not* include a decimal marker.

172 Rounding floating point values

The module can round floating point values to either decimal places or significant figures using the usual method in which exact halves are rounded up.

<hr/>	
<code>\fp_round_figures:Nn</code>	<code>\fp_round_figures:Nn</code> <i><floating point variable></i> <i>{<target>}</i>
<code>\fp_round_figures:cn</code>	
<code>\fp_ground_figures:Nn</code>	Rounds the <i><floating point variable></i> to the <i><target></i> number of significant figures (an integer expression).
<code>\fp_ground_figures:cn</code>	
<hr/>	
<code>\fp_round_places:Nn</code>	<code>\fp_round_places:Nn</code> <i><floating point variable></i> <i>{<target>}</i>
<code>\fp_round_places:cn</code>	
<code>\fp_ground_places:Nn</code>	Rounds the <i><floating point variable></i> to the <i><target></i> number of decimal places (an integer expression).
<code>\fp_ground_places:cn</code>	

173 Floating-point conditionals

<hr/>	
<code>\fp_if_undefined_p:N</code> ☆	<code>\fp_if_undefined_p:N</code> <i><fixed-point></i>
<code>\fp_if_undefined:NTF</code> ☆	<code>\fp_if_undefined:NTF</code> <i><fixed-point></i> <i>{<true code>}</i> <i>{<false code>}</i>
<hr/>	
	Tests if <i><floating point></i> is undefined (<i>i.e.</i> equal to the special <code>\c_undefined_fp</code> variable).

<code>\fp_if_zero_p:N</code> ★	<code>\fp_if_zero_p:N</code> $\langle fixed\text{-}point \rangle$
<code>\fp_if_zero:NTF</code> ★	<code>\fp_if_zero:NTF</code> $\langle fixed\text{-}point \rangle$ $\{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$

Tests if $\langle floating\ point \rangle$ is equal to zero (*i.e.* equal to the special `\c_zero_fp` variable).

<code>\fp_compare:nNnTF</code>	<code>\fp_compare:nNnTF</code> $\{\langle floating\ point_1 \rangle\} \langle relation \rangle \{\langle floating\ point_2 \rangle\}$ $\{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$
--------------------------------	--

This function compared the two $\langle floating\ point \rangle$ values, which may be stored as `fp` variables, using the $\langle relation \rangle$:

Equal	=
Greater than	>
Less than	<

The tests treat undefined floating points as zero as the comparison is intended for real numbers only.

<code>\fp_compare:nTF</code>	<code>\fp_compare:nTF</code> $\{\langle floating\ point_1 \rangle \langle relation \rangle \langle floating\ point_2 \rangle\}$ $\{\langle true\ code \rangle\} \{\langle false\ code \rangle\}$
------------------------------	--

This function compared the two $\langle floating\ point \rangle$ values, which may be stored as `fp` variables, using the $\langle relation \rangle$:

Equal	= or ==
Greater than	>
Greater than or equal	>=
Less than	<
Less than or equal	<=
Not equal	!=

The tests treat undefined floating points as zero as the comparison is intended for real numbers only.

174 Unary floating-point operations

The unary operations alter the value stored within an `fp` variable.

<code>\fp_abs:N</code>	<code>\fp_abs:N</code> $\langle floating\ point\ variable \rangle$
<code>\fp_abs:c</code>	
<code>\fp_gabs:N</code>	Converts the $\langle floating\ point\ variable \rangle$ to its absolute value.
<code>\fp_gabs:c</code>	

<code>\fp_neg:N</code>	<code>\fp_neg:N</code> $\langle floating\ point\ variable \rangle$
<code>\fp_neg:c</code>	
<code>\fp_gneg:N</code>	Reverse the sign of the $\langle floating\ point\ variable \rangle$.
<code>\fp_gneg:c</code>	

175 Floating-point arithmetic

Binary arithmetic operations act on the value stored in an `fp`, so for example

```
\fp_set:Nn \l_my_fp { 1.234 }  
\fp_sub:Nn \l_my_fp { 5.678 }
```

sets `\l_my_fp` to the result of $1.234 - 5.678$ (*i.e.* -4.444).

<code>\fp_add:Nn</code>	<code>\fp_add:Nn <floating point> {<value>}</code>
<code>\fp_add:cn</code>	Adds the <i><value></i> to the <i><floating point></i> .
<code>\fp_gadd:Nn</code>	
<code>\fp_gadd:cn</code>	

<code>\fp_sub:Nn</code>	<code>\fp_sub:Nn <floating point> {<value>}</code>
<code>\fp_sub:cn</code>	Subtracts the <i><value></i> from the <i><floating point></i> .
<code>\fp_gsub:Nn</code>	
<code>\fp_gsub:cn</code>	

<code>\fp_mul:Nn</code>	<code>\fp_mul:Nn <floating point> {<value>}</code>
<code>\fp_mul:cn</code>	
<code>\fp_gmul:Nn</code>	Multiplies the <i><floating point></i> by the <i><value></i> .
<code>\fp_gmul:cn</code>	

<code>\fp_div:Nn</code>	<code>\fp_div:Nn <floating point> {<value>}</code>
<code>\fp_div:cn</code>	
<code>\fp_gdiv:Nn</code>	Divides the <i><floating point></i> by the <i><value></i> , making the assignment within the current <code>TeX</code>
<code>\fp_gdiv:cn</code>	group level. If the <i><value></i> is zero, the <i><floating point></i> will be set to <code>\c_undefined_fp</code> .

176 Floating-point power operations

<code>\fp_pow:Nn</code>	<code>\fp_pow:Nn <floating point> {<value>}</code>
<code>\fp_pow:cn</code>	
<code>\fp_gpow:Nn</code>	Raises the <i><floating point></i> to the given <i><value></i> . If the <i><floating point></i> is negative, then the
<code>\fp_gpow:cn</code>	<i><value></i> should be either a positive real number or a negative integer. If the <i><floating point></i>

is positive, then the *<value>* may be any real value. Mathematically invalid operations such as 0^0 will give set the *<floating point>* to to `\c_undefined_fp`.

177 Exponential and logarithm functions

<code>\fp_exp:Nn</code>	<code>\fp_exp:Nn <floating point> {<value>}</code>
<code>\fp_exp:cn</code>	
<code>\fp_gexp:Nn</code>	Calculates the exponential of the <i><value></i> and assigns this to the <i><floating point></i> .
<code>\fp_gexp:cn</code>	

<u>\fp_ln:Nn</u>	\fp_ln:Nn <i><floating point></i> { <i><value></i> }
<u>\fp_ln:cn</u>	
<u>\fp_gln:Nn</u>	Calculates the natural logarithm of the <i><value></i> and assigns this to the <i><floating point></i> .
<u>\fp_gln:cn</u>	

178 Trigonometric functions

The trigonometric functions all work in radians. They accept a maximum input value of 100 000 000, as there are issues with range reduction and very large input values.

<u>\fp_sin:Nn</u>	\fp_sin:Nn <i><floating point></i> { <i><value></i> }
<u>\fp_sin:cn</u>	
<u>\fp_gsin:Nn</u>	Assigns the sine of the <i><value></i> to the <i><floating point></i> . The <i><value></i> should be given in radians.
<u>\fp_gsin:cn</u>	

<u>\fp_cos:Nn</u>	\fp_cos:Nn <i><floating point></i> { <i><value></i> }
<u>\fp_cos:cn</u>	
<u>\fp_gcos:Nn</u>	Assigns the cosine of the <i><value></i> to the <i><floating point></i> . The <i><value></i> should be given in radians.
<u>\fp_gcos:cn</u>	

<u>\fp_tan:Nn</u>	\fp_tan:Nn <i><floating point></i> { <i><value></i> }
<u>\fp_tan:cn</u>	
<u>\fp_gtan:Nn</u>	Assigns the tangent of the <i><value></i> to the <i><floating point></i> . The <i><value></i> should be given in radians.
<u>\fp_gtan:cn</u>	

179 Constant floating point values

<u>\c_e_fp</u>	The value of the base of natural numbers, e.
<u>\c_one_fp</u>	A floating point variable with permanent value 1: used for speeding up some comparisons.
<u>\c_pi_fp</u>	The value of π .
<u>\c_undefined_fp</u>	A special marker floating point variable representing the result of an operation which does not give a defined result (such as division by 0).
<u>\c_zero_fp</u>	A permanently zero floating point variable.

180 Notes on the floating point unit

As calculation of the elemental transcendental functions is computationally expensive compared to storage of results, after calculating a trigonometric function, exponent, *etc.* the module stored the result for reuse. Thus the performance of the module for repeated operations, most probably trigonometric functions, should be much higher than if the values were re-calculated every time they were needed.

Anyone with experience of programming floating point calculations will know that this is a complex area. The aim of the unit is to be accurate enough for the likely applications in a typesetting context. The arithmetic operations are therefore intended to provide ten digit accuracy with the last digit accurate to ± 1 . The elemental transcendental functions may not provide such high accuracy in every case, although the design aim has been to provide 10 digit accuracy for cases likely to be relevant in typesetting situations. A good overview of the challenges in this area can be found in J.-M. Muller, *Elementary functions: algorithms and implementation*, 2nd edition, Birkhäuser Boston, New York, USA, 2006.

The internal representation of numbers is tuned to the needs of the underlying \TeX system. This means that the format is somewhat different from that used in, for example, computer floating point units. Programming in \TeX makes it most convenient to use a radix 10 system, using \TeX `count` registers for storage and taking advantage where possible of delimited arguments.

Part XXIII

The l3_{lua}tex package

LuaTeX-specific functions

181 Breaking out to Lua

The LuaTeX engine provides access to the Lua programming language, and with it access to the “internals” of T_EX. In order to use this within the framework provided here, a family of functions is available. When used with pdfT_EX or XeT_EX these will raise an error: use `\luatex_if_engine:T` to avoid this. Details of coding the LuaTeX engine are detailed in the LuaTeX manual.

<code>\lua_now:n</code>	★	<code>\lua_now:n {⟨token list⟩}</code>
-------------------------	---	--

<code>\lua_now:x</code>	★	
-------------------------	---	--

The *⟨token list⟩* is first tokenized by T_EX, which will include converting line ends to spaces in the usual T_EX manner and which respects currently-applicable T_EX category codes. The resulting *⟨Lua input⟩* is passed to the Lua interpreter for processing. Each `\lua_now:n` block is treated by Lua as a separate chunk. The Lua interpreter will execute the *⟨Lua input⟩* immediately, and in an expandable manner.

T_EXhackers note: `\lua_now:x` is the LuaTeX primitive `\directlua` renamed.

<code>\lua_shipout:n</code>		<code>\lua_shipout:x {⟨token list⟩}</code>
-----------------------------	--	--

<code>\lua_shipout:x</code>		
-----------------------------	--	--

The *⟨token list⟩* is first tokenized by T_EX, which will include converting line ends to spaces in the usual T_EX manner and which respects currently-applicable T_EX category codes. The resulting *⟨Lua input⟩* is passed to the Lua interpreter when the current page is finalised (*i.e.* at shipout). Each `\lua_shipout:n` block is treated by Lua as a separate chunk. The Lua interpreter will execute the *⟨Lua input⟩* during the page-building routine: no T_EX expansion of the *⟨Lua input⟩* will occur at this stage.

T_EXhackers note: At a T_EX level, the *⟨Lua input⟩* is stored as a “whatsit”.

<code>\lua_shipout_x:n</code>	<code>\lua_shipout:n {⟨token list⟩}</code>
<code>\lua_shipout_x:x</code>	

The *⟨token list⟩* is first tokenized by \TeX , which will include converting line ends to spaces in the usual \TeX manner and which respects currently-applicable \TeX category codes. The resulting *⟨Lua input⟩* is passed to the Lua interpreter when the current page is finalised (*i.e.* at shipout). Each `\lua_shipout:n` block is treated by Lua as a separate chunk. The Lua interpreter will execute the *⟨Lua input⟩* during the page-building routine: the *⟨Lua input⟩* is expanded during this process in addition to any expansion when the argument was read. This makes these functions suitable for including material finalised during the page building process (such as the page number).

\TeX hackers note: `\lua_shipout_x:n` is the Lua \TeX primitive `\latelua` named using the \LaTeX 3 scheme.

At a \TeX level, the *⟨Lua input⟩* is stored as a “whatsit”.

182 Category code tables

As well as providing methods to break out into Lua, there are places where additional \LaTeX 3 functions are provided by the Lua \TeX engine. In particular, Lua \TeX provides category code tables. These can be used to ensure that a set of category codes are in force in a more robust way than is possible with other engines. These are therefore used by `\ExplSyntaxOn` and `\ExplSyntaxOff` when using the Lua \TeX engine.

<code>\cctab_new:N</code>	<code>\cctab_new:N ⟨category code table⟩</code>
	Creates a new category code table, initially with the codes as used by <code>\InitEX</code> .

<code>\cctab_gset:Nn</code>	<code>\cctab_gset:Nn ⟨category code table⟩ {⟨category code set up⟩}</code>
	Sets the <i>⟨category code table⟩</i> to apply the category codes which apply when the prevailing regime is modified by the <i>⟨category code set up⟩</i> . Thus within a standard code block the starting point will be the code applied by <code>\c_code_cctab</code> . The assignment of the table is global: the underlying primitive does not respect grouping.

<code>\cctab_begin:N</code>	<code>\cctab_begin:N ⟨category code table⟩</code>
	Switches the category codes in force to those stored in the <i>⟨category code table⟩</i> . The prevailing codes before the function is called are added to a stack, for use with <code>\cctab_end:</code> .

<code>\cctab_end</code>	<code>\cctab_end:</code>
	Ends the scope of a <i>⟨category code table⟩</i> started using <code>\cctab_begin:N</code> , retuning the codes to those in force before the matching <code>\cctab_begin:N</code> was used.

<code>\c_code_cctab</code>	Category code table for the code environment. This does not include setting the behaviour of the line-end character, which is only altered by <code>\ExplSyntaxOn</code> .
----------------------------	--

<hr/> <hr/> <code>\c_document_cctab</code>	Category code table for a standard L ^A T _E X document. This does not include setting the behaviour of the line-end character, which is only altered by <code>\ExplSyntaxOff</code> .
<hr/> <hr/> <code>\c_initex_cctab</code>	Category code table as set up by IniT _E X.
<hr/> <hr/> <code>\c_other_cctab</code>	Category code table where all characters have category code 12 (other).
<hr/> <hr/> <code>\c_str_cctab</code>	Category code table where all characters have category code 12 (other) with the exception of spaces, which have category code 10 (space).

Part XXIV

Implementation

183 l3bootstrap implementation

```
1 <*initex | package>
```

183.1 Format-specific code

The very first thing to do is to bootstrap the IniT_EX system so that everything else will actually work. T_EX does not start with some pretty basic character codes set up.

```
2 <*initex>
3 \catcode '\{ = 1 \relax
4 \catcode '\} = 2 \relax
5 \catcode '\# = 6 \relax
6 \catcode '\^ = 7 \relax
7 </initex>
```

Tab characters should not show up in the code, but to be on the safe side.

```
8 <*initex>
9 \catcode '\^^I = 10 \relax
10 </initex>
```

For LuaT_EX the extra primitives need to be enabled before they can be use. No `\ifdefined` yet, so do it the old-fashioned way. The primitive `\strcmp` is simulated using some Lua code, which currently has to be applied to every job as the Lua code is not part of the format. Thanks to Taco Hoekwater for this code. The odd `\csname` business is needed so that the later deletion code will work.

```
11 <*initex>
12 \begingroup\expandafter\expandafter\expandafter\endgroup
13 \expandafter\ifx\csname directlua\endcsname\relax
14 \else
15   \directlua
16     {
```

```

17 tex.enableprimitives('',tex.extraprimitives ())
18 lua.bytecode[1] = function ()
19   function strcmp (A, B)
20     if A == B then
21       tex.write("0")
22     elseif A < B then
23       tex.write("-1")
24     else
25       tex.write("1")
26     end
27   end
28 end
29 lua.bytecode[1]()
30 }
31 \everyjob\expandafter
32 {\csname\detokenize{luatex_directlua:D}\endcsname{lua.bytecode[1]()}}
33 \long\edef\pdfstrcmp#1#2%
34 {%
35   \expandafter\noexpand\csname\detokenize{luatex_directlua:D}\endcsname
36   {%
37     strcmp%
38     (%
39       "\noexpand\luaescapestring{#1}",%
40       "\noexpand\luaescapestring{#2}"%
41     )%
42   }%
43 }
44 \fi
45 \</initex>

```

183.2 Package-specific code

The package starts by identifying itself: the information itself is taken from the SVN Id string at the start of the source file.

```

46 <*package>
47 \ProvidesPackage{l3bootstrap}
48 [%
49   \ExplFileDate\space v\ExplFileVersion\space
50   L3 Experimental bootstrap code%
51 ]
52 </package>

```

For LuaTeX the functionality of the `\pdfstrcmp` primitive needs to be provided: the `pdftexmcds` package is used to do this if necessary. At present, there is also a need to deal with some low-level allocation stuff that could usefully be added to `lualatex.ini`. As it is currently not, load Heiko Oberdiek's `luatex` package instead.

```

53 <*package>
54 \def\@tempa%
55 {%

```

```

56 \def\@tempa{}%
57 \RequirePackage{luatex}%
58 \RequirePackage{pdfsync}%
59 \let\pdfsync\pdf@sync
60 }
61 \begingroup\expandafter\expandafter\expandafter\endgroup
62 \expandafter\ifx\csname directlua\endcsname\relax
63 \else
64 \expandafter\@tempa
65 \fi
66 \end{package}

```

`\ExplSyntaxOff` Experimental syntax switching is set up here for the package-loading process. These are
`\ExplSyntaxOn` redefined in `expl3` for the package and in `l3final` for the format.

```

67 \begin{package}
68 \protected\def\ExplSyntaxOff
69 {
70 \catcode 9 = \the\catcode 9\relax
71 \catcode 32 = \the\catcode 32\relax
72 \catcode 34 = \the\catcode 34\relax
73 \catcode 38 = \the\catcode 38\relax
74 \catcode 58 = \the\catcode 58\relax
75 \catcode 94 = \the\catcode 94\relax
76 \catcode 95 = \the\catcode 95\relax
77 \catcode 124 = \the\catcode 124\relax
78 \catcode 126 = \the\catcode 126\relax
79 \endlinechar = \the\endlinechar\relax
80 \chardef\csname\detokenize{l_expl_status_bool}\endcsname = 0 \relax
81 }
82 \protected\def\ExplSyntaxOn
83 {
84 \catcode 9 = 9 \relax
85 \catcode 32 = 9 \relax
86 \catcode 34 = 12 \relax
87 \catcode 58 = 11 \relax
88 \catcode 94 = 7 \relax
89 \catcode 95 = 11 \relax
90 \catcode 124 = 12 \relax
91 \catcode 126 = 10 \relax
92 \endlinechar = 32 \relax
93 \chardef\csname\detokenize{l_expl_status_bool}\endcsname = 1 \relax
94 }
95 \end{package}

```

(End definition for `\ExplSyntaxOff` and `\ExplSyntaxOn`. These functions are documented on page

6.)

`\l_expl_status_bool` The status for experimental code syntax: this is off at present. This code is used by both
the package and the format.

```

96 \expandafter\chardef\csname\detokenize{l_expl_status_bool}\endcsname = 0 \relax
(End definition for \l_expl_status_bool. This function is documented on page ??.)

```

183.3 Dealing with package-mode meta-data

`\GetIdInfo` Functions for collecting up meta-data from the SVN information used by the L^AT_EX3 Project.

`\GetIdInfoFull`

`\GetIdInfoAuxI`

`\GetIdInfoAuxII`

`\GetIdInfoAuxIII`

`\GetIdInfoAuxCVS`

`\GetIdInfoAuxSVN`

```

97  <*package>
98  \protected\def\GetIdInfo
99  {
100    \begingroup
101    \catcode 32 = 10 \relax
102    \GetIdInfoAuxI
103  }
104  \protected\def\GetIdInfoAuxI$#1$#2%
105  {
106    \def\tempa{#1}%
107    \def\tempb{Id}%
108    \ifx\tempa\tempb
109      \def\tempa
110      {%
111        \endgroup
112        \def\ExplFileName{9999/99/99}%
113        \def\ExplFileDescription{#2}%
114        \def\ExplFileName{[unknown name]}%
115        \def\ExplFileVersion{999}%
116      }%
117    \else
118      \def\tempa
119      {%
120        \endgroup
121        \GetIdInfoAuxII$#1$#2}%
122      }%
123    \fi
124    \tempa
125  }
126  \protected\def\GetIdInfoAuxII$#1 #2.#3 #4 #5 #6 #7 #8$#9%
127  {%
128    \def\ExplFileName{#2}%
129    \def\ExplFileVersion{#4}%
130    \def\ExplFileDescription{#9}%
131    \GetIdInfoAuxIII#5\relax#3\relax#5\relax#6\relax
132  }
133  \protected\def\GetIdInfoAuxIII#1#2#3#4#5#6\relax
134  {%
135    \ifx#5/%
136      \expandafter\GetIdInfoAuxCVS
137    \else
138      \expandafter\GetIdInfoAuxSVN
139    \fi
140  }
141  \protected\def\GetIdInfoAuxCVS#1,v\relax#2\relax#3\relax
142  {\def\ExplFileName{#2}}

```

```

143 \protected\def\GetIdInfoAuxSVN#1\relax#2-#3-#4\relax#5Z\relax
144   {\def\ExplFileDate{#2/#3/#4}}
145 \</package>
      (End definition for \GetIdInfo. This function is documented on page ??.)

```

\ProvidesExplPackage For other packages and classes building on this one it is convenient not to need
 \ProvidesExplClass \ExplSyntaxOn each time.
 \ProvidesExplFile

```

146 <*package>
147 \protected\def\ProvidesExplPackage#1#2#3#4%
148   {%
149     \ProvidesPackage{#1}[#2 v#3 #4]%
150     \ExplSyntaxOn
151   }
152 \protected\def\ProvidesExplClass#1#2#3#4%
153   {%
154     \ProvidesClass{#1}[#2 v#3 #4]%
155     \ExplSyntaxOn
156   }
157 \protected\def\ProvidesExplFile#1#2#3#4%
158   {%
159     \ProvidesFile{#1}[#2 v#3 #4]%
160     \ExplSyntaxOn
161   }
162 \</package>

```

(End definition for \ProvidesExplPackage, \ProvidesExplClass, and \ProvidesExplFile. These functions are documented on page 6.)

\@pushfilename The idea here is to use L^AT_EX 2_ε's \@pushfilename and \@popfilename to track the
 \@popfilename current syntax status. This can be achieved by saving the current status flag at each
 push to a stack, then recovering it at the pop stage and checking if the code environment
 should still be active.

```

163 <*package>
164 \edef\@pushfilename
165   {%
166     \edef\expandafter\noexpand
167     \csname\detokenize{l_expl_status_stack_tl}\endcsname
168     {%
169       \noexpand\ifodd\expandafter\noexpand
170       \csname\detokenize{l_expl_status_bool}\endcsname
171       1%
172     \noexpand\else
173     0%
174     \noexpand\fi
175     \expandafter\noexpand
176     \csname\detokenize{l_expl_status_stack_tl}\endcsname
177   }%
178   \ExplSyntaxOff
179   \unexpanded\expandafter{\@pushfilename}%
180 }

```

```

181 \edef\@popfilename
182 {%
183   \unexpanded\expandafter{\@popfilename}%
184   \noexpand\if a\expandafter\noexpand\csname
185     \detokenize{l_expl_status_stack_tl}\endcsname a%
186     \ExplSyntaxOff
187   \noexpand\else
188     \noexpand\expandafter
189     \expandafter\noexpand\csname
190       \detokenize{expl_status_pop:w}\endcsname
191       \expandafter\noexpand\csname
192         \detokenize{l_expl_status_stack_tl}\endcsname
193         \noexpand\@nil
194   \noexpand\fi
195 }
196 \</package>

```

(End definition for \@pushfilename and \@popfilename. These functions are documented on page ??.)

`\l_expl_status_stack_tl` As expl3 itself cannot be loaded with the code environment already active, at the end of the package `\ExplSyntaxOff` can safely be called.

```

197 \<package>
198 \@namedef{\detokenize{l_expl_status_stack_tl}}{0}
199 \</package>

```

(End definition for \l_expl_status_stack_tl. This function is documented on page ??.)

`\expl_status_pop:w` The pop auxiliary function removes the first item from the stack, saves the rest of the stack and then does the test. As `\ExplSyntaxOff` is already defined as a protected macro, there is no need for `\noexpand` here.

```

200 \<package>
201 \expandafter\edef\csname\detokenize{expl_status_pop:w}\endcsname#1#2\@nil
202 {%
203   \def\expandafter\noexpand
204     \csname\detokenize{l_expl_status_stack_tl}\endcsname{#2}%
205   \noexpand\ifodd#1\space
206     \noexpand\expandafter\noexpand\ExplSyntaxOn
207   \noexpand\else
208     \noexpand\expandafter\ExplSyntaxOff
209   \noexpand\fi
210 }
211 \</package>

```

(End definition for \expl_status_pop:w. This function is documented on page ??.)

We want the expl3 bundle to be loaded “as one”; this command is used to ensure that one of the 13 packages isn’t loaded on its own.

```

212 \<package>
213 \expandafter\protected\expandafter\def
214   \csname\detokenize{package_check_loaded_expl:}\endcsname
215   {%

```

```

216 \@ifpackageloaded{expl3}
217 {}
218 {%
219 \PackageError{expl3}
220 {Cannot load the expl3 modules separately}
221 {%
222 The expl3 modules cannot be loaded separately;\MessageBreak
223 please \string\usepackage\string{expl3\string} instead.
224 }%
225 }%
226 }
227 \</package>

```

183.4 The `\pdfstrcmp` primitive in X_YTeX

Only pdfTeX has a primitive called `\pdfstrcmp`. The X_YTeX version is just `\strcmp`, so there is some shuffling to do.

```

228 \begingroup\expandafter\expandafter\expandafter\endgroup
229 \expandafter\ifx\csname pdfstrcmp\endcsname\relax
230 \let\pdfstrcmp\strcmp
231 \fi

```

183.5 Engine requirements

The code currently requires functionality equivalent to `\pdfstrcmp` in addition to ε -TeX. The former is therefore used as a test for a suitable engine.

```

232 \begingroup\expandafter\expandafter\expandafter\endgroup
233 \expandafter\ifx\csname pdfstrcmp\endcsname\relax
234 \*package>
235 \PackageError{!3names}{Required primitive not found: \protect\pdfstrcmp}
236 {%
237 LaTeX3 requires the e-TeX primitives and
238 \string\pdfstrcmp.\MessageBreak
239 These are available in engine versions: \MessageBreak
240 - pdfTeX 1.30 \MessageBreak
241 - XeTeX 0.9994 \MessageBreak
242 - LuaTeX 0.60 \MessageBreak
243 or later. \MessageBreak
244 \MessageBreak
245 Loading of expl3 will abort!
246 }
247 \</package>
248 \*initex>
249 \newlinechar'\^^J\relax
250 \errhelp{%
251 LaTeX3 requires the e-TeX primitives and
252 \string\pdfstrcmp. ^^J
253 These are available in engine versions: ^^J
254 - pdfTeX 1.30 ^^J

```

```

255     - XeTeX 0.9994 ^^J
256     - LuaTeX 0.60 ^^J
257     or later. ^^J
258     For pdfTeX and XeTeX the '-etex' command-line switch is also
259     needed. ^^J
260     ^^J
261     Format building will abort!
262 }
263 </initex>
264 \expandafter\endinput
265 \fi

```

183.6 The L^AT_EX3 code environment

`\ExplSyntaxNamesOn` These can be set up early, as they are not used anywhere in the package or format itself.
`\ExplSyntaxNamesOff` Using an `\edef` here makes the definitions that bit clearer later.

```

266 \protected\edef\ExplSyntaxNamesOn
267 {%
268   \expandafter\noexpand
269   \csname\detokenize{char_set_catcode_letter:n}\endcsname{58}%
270   \expandafter\noexpand
271   \csname\detokenize{char_set_catcode_letter:n}\endcsname{95}%
272 }
273 \protected\edef\ExplSyntaxNamesOff
274 {%
275   \expandafter\noexpand
276   \csname\detokenize{char_set_catcode_other:n}\endcsname{58}%
277   \expandafter\noexpand
278   \csname\detokenize{char_set_catcode_math_subscript:n}\endcsname{95}%
279 }

```

(End definition for `\ExplSyntaxNamesOn` and `\ExplSyntaxNamesOff`. These functions are documented on page 6.)

The code environment is now set up for the format: the package deals with this using `\ProvidesExplPackage`.

```

280 <*initex>
281 \catcode 9 = 9 \relax
282 \catcode 32 = 9 \relax
283 \catcode 34 = 12 \relax
284 \catcode 58 = 11 \relax
285 \catcode 94 = 7 \relax
286 \catcode 95 = 11 \relax
287 \catcode 124 = 12 \relax
288 \catcode 126 = 10 \relax
289 \endlinechar = 32 \relax
290 </initex>

```

`\ExplSyntaxOn` The idea here is that multiple `\ExplSyntaxOn` calls are not going to mess up category
`\ExplSyntaxOff` codes, and that multiple calls to `\ExplSyntaxOff` are also not wasting time.


```

291 <*initex>
292 \protected \def \ExplSyntaxOn
293 {
294   \bool_if:NF \l_expl_status_bool
295   {
296     \cs_set_protected_nopar:Npx \ExplSyntaxOff
297     {
298       \char_set_catcode:nn { 9 } { \char_value_catcode:n { 9 } }
299       \char_set_catcode:nn { 32 } { \char_value_catcode:n { 32 } }
300       \char_set_catcode:nn { 34 } { \char_value_catcode:n { 34 } }
301       \char_set_catcode:nn { 38 } { \char_value_catcode:n { 38 } }
302       \char_set_catcode:nn { 58 } { \char_value_catcode:n { 58 } }
303       \char_set_catcode:nn { 94 } { \char_value_catcode:n { 94 } }
304       \char_set_catcode:nn { 95 } { \char_value_catcode:n { 95 } }
305       \char_set_catcode:nn { 124 } { \char_value_catcode:n { 124 } }
306       \char_set_catcode:nn { 126 } { \char_value_catcode:n { 126 } }
307       \tex_endlinechar:D =
308       \tex_the:D \tex_endlinechar:D \scan_stop:
309       \bool_set_false:N \l_expl_status_bool
310       \cs_set_protected_nopar:Npn \ExplSyntaxOff { }
311     }
312   }
313   \char_set_catcode_ignore:n { 9 } % tab
314   \char_set_catcode_ignore:n { 32 } % space
315   \char_set_catcode_other:n { 34 } % double quote
316   \char_set_catcode_alignment:n { 38 } % ampersand
317   \char_set_catcode_letter:n { 58 } % colon
318   \char_set_catcode_math_superscript:n { 94 } % circumflex
319   \char_set_catcode_letter:n { 95 } % underscore
320   \char_set_catcode_other:n { 124 } % pipe
321   \char_set_catcode_space:n { 126 } % tilde
322   \tex_endlinechar:D = 32 \scan_stop:
323   \bool_set_true:N \l_expl_status_bool
324 }
325 \protected \def \ExplSyntaxOff { }
326 </initex>

```

(End definition for \ExplSyntaxOn and \ExplSyntaxOff. These functions are documented on page 6.)

\l_expl_status_bool A flag to show the current syntax status.

```

327 <*initex>
328 \chardef \l_expl_status_bool = 0 ~
329 </initex>

```

(End definition for \l_expl_status_bool. This function is documented on page ??.)

```

330 </initex | package>

```

184 l3names implementation

```

331 <*initex | package>
332 <*package>
333 \ProvidesExplPackage
334   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
335 </package>

```

The code here simply renames all of the primitives to new, internal, names. In format mode, it also deletes all of the existing names (although some do come back later).

`\tex_undefined:D` This function does not exist at all, but is the name used by the plain T_EX format for an undefined function. So it should be marked here as “taken”.

(End definition for \tex_undefined:D. This function is documented on page ??.)

The `\let` primitive is renamed by hand first as it is essential for the entire process to follow. This also uses `\global`, as that way we avoid leaving an unneeded csname in the hash table.

```

336 \let \tex_global:D \global
337 \let \tex_let:D \let

```

Everything is inside a (rather long) group, which keeps `\name_primitive:NN` trapped.

```

338 \begingroup

```

`\name_primitive:NN` A temporary function to actually do the renaming. This also allows the original names to be removed in format mode.

```

339 \long \def \name_primitive:NN #1#2
340 {
341   \tex_global:D \tex_let:D #2 #1
342 <*initex>
343   \tex_global:D \tex_let:D #1 \tex_undefined:D
344 </initex>
345 }

```

(End definition for \name_primitive:NN. This function is documented on page ??.)

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.

```

346 \name_primitive:NN \tex_space:D
347 \name_primitive:NN \tex_italiccor:D
348 \name_primitive:NN \tex_hyphen:D

```

Now all the other primitives.

```

349 \name_primitive:NN \tex_let:D
350 \name_primitive:NN \tex_def:D
351 \name_primitive:NN \tex_edef:D
352 \name_primitive:NN \tex_gdef:D
353 \name_primitive:NN \tex_xdef:D
354 \name_primitive:NN \tex_chardef:D
355 \name_primitive:NN \tex_countdef:D
356 \name_primitive:NN \tex_dimendef:D
357 \name_primitive:NN \tex_skipdef:D
358 \name_primitive:NN \tex_muskipdef:D

```

359	\name_primitive:NN \mathchardef	\tex_mathchardef:D
360	\name_primitive:NN \toksdef	\tex_toksdef:D
361	\name_primitive:NN \futurelet	\tex_futurelet:D
362	\name_primitive:NN \advance	\tex_advance:D
363	\name_primitive:NN \divide	\tex_divide:D
364	\name_primitive:NN \multiply	\tex_multiply:D
365	\name_primitive:NN \font	\tex_font:D
366	\name_primitive:NN \fam	\tex_fam:D
367	\name_primitive:NN \global	\tex_global:D
368	\name_primitive:NN \long	\tex_long:D
369	\name_primitive:NN \outer	\tex_outer:D
370	\name_primitive:NN \setlanguage	\tex_setlanguage:D
371	\name_primitive:NN \globaldefs	\tex_globaldefs:D
372	\name_primitive:NN \afterassignment	\tex_afterassignment:D
373	\name_primitive:NN \aftergroup	\tex_aftergroup:D
374	\name_primitive:NN \expandafter	\tex_expandafter:D
375	\name_primitive:NN \noexpand	\tex_noexpand:D
376	\name_primitive:NN \begingroup	\tex_begingroup:D
377	\name_primitive:NN \endgroup	\tex_endgroup:D
378	\name_primitive:NN \halign	\tex_halign:D
379	\name_primitive:NN \valign	\tex_valign:D
380	\name_primitive:NN \cr	\tex_cr:D
381	\name_primitive:NN \crcr	\tex_crcr:D
382	\name_primitive:NN \noalign	\tex_noalign:D
383	\name_primitive:NN \omit	\tex_omit:D
384	\name_primitive:NN \span	\tex_span:D
385	\name_primitive:NN \tabskip	\tex_tabskip:D
386	\name_primitive:NN \everycr	\tex_everycr:D
387	\name_primitive:NN \if	\tex_if:D
388	\name_primitive:NN \ifcase	\tex_ifcase:D
389	\name_primitive:NN \ifcat	\tex_ifcat:D
390	\name_primitive:NN \ifnum	\tex_ifnum:D
391	\name_primitive:NN \ifodd	\tex_ifodd:D
392	\name_primitive:NN \ifdim	\tex_ifdim:D
393	\name_primitive:NN \ifeof	\tex_ifeof:D
394	\name_primitive:NN \ifhbox	\tex_ifhbox:D
395	\name_primitive:NN \ifvbox	\tex_ifvbox:D
396	\name_primitive:NN \ifvoid	\tex_ifvoid:D
397	\name_primitive:NN \ifx	\tex_ifx:D
398	\name_primitive:NN \iffalse	\tex_iffalse:D
399	\name_primitive:NN \iftrue	\tex_iftrue:D
400	\name_primitive:NN \ifhmode	\tex_ifhmode:D
401	\name_primitive:NN \ifmmode	\tex_ifmmode:D
402	\name_primitive:NN \ifvmode	\tex_ifvmode:D
403	\name_primitive:NN \ifinner	\tex_ifinner:D
404	\name_primitive:NN \else	\tex_else:D
405	\name_primitive:NN \fi	\tex_fi:D
406	\name_primitive:NN \or	\tex_or:D
407	\name_primitive:NN \immediate	\tex_immediate:D
408	\name_primitive:NN \closeout	\tex_closeout:D

409	\name_primitive:NN \openin	\tex_openin:D
410	\name_primitive:NN \openout	\tex_openout:D
411	\name_primitive:NN \read	\tex_read:D
412	\name_primitive:NN \write	\tex_write:D
413	\name_primitive:NN \closein	\tex_closein:D
414	\name_primitive:NN \newlinechar	\tex_newlinechar:D
415	\name_primitive:NN \input	\tex_input:D
416	\name_primitive:NN \endinput	\tex_endinput:D
417	\name_primitive:NN \inputlineno	\tex_inputlineno:D
418	\name_primitive:NN \errmessage	\tex_errmessage:D
419	\name_primitive:NN \message	\tex_message:D
420	\name_primitive:NN \show	\tex_show:D
421	\name_primitive:NN \showthe	\tex_showthe:D
422	\name_primitive:NN \showbox	\tex_showbox:D
423	\name_primitive:NN \showlists	\tex_showlists:D
424	\name_primitive:NN \errhelp	\tex_errhelp:D
425	\name_primitive:NN \errorcontextlines	\tex_errorcontextlines:D
426	\name_primitive:NN \tracingcommands	\tex_tracingcommands:D
427	\name_primitive:NN \tracinglostchars	\tex_tracinglostchars:D
428	\name_primitive:NN \tracingmacros	\tex_tracingmacros:D
429	\name_primitive:NN \tracingonline	\tex_tracingonline:D
430	\name_primitive:NN \tracingoutput	\tex_tracingoutput:D
431	\name_primitive:NN \tracingpages	\tex_tracingpages:D
432	\name_primitive:NN \tracingparagraphs	\tex_tracingparagraphs:D
433	\name_primitive:NN \tracingrestores	\tex_tracingrestores:D
434	\name_primitive:NN \tracingstats	\tex_tracingstats:D
435	\name_primitive:NN \pausing	\tex_pausing:D
436	\name_primitive:NN \showboxbreadth	\tex_showboxbreadth:D
437	\name_primitive:NN \showboxdepth	\tex_showboxdepth:D
438	\name_primitive:NN \batchmode	\tex_batchmode:D
439	\name_primitive:NN \errorstopmode	\tex_errorstopmode:D
440	\name_primitive:NN \nonstopmode	\tex_nonstopmode:D
441	\name_primitive:NN \scrollmode	\tex_scrollmode:D
442	\name_primitive:NN \end	\tex_end:D
443	\name_primitive:NN \csname	\tex_csname:D
444	\name_primitive:NN \endcsname	\tex_endcsname:D
445	\name_primitive:NN \ignorespaces	\tex_ignorespaces:D
446	\name_primitive:NN \relax	\tex_relax:D
447	\name_primitive:NN \the	\tex_the:D
448	\name_primitive:NN \mag	\tex_mag:D
449	\name_primitive:NN \language	\tex_language:D
450	\name_primitive:NN \mark	\tex_mark:D
451	\name_primitive:NN \topmark	\tex_topmark:D
452	\name_primitive:NN \firstmark	\tex_firstmark:D
453	\name_primitive:NN \botmark	\tex_botmark:D
454	\name_primitive:NN \splitfirstmark	\tex_splitfirstmark:D
455	\name_primitive:NN \splitbotmark	\tex_splitbotmark:D
456	\name_primitive:NN \fontname	\tex_fontname:D
457	\name_primitive:NN \escapechar	\tex_escapechar:D
458	\name_primitive:NN \endlinechar	\tex_endlinechar:D

459	\name_primitive:NN \mathchoice	\tex_mathchoice:D
460	\name_primitive:NN \delimiter	\tex_delimiter:D
461	\name_primitive:NN \mathaccent	\tex_mathaccent:D
462	\name_primitive:NN \mathchar	\tex_mathchar:D
463	\name_primitive:NN \mskip	\tex_mskip:D
464	\name_primitive:NN \radical	\tex_radical:D
465	\name_primitive:NN \vcenter	\tex_vcenter:D
466	\name_primitive:NN \mkern	\tex_mkern:D
467	\name_primitive:NN \above	\tex_above:D
468	\name_primitive:NN \abovewithdelims	\tex_abovewithdelims:D
469	\name_primitive:NN \atop	\tex_atop:D
470	\name_primitive:NN \atopwithdelims	\tex_atopwithdelims:D
471	\name_primitive:NN \over	\tex_over:D
472	\name_primitive:NN \overwithdelims	\tex_overwithdelims:D
473	\name_primitive:NN \displaystyle	\tex_displaystyle:D
474	\name_primitive:NN \textstyle	\tex_textstyle:D
475	\name_primitive:NN \scriptstyle	\tex_scriptstyle:D
476	\name_primitive:NN \scriptscriptstyle	\tex_scriptscriptstyle:D
477	\name_primitive:NN \nonscript	\tex_nonscript:D
478	\name_primitive:NN \eqno	\tex_eqno:D
479	\name_primitive:NN \leqno	\tex_leqno:D
480	\name_primitive:NN \abovedisplayshortskip	\tex_abovedisplayshortskip:D
481	\name_primitive:NN \abovedisplayskip	\tex_abovedisplayskip:D
482	\name_primitive:NN \belowdisplayshortskip	\tex_belowdisplayshortskip:D
483	\name_primitive:NN \belowdisplayskip	\tex_belowdisplayskip:D
484	\name_primitive:NN \displaywidowpenalty	\tex_displaywidowpenalty:D
485	\name_primitive:NN \displayindent	\tex_displayindent:D
486	\name_primitive:NN \displaywidth	\tex_displaywidth:D
487	\name_primitive:NN \everydisplay	\tex_everydisplay:D
488	\name_primitive:NN \predisplaysize	\tex_predisplaysize:D
489	\name_primitive:NN \predisplaypenalty	\tex_predisplaypenalty:D
490	\name_primitive:NN \postdisplaypenalty	\tex_postdisplaypenalty:D
491	\name_primitive:NN \mathbin	\tex_mathbin:D
492	\name_primitive:NN \mathclose	\tex_mathclose:D
493	\name_primitive:NN \mathinner	\tex_mathinner:D
494	\name_primitive:NN \mathop	\tex_mathop:D
495	\name_primitive:NN \displaylimits	\tex_displaylimits:D
496	\name_primitive:NN \limits	\tex_limits:D
497	\name_primitive:NN \nolimits	\tex_nolimits:D
498	\name_primitive:NN \mathopen	\tex_mathopen:D
499	\name_primitive:NN \mathord	\tex_mathord:D
500	\name_primitive:NN \mathpunct	\tex_mathpunct:D
501	\name_primitive:NN \mathrel	\tex_mathrel:D
502	\name_primitive:NN \overline	\tex_overline:D
503	\name_primitive:NN \underline	\tex_underline:D
504	\name_primitive:NN \left	\tex_left:D
505	\name_primitive:NN \right	\tex_right:D
506	\name_primitive:NN \binoppenalty	\tex_binoppenalty:D
507	\name_primitive:NN \relpenalty	\tex_relpenalty:D
508	\name_primitive:NN \delimitershortfall	\tex_delimitershortfall:D

509	\name_primitive:NN \delimiterfactor	\tex_delimiterfactor:D
510	\name_primitive:NN \nulldelimiterspace	\tex_nulldelimiterspace:D
511	\name_primitive:NN \everymath	\tex_everymath:D
512	\name_primitive:NN \mathsurround	\tex_mathsurround:D
513	\name_primitive:NN \medmuskip	\tex_medmuskip:D
514	\name_primitive:NN \thinmuskip	\tex_thinmuskip:D
515	\name_primitive:NN \thickmuskip	\tex_thickmuskip:D
516	\name_primitive:NN \scriptspace	\tex_scriptspace:D
517	\name_primitive:NN \noboundary	\tex_noboundary:D
518	\name_primitive:NN \accent	\tex_accent:D
519	\name_primitive:NN \char	\tex_char:D
520	\name_primitive:NN \discretionary	\tex_discretionary:D
521	\name_primitive:NN \hfil	\tex_hfil:D
522	\name_primitive:NN \hfilneg	\tex_hfilneg:D
523	\name_primitive:NN \hfill	\tex_hfill:D
524	\name_primitive:NN \hskip	\tex_hskip:D
525	\name_primitive:NN \hss	\tex_hss:D
526	\name_primitive:NN \vfil	\tex_vfil:D
527	\name_primitive:NN \vfilneg	\tex_vfilneg:D
528	\name_primitive:NN \vfill	\tex_vfill:D
529	\name_primitive:NN \vskip	\tex_vskip:D
530	\name_primitive:NN \vss	\tex_vss:D
531	\name_primitive:NN \unskip	\tex_unskip:D
532	\name_primitive:NN \kern	\tex_kern:D
533	\name_primitive:NN \unkern	\tex_unkern:D
534	\name_primitive:NN \hrule	\tex_hrule:D
535	\name_primitive:NN \vrule	\tex_vrule:D
536	\name_primitive:NN \leaders	\tex_leaders:D
537	\name_primitive:NN \cleaders	\tex_cleaders:D
538	\name_primitive:NN \xleaders	\tex_xleaders:D
539	\name_primitive:NN \lastkern	\tex_lastkern:D
540	\name_primitive:NN \lastskip	\tex_lastskip:D
541	\name_primitive:NN \indent	\tex_indent:D
542	\name_primitive:NN \par	\tex_par:D
543	\name_primitive:NN \noindent	\tex_noindent:D
544	\name_primitive:NN \vadjust	\tex_vadjust:D
545	\name_primitive:NN \baselineskip	\tex_baselineskip:D
546	\name_primitive:NN \lineskip	\tex_lineskip:D
547	\name_primitive:NN \lineskiplimit	\tex_lineskiplimit:D
548	\name_primitive:NN \clubpenalty	\tex_clubpenalty:D
549	\name_primitive:NN \widowpenalty	\tex_widowpenalty:D
550	\name_primitive:NN \exhyphenpenalty	\tex_exhyphenpenalty:D
551	\name_primitive:NN \hyphenpenalty	\tex_hyphenpenalty:D
552	\name_primitive:NN \linepenalty	\tex_linepenalty:D
553	\name_primitive:NN \doublehyphendemerits	\tex_doublehyphendemerits:D
554	\name_primitive:NN \finalhyphendemerits	\tex_finalhyphendemerits:D
555	\name_primitive:NN \adjdemerits	\tex_adjdemerits:D
556	\name_primitive:NN \hangafter	\tex_hangafter:D
557	\name_primitive:NN \hangindent	\tex_hangindent:D
558	\name_primitive:NN \parshape	\tex_parshape:D

559	\name_primitive:NN \hsize	\tex_hsize:D
560	\name_primitive:NN \lefthyphenmin	\tex_lefthyphenmin:D
561	\name_primitive:NN \righthyphenmin	\tex_righthyphenmin:D
562	\name_primitive:NN \leftskip	\tex_leftskip:D
563	\name_primitive:NN \rightskip	\tex_rightskip:D
564	\name_primitive:NN \looseness	\tex_looseness:D
565	\name_primitive:NN \parskip	\tex_parskip:D
566	\name_primitive:NN \parindent	\tex_parindent:D
567	\name_primitive:NN \uchyph	\tex_uchyph:D
568	\name_primitive:NN \emergencystretch	\tex_emergencystretch:D
569	\name_primitive:NN \pretolerance	\tex_pretolerance:D
570	\name_primitive:NN \tolerance	\tex_tolerance:D
571	\name_primitive:NN \spaceskip	\tex_spaceskip:D
572	\name_primitive:NN \xspaceskip	\tex_xspaceskip:D
573	\name_primitive:NN \parfillskip	\tex_parfillskip:D
574	\name_primitive:NN \everypar	\tex_everypar:D
575	\name_primitive:NN \prevgraf	\tex_prevgraf:D
576	\name_primitive:NN \spacefactor	\tex_spacefactor:D
577	\name_primitive:NN \shipout	\tex_shipout:D
578	\name_primitive:NN \vsize	\tex_vsize:D
579	\name_primitive:NN \interlinepenalty	\tex_interlinepenalty:D
580	\name_primitive:NN \brokenpenalty	\tex_brokenpenalty:D
581	\name_primitive:NN \topskip	\tex_topskip:D
582	\name_primitive:NN \maxdeadcycles	\tex_maxdeadcycles:D
583	\name_primitive:NN \maxdepth	\tex_maxdepth:D
584	\name_primitive:NN \output	\tex_output:D
585	\name_primitive:NN \deadcycles	\tex_deadcycles:D
586	\name_primitive:NN \pagedepth	\tex_pagedepth:D
587	\name_primitive:NN \pagestretch	\tex_pagestretch:D
588	\name_primitive:NN \pagefilstretch	\tex_pagefilstretch:D
589	\name_primitive:NN \pagefillstretch	\tex_pagefillstretch:D
590	\name_primitive:NN \pagefillllstretch	\tex_pagefillllstretch:D
591	\name_primitive:NN \pageshrink	\tex_pageshrink:D
592	\name_primitive:NN \pagegoal	\tex_pagegoal:D
593	\name_primitive:NN \pagetotal	\tex_pagetotal:D
594	\name_primitive:NN \outputpenalty	\tex_outputpenalty:D
595	\name_primitive:NN \hoffset	\tex_hoffset:D
596	\name_primitive:NN \voffset	\tex_voffset:D
597	\name_primitive:NN \insert	\tex_insert:D
598	\name_primitive:NN \holdinginserts	\tex_holdinginserts:D
599	\name_primitive:NN \floatingpenalty	\tex_floatingpenalty:D
600	\name_primitive:NN \insertpenalties	\tex_insertpenalties:D
601	\name_primitive:NN \lower	\tex_lower:D
602	\name_primitive:NN \moveleft	\tex_moveleft:D
603	\name_primitive:NN \moveright	\tex_moveright:D
604	\name_primitive:NN \raise	\tex_raise:D
605	\name_primitive:NN \copy	\tex_copy:D
606	\name_primitive:NN \lastbox	\tex_lastbox:D
607	\name_primitive:NN \vsplit	\tex_vsplit:D
608	\name_primitive:NN \unhbox	\tex_unhbox:D

609	\name_primitive:NN \unhcopy	\tex_unhcopy:D
610	\name_primitive:NN \unvbox	\tex_unvbox:D
611	\name_primitive:NN \unvcopy	\tex_unvcopy:D
612	\name_primitive:NN \setbox	\tex_setbox:D
613	\name_primitive:NN \hbox	\tex_hbox:D
614	\name_primitive:NN \vbox	\tex_vbox:D
615	\name_primitive:NN \vtop	\tex_vtop:D
616	\name_primitive:NN \prevdepth	\tex_prevdepth:D
617	\name_primitive:NN \badness	\tex_badness:D
618	\name_primitive:NN \hbadness	\tex_hbadness:D
619	\name_primitive:NN \vbadness	\tex_vbadness:D
620	\name_primitive:NN \hfuzz	\tex_hfuzz:D
621	\name_primitive:NN \vfuzz	\tex_vfuzz:D
622	\name_primitive:NN \overfullrule	\tex_overfullrule:D
623	\name_primitive:NN \boxmaxdepth	\tex_boxmaxdepth:D
624	\name_primitive:NN \splitmaxdepth	\tex_splitmaxdepth:D
625	\name_primitive:NN \splittopskip	\tex_splittopskip:D
626	\name_primitive:NN \everyhbox	\tex_everyhbox:D
627	\name_primitive:NN \everyvbox	\tex_everyvbox:D
628	\name_primitive:NN \nullfont	\tex_nullfont:D
629	\name_primitive:NN \textfont	\tex_textfont:D
630	\name_primitive:NN \scriptfont	\tex_scriptfont:D
631	\name_primitive:NN \scriptscriptfont	\tex_scriptscriptfont:D
632	\name_primitive:NN \fontdimen	\tex_fontdimen:D
633	\name_primitive:NN \hyphenchar	\tex_hyphenchar:D
634	\name_primitive:NN \skewchar	\tex_skewchar:D
635	\name_primitive:NN \defaultthyphenchar	\tex_defaultthyphenchar:D
636	\name_primitive:NN \defaultskewchar	\tex_defaultskewchar:D
637	\name_primitive:NN \number	\tex_number:D
638	\name_primitive:NN \romannumeral	\tex_romannumeral:D
639	\name_primitive:NN \string	\tex_string:D
640	\name_primitive:NN \lowercase	\tex_lowercase:D
641	\name_primitive:NN \uppercase	\tex_uppercase:D
642	\name_primitive:NN \meaning	\tex_meaning:D
643	\name_primitive:NN \penalty	\tex_penalty:D
644	\name_primitive:NN \unpenalty	\tex_unpenalty:D
645	\name_primitive:NN \lastpenalty	\tex_lastpenalty:D
646	\name_primitive:NN \special	\tex_special:D
647	\name_primitive:NN \dump	\tex_dump:D
648	\name_primitive:NN \patterns	\tex_patterns:D
649	\name_primitive:NN \hyphenation	\tex_hyphenation:D
650	\name_primitive:NN \time	\tex_time:D
651	\name_primitive:NN \day	\tex_day:D
652	\name_primitive:NN \month	\tex_month:D
653	\name_primitive:NN \year	\tex_year:D
654	\name_primitive:NN \jobname	\tex_jobname:D
655	\name_primitive:NN \everyjob	\tex_everyjob:D
656	\name_primitive:NN \count	\tex_count:D
657	\name_primitive:NN \dimen	\tex_dimen:D
658	\name_primitive:NN \skip	\tex_skip:D

659	\name_primitive:NN \toks	\tex_toks:D
660	\name_primitive:NN \muskip	\tex_muskip:D
661	\name_primitive:NN \box	\tex_box:D
662	\name_primitive:NN \wd	\tex_wd:D
663	\name_primitive:NN \ht	\tex_ht:D
664	\name_primitive:NN \dp	\tex_dp:D
665	\name_primitive:NN \catcode	\tex_catcode:D
666	\name_primitive:NN \delcode	\tex_delcode:D
667	\name_primitive:NN \sfcode	\tex_sfcode:D
668	\name_primitive:NN \lccode	\tex_lccode:D
669	\name_primitive:NN \uccode	\tex_uccode:D
670	\name_primitive:NN \mathcode	\tex_mathcode:D

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

671	\name_primitive:NN \ifdefined	\etex_ifdefined:D
672	\name_primitive:NN \ifcsname	\etex_ifcsname:D
673	\name_primitive:NN \unless	\etex_unless:D
674	\name_primitive:NN \eTeXversion	\etex_eTeXversion:D
675	\name_primitive:NN \eTeXrevision	\etex_eTeXrevision:D
676	\name_primitive:NN \marks	\etex_marks:D
677	\name_primitive:NN \topmarks	\etex_topmarks:D
678	\name_primitive:NN \firstmarks	\etex_firstmarks:D
679	\name_primitive:NN \botmarks	\etex_botmarks:D
680	\name_primitive:NN \splitfirstmarks	\etex_splitfirstmarks:D
681	\name_primitive:NN \splitbotmarks	\etex_splitbotmarks:D
682	\name_primitive:NN \unexpanded	\etex_unexpanded:D
683	\name_primitive:NN \detokenize	\etex_detokenize:D
684	\name_primitive:NN \scantokens	\etex_scantokens:D
685	\name_primitive:NN \showtokens	\etex_showtokens:D
686	\name_primitive:NN \readline	\etex_readline:D
687	\name_primitive:NN \tracingassigns	\etex_tracingassigns:D
688	\name_primitive:NN \tracingscantokens	\etex_tracingscantokens:D
689	\name_primitive:NN \tracingnesting	\etex_tracingnesting:D
690	\name_primitive:NN \tracingifs	\etex_tracingifs:D
691	\name_primitive:NN \currentiflevel	\etex_currentiflevel:D
692	\name_primitive:NN \currentifbranch	\etex_currentifbranch:D
693	\name_primitive:NN \currentifttype	\etex_currentifttype:D
694	\name_primitive:NN \tracinggroups	\etex_tracinggroups:D
695	\name_primitive:NN \currentgrouplevel	\etex_currentgrouplevel:D
696	\name_primitive:NN \currentgrouptype	\etex_currentgrouptype:D
697	\name_primitive:NN \showgroups	\etex_showgroups:D
698	\name_primitive:NN \showifs	\etex_showifs:D
699	\name_primitive:NN \interactionmode	\etex_interactionmode:D
700	\name_primitive:NN \lastnodetype	\etex_lastnodetype:D
701	\name_primitive:NN \iffontchar	\etex_iffontchar:D
702	\name_primitive:NN \fontcharht	\etex_fontcharht:D
703	\name_primitive:NN \fontchardp	\etex_fontchardp:D
704	\name_primitive:NN \fontcharwd	\etex_fontcharwd:D
705	\name_primitive:NN \fontcharic	\etex_fontcharic:D

706	\name_primitive:NN \parshapeindent	\etex_parshapeindent:D
707	\name_primitive:NN \parshapelength	\etex_parshapelength:D
708	\name_primitive:NN \parshapedimen	\etex_parshapedimen:D
709	\name_primitive:NN \numexpr	\etex_numexpr:D
710	\name_primitive:NN \dimexpr	\etex_dimexpr:D
711	\name_primitive:NN \glueexpr	\etex_glueexpr:D
712	\name_primitive:NN \muexpr	\etex_muexpr:D
713	\name_primitive:NN \gluestretch	\etex_gluestretch:D
714	\name_primitive:NN \glueshrink	\etex_glueshrink:D
715	\name_primitive:NN \gluestretchorder	\etex_gluestretchorder:D
716	\name_primitive:NN \glueshrinkorder	\etex_glueshrinkorder:D
717	\name_primitive:NN \gluetomu	\etex_gluetomu:D
718	\name_primitive:NN \mutoglu	\etex_mutoglu:D
719	\name_primitive:NN \lastlinefit	\etex_lastlinefit:D
720	\name_primitive:NN \interlinepenalties	\etex_interlinepenalties:D
721	\name_primitive:NN \clubpenalties	\etex_clubpenalties:D
722	\name_primitive:NN \widowpenalties	\etex_widowpenalties:D
723	\name_primitive:NN \displaywidowpenalties	\etex_displaywidowpenalties:D
724	\name_primitive:NN \middle	\etex_middle:D
725	\name_primitive:NN \savinghyphcodes	\etex_savinghyphcodes:D
726	\name_primitive:NN \savingvdiscards	\etex_savingvdiscards:D
727	\name_primitive:NN \pagediscards	\etex_pagediscards:D
728	\name_primitive:NN \splitdiscards	\etex_splitdiscards:D
729	\name_primitive:NN \TeXstate	\etex_TeXstate:D
730	\name_primitive:NN \beginL	\etex_beginL:D
731	\name_primitive:NN \endL	\etex_endL:D
732	\name_primitive:NN \beginR	\etex_beginR:D
733	\name_primitive:NN \endR	\etex_endR:D
734	\name_primitive:NN \predisplaydirection	\etex_predisplaydirection:D
735	\name_primitive:NN \everyeof	\etex_everyeof:D
736	\name_primitive:NN \protected	\etex_protected:D

The newer primitives are more complex: there are an awful lot of them, and we don't use them all at the moment. So the following is selective. In the case of the pdfTeX primitives, we retain pdf at the start of the names *only* for directly PDF-related primitives, as there are a lot of pdfTeX primitives that start \pdf... but are not related to PDF output. These ones related to PDF output.

737	\name_primitive:NN \pdfcreationdate	\pdfTEX_pdfcreationdate:D
738	\name_primitive:NN \pdfcolorstack	\pdfTEX_pdfcolorstack:D
739	\name_primitive:NN \pdfcompresslevel	\pdfTEX_pdfcompresslevel:D
740	\name_primitive:NN \pdfdecimaldigits	\pdfTEX_pdfdecimaldigits:D
741	\name_primitive:NN \pdfhorigin	\pdfTEX_pdfhorigin:D
742	\name_primitive:NN \pdfinfo	\pdfTEX_pdfinfo:D
743	\name_primitive:NN \pdflastxform	\pdfTEX_pdflastxform:D
744	\name_primitive:NN \pdfliteral	\pdfTEX_pdfliteral:D
745	\name_primitive:NN \pdfminorversion	\pdfTEX_pdfminorversion:D
746	\name_primitive:NN \pdfobjcompresslevel	\pdfTEX_pdfobjcompresslevel:D
747	\name_primitive:NN \pdfoutput	\pdfTEX_pdfoutput:D
748	\name_primitive:NN \pdfrefxform	\pdfTEX_pdfrefxform:D
749	\name_primitive:NN \pdfrestore	\pdfTEX_pdfrestore:D

```

750 \name_primitive:NN \pdfsave \pdfTeX_pdfsave:D
751 \name_primitive:NN \pdfsetmatrix \pdfTeX_pdfsetmatrix:D
752 \name_primitive:NN \pdfpkresolution \pdfTeX_pdfpkresolution:D
753 \name_primitive:NN \pdfTeXrevision \pdfTeX_pdfTeXrevision:D
754 \name_primitive:NN \pdfvorigin \pdfTeX_pdfvorigin:D
755 \name_primitive:NN \pdfxform \pdfTeX_pdfxform:D

```

While these are not.

```

756 \name_primitive:NN \pdfstrcmp \pdfTeX_strcmp:D

```

X_YTeX-specific primitives. Note that X_YTeX’s \strcmp is handled earlier and is “rolled up” into \pdfstrcmp.

```

757 \name_primitive:NN \XeTeXversion \xetex_XeTeXversion:D

```

Primitives from LuaTeX.

```

758 \name_primitive:NN \catcodetable \luaTeX_catcodetable:D
759 \name_primitive:NN \directlua \luaTeX_directlua:D
760 \name_primitive:NN \initcatcodetable \luaTeX_initcatcodetable:D
761 \name_primitive:NN \lattelua \luaTeX_lattelua:D
762 \name_primitive:NN \luaTeXversion \luaTeX_luaTeXversion:D
763 \name_primitive:NN \savecatcodetable \luaTeX_savecatcodetable:D

```

The job is done: close the group (using the primitive renamed!).

```

764 \tex_endgroup:D

```

L^AT_EX 2_ε will have moved a few primitives, so these are sorted out.

```

765 <*package>
766 \tex_let:D \tex_end:D @@end
767 \tex_let:D \tex_everydisplay:D \frozen@everydisplay
768 \tex_let:D \tex_everymath:D \frozen@everymath
769 \tex_let:D \tex_hyphen:D @@hyph
770 \tex_let:D \tex_input:D @@input
771 \tex_let:D \tex_italic_correction:D @@italiccorr
772 \tex_let:D \tex_underline:D @@underline

```

That is also true for the luatex package for L^AT_EX 2_ε.

```

773 \tex_let:D \luaTeX_catcodetable:D \luaTeXcatcodetable
774 \tex_let:D \luaTeX_initcatcodetable:D \luaTeXinitcatcodetable
775 \tex_let:D \luaTeX_lattelua:D \luaTeXlattelua
776 \tex_let:D \luaTeX_savecatcodetable:D \luaTeXsavecatcodetable
777 </package>
778 </initex | package>

```

185 l3basics implementation

```

779 <*initex | package>
780 <*package>
781 \ProvidesExplPackage
782   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
783 \package_check_loaded_expl:

```

784 `\package`

185.1 Renaming some T_EX primitives (again)

Having given all the T_EX 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.²

```

\if_true: Then some conditionals.
\if_false: 785 \tex_let:D \if_true:      \tex_iftrue:D
      \or: 786 \tex_let:D \if_false:    \tex_iffalse:D
      \else: 787 \tex_let:D \or:        \tex_or:D
      \fi: 788 \tex_let:D \else:      \tex_else:D
\reverse_if:N 789 \tex_let:D \fi:      \tex_fi:D
      \if:w 790 \tex_let:D \reverse_if:N \etex_unless:D
\if_charcode:w 791 \tex_let:D \if:w      \tex_if:D
\if_catcode:w 792 \tex_let:D \if_charcode:w \tex_if:D
\if_meaning:w 793 \tex_let:D \if_catcode:w \tex_ifcat:D
      794 \tex_let:D \if_meaning:w \tex_ifx:D

```

(End definition for \if_true:. This function is documented on page 22.)

```

\if_mode_math: TEX lets us detect some if its modes.
\if_mode_horizontal: 795 \tex_let:D \if_mode_math:      \tex_ifmmode:D
\if_mode_vertical: 796 \tex_let:D \if_mode_horizontal: \tex_ifhmode:D
\if_mode_inner: 797 \tex_let:D \if_mode_vertical:      \tex_ifvmode:D
      798 \tex_let:D \if_mode_inner:      \tex_ifinner:D

```

(End definition for \if_mode_math:. This function is documented on page ??.)

```

\if_cs_exist:N
\if_cs_exist:w 799 \tex_let:D \if_cs_exist:N      \etex_ifdefined:D
      800 \tex_let:D \if_cs_exist:w      \etex_ifcurname:D

```

(End definition for \if_cs_exist:N. This function is documented on page ??.)

```

\exp_after:wN The three \exp_ functions are used in the l3expan module where they are described.
\exp_not:N 801 \tex_let:D \exp_after:wN      \tex_expandafter:D
\exp_not:n 802 \tex_let:D \exp_not:N      \tex_noexpand:D
      803 \tex_let:D \exp_not:n      \etex_unexpanded:D

```

(End definition for \exp_after:wN. This function is documented on page 30.)

```

\token_to_meaning:N
\token_to_str:N 804 \tex_let:D \token_to_meaning:N \tex_meaning:D
      \cs:w 805 \tex_let:D \token_to_str:N      \tex_string:D
      \cs_end: 806 \tex_let:D \cs:w      \tex_csname:D
\cs_meaning:N 807 \tex_let:D \cs_end:      \tex_endcsname:D
\cs_show:N 808 \tex_let:D \cs_meaning:N      \tex_meaning:D
      809 \tex_let:D \cs_show:N      \tex_show:D

```

²This 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.

(End definition for `\token_to_meaning:N`. This function is documented on page 15.)

`\scan_stop:` The next three are basic functions for which there also exist versions that are safe inside alignments. These safe versions are defined in the `l3prg` module.

```
\group_begin:
\group_end:
810 \tex_let:D \scan_stop:      \tex_relax:D
811 \tex_let:D \group_begin:   \tex_begingroup:D
812 \tex_let:D \group_end:     \tex_endgroup:D
```

(End definition for `\scan_stop:`. This function is documented on page ??.)

`\if_int_compare:w`
`\int_to_roman:w`

```
813 \tex_let:D \if_int_compare:w \tex_ifnum:D
814 \tex_let:D \int_to_roman:w   \tex_romannumeral:D
```

(End definition for `\if_int_compare:w`. This function is documented on page 69.)

`\group_insert_after:N`

```
815 \tex_let:D \group_insert_after:N \tex_aftergroup:D
```

(End definition for `\group_insert_after:N`. This function is documented on page 9.)

`\tex_global:D`
`\tex_long:D`
`\tex_protected:D`

```
816 \tex_let:D \tex_global:D      \tex_global:D
817 \tex_let:D \tex_long:D       \tex_long:D
818 \tex_let:D \tex_protected:D   \etex_protected:D
```

(End definition for `\tex_global:D`. This function is documented on page ??.)

`\exp_args:Nc` Discussed in `l3expan`, but needed much earlier.

```
819 \tex_long:D \tex_def:D \exp_args:Nc #1#2 { \exp_after:wN #1 \cs:w #2 \cs_end: }
```

(End definition for `\exp_args:Nc`. This function is documented on page 26.)

`\token_to_str:c` A small number of variants by hand. Some of the necessary functions (`\use_i:nn`, `\use_ii:nn`, and `\exp_args:NNc`) are not defined at that point yet, but will be defined before those variants are used. The `\cs_meaning:c` command must check for an undefined control sequence to avoid defining it mistakenly. The `\cs_show:c` command is “protected” because its action is not expandable. Also, the conversion of its argument to a control sequence is done within a group to avoid converting it to `\relax`.

```
820 \tex_def:D \token_to_str:c { \exp_args:Nc \token_to_str:N }
821 \tex_long:D \tex_def:D \cs_meaning:c #1
822 {
823   \if_cs_exist:w #1 \cs_end:
824     \exp_after:wN \use_i:nn
825   \else:
826     \exp_after:wN \use_ii:nn
827   \fi:
828   { \exp_args:Nc \cs_meaning:N {#1} }
829   { \tl_to_str:n {undefined} }
830 }
831 \tex_protected:D \tex_def:D \cs_show:c
832 { \group_begin: \exp_args:NNc \group_end: \cs_show:N }
```

(End definition for `\token_to_str:c`. This function is documented on page ??.)

185.2 Defining some constants

`\c_minus_one` We need the constants `\c_minus_one` and `\c_sixteen` now for writing information to the log and the terminal and `\c_zero` which is used by some functions in the `l3alloc` module.
`\c_zero` log and the terminal and `\c_zero` which is used by some functions in the `l3alloc` module.
`\c_sixteen` The rest are defined in the `l3int` module – at least for the ones that can be defined
`\c_six` with `\tex_chardef:D` or `\tex_mathchardef:D`. For other constants the `l3int` module is
`\c_seven` required but it can't be used until the allocation has been set up properly! The actual
`\c_twelve` 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`.

```

833 <*package>
834 \tex_let:D \c_minus_one \m@ne
835 </package>
836 <*initex>
837 \tex_countdef:D \c_minus_one = 10 ~
838 \c_minus_one = -1 ~
839 </initex>
840 \tex_chardef:D \c_sixteen = 16~
841 \tex_chardef:D \c_zero = 0~
842 \tex_chardef:D \c_six = 6~
843 \tex_chardef:D \c_seven = 7~
844 \tex_chardef:D \c_twelve = 12~

```

(End definition for `\c_minus_one`, `\c_zero`, and `\c_sixteen`. These functions are documented on page 68.)

`\c_max_register_int` This is here as this particular integer is needed both in package mode and to bootstrap `l3alloc`.

```

845 \etex_ifdefined:D \luatex luatexversion:D
846 \tex_chardef:D \c_max_register_int = 65 535 ~
847 \tex_else:D
848 \tex_mathchardef:D \c_max_register_int = 32 767 ~
849 \tex_fi:D

```

(End definition for `\c_max_register_int`. This function is documented on page 68.)

185.3 Defining functions

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

`\cs_set_nopar:Npn` All assignment functions in $\text{\LaTeX}3$ should be naturally robust; after all, the \TeX primitives for assignments are and it can be a cause of problems if others aren't.
`\cs_set_nopar:Npx`
`\cs_set:Npn`
`\cs_set:Npx`
`\cs_set_protected_nopar:Npn`
`\cs_set_protected_nopar:Npx`
`\cs_set_protected:Npn`
`\cs_set_protected:Npx`

```

850 \tex_let:D \cs_set_nopar:Npn \tex_def:D
851 \tex_let:D \cs_set_nopar:Npx \tex_edef:D
852 \tex_protected:D \cs_set_nopar:Npn \cs_set:Npn
853 { \tex_long:D \cs_set_nopar:Npn }
854 \tex_protected:D \cs_set_nopar:Npn \cs_set:Npx
855 { \tex_long:D \cs_set_nopar:Npx }
856 \tex_protected:D \cs_set_nopar:Npn \cs_set_protected_nopar:Npn
857 { \tex_protected:D \cs_set_nopar:Npn }
858 \tex_protected:D \cs_set_nopar:Npn \cs_set_protected_nopar:Npx

```

```

859 { \tex_protected:D \cs_set_nopar:Npx }
860 \cs_set_protected_nopar:Npn \cs_set_protected:Npn
861 { \tex_protected:D \tex_long:D \cs_set_nopar:Npn }
862 \cs_set_protected_nopar:Npn \cs_set_protected:Npx
863 { \tex_protected:D \tex_long:D \cs_set_nopar:Npx }
      (End definition for \cs_set_nopar:Npn. This function is documented on page ??.)

```

```

\cs_gset_nopar:Npn Global versions of the above functions.
\cs_gset_nopar:Npx
  \cs_gset:Npn
  \cs_gset:Npx
\cs_gset_protected_nopar:Npn
\cs_gset_protected_nopar:Npx
  \cs_gset_protected:Npn
  \cs_gset_protected:Npx
864 \tex_let:D \cs_gset_nopar:Npn          \tex_gdef:D
865 \tex_let:D \cs_gset_nopar:Npx          \tex_xdef:D
866 \cs_set_protected_nopar:Npn \cs_gset:Npn
867 { \tex_long:D \cs_gset_nopar:Npn }
868 \cs_set_protected_nopar:Npn \cs_gset:Npx
869 { \tex_long:D \cs_gset_nopar:Npx }
870 \cs_set_protected_nopar:Npn \cs_gset_protected_nopar:Npn
871 { \tex_protected:D \cs_gset_nopar:Npn }
872 \cs_set_protected_nopar:Npn \cs_gset_protected_nopar:Npx
873 { \tex_protected:D \cs_gset_nopar:Npx }
874 \cs_set_protected_nopar:Npn \cs_gset_protected:Npn
875 { \tex_protected:D \tex_long:D \cs_gset_nopar:Npn }
876 \cs_set_protected_nopar:Npn \cs_gset_protected:Npx
877 { \tex_protected:D \tex_long:D \cs_gset_nopar:Npx }
      (End definition for \cs_gset_nopar:Npn. This function is documented on page ??.)

```

185.4 Selecting tokens

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

```

878 \cs_set:Npn \use:c #1 { \cs:w #1 \cs_end: }
      (End definition for \use:c. This function is documented on page 16.)

```

\use:x Fully expands its argument and passes it to the input stream. Uses **\cs_tmp:w** as a scratch register but does not affect it.

```

879 \cs_set_protected:Npn \use:x #1
880 {
881   \group_begin:
882     \cs_set_nopar:Npx \cs_tmp:w {#1}
883     \exp_after:wN
884     \group_end:
885     \cs_tmp:w
886 }
887 \cs_set:Npn \cs_tmp:w { }

```

\use:n These macro grabs its arguments and returns it back to the input (with outer braces removed).

```

\use:nnn
\use:nnnn
888 \cs_set:Npn \use:n #1 {#1}
889 \cs_set:Npn \use:nn #1#2 {#1#2}
890 \cs_set:Npn \use:nnn #1#2#3 {#1#2#3}
891 \cs_set:Npn \use:nnnn #1#2#3#4 {#1#2#3#4}

```

`\use_i:nn` The equivalent to L^AT_EX 2_ε's `\@firstoftwo` and `\@secondoftwo`.

`\use_ii:nn` 892 `\cs_set:Npn \use_i:nn #1#2 {#1}`
 893 `\cs_set:Npn \use_ii:nn #1#2 {#2}`

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

`\use_ii:nnn` 894 `\cs_set:Npn \use_i:nnn #1#2#3 {#1}`
`\use_iii:nnn` 895 `\cs_set:Npn \use_ii:nnn #1#2#3 {#2}`
`\use_i_ii:nnn` 896 `\cs_set:Npn \use_iii:nnn #1#2#3 {#3}`
`\use_i:nnnn` 897 `\cs_set:Npn \use_i_ii:nnn #1#2#3 {#1#2}`
`\use_ii:nnnn` 898 `\cs_set:Npn \use_i:nnnn #1#2#3#4 {#1}`
`\use_iii:nnnn` 899 `\cs_set:Npn \use_ii:nnnn #1#2#3#4 {#2}`
`\use_iv:nnnn` 900 `\cs_set:Npn \use_iii:nnnn #1#2#3#4 {#3}`
 901 `\cs_set:Npn \use_iv:nnnn #1#2#3#4 {#4}`

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

`\use_none_delimit_by_q_stop:w` 902 `\cs_set:Npn \use_none_delimit_by_q_nil:w #1 \q_nil { }`
`\use_none_delimit_by_q_recursion_stop:w` 903 `\cs_set:Npn \use_none_delimit_by_q_stop:w #1 \q_stop { }`
 904 `\cs_set:Npn \use_none_delimit_by_q_recursion_stop:w #1 \q_recursion_stop { }`

`\use_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.

905 `\cs_set:Npn \use_i_delimit_by_q_nil:nw #1#2 \q_nil {#1}`
 906 `\cs_set:Npn \use_i_delimit_by_q_stop:nw #1#2 \q_stop {#1}`
 907 `\cs_set:Npn \use_i_delimit_by_q_recursion_stop:nw #1#2 \q_recursion_stop {#1}`

185.5 Gobbling tokens from input

`\use_none:n` To gobble tokens from the input we use a standard naming convention: the number of tokens gobbled is given by the number of `n`'s following the `:` in the name. Although defining `\use_none:nnn` and above as separate calls of `\use_none:n` and `\use_none:nn` is slightly faster, this is very non-intuitive to the programmer who will assume that expanding such a function once will take care of gobbling all the tokens in one go.

`\use_none:nnnn` 908 `\cs_set:Npn \use_none:n #1 { }`
`\use_none:nnnnnn` 909 `\cs_set:Npn \use_none:nn #1#2 { }`
`\use_none:nnnnnnnn` 910 `\cs_set:Npn \use_none:nnn #1#2#3 { }`
`\use_none:nnnnnnnnn` 911 `\cs_set:Npn \use_none:nnnn #1#2#3#4 { }`
 912 `\cs_set:Npn \use_none:nnnnnn #1#2#3#4#5 { }`
 913 `\cs_set:Npn \use_none:nnnnnnn #1#2#3#4#5#6 { }`
 914 `\cs_set:Npn \use_none:nnnnnnnn #1#2#3#4#5#6#7 { }`
 915 `\cs_set:Npn \use_none:nnnnnnnnn #1#2#3#4#5#6#7#8 { }`
 916 `\cs_set:Npn \use_none:nnnnnnnnn #1#2#3#4#5#6#7#8#9 { }`

185.6 Conditional processing and definitions

Underneath any predicate function (`_p`) or other conditional forms (TF, etc.) is a built-in logic saying that it after all of the testing and processing must return the *state* this leaves `TeX` in. Therefore, a simple user interface could be something like

```
\if_meaning:w #1#2 \prg_return_true: \else:
  \if_meaning:w #1#3 \prg_return_true: \else:
    \prg_return_false:
\fi: \fi:
```

Usually, a `TeX` programmer would have to insert a number of `\exp_after:wN`s to ensure the state value is returned at exactly the point where the last conditional is finished. However, that obscures the code and forces the `TeX` programmer to prove that he/she knows the $2^n - 1$ table. We therefore provide the simpler interface.

`\prg_return_true:` The idea here is that `\int_to_roman:w` will expand fully any `\else:` and the `\fi:` that
`\prg_return_false:` are waiting to be discarded, before reaching the `\c_zero` which will leave the expansion null. The code can then leave either the first or second argument in the input stream. This means that all of the branching code has to contain at least two tokens: see how the logical tests are actually implemented to see this.

```
917 \cs_set_nopar:Npn \prg_return_true:
918 { \exp_after:wN \use_i:nn \int_to_roman:w }
919 \cs_set_nopar:Npn \prg_return_false:
920 { \exp_after:wN \use_ii:nn \int_to_roman:w }
```

An extended state space could be implemented by including a more elaborate function in place of `\use_i:nn`/`\use_ii:nn`. Provided two arguments are absorbed then the code will work.

`\prg_set_conditional:Npnn` The user functions for the types using parameter text from the programmer. Call aux
`\prg_new_conditional:Npnn` function to grab parameters, split the base function into name and signature and then
`\prg_set_protected_conditional:Npnn` use, e.g., `\cs_set:Npn` to define it with.

```
\prg_new_protected_conditional:Npnn
921 \cs_set_protected:Npn \prg_set_conditional:Npnn #1
922 {
923   \prg_get_parm_aux:nw
924   {
925     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
926     \cs_set:Npn { parm }
927   }
928 }
929 \cs_set_protected:Npn \prg_new_conditional:Npnn #1
930 {
931   \prg_get_parm_aux:nw
932   {
933     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
934     \cs_new:Npn { parm }
935   }
936 }
```

```

937 \cs_set_protected:Npn \prg_set_protected_conditional:Npnn #1
938 {
939   \prg_get_parm_aux:nw{
940     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
941     \cs_set_protected:Npn { parm }
942   }
943 }
944 \cs_set_protected:Npn \prg_new_protected_conditional:Npnn #1
945 {
946   \prg_get_parm_aux:nw
947   {
948     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
949     \cs_new_protected:Npn { parm }
950   }
951 }

```

\prg_set_conditional:Nnn
\prg_new_conditional:Nnn
\prg_set_protected_conditional:Nnn
\prg_new_protected_conditional:Nnn

The user functions for the types automatically inserting the correct parameter text based on the signature. Call aux function after calculating number of arguments, split the base function into name and signature and then use, *e.g.*, \cs_set:Npn to define it with.

```

952 \cs_set_protected:Npn \prg_set_conditional:Nnn #1
953 {
954   \exp_args:Nnf \prg_get_count_aux:nn
955   {
956     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
957     \cs_set:Npn { count }
958   }
959   { \cs_get_arg_count_from_signature:N #1 }
960 }
961 \cs_set_protected:Npn \prg_new_conditional:Nnn #1
962 {
963   \exp_args:Nnf \prg_get_count_aux:nn
964   {
965     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
966     \cs_new:Npn { count }
967   }
968   { \cs_get_arg_count_from_signature:N #1 }
969 }
970
971 \cs_set_protected:Npn \prg_set_protected_conditional:Nnn #1{
972   \exp_args:Nnf \prg_get_count_aux:nn{
973     \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
974     \cs_set_protected:Npn {count}
975   }{\cs_get_arg_count_from_signature:N #1}
976 }
977
978 \cs_set_protected:Npn \prg_new_protected_conditional:Nnn #1
979 {
980   \exp_args:Nnf \prg_get_count_aux:nn
981   {

```

```

982         \cs_split_function:NN #1 \prg_generate_conditional_aux:nnNNnnnn
983         \cs_new_protected:Npn {count}
984     }
985     { \cs_get_arg_count_from_signature:N #1 }
986 }

```

\prg_set_eq_conditional:NNn The obvious setting-equal functions.

```

\prg_new_eq_conditional:NNn
987 \cs_set_protected:Npn \prg_set_eq_conditional:NNn #1#2#3
988 { \prg_set_eq_conditional_aux:NNNn \cs_set_eq:cc #1#2 {#3} }
989 \cs_set_protected:Npn \prg_new_eq_conditional:NNn #1#2#3
990 { \prg_set_eq_conditional_aux:NNNn \cs_new_eq:cc #1#2 {#3} }

```

\prg_get_parm_aux:nw For the Npnn type we must grab the parameter text before continuing. We make this a very generic function that takes one argument before reading everything up to a left brace. Something similar for the Nnn type.

```

991 \cs_set:Npn \prg_get_count_aux:nn #1#2 { #1 {#2} }
992 \cs_set:Npn \prg_get_parm_aux:nw #1#2# { #1 {#2} }

```

\prg_generate_conditional_parm_aux:nnNNnnnn The workhorse here is going through a list of desired forms, *i.e.*, p, TF, T and F. The first three arguments come from splitting up the base form of the conditional, which gives the name, signature and a boolean to signal whether or not there was a colon in the name. For the time being, we do not use this piece of information but could well throw an error. The fourth argument is how to define this function, the fifth is the text **parm** or **count** for which version to use to define the functions, the sixth is the parameters to use (possibly empty) or number of arguments, the seventh is the list of forms to define, the eighth is the replacement text which we will augment when defining the forms.

```

993 \cs_set_protected:Npn \prg_generate_conditional_aux:nnNNnnnn #1#2#3#4#5#6#7#8
994 {
995     \prg_generate_conditional_aux:nnw {#5}
996     {
997         #4 {#1} {#2} {#6} {#8}
998     }
999     #7 , ? , \q_recursion_stop
1000 }

```

Looping through the list of desired forms. First is the text **parm** or **count**, second is five arguments packed together and third is the form. Use text and form to call the correct type.

```

1001 \cs_set_protected:Npn \prg_generate_conditional_aux:nnw #1#2#3 ,
1002 {
1003     \if:w ?#3
1004         \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
1005     \fi:
1006     \use:c { prg_generate_#3_form_#1:Nnnnn } #2
1007     \prg_generate_conditional_aux:nnw {#1} {#2}
1008 }

```

`\prg_generate_p_form_parm:Nnnnn` How to generate the various forms. The `parm` types here takes the following arguments:
`\prg_generate_TF_form_parm:Nnnnn` 1: how to define (an N-type), 2: name, 3: signature, 4: parameter text (or empty), 5:
`\prg_generate_T_form_parm:Nnnnn` replacement. Remember that the logic-returning functions expect two arguments to be
`\prg_generate_F_form_parm:Nnnnn` present after `\c_zero`: notice the construction of the different variants relies on this, and
that the TF variant will be slightly faster than the T version.

```

1009 \cs_set_protected:Npn \prg_generate_p_form_parm:Nnnnn #1#2#3#4#5
1010 {
1011   \exp_args:Nc #1 { #2 _p: #3 } #4
1012   {
1013     #5 \c_zero
1014     \c_true_bool \c_false_bool
1015   }
1016 }
1017 \cs_set_protected:Npn \prg_generate_T_form_parm:Nnnnn #1#2#3#4#5
1018 {
1019   \exp_args:Nc #1 { #2 : #3 T } #4
1020   {
1021     #5 \c_zero
1022     \use:n \use_none:n
1023   }
1024 }
1025 \cs_set_protected:Npn \prg_generate_F_form_parm:Nnnnn #1#2#3#4#5
1026 {
1027   \exp_args:Nc #1 { #2 : #3 F } #4
1028   {
1029     #5 \c_zero
1030     { }
1031   }
1032 }
1033 \cs_set_protected:Npn \prg_generate_TF_form_parm:Nnnnn #1#2#3#4#5
1034 {
1035   \exp_args:Nc #1 { #2 : #3 TF } #4
1036   { #5 \c_zero }
1037 }

```

`\prg_generate_p_form_count:Nnnnn` The `count` form is similar, but of course requires a number rather than a primitive
`\prg_generate_TF_form_count:Nnnnn` argument specification.

```

1038 \cs_set_protected:Npn \prg_generate_p_form_count:Nnnnn #1#2#3#4#5
1039 {
1040   \cs_generate_from_arg_count:cNnn { #2 _p: #3 } #1 {#4}
1041   {
1042     #5 \c_zero
1043     \c_true_bool \c_false_bool
1044   }
1045 }
1046 \cs_set_protected:Npn \prg_generate_T_form_count:Nnnnn #1#2#3#4#5
1047 {
1048   \cs_generate_from_arg_count:cNnn { #2 : #3 T } #1 {#4}
1049   {

```

```

1050         #5 \c_zero
1051         \use:n \use_none:n
1052     }
1053 }
1054 \cs_set_protected:Npn \prg_generate_F_form_count:Nnnnn #1#2#3#4#5
1055 {
1056     \cs_generate_from_arg_count:cNnn { #2 : #3 F } #1 {#4}
1057     {
1058         #5 \c_zero
1059         { }
1060     }
1061 }
1062 \cs_set_protected:Npn \prg_generate_TF_form_count:Nnnnn #1#2#3#4#5
1063 {
1064     \cs_generate_from_arg_count:cNnn { #2 : #3 TF } #1 {#4}
1065     { #5 \c_zero }
1066 }

```

\prg_set_eq_conditional_aux:NNNn

\prg_set_eq_conditional_aux:NNNw

```

1067 \cs_set_protected:Npn \prg_set_eq_conditional_aux:NNNn #1#2#3#4
1068 { \prg_set_eq_conditional_aux:NNNw #1#2#3#4 , ? , \q_recursion_stop }

```

Manual clist loop over argument #4.

```

1069 \cs_set_protected:Npn \prg_set_eq_conditional_aux:NNNw #1#2#3#4 ,
1070 {
1071     \if:w ? #4 \scan_stop:
1072     \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
1073     \fi:
1074     #1
1075     { \exp_args:NNc \cs_split_function:NN #2 { prg_conditional_form_#4:nnn } }
1076     { \exp_args:NNc \cs_split_function:NN #3 { prg_conditional_form_#4:nnn } }
1077     \prg_set_eq_conditional_aux:NNNw #1 {#2} {#3}
1078 }
1079 \cs_set:Npn \prg_conditional_form_p:nnn #1#2#3 { #1 _p : #2 }
1080 \cs_set:Npn \prg_conditional_form_TF:nnn #1#2#3 { #1 : #2 TF }
1081 \cs_set:Npn \prg_conditional_form_T:nnn #1#2#3 { #1 : #2 T }
1082 \cs_set:Npn \prg_conditional_form_F:nnn #1#2#3 { #1 : #2 F }

```

All that is left is to define the canonical boolean true and false. 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 could be used in logical operations. Later again they were changed to being numerical constants with values of 1 for true and 0 for false. We need this from the get-go.

\c_true_bool Here are the canonical boolean values.

\c_false_bool

```

1083 \tex_chardef:D \c_true_bool = 1~
1084 \tex_chardef:D \c_false_bool = 0~

```

185.7 Dissecting a control sequence

`\cs_to_str:N` This converts a control sequence into the character string of its name, removing the leading escape character. This turns out to be a non-trivial matter as there are different cases:

- The usual case of a printable escape character;
- the case of a non-printable escape character, e.g., when the value of `\tex_escapechar:D` is negative;
- when the escape character is a space.

One approach to solve this is to test how many tokens result from `\token_to_str:N \a`. If there are two tokens, then the escape character is printable, while if it is non-printable then only one is present.

However, there is an additional complication: the control sequence itself may start with a space. Clearly that should *not* be lost in the process of converting to a string. So the approach adopted is a little more intricate still. When the escape character is printable, `\token_to_str:N__` yields the escape character itself and a space. The character codes are different, thus the `\if:w` test is false, and TeX reads `\cs_to_str_aux:N` after turning the following control sequence into a string; this auxiliary removes the escape character, and stops the expansion of the initial `\int_to_roman:w`. The second case is that the escape character is not printable. Then the `\if:w` test is unfinished after reading the space from `\token_to_str:N__`, and the auxiliary `\cs_to_str_aux:w` is expanded, feeding - as a second character for the test; the test is false, and TeX skips to `\fi:`, then performs `\token_to_str:N`, and stops the `\int_to_roman:w` with `\c_zero`. The last case is that the escape character is itself a space. In this case, the `\if:w` test is true, and the auxiliary `\cs_to_str_aux:w` comes into play, inserting `-\int_value:w`, which expands `\c_zero` to the character 0. The initial `\int_to_roman:w` then sees 0, which is not a terminated number, followed by the escape character, a space, which is removed, terminating the argument of `\int_to_roman:w`. In all three cases, `\cs_to_str:N` takes two expansion steps to be fully expanded.

```

1085 \cs_set_nopar:Npn \cs_to_str:N
1086 {
1087   \int_to_roman:w
1088     \if:w \token_to_str:N \ \cs_to_str_aux:w \fi:
1089     \exp_after:wN \cs_to_str_aux:N \token_to_str:N
1090 }
1091 \cs_set:Npn \cs_to_str_aux:N #1 { \c_zero }
1092 \cs_set:Npn \cs_to_str_aux:w #1 \cs_to_str_aux:N
1093 { - \int_value:w \fi: \exp_after:wN \c_zero }

```

`\cs_split_function:NN` This function takes a function name and splits it into name with the escape char removed
`\cs_split_function_aux:w` and argument specification. In addition to this, a third argument, a boolean `<true>`
`\cs_split_function_auxii:w` or `<false>` is returned with `<true>` for when there is a colon in the function and `<false>` if there is not. Lastly, the second argument of `\cs_split_function:NN` is supposed to be a function taking three variables, one for name, one for signature, and one for

the boolean. For example, `\cs_split_function:NN\foo_bar:cnx\use_i:nnn` as input becomes `\use_i:nnn {foo_bar}{cnx}\c_true_bool`.

Can't use a literal `:` because it has the wrong catcode here, so it's transformed from `@` with `\tex_lowercase:D`.

```

1094 \group_begin:
1095   \tex_lccode:D '@ = '\: \scan_stop:
1096   \tex_catcode:D '@ = 12~
1097   \tex_lowercase:D
1098   {
1099   \group_end:

```

First ensure that we actually get a properly evaluated str by expanding `\cs_to_str:N` twice. Insert extra colon to catch the error cases.

```

1100   \cs_set:Npn \cs_split_function:NN #1#2
1101   {
1102     \exp_after:wN \exp_after:wN
1103     \exp_after:wN \cs_split_function_aux:w
1104     \cs_to_str:N #1 @ a \q_stop #2
1105   }

```

If no colon in the name, `#2` is a with catcode 11 and `#3` is empty. If colon in the name, then either `#2` is a colon or the first letter of the signature. The letters here have catcode 12. If a colon was given we need to a) split off the colon and quark at the end and b) ensure we return the name, signature and boolean true. We can't use `\quark_if_no_value:NTF` yet but this is very safe anyway as all tokens have catcode 12.

```

1106   \cs_set:Npn \cs_split_function_aux:w #1 @ #2#3 \q_stop #4
1107   {
1108     \if_meaning:w a #2
1109     \exp_after:wN \use_i:nn
1110     \else:
1111     \exp_after:wN \use_ii:nn
1112     \fi:
1113     { #4 {#1} { } \c_false_bool }
1114     { \cs_split_function_auxii:w #2#3 \q_stop #4 {#1} }
1115   }
1116   \cs_set:Npn \cs_split_function_auxii:w #1 @a \q_stop #2#3
1117   { #2{#3}{#1}\c_true_bool }

```

End of lowercase

```

1118 }

```

`\cs_get_function_name:N` Now returning the name is trivial: just discard the last two arguments. Similar for
`\cs_get_function_signature:N` signature.

```

1119 \cs_set:Npn \cs_get_function_name:N #1
1120 { \cs_split_function:NN #1 \use_i:nnn }
1121 \cs_set:Npn \cs_get_function_signature:N #1
1122 { \cs_split_function:NN #1 \use_ii:nnn }

```

185.8 Exist or free

A control sequence is said to *exist* (to be used) if has an entry in the hash table and its meaning is different from the primitive `\tex_relax:D` token. A control sequence is said to be *free* (to be defined) if it does not already exist.

`\cs_if_exist:N` Two versions for checking existence. For the `N` form we firstly check for `\scan_stop:` and
`\cs_if_exist:c` then if it is in the hash table. There is no problem when inputting something like `\else:`
or `\fi:` as T_EX will only ever skip input in case the token tested against is `\scan_stop:`.

```

1123 \prg_set_conditional:Npnn \cs_if_exist:N #1 { p , T , F , TF }
1124 {
1125   \if_meaning:w #1 \scan_stop:
1126   \prg_return_false:
1127   \else:
1128     \if_cs_exist:N #1
1129     \prg_return_true:
1130   \else:
1131     \prg_return_false:
1132   \fi:
1133 \fi:
1134 }

```

For the `c` form we firstly check if it is in the hash table and then for `\scan_stop:` so that we do not add it to the hash table unless it was already there. Here we have to be careful as the text to be skipped if the first test is false may contain tokens that disturb the scanner. Therefore, we ensure that the second test is performed after the first one has concluded completely.

```

1135 \prg_set_conditional:Npnn \cs_if_exist:c #1 { p , T , F , TF }
1136 {
1137   \if_cs_exist:w #1 \cs_end:
1138   \exp_after:wN \use_i:nn
1139   \else:
1140     \exp_after:wN \use_ii:nn
1141   \fi:
1142   {
1143     \exp_after:wN \if_meaning:w \cs:w #1 \cs_end: \scan_stop:
1144     \prg_return_false:
1145   \else:
1146     \prg_return_true:
1147   \fi:
1148   }
1149   \prg_return_false:
1150 }

```

(End definition for `\use:x`. This function is documented on page ??.)

`\cs_if_free:N` The logical reversal of the above.

```

\cs_if_free:c 1151 \prg_set_conditional:Npnn \cs_if_free:N #1 { p , T , F , TF }
1152 {
1153   \if_meaning:w #1 \scan_stop:

```



```

1154     \prg_return_true:
1155 \else:
1156     \if_cs_exist:N #1
1157     \prg_return_false:
1158 \else:
1159     \prg_return_true:
1160 \fi:
1161 \fi:
1162 }
1163 \prg_set_conditional:Npnn \cs_if_free:c #1 { p , T , F , TF }
1164 {
1165     \if_cs_exist:w #1 \cs_end:
1166     \exp_after:wN \use_i:nn
1167 \else:
1168     \exp_after:wN \use_ii:nn
1169 \fi:
1170 {
1171     \exp_after:wN \if_meaning:w \cs:w #1 \cs_end: \scan_stop:
1172     \prg_return_true:
1173 \else:
1174     \prg_return_false:
1175 \fi:
1176 }
1177 { \prg_return_true: }
1178 }

```

(End definition for \cs_if_free:N and \cs_if_free:c. These functions are documented on page ??.)

\cs_if_exist_use:N The \cs_if_exist_use:... functions cannot be implemented as conditionals because
\cs_if_exist_use:c the true branch must leave both the control sequence itself and the true code in the input
\cs_if_exist_use:N stream. For the c variants, we are careful not to put the control sequence in the hash
\cs_if_exist_use:c table if it does not exist.

```

1179 \cs_set:Npn \cs_if_exist_use:NTF #1#2
1180 { \cs_if_exist:NTF #1 { #1 #2 } }
1181 \cs_set:Npn \cs_if_exist_use:NF #1
1182 { \cs_if_exist:NTF #1 { #1 } }
1183 \cs_set:Npn \cs_if_exist_use:NT #1 #2
1184 { \cs_if_exist:NTF #1 { #1#2 } { } }
1185 \cs_set:Npn \cs_if_exist_use:N #1
1186 { \cs_if_exist:NTF #1 { #1 } { } }
1187 \cs_set:Npn \cs_if_exist_use:cTF #1#2
1188 { \cs_if_exist:cTF {#1} { \use:c {#1} #2 } }
1189 \cs_set:Npn \cs_if_exist_use:cF #1
1190 { \cs_if_exist:cTF {#1} { \use:c {#1} } }
1191 \cs_set:Npn \cs_if_exist_use:cT #1#2
1192 { \cs_if_exist:cTF {#1} { \use:c {#1} #2 } { } }
1193 \cs_set:Npn \cs_if_exist_use:c #1
1194 { \cs_if_exist:cTF {#1} { \use:c {#1} } { } }

```

(End definition for \cs_if_exist_use:N and \cs_if_exist_use:c. These functions are documented on page ??.)

185.9 Defining and checking (new) functions

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.

`\iow_log:x` We define a routine to write only to the log file. And a similar one for writing to both
`\iow_term:x` the log file and the terminal. These will be redefined later by `l3io`.

```
1195 \cs_set_protected_nopar:Npn \iow_log:x
1196 { \tex_immediate:D \tex_write:D \c_minus_one }
1197 \cs_set_protected_nopar:Npn \iow_term:x
1198 { \tex_immediate:D \tex_write:D \c_sixteen }
      (End definition for \iow_log:x. This function is documented on page ??.)
```

`\msg_kernel_error:nxxx` If an internal error occurs before L^AT_EX3 has loaded `l3msg` then the code should issue a
`\msg_kernel_error:nxx` usable if terse error message and halt. This can only happen if a coding error is made by
`\msg_kernel_error:nn` the team, so this is a reasonable response.

```
1199 \cs_set_protected:Npn \msg_kernel_error:nxxx #1#2#3#4
1200 {
1201   \tex_errmessage:D
1202   {
1203     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!~! ^^J
1204     Argh,~internal~LaTeX3~error! ^^J ^^J
1205     Module ~ #1 , ~ message-name-~"#2": ^^J
1206     Arguments~'#3'~and~'#4' ^^J ^^J
1207     This~is~one~for~The~LaTeX3~Project:~bailing~out
1208   }
1209   \tex_end:D
1210 }
1211 \cs_set_protected:Npn \msg_kernel_error:nxx #1#2#3
1212 { \msg_kernel_error:nxxx {#1} {#2} {#3} { } }
1213 \cs_set_protected:Npn \msg_kernel_error:nn #1#2
1214 { \msg_kernel_error:nxxx {#1} {#2} { } { } }
      (End definition for \msg_kernel_error:nxxx. This function is documented on page ??.)
```

`\msg_line_context:` Another one from `l3msg` which will be altered later.

```
1215 \cs_set_nopar:Npn \msg_line_context:
1216 { on~line~\tex_the:D \tex_inputlineno:D }
      (End definition for \msg_line_context:. This function is documented on page ??.)
```

`\chk_if_free_cs:N` This command is called by `\cs_new_nopar:Npn` and `\cs_new_eq:NN` etc. to make sure
`\chk_if_free_cs:c` that the argument sequence is not already in use. If it is, an error is signalled. It checks if `<cname>` 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!

```

1217 \cs_set_protected:Npn \chk_if_free_cs:N #1
1218 {
1219   \cs_if_free:NF #1
1220   {
1221     \msg_kernel_error:nnxx { kernel } { command-already-defined }
1222     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1223   }
1224 }
1225 <*package>
1226 \tex_ifodd:D \l@expl@log@functions@bool
1227 \cs_set_protected:Npn \chk_if_free_cs:N #1
1228 {
1229   \cs_if_free:NF #1
1230   {
1231     \msg_kernel_error:nnxx { kernel } { command-already-defined }
1232     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1233   }
1234   \iow_log:x { Defining~\token_to_str:N #1~ \msg_line_context: }
1235 }
1236 \fi:
1237 </package>
1238 \cs_set_protected_nopar:Npn \chk_if_free_cs:c
1239 { \exp_args:Nc \chk_if_free_cs:N }

```

(End definition for \chk_if_free_cs:N and \chk_if_free_cs:c. These functions are documented on page ??.)

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

```

1240 \cs_set_protected:Npn \chk_if_exist_cs:N #1
1241 {
1242   \cs_if_exist:NF #1
1243   {
1244     \msg_kernel_error:nnxx { kernel } { command-not-defined }
1245     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1246   }
1247 }
1248 \cs_set_protected_nopar:Npn \chk_if_exist_cs:c
1249 { \exp_args:Nc \chk_if_exist_cs:N }

```

(End definition for \chk_if_exist_cs:N and \chk_if_exist_cs:c. These functions are documented on page ??.)

185.10 More new definitions

Global versions of the above functions.

```

\cs_new_nopar:Npn
\cs_new_nopar:Npx
  \cs_new:Npn
  \cs_new:Npx
\cs_new_protected_nopar:Npn
\cs_new_protected_nopar:Npx
  \cs_new_protected:Npn
  \cs_new_protected:Npx

```

```

1250 \cs_set:Npn \cs_tmp:w #1#2
1251 {
1252   \cs_set_protected:Npn #1 ##1
1253   {
1254     \chk_if_free_cs:N ##1

```

```

1255         #2 ##1
1256     }
1257 }
1258 \cs_tmp:w \cs_new_nopar:Npn          \cs_gset_nopar:Npn
1259 \cs_tmp:w \cs_new_nopar:Npx          \cs_gset_nopar:Npx
1260 \cs_tmp:w \cs_new:Npn                \cs_gset:Npn
1261 \cs_tmp:w \cs_new:Npx                \cs_gset:Npx
1262 \cs_tmp:w \cs_new_protected_nopar:Npn \cs_gset_protected_nopar:Npn
1263 \cs_tmp:w \cs_new_protected_nopar:Npx \cs_gset_protected_nopar:Npx
1264 \cs_tmp:w \cs_new_protected:Npn      \cs_gset_protected:Npn
1265 \cs_tmp:w \cs_new_protected:Npx      \cs_gset_protected:Npx

```

(End definition for `\cs_new_nopar:Npn`. This function is documented on page ??.)

`\cs_set_nopar:cpn` Like `\cs_set_nopar:Npn` and `\cs_new_nopar: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 `csname` argument, see the expansion module). Global versions are also provided.

`\cs_set_nopar:cpn` $\langle string \rangle \langle rep-text \rangle$ will turn $\langle string \rangle$ into a `csname` and then assign $\langle rep-text \rangle$ to it by using `\cs_set_nopar:Npn`. This means that there might be a parameter string between the two arguments.

```

1266 \cs_set:Npn \cs_tmp:w #1#2
1267 { \cs_new_protected_nopar:Npn #1 { \exp_args:Nc #2 } }
1268 \cs_tmp:w \cs_set_nopar:cpn \cs_set_nopar:Npn
1269 \cs_tmp:w \cs_set_nopar:cpx \cs_set_nopar:Npx
1270 \cs_tmp:w \cs_gset_nopar:cpn \cs_gset_nopar:Npn
1271 \cs_tmp:w \cs_gset_nopar:cpx \cs_gset_nopar:Npx
1272 \cs_tmp:w \cs_new_nopar:cpn \cs_new_nopar:Npn
1273 \cs_tmp:w \cs_new_nopar:cpx \cs_new_nopar:Npx

```

(End definition for `\cs_set_nopar:cpn`. This function is documented on page ??.)

`\cs_set:cpn` Variants of the `\cs_set:Npn` versions which make a `csname` out of the first arguments.

`\cs_set:cpx` We may also do this globally.

```

1274 \cs_tmp:w \cs_set:cpn \cs_set:Npn
1275 \cs_tmp:w \cs_set:cpx \cs_set:Npx
1276 \cs_tmp:w \cs_gset:cpn \cs_gset:Npn
1277 \cs_tmp:w \cs_gset:cpx \cs_gset:Npx
1278 \cs_tmp:w \cs_new:cpn \cs_new:Npn
1279 \cs_tmp:w \cs_new:cpx \cs_new:Npx

```

(End definition for `\cs_set:cpn`. This function is documented on page ??.)

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

```

1280 \cs_tmp:w \cs_set_protected_nopar:cpn \cs_set_protected_nopar:Npn
1281 \cs_tmp:w \cs_set_protected_nopar:cpx \cs_set_protected_nopar:Npx
1282 \cs_tmp:w \cs_gset_protected_nopar:cpn \cs_gset_protected_nopar:Npn
1283 \cs_tmp:w \cs_gset_protected_nopar:cpx \cs_gset_protected_nopar:Npx
1284 \cs_tmp:w \cs_new_protected_nopar:cpn \cs_new_protected_nopar:Npn
1285 \cs_tmp:w \cs_new_protected_nopar:cpx \cs_new_protected_nopar:Npx

```

(End definition for `\cs_set_protected_nopar:cpn`. This function is documented on page ??.)

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

```

\cs_set_protected:cpn 1286 \cs_tmp:w \cs_set_protected:cpn \cs_set_protected:Npn
\cs_set_protected:cpx 1287 \cs_tmp:w \cs_set_protected:cpx \cs_set_protected:Npx
\cs_gset_protected:cpn 1288 \cs_tmp:w \cs_gset_protected:cpn \cs_gset_protected:Npn
\cs_gset_protected:cpx 1289 \cs_tmp:w \cs_gset_protected:cpx \cs_gset_protected:Npx
\cs_new_protected:cpn 1290 \cs_tmp:w \cs_new_protected:cpn \cs_new_protected:Npn
\cs_new_protected:cpx 1291 \cs_tmp:w \cs_new_protected:cpx \cs_new_protected:Npx

```

(End definition for `\cs_set_protected:cpn`. This function is documented on page ??.)

185.11 Copying definitions

`\cs_set_eq:NN` These macros allow us to copy the definition of a control sequence to another control sequence.

`\cs_set_eq:cN` The = sign allows us to define funny char tokens like = itself or `_` with this function.

`\cs_set_eq:Nc` For the definition of `\c_space_char{~}` to work we need the ~ after the =.

`\cs_set_eq:cc` `\cs_set_eq:NN` is long to avoid problems with a literal argument of `\par`. While `\cs_new_eq:NN` will probably never be correct with a first argument of `\par`, define it long in order to throw an “already defined” error rather than “runaway argument”.

```

1292 \cs_new_protected:Npn \cs_set_eq:NN #1 { \tex_let:D #1 =~ }
1293 \cs_new_protected_nopar:Npn \cs_set_eq:cN { \exp_args:Nc \cs_set_eq:NN }
1294 \cs_new_protected_nopar:Npn \cs_set_eq:Nc { \exp_args:NNc \cs_set_eq:NN }
1295 \cs_new_protected_nopar:Npn \cs_set_eq:cc { \exp_args:Ncc \cs_set_eq:NN }

```

(End definition for `\cs_set_eq:NN`. This function is documented on page ??.)

`\cs_new_eq:NN`

```

\cs_new_eq:cN 1296 \cs_new_protected:Npn \cs_new_eq:NN #1
\cs_new_eq:Nc 1297 {
\cs_new_eq:cc 1298   \chk_if_free_cs:N #1
1299   \tex_global:D \cs_set_eq:NN #1
1300 }
1301 \cs_new_protected_nopar:Npn \cs_new_eq:cN { \exp_args:Nc \cs_new_eq:NN }
1302 \cs_new_protected_nopar:Npn \cs_new_eq:Nc { \exp_args:NNc \cs_new_eq:NN }
1303 \cs_new_protected_nopar:Npn \cs_new_eq:cc { \exp_args:Ncc \cs_new_eq:NN }

```

(End definition for `\cs_new_eq:NN`. This function is documented on page ??.)

`\cs_gset_eq:NN`

```

\cs_gset_eq:cN 1304 \cs_new_protected_nopar:Npn \cs_gset_eq:NN { \tex_global:D \cs_set_eq:NN }
\cs_gset_eq:Nc 1305 \cs_new_protected_nopar:Npn \cs_gset_eq:Nc { \exp_args:NNc \cs_gset_eq:NN }
\cs_gset_eq:cc 1306 \cs_new_protected_nopar:Npn \cs_gset_eq:cN { \exp_args:Nc \cs_gset_eq:NN }
1307 \cs_new_protected_nopar:Npn \cs_gset_eq:cc { \exp_args:Ncc \cs_gset_eq:NN }

```

(End definition for `\cs_gset_eq:NN`. This function is documented on page ??.)

185.12 Undefined functions

`\cs_undefine:N` The following function is used to free the main memory from the definition of some
`\cs_undefine:c` function that isn't in use any longer. The `c` variant is careful not to add the control sequence to the hash table if it isn't there yet, and it also avoids nesting \TeX conditionals in case `#1` is unbalanced in this matter.

```

1308 \cs_new_protected:Npn \cs_undefine:N #1
1309 { \cs_gset_eq:NN #1 \c_undefined:D }
1310 \cs_new_protected:Npn \cs_undefine:c #1
1311 {
1312   \if_cs_exist:w #1 \cs_end:
1313     \exp_after:wN \use:n
1314   \else:
1315     \exp_after:wN \use_none:n
1316   \fi:
1317   { \cs_gset_eq:cN {#1} \c_undefined:D }
1318 }

```

(End definition for `\cs_undefine:N` and `\cs_undefine:c`. These functions are documented on page ??.)

185.13 Defining functions from a given number of arguments

`\cs_get_arg_count_from_signature:N` Counting the number of tokens in the signature, i.e., the number of arguments the func-
`\cs_get_arg_count_from_signature_aux:nnN` tion should take. If there is no signature, we return that there is -1 arguments to signal
`\cs_get_arg_count_from_signature_auxii:w` an error. Otherwise we insert the string 9876543210 after the signature. If the signature is empty, the number we want is 0 so we remove the first nine tokens and return the tenth. Similarly, if the signature is `nnn` we want to remove the nine tokens `nnn987654` and return 3. Therefore, we simply remove the first nine tokens and then return the tenth.

```

1319 \cs_new:Npn \cs_get_arg_count_from_signature:N #1
1320 { \cs_split_function:NN #1 \cs_get_arg_count_from_signature_aux:nnN }
1321 \cs_new:Npn \cs_get_arg_count_from_signature_aux:nnN #1#2#3
1322 {
1323   \if_meaning:w \c_true_bool #3
1324     \exp_after:wN \use_i:nn
1325   \else:
1326     \exp_after:wN \use_ii:nn
1327   \fi:
1328   {
1329     \exp_after:wN \cs_get_arg_count_from_signature_auxii:w
1330     \use_none:nnnnnnnn #2 9876543210 \q_stop
1331   }
1332   { -1 }
1333 }
1334 \cs_new:Npn \cs_get_arg_count_from_signature_auxii:w #1#2 \q_stop {#1}

```

A variant form we need right away.

```

1335 \cs_new_nopar:Npn \cs_get_arg_count_from_signature:c
1336 { \exp_args:Nc \cs_get_arg_count_from_signature:N }

```

(End definition for `\cs_get_arg_count_from_signature:N`. This function is documented on page ??.)

`\cs_generate_from_arg_count:NNnn`
`\cs_generate_from_arg_count_error_msg:Nn`
`\cs_generate_from_arg_count_aux:nwn`

We provide a constructor function for defining functions with a given number of arguments. For this we need to choose the correct parameter text and then use that when defining. Since TeX supports from zero to nine arguments, we use a simple switch to choose the correct parameter text, ensuring the result is returned after finishing the conditional. If it is not between zero and nine, we throw an error.

1: function to define, 2: with what to define it, 3: the number of args it requires and 4: the replacement text

```

1337 \cs_new_protected:Npn \cs_generate_from_arg_count:NNnn #1#2#3#4
1338 {
1339   \if_case:w \int_eval:w #3 \int_eval_end:
1340     \cs_generate_from_arg_count_aux:nwn {}
1341   \or: \cs_generate_from_arg_count_aux:nwn {##1}
1342   \or: \cs_generate_from_arg_count_aux:nwn {##1##2}
1343   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3}
1344   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4}
1345   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5}
1346   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6}
1347   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7}
1348   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7##8}
1349   \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7##8##9}
1350   \else:
1351     \cs_generate_from_arg_count_error_msg:Nn #1 {#3}
1352     \use_i:nnn
1353   \fi:
1354   {#2#1}
1355   {#4}
1356 }
1357 \cs_new_protected:Npn
1358   \cs_generate_from_arg_count_aux:nwn #1 #2 \fi: #3
1359   { \fi: #3 #1 }

```

A variant form we need right away.

```

1360 \cs_new_nopar:Npn \cs_generate_from_arg_count:cNnn
1361   { \exp_args:Nc \cs_generate_from_arg_count:NNnn }

```

The error message. Elsewhere we use the value of `-1` to signal a missing colon in a function, so provide a hint for help on this.

```

1362 \cs_new:Npn \cs_generate_from_arg_count_error_msg:Nn #1#2
1363 {
1364   \msg_kernel_error:nxxx { kernel } { bad-number-of-arguments }
1365   { \token_to_str:N #1 } { \int_eval:n {#2} }
1366 }

```

(End definition for `\cs_generate_from_arg_count:NNnn`. This function is documented on page ??.)

185.14 Using the signature to define functions

We can now combine some of the tools we have to provide a simple interface for defining functions. We define some simpler functions with user interface `\cs_set:Nn \foo_bar:nn {#1,#2}`, *i.e.*, the number of arguments is read from the signature.

We want to define `\cs_set:Nn` as

```

\cs_set:Nn
\cs_set:Nx
\cs_set_nopar:Nn
\cs_set_nopar:Nx
\cs_set_protected:Nn
\cs_set_protected:Nx
\cs_set_protected_nopar:Nn
\cs_set_protected_nopar:Nx
\cs_gset:Nn
\cs_gset:Nx
\cs_gset_nopar:Nn
\cs_gset_nopar:Nx
\cs_gset_protected:Nn
\cs_gset_protected:Nx
\cs_gset_protected_nopar:Nn
\cs_gset_protected_nopar:Nx

```

In short, to define `\cs_set:Nn` we need just use `\cs_set:Npn`, everything else is the same for each variant. Therefore, we can make it simpler by temporarily defining a function to do this for us.

```

1367 \cs_set:Npn \cs_tmp:w #1#2#3
1368 {
1369   \cs_set_protected:cpx { cs_ #1 : #2 } ##1##2
1370   {
1371     \exp_not:N \cs_generate_from_arg_count:NNnn ##1
1372     \exp_after:wN \exp_not:N \cs:w cs_#1 : #3 \cs_end:
1373     { \exp_not:N \cs_get_arg_count_from_signature:N ##1 }{##2}
1374   }
1375 }

```

Then we define the 32 variants beginning with N.

```

1376 \cs_tmp:w { set } { Nn } { Npn }
1377 \cs_tmp:w { set } { Nx } { Npx }
1378 \cs_tmp:w { set_nopar } { Nn } { Npn }
1379 \cs_tmp:w { set_nopar } { Nx } { Npx }
1380 \cs_tmp:w { set_protected } { Nn } { Npn }
1381 \cs_tmp:w { set_protected } { Nx } { Npx }
1382 \cs_tmp:w { set_protected_nopar } { Nn } { Npn }
1383 \cs_tmp:w { set_protected_nopar } { Nx } { Npx }
1384 \cs_tmp:w { gset } { Nn } { Npn }
1385 \cs_tmp:w { gset } { Nx } { Npx }
1386 \cs_tmp:w { gset_nopar } { Nn } { Npn }
1387 \cs_tmp:w { gset_nopar } { Nx } { Npx }
1388 \cs_tmp:w { gset_protected } { Nn } { Npn }
1389 \cs_tmp:w { gset_protected } { Nx } { Npx }
1390 \cs_tmp:w { gset_protected_nopar } { Nn } { Npn }
1391 \cs_tmp:w { gset_protected_nopar } { Nx } { Npx }

```

(End definition for \cs_set:Nn. This function is documented on page ??.)

```

\cs_new:Nn
\cs_new:Nx
\cs_new_nopar:Nn
\cs_new_nopar:Nx
\cs_new_protected:Nn
\cs_new_protected:Nx
\cs_new_protected_nopar:Nn
\cs_new_protected_nopar:Nx

```

```

1392 \cs_tmp:w { new } { Nn } { Npn }
1393 \cs_tmp:w { new } { Nx } { Npx }
1394 \cs_tmp:w { new_nopar } { Nn } { Npn }

```



```

1395 \cs_tmp:w { new_nopar } { Nx } { Npx }
1396 \cs_tmp:w { new_protected } { Nn } { Npn }
1397 \cs_tmp:w { new_protected } { Nx } { Npx }
1398 \cs_tmp:w { new_protected_nopar } { Nn } { Npn }
1399 \cs_tmp:w { new_protected_nopar } { Nx } { Npx }

```

(End definition for \cs_new:Nn. This function is documented on page ??.)

Then something similar for the c variants.

```

\cs_set_protected:Npn \cs_set:cn #1#2
{
  \cs_generate_from_arg_count:cNnn {#1} \cs_set:Npn
  { \cs_get_arg_count_from_signature:c {#1} } {#2}
}

```

```

1400 \cs_set:Npn \cs_tmp:w #1#2#3
1401 {
1402   \cs_set_protected:cpx {cs_#1:#2}##1##2{
1403     \exp_not:N\cs_generate_from_arg_count:cNnn {##1}
1404     \exp_after:wN \exp_not:N \cs:w cs_#1:#3 \cs_end:
1405     { \exp_not:N \cs_get_arg_count_from_signature:c {##1} } {##2}
1406   }
1407 }

```

\cs_set:cn The 32 c variants.

```

\cs_set:cx 1408 \cs_tmp:w { set } { cn } { Npn }
\cs_set_nopar:cn 1409 \cs_tmp:w { set } { cx } { Npx }
\cs_set_nopar:cx 1410 \cs_tmp:w { set_nopar } { cn } { Npn }
\cs_set_protected:cn 1411 \cs_tmp:w { set_nopar } { cx } { Npx }
\cs_set_protected:cx 1412 \cs_tmp:w { set_protected } { cn } { Npn }
\cs_set_protected_nopar:cn 1413 \cs_tmp:w { set_protected } { cx } { Npx }
\cs_set_protected_nopar:cx 1414 \cs_tmp:w { set_protected_nopar } { cn } { Npn }
\cs_gset:cn 1415 \cs_tmp:w { set_protected_nopar } { cx } { Npx }
\cs_gset:cx 1416 \cs_tmp:w { gset } { cn } { Npn }
\cs_gset_nopar:cn 1417 \cs_tmp:w { gset } { cx } { Npx }
\cs_gset_nopar:cx 1418 \cs_tmp:w { gset_nopar } { cn } { Npn }
\cs_gset_protected:cn 1419 \cs_tmp:w { gset_nopar } { cx } { Npx }
\cs_gset_protected:cx 1420 \cs_tmp:w { gset_protected } { cn } { Npn }
\cs_gset_protected_nopar:cn 1421 \cs_tmp:w { gset_protected } { cx } { Npx }
\cs_gset_protected_nopar:cx 1422 \cs_tmp:w { gset_protected_nopar } { cn } { Npn }
1423 \cs_tmp:w { gset_protected_nopar } { cx } { Npx }

```

(End definition for \cs_set:cn. This function is documented on page ??.)

```

\cs_new:cn
\cs_new:cx 1424 \cs_tmp:w { new } { cn } { Npn }
\cs_new_nopar:cn 1425 \cs_tmp:w { new } { cx } { Npx }
\cs_new_nopar:cx 1426 \cs_tmp:w { new_nopar } { cn } { Npn }
\cs_new_protected:cn 1427 \cs_tmp:w { new_nopar } { cx } { Npx }
\cs_new_protected:cx 1428 \cs_tmp:w { new_protected } { cn } { Npn }
\cs_new_protected_nopar:cn 1429 \cs_tmp:w { new_protected } { cx } { Npx }
\cs_new_protected_nopar:cx

```

```

1430 \cs_tmp:w { new_protected_nopar } { cn } { Npn }
1431 \cs_tmp:w { new_protected_nopar } { cx } { Npx }
      (End definition for \cs_new:cn. This function is documented on page ??.)

```

185.15 Checking control sequence equality

```

\cs_if_eq:NN Check if two control sequences are identical.
\cs_if_eq:cN 1432 \prg_new_conditional:Npnn \cs_if_eq:NN #1#2 { p , T , F , TF }
\cs_if_eq:Nc 1433 {
\cs_if_eq:cc 1434   \if_meaning:w #1#2
              1435   \prg_return_true: \else: \prg_return_false: \fi:
              1436 }
1437 \cs_new_nopar:Npn \cs_if_eq_p:cN { \exp_args:Nc \cs_if_eq_p:NN }
1438 \cs_new_nopar:Npn \cs_if_eq:cNTF { \exp_args:Nc \cs_if_eq:NNTF }
1439 \cs_new_nopar:Npn \cs_if_eq:cNT { \exp_args:Nc \cs_if_eq:NNT }
1440 \cs_new_nopar:Npn \cs_if_eq:cNF { \exp_args:Nc \cs_if_eq:NNF }
1441 \cs_new_nopar:Npn \cs_if_eq_p:Nc { \exp_args:NNc \cs_if_eq_p:NN }
1442 \cs_new_nopar:Npn \cs_if_eq:NcTF { \exp_args:NNc \cs_if_eq:NNTF }
1443 \cs_new_nopar:Npn \cs_if_eq:NcT { \exp_args:NNc \cs_if_eq:NNT }
1444 \cs_new_nopar:Npn \cs_if_eq:NcF { \exp_args:NNc \cs_if_eq:NNF }
1445 \cs_new_nopar:Npn \cs_if_eq_p:cc { \exp_args:Ncc \cs_if_eq_p:NN }
1446 \cs_new_nopar:Npn \cs_if_eq:ccTF { \exp_args:Ncc \cs_if_eq:NNTF }
1447 \cs_new_nopar:Npn \cs_if_eq:ccT { \exp_args:Ncc \cs_if_eq:NNT }
1448 \cs_new_nopar:Npn \cs_if_eq:ccF { \exp_args:Ncc \cs_if_eq:NNF }
      (End definition for \cs_if_eq:NN and others. These functions are documented on page ??.)

```

185.16 Diagnostic wrapper functions

```

\kernel_register_show:N
\kernel_register_show:c 1449 \cs_new:Npn \kernel_register_show:N #1
                        1450 {
                        1451   \cs_if_exist:NTF #1
                        1452   { \tex_showthe:D #1 }
                        1453   {
                        1454     \msg_kernel_error:nxx { kernel } { variable-not-defined }
                        1455     { \token_to_str:N #1 }
                        1456   }
                        1457 }
1458 \cs_new_nopar:Npn \kernel_register_show:c
1459 { \exp_args:Nc \kernel_register_show:N }
      (End definition for \kernel_register_show:N and \kernel_register_show:c. These functions are
      documented on page ??.)

```

185.17 Engine specific definitions

```

\xetex_if_engine: In some cases it will be useful to know which engine we're running. This can all be
\luatex_if_engine: hard-coded for speed.
\pdftex_if_engine: 1460 \cs_new_eq:NN \luatex_if_engine:T \use_none:n

```

```

1461 \cs_new_eq:NN \luatex_if_engine:F \use:n
1462 \cs_new_eq:NN \luatex_if_engine:TF \use_ii:nn
1463 \cs_new_eq:NN \pdftex_if_engine:T \use:n
1464 \cs_new_eq:NN \pdftex_if_engine:F \use_none:n
1465 \cs_new_eq:NN \pdftex_if_engine:TF \use_i:nn
1466 \cs_new_eq:NN \xetex_if_engine:T \use_none:n
1467 \cs_new_eq:NN \xetex_if_engine:F \use:n
1468 \cs_new_eq:NN \xetex_if_engine:TF \use_ii:nn
1469 \cs_new_eq:NN \luatex_if_engine_p: \c_false_bool
1470 \cs_new_eq:NN \pdftex_if_engine_p: \c_true_bool
1471 \cs_new_eq:NN \xetex_if_engine_p: \c_false_bool
1472 \cs_if_exist:NT \xetex_XeTeXversion:D
1473 {
1474     \cs_set_eq:NN \pdftex_if_engine:T \use_none:n
1475     \cs_set_eq:NN \pdftex_if_engine:F \use:n
1476     \cs_set_eq:NN \pdftex_if_engine:TF \use_ii:nn
1477     \cs_set_eq:NN \xetex_if_engine:T \use:n
1478     \cs_set_eq:NN \xetex_if_engine:F \use_none:n
1479     \cs_set_eq:NN \xetex_if_engine:TF \use_i:nn
1480     \cs_set_eq:NN \pdftex_if_engine_p: \c_false_bool
1481     \cs_set_eq:NN \xetex_if_engine_p: \c_true_bool
1482 }
1483 \cs_if_exist:NT \luatex_directlua:D
1484 {
1485     \cs_set_eq:NN \luatex_if_engine:T \use:n
1486     \cs_set_eq:NN \luatex_if_engine:F \use_none:n
1487     \cs_set_eq:NN \luatex_if_engine:TF \use_i:nn
1488     \cs_set_eq:NN \pdftex_if_engine:T \use_none:n
1489     \cs_set_eq:NN \pdftex_if_engine:F \use:n
1490     \cs_set_eq:NN \pdftex_if_engine:TF \use_ii:nn
1491     \cs_set_eq:NN \luatex_if_engine_p: \c_true_bool
1492     \cs_set_eq:NN \pdftex_if_engine_p: \c_false_bool
1493 }

```

(End definition for `\xetex_if_engine:`, `\luatex_if_engine:`, and `\pdftex_if_engine:`. These functions are documented on page ??.)

185.18 Doing nothing functions

`\prg_do_nothing:` This does not fit anywhere else!

```

1494 \cs_new_nopar:Npn \prg_do_nothing: { }

```

(End definition for `\prg_do_nothing:`. This function is documented on page ??.)

185.19 String comparisons

`\str_if_eq:nn` Modern engines provide a direct way of comparing two token lists, but returning a number. This set of conditionals therefore make life a bit clearer. The `nn` and `xx` versions are created directly as this is most efficient. These should eventually move somewhere else.

```

1495 \prg_new_conditional:Npnn \str_if_eq:nn #1#2 { p , T , F , TF }

```

```

1496 {
1497   \if_int_compare:w \pdfTeX_strcmp:D { \exp_not:n {#1} } { \exp_not:n {#2} }
1498   = \c_zero
1499   \prg_return_true: \else: \prg_return_false: \fi:
1500 }
1501 \prg_new_conditional:Npnn \str_if_eq:xx #1#2 { p , T , F , TF }
1502 {
1503   \if_int_compare:w \pdfTeX_strcmp:D {#1} {#2} = \c_zero
1504   \prg_return_true: \else: \prg_return_false: \fi:
1505 }

```

(End definition for \str_if_eq:nn. This function is documented on page ??.)

185.20 Breaking out of mapping functions

`\prg_break_point:n` In inline mappings, the nesting level must be reset at the end of the mapping, even when the user decides to break out. This is done by putting the code that must be performed as an argument of `\prg_break_point:n`. The breaking functions are then defined to jump to that point and perform the argument of `\prg_break_point:n`, before the user's code (if any).

```

1506 \cs_new_eq:NN \prg_break_point:n \use:n
1507 \cs_new:Npn \prg_map_break: #1 \prg_break_point:n #2 { #2 }
1508 \cs_new:Npn \prg_map_break:n #1 #2 \prg_break_point:n #3 { #3 #1 }

```

(End definition for \prg_break_point:n. This function is documented on page 42.)

185.21 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

```

1509 <*deprecated>
1510 \cs_new_eq:NN \cs_gnew_nopar:Npn \cs_new_nopar:Npn
1511 \cs_new_eq:NN \cs_gnew:Npn \cs_new:Npn
1512 \cs_new_eq:NN \cs_gnew_protected_nopar:Npn \cs_new_protected_nopar:Npn
1513 \cs_new_eq:NN \cs_gnew_protected:Npn \cs_new_protected:Npn
1514 \cs_new_eq:NN \cs_gnew_nopar:Npx \cs_new_nopar:Npx
1515 \cs_new_eq:NN \cs_gnew:Npx \cs_new:Npx
1516 \cs_new_eq:NN \cs_gnew_protected_nopar:Npx \cs_new_protected_nopar:Npx
1517 \cs_new_eq:NN \cs_gnew_protected:Npx \cs_new_protected:Npx
1518 \cs_new_eq:NN \cs_gnew_nopar:cpn \cs_new_nopar:cpn
1519 \cs_new_eq:NN \cs_gnew:cpn \cs_new:cpn
1520 \cs_new_eq:NN \cs_gnew_protected_nopar:cpn \cs_new_protected_nopar:cpn
1521 \cs_new_eq:NN \cs_gnew_protected:cpn \cs_new_protected:cpn
1522 \cs_new_eq:NN \cs_gnew_nopar:cpx \cs_new_nopar:cpx
1523 \cs_new_eq:NN \cs_gnew:cpx \cs_new:cpx
1524 \cs_new_eq:NN \cs_gnew_protected_nopar:cpx \cs_new_protected_nopar:cpx
1525 \cs_new_eq:NN \cs_gnew_protected:cpx \cs_new_protected:cpx
1526 </deprecated>
1527 <*deprecated>
1528 \cs_new_eq:NN \cs_gnew_eq:NN \cs_new_eq:NN
1529 \cs_new_eq:NN \cs_gnew_eq:cN \cs_new_eq:cN

```

```

1530 \cs_new_eq:NN \cs_gnew_eq:Nc \cs_new_eq:Nc
1531 \cs_new_eq:NN \cs_gnew_eq:cc \cs_new_eq:cc
1532 </deprecated>

1533 <*deprecated>
1534 \cs_new_eq:NN \cs_gundefine:N \cs_undefine:N
1535 \cs_new_eq:NN \cs_gundefine:c \cs_undefine:c
1536 </deprecated>

1537 <*deprecated>
1538 \cs_new_eq:NN \group_execute_after:N \group_insert_after:N
1539 </deprecated>

```

Deprecated 2011-09-06, for removal by 2012-09-05.

```

\c_pdftex_is_engine_bool
\c_luatex_is_engine_bool
\c_xetex_is_engine_bool

```

Predicates are better

```

1540 \cs_new_eq:NN \c_luatex_is_engine_bool \luatex_if_engine_p:
1541 \cs_new_eq:NN \c_pdftex_is_engine_bool \pdftex_if_engine_p:
1542 \cs_new_eq:NN \c_xetex_is_engine_bool \xetex_if_engine_p:

```

(End definition for \c_pdftex_is_engine_bool, \c_luatex_is_engine_bool, and \c_xetex_is_engine_bool. These functions are documented on page ??.)

Deprecated 2011-09-06, for removal by 2012-10-06.

```

\use_i_after_fi:nw
\use_i_after_else:nw
\use_i_after_or:nw
\use_i_after_orelse:nw

```

These functions return the first argument after ending the conditional. This is rather specialized, and we want to de-emphasize the use of primitive T_EX conditionals.

```

1543 \cs_set:Npn \use_i_after_fi:nw #1 \fi: { \fi: #1 }
1544 \cs_set:Npn \use_i_after_else:nw #1 \else: #2 \fi: { \fi: #1 }
1545 \cs_set:Npn \use_i_after_or:nw #1 \or: #2 \fi: { \fi: #1 }
1546 \cs_set:Npn \use_i_after_orelse:nw #1#2#3 \fi: { \fi: #1 }

```

(End definition for \use_i_after_fi:nw. This function is documented on page ??.)

Deprecated 2011-09-07, for removal by 2011-10-07.

```
\cs_set_eq:NwN
```

```

1547 \tex_let:D \cs_set_eq:NwN \tex_let:D

```

(End definition for \cs_set_eq:NwN. This function is documented on page ??.)

```

1548 </initex | package>

```

186 l3expan implementation

```
1549 <*initex | package>
```

We start by ensuring that the required packages are loaded.

```

1550 <*package>
1551 \ProvidesExplPackage
1552   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
1553 \package_check_loaded_expl:
1554 </package>

```

```

\exp_after:wN
\exp_not:N
\exp_not:n

```

These are defined in l3basics.

(End definition for \exp_after:wN. This function is documented on page 30.)

186.1 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 `\cs_set_nopar:Npx` at some point, and so is never going to be expandable.³)

The definition of expansion functions with this technique happens in section 186.3. In section 186.2 some common cases are coded by a more direct method for efficiency, typically using calls to `\exp_after:wN`.

`\l_exp_tl` We need a scratch token list variable. We don't use `tl` methods so that `l3expan` can be loaded earlier.

```
1555 \cs_new_nopar:Npn \l_exp_tl { }
```

(End definition for `\l_exp_tl`. This function is documented on page 31.)

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.

An argument manipulator `\::⟨Z⟩` always has signature `#1\:::#2#3` where `#1` holds the remaining argument manipulations to be performed, `\:::` serves as an end marker for the list of manipulations, `#2` is the carried over result of the previous expansion steps and `#3` is the argument about to be processed.

`\exp_arg_next:nnn` `\exp_arg_next_nobrace:nnn` `#1` is the result of an expansion step, `#2` is the remaining argument manipulations and `#3` is the current result of the expansion chain. This auxiliary function moves `#1` back after `#3` in the input stream and checks if any expansion is left to be done by calling `#2`. In by far the most cases we will require to add a set of braces to the result of an argument manipulation so it is more effective to do it directly here. Actually, so far only the `c` of the final argument manipulation variants does not require a set of braces.

```
1556 \cs_new:Npn \exp_arg_next:nnn #1#2#3 { #2 \::: { #3 {#1} } }
```

```
1557 \cs_new:Npn \exp_arg_next_nobrace:nnn #1#2#3 { #2 \::: { #3 #1 } }
```

(End definition for `\exp_arg_next:nnn`. This function is documented on page ??.)

`\:::` The end marker is just another name for the identity function.

```
1558 \cs_new:Npn \::: #1 {#1}
```

(End definition for `\:::`. This function is documented on page 31.)

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

```
1559 \cs_new:Npn \::n #1 \::: #2#3 { #1 \::: { #2 {#3} } }
```

(End definition for `\::n`. This function is documented on page 31.)

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

```
1560 \cs_new:Npn \::N #1 \::: #2#3 { #1 \::: {#2#3} }
```

(End definition for `\::N`. This function is documented on page 31.)

³However, 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.

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

```
1561 \cs_new:Npn \::c #1 \::: #2#3
1562 { \exp_after:wN \exp_arg_next_nobrace:nnn \cs:w #3 \cs_end: {#1} {#2} }
(End definition for \::c. This function is documented on page 31.)
```

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

```
1563 \cs_new:Npn \::o #1 \::: #2#3
1564 { \exp_after:wN \exp_arg_next:nnn \exp_after:wN {#3} {#1} {#2} }
(End definition for \::o. This function is documented on page 31.)
```

`\::f` This function is used to expand a token list until the first unexpandable token is found.
`\exp_stop_f:` The underlying `\romannumeral -'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 T_EX had fully expanded `\cs_set_eq:Nc \aaa { b \l_tmpa_tl b }` into `\cs_set_eq:NN \aaa = \blurb` which then turned out to contain the non-expandable token `\cs_set_eq:NN`. Since the expansion of `\romannumeral -'0` is $\langle null \rangle$, we wind up with a fully expanded list, only T_EX has not tried to execute any of the non-expandable tokens. This is what differentiates this function from the `x` argument type.

```
1565 \cs_new:Npn \::f #1 \::: #2#3
1566 {
1567   \exp_after:wN \exp_arg_next:nnn
1568   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1569   {#1} {#2}
1570 }
1571 \use:nn { \cs_new_eq:NN \exp_stop_f: } { ~ }
(End definition for \::f. This function is documented on page ??.)
```

`\::x` This function is used to expand an argument fully.

```
1572 \cs_new_protected:Npn \::x #1 \::: #2#3
1573 {
1574   \cs_set_nopar:Npx \l_exp_tl { {#3} }
1575   \exp_after:wN \exp_arg_next:nnn \l_exp_tl {#1} {#2}
1576 }
(End definition for \::x. This function is documented on page 31.)
```

`\::v` These functions return the value of a register, i.e., one of `tl`, `num`, `int`, `skip`, `dim` and `\::V muskip`. The `V` version expects a single token whereas `v` like `c` creates a `csname` from its argument given in braces and then evaluates it as if it was a `V`. The primitive `\romannumeral` sets off an expansion similar to an `f` type expansion, which we will terminate using `\c-zero`. The argument is returned in braces.

```
1577 \cs_new:Npn \::V #1 \::: #2#3
1578 {
```

```

1579     \exp_after:wN \exp_arg_next:nnn
1580     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1581     {#1} {#2}
1582 }
1583 \cs_new:Npn \::v # 1\::: #2#3
1584 {
1585     \exp_after:wN \exp_arg_next:nnn
1586     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#3} }
1587     {#1} {#2}
1588 }

```

(End definition for \::v. This function is documented on page 31.)

`\exp_eval_register:N` This function evaluates a register. Now a register might exist as one of two things: A parameter-less macro or a built-in TeX register such as `\count`. For the TeX registers we have to utilize a `\the` whereas for the macros we merely have to expand them once. The trick is to find out when to use `\the` and when not to. What we do here is try to find out whether the token will expand to something else when hit with `\exp_after:wN`. The technique is to compare the meaning of the register in question when it has been prefixed with `\exp_not:N` and the register itself. If it is a macro, the prefixed `\exp_not:N` will temporarily turn it into the primitive `\scan_stop:`.

```

1589 \cs_new:Npn \exp_eval_register:N #1
1590 {
1591     \exp_after:wN \if_meaning:w \exp_not:N #1 #1

```

If the token was not a macro it may be a malformed variable from a `c` expansion in which case it is equal to the primitive `\scan_stop:`. In that case we throw an error. We could let TeX do it for us but that would result in the rather obscure

! You can't use ‘\relax’ after \the.

which while quite true doesn't give many hints as to what actually went wrong. We provide something more sensible.

```

1592     \if_meaning:w \scan_stop: #1
1593     \exp_eval_error_msg:w
1594     \fi:

```

The next bit requires some explanation. The function must be initiated by the primitive `\romannumeral` and we want to terminate this expansion chain by inserting the `\c_zero` integer constant. However, we have to expand the register `#1` before we do that. If it is a TeX register, we need to execute the sequence `\exp_after:wN \c_zero \tex_the:D #1` and if it is a macro we need to execute `\exp_after:wN \c_zero #1`. We therefore issue the longer of the two sequences and if the register is a macro, we remove the `\tex_the:D`.

```

1595     \else:
1596     \exp_after:wN \use_i_ii:nnn
1597     \fi:
1598     \exp_after:wN \c_zero \tex_the:D #1
1599 }
1600 \cs_new:Npn \exp_eval_register:c #1
1601 { \exp_after:wN \exp_eval_register:N \cs:w #1 \cs_end: }

```


Clean up nicely, then call the undefined control sequence. The result is an error message looking like this:

```

! Undefined control sequence.
<argument> \LaTeX3 error:
                               Erroneous variable used!
1.55 \tl_set:Nv \l_tmpa_tl {undefined_tl}

1602 \cs_new:Npn \exp_eval_error_msg:w #1 \tex_the:D #2
1603 {
1604   \fi:
1605   \fi:
1606   \msg_expandable_kernel_error:nnn { kernel } { bad-var } {#2}
1607   \c_zero
1608 }

(End definition for \exp_eval_register:N and \exp_eval_register:c. These functions are doc-
umented on page ??.)

```

186.2 Hand-tuned definitions

One of the most important features of these functions is that they are fully expandable and therefore allow to prefix them with `\tex_global:D` for example.

```

\exp_args:No Those lovely runs of expansion!
\exp_args:NNo 1609 \cs_new:Npn \exp_args:No #1#2 { \exp_after:wN #1 \exp_after:wN {#2} }
\exp_args:NNNo 1610 \cs_new:Npn \exp_args:NNo #1#2#3
1611 { \exp_after:wN #1 \exp_after:wN #2 \exp_after:wN {#3} }
1612 \cs_new:Npn \exp_args:NNNo #1#2#3#4
1613 { \exp_after:wN #1 \exp_after:wN#2 \exp_after:wN #3 \exp_after:wN {#4} }

(End definition for \exp_args:No. This function is documented on page 28.)

\exp_args:Nc In l3basics
(End definition for \exp_args:Nc. This function is documented on page 26.)

\exp_args:cc Here are the functions that turn their argument into csnames but are expandable.
\exp_args:NNc 1614 \cs_new:Npn \exp_args:cc #1#2
\exp_args:Ncc 1615 { \cs:w #1 \exp_after:wN \cs_end: \cs:w #2 \cs_end: }
\exp_args:Nccc 1616 \cs_new:Npn \exp_args:NNc #1#2#3
1617 { \exp_after:wN #1 \exp_after:wN #2 \cs:w # 3\cs_end: }
1618 \cs_new:Npn \exp_args:Ncc #1#2#3
1619 { \exp_after:wN #1 \cs:w #2 \exp_after:wN \cs_end: \cs:w #3 \cs_end: }
1620 \cs_new:Npn \exp_args:Nccc #1#2#3#4
1621 {
1622   \exp_after:wN #1
1623   \cs:w #2 \exp_after:wN \cs_end:
1624   \cs:w #3 \exp_after:wN \cs_end:
1625   \cs:w #4 \cs_end:
1626 }

(End definition for \exp_args:cc and others. These functions are documented on page ??.)

```

```

\exp_args:Nf
\exp_args:Nv 1627 \cs_new:Npn \exp_args:Nf #1#2
\exp_args:Nv 1628 { \exp_after:wN #1 \exp_after:wN { \tex_romannumeral:D -'0 #2 } }
\exp_args:Nx 1629 \cs_new:Npn \exp_args:Nv #1#2
1630 {
1631   \exp_after:wN #1 \exp_after:wN
1632   { \tex_romannumeral:D \exp_eval_register:c {#2} }
1633 }
1634 \cs_new:Npn \exp_args:NV #1#2
1635 {
1636   \exp_after:wN #1 \exp_after:wN
1637   { \tex_romannumeral:D \exp_eval_register:N #2 }
1638 }

```

(End definition for `\exp_args:Nf` and others. These functions are documented on page 27.)

`\exp_args:NNV` Some more hand-tuned function with three arguments. If we force that an `o` argument always has braces, we could implement `\exp_args:Nco` with less tokens and only two arguments.

```

\exp_args:NNV 1639 \cs_new:Npn \exp_args:NNf #1#2#3
\exp_args:Ncf 1640 {
\exp_args:Nco 1641   \exp_after:wN #1
1642   \exp_after:wN #2
1643   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1644 }
1645 \cs_new:Npn \exp_args:NNv #1#2#3
1646 {
1647   \exp_after:wN #1
1648   \exp_after:wN #2
1649   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#3} }
1650 }
1651 \cs_new:Npn \exp_args:NNV #1#2#3
1652 {
1653   \exp_after:wN #1
1654   \exp_after:wN #2
1655   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1656 }
1657 \cs_new:Npn \exp_args:Nco #1#2#3
1658 {
1659   \exp_after:wN #1
1660   \cs:w #2 \exp_after:wN \cs_end:
1661   \exp_after:wN {#3}
1662 }
1663 \cs_new:Npn \exp_args:Ncf #1#2#3
1664 {
1665   \exp_after:wN #1
1666   \cs:w #2 \exp_after:wN \cs_end:
1667   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1668 }
1669 \cs_new:Npn \exp_args:NVV #1#2#3

```

```

1670 {
1671   \exp_after:wN #1
1672   \exp_after:wN { \tex_romannumeral:D \exp_after:wN
1673     \exp_eval_register:N \exp_after:wN #2 \exp_after:wN }
1674   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1675 }

```

(End definition for `\exp_args:NNV` and others. These functions are documented on page ??.)

`\exp_args:Ncco` A few more that we can hand-tune.

```

\exp_args:NcNc 1676 \cs_new:Npn \exp_args:NNNV #1#2#3#4
\exp_args:NcNo 1677 {
\exp_args:NNNV 1678   \exp_after:wN #1
1679   \exp_after:wN #2
1680   \exp_after:wN #3
1681   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #4 }
1682 }
1683 \cs_new:Npn \exp_args:NcNc #1#2#3#4
1684 {
1685   \exp_after:wN #1
1686   \cs:w #2 \exp_after:wN \cs_end:
1687   \exp_after:wN #3
1688   \cs:w #4 \cs_end:
1689 }
1690 \cs_new:Npn \exp_args:NcNo #1#2#3#4
1691 {
1692   \exp_after:wN #1
1693   \cs:w #2 \exp_after:wN \cs_end:
1694   \exp_after:wN #3
1695   \exp_after:wN {#4}
1696 }
1697 \cs_new:Npn \exp_args:Ncco #1#2#3#4
1698 {
1699   \exp_after:wN #1
1700   \cs:w #2 \exp_after:wN \cs_end:
1701   \cs:w #3 \exp_after:wN \cs_end:
1702   \exp_after:wN {#4}
1703 }

```

(End definition for `\exp_args:Ncco` and others. These functions are documented on page ??.)

186.3 Definitions with the automated technique

Some of these could be done more efficiently, but the complexity of coding then becomes an issue. Notice that the auto-generated functions are all not long: they don't actually take any arguments themselves.

`\exp_args:Nx`

```

1704 \cs_new_protected_nopar:Npn \exp_args:Nx { \::x \::: }

```

(End definition for `\exp_args:Nx`. This function is documented on page 27.)

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

```

\exp_args:Nnc 1705 \cs_new_nopar:Npn \exp_args:Nnc { \::n \::c \::: }
\exp_args:Ncx 1706 \cs_new_nopar:Npn \exp_args:Nfo { \::f \::o \::: }
\exp_args:Nfo 1707 \cs_new_nopar:Npn \exp_args:Nff { \::f \::f \::: }
\exp_args:Nff 1708 \cs_new_nopar:Npn \exp_args:Nnf { \::n \::f \::: }
\exp_args:Nnf 1709 \cs_new_nopar:Npn \exp_args:Nno { \::n \::o \::: }
\exp_args:Nno 1710 \cs_new_nopar:Npn \exp_args:NnV { \::n \::V \::: }
\exp_args:NnV 1711 \cs_new_nopar:Npn \exp_args:Noc { \::o \::c \::: }
\exp_args:NnV 1712 \cs_new_nopar:Npn \exp_args:Noo { \::o \::o \::: }
\exp_args:Nnx 1713 \cs_new_protected_nopar:Npn \exp_args:NNx { \::N \::x \::: }
\exp_args:Noo 1714 \cs_new_protected_nopar:Npn \exp_args:Ncx { \::c \::x \::: }
\exp_args:Noc 1715 \cs_new_protected_nopar:Npn \exp_args:Nnx { \::n \::x \::: }
\exp_args:Nox 1716 \cs_new_protected_nopar:Npn \exp_args:Nox { \::o \::x \::: }
\exp_args:Nxo 1717 \cs_new_protected_nopar:Npn \exp_args:Nxo { \::x \::o \::: }
\exp_args:Nxx 1718 \cs_new_protected_nopar:Npn \exp_args:Nxx { \::x \::x \::: }

```

(End definition for `\exp_args:NNx` and others. These functions are documented on page ??.)

```

\exp_args:Nccx
\exp_args:Ncnx 1719 \cs_new_nopar:Npn \exp_args:NNno { \::N \::n \::o \::: }
\exp_args:NNno 1720 \cs_new_nopar:Npn \exp_args:NNoo { \::N \::o \::o \::: }
\exp_args:Nnno 1721 \cs_new_nopar:Npn \exp_args:Nnnc { \::n \::n \::c \::: }
\exp_args:Nnnx 1722 \cs_new_nopar:Npn \exp_args:Nnno { \::n \::n \::o \::: }
\exp_args:Nnox 1723 \cs_new_nopar:Npn \exp_args:Nooo { \::o \::o \::o \::: }
\exp_args:Nooo 1724 \cs_new_protected_nopar:Npn \exp_args:NNnx { \::N \::n \::x \::: }
\exp_args:Noox 1725 \cs_new_protected_nopar:Npn \exp_args:NNox { \::N \::o \::x \::: }
\exp_args:Nnnc 1726 \cs_new_protected_nopar:Npn \exp_args:Nnnx { \::n \::n \::x \::: }
\exp_args:Nnnx 1727 \cs_new_protected_nopar:Npn \exp_args:Nnox { \::n \::o \::x \::: }
\exp_args:NNcx 1728 \cs_new_protected_nopar:Npn \exp_args:Nccx { \::c \::c \::x \::: }
\exp_args:NNoo 1729 \cs_new_protected_nopar:Npn \exp_args:Ncnx { \::c \::n \::x \::: }
\exp_args:NNox 1730 \cs_new_protected_nopar:Npn \exp_args:Noox { \::o \::o \::x \::: }

```

(End definition for `\exp_args:Nccx` and others. These functions are documented on page ??.)

186.4 Last-unbraced versions

`\exp_arg_last_unbraced:nn` There are a few places where the last argument needs to be available unbraced. First some helper macros.

```

\::f_unbraced
\::o_unbraced 1731 \cs_new:Npn \exp_arg_last_unbraced:nn #1#2 { #2#1 }
\::V_unbraced 1732 \cs_new:Npn \::f_unbraced \::: #1#2
\::v_unbraced 1733 {
\exp_after:wN \exp_arg_last_unbraced:nn
\exp_after:wN { \tex_romannumeral:D -'0 #2 } {#1}
1736 }
1737 \cs_new:Npn \::o_unbraced \::: #1#2
1738 { \exp_after:wN \exp_arg_last_unbraced:nn \exp_after:wN {#2} {#1} }
1739 \cs_new:Npn \::V_unbraced \::: #1#2
1740 {
1741 \exp_after:wN \exp_arg_last_unbraced:nn
1742 \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #2 } {#1}

```

```

1743 }
1744 \cs_new:Npn \::v_unbraced \::: #1#2
1745 {
1746   \exp_after:wN \exp_arg_last_unbraced:nn
1747   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#2} } {#1}
1748 }
1749 \cs_new_protected:Npn \::x_unbraced \::: #1#2
1750 {
1751   \cs_set_nopar:Npx \l_exp_tl { \exp_not:n {#1} #2 }
1752   \l_exp_tl
1753 }

```

(End definition for \exp_arg_last_unbraced:nn. This function is documented on page ??.)

\exp_last_unbraced:NV Now the business end: most of these are hand-tuned for speed, but the general system is in place.

```

\exp_last_unbraced:Nv
\exp_last_unbraced:Nf
\exp_last_unbraced:No
\exp_last_unbraced:Nco
\exp_last_unbraced:NcV
\exp_last_unbraced:NNV
\exp_last_unbraced:NNo
\exp_last_unbraced:NNNV
\exp_last_unbraced:NNNo
\exp_last_unbraced:Nno
\exp_last_unbraced:Noo
\exp_last_unbraced:Nfo
\exp_last_unbraced:Nx
1754 \cs_new:Npn \exp_last_unbraced:NV #1#2
1755 { \exp_after:wN #1 \tex_romannumeral:D \exp_eval_register:N #2 }
1756 \cs_new:Npn \exp_last_unbraced:Nv #1#2
1757 { \exp_after:wN #1 \tex_romannumeral:D \exp_eval_register:c {#2} }
1758 \cs_new:Npn \exp_last_unbraced:No #1#2 { \exp_after:wN #1 #2 }
1759 \cs_new:Npn \exp_last_unbraced:Nf #1#2
1760 { \exp_after:wN #1 \tex_romannumeral:D -'0 #2 }
1761 \cs_new:Npn \exp_last_unbraced:Nco #1#2#3
1762 { \exp_after:wN #1 \cs:w #2 \exp_after:wN \cs_end: #3 }
1763 \cs_new:Npn \exp_last_unbraced:NcV #1#2#3
1764 {
1765   \exp_after:wN #1
1766   \cs:w #2 \exp_after:wN \cs_end:
1767   \tex_romannumeral:D \exp_eval_register:N #3
1768 }
1769 \cs_new:Npn \exp_last_unbraced:NNV #1#2#3
1770 {
1771   \exp_after:wN #1
1772   \exp_after:wN #2
1773   \tex_romannumeral:D \exp_eval_register:N #3
1774 }
1775 \cs_new:Npn \exp_last_unbraced:NNo #1#2#3
1776 { \exp_after:wN #1 \exp_after:wN #2 #3 }
1777 \cs_new:Npn \exp_last_unbraced:NNNV #1#2#3#4
1778 {
1779   \exp_after:wN #1
1780   \exp_after:wN #2
1781   \exp_after:wN #3
1782   \tex_romannumeral:D \exp_eval_register:N #4
1783 }
1784 \cs_new:Npn \exp_last_unbraced:NNNo #1#2#3#4
1785 { \exp_after:wN #1 \exp_after:wN #2 \exp_after:wN #3 #4 }
1786 \cs_new_nopar:Npn \exp_last_unbraced:Nno { \::n \::o_unbraced \::: }
1787 \cs_new_nopar:Npn \exp_last_unbraced:Noo { \::o \::o_unbraced \::: }

```

```

1788 \cs_new_nopar:Npn \exp_last_unbraced:Nfo { \::f \::o_unbraced \::: }
1789 \cs_new_protected_nopar:Npn \exp_last_unbraced:Nx { \::x_unbraced \::: }
      (End definition for \exp_last_unbraced:NV. This function is documented on page 29.)

```

`\exp_last_two_unbraced:Noo` If #2 is a single token then this can be implemented as

```

\cs_new:Npn \exp_last_two_unbraced:Noo #1 #2 #3
  { \exp_after:wN \exp_after:wN \exp_after:wN #1 \exp_after:wN #2 #3 }

```

However, for robustness this is not suitable. Instead, a bit of a shuffle is used to ensure that #2 can be multiple tokens.

```

1790 \cs_new:Npn \exp_last_two_unbraced:Noo #1#2#3
1791   { \exp_after:wN \exp_last_two_unbraced_aux:noN \exp_after:wN {#3} {#2} #1 }
1792 \cs_new:Npn \exp_last_two_unbraced_aux:noN #1#2#3
1793   { \exp_after:wN #3 #2 #1 }
      (End definition for \exp_last_two_unbraced:Noo. This function is documented on page 29.)

```

186.5 Preventing expansion

```

\exp_not:o
\exp_not:f 1794 \cs_new:Npn \exp_not:o #1 { \etex_unexpanded:D \exp_after:wN {#1} }
\exp_not:V 1795 \cs_new:Npn \exp_not:f #1
\exp_not:v 1796   { \etex_unexpanded:D \exp_after:wN { \tex_romannumeral:D -‘0 #1 } }
1797 \cs_new:Npn \exp_not:V #1
1798   {
1799     \etex_unexpanded:D \exp_after:wN
1800     { \tex_romannumeral:D \exp_eval_register:N #1 }
1801   }
1802 \cs_new:Npn \exp_not:v #1
1803   {
1804     \etex_unexpanded:D \exp_after:wN
1805     { \tex_romannumeral:D \exp_eval_register:c {#1} }
1806   }
      (End definition for \exp_not:o. This function is documented on page 30.)

```

`\exp_not:c` A helper function.

```

1807 \cs_new:Npn \exp_not:c #1 { \exp_after:wN \exp_not:N \cs:w #1 \cs_end: }
      (End definition for \exp_not:c. This function is documented on page 30.)

```

186.6 Defining function variants

```

\cs_generate_variant:Nn #1 : Base form of a function; e.g., \tl_set:Nn
\cs_generate_variant_aux:nnNNn #2 : One or more variant argument specifiers; e.g., {Nx,c,cx}
\cs_generate_variant_aux:Nnnw Test whether the base function is protected or not and define \cs_tmp:w as either
\cs_generate_variant_aux:NNn \cs_new_nopar:Npx or \cs_new_protected_nopar:Npx, then used to define all the vari-
ants. Split up the original base function to grab its name and signature consisting of k
letters. Then we wish to iterate through the list of variant argument specifiers, and for

```

each one construct a new function name using the original base name, the variant signature consisting of l letters and the last $k - l$ letters of the base signature. For example, for a base function `\tl_set:Nn` which needs a `c` variant form, we want the new signature to be `cn`.

```

1808 \cs_new_protected:Npn \cs_generate_variant:Nn #1
1809 {
1810   \chk_if_exist_cs:N #1
1811   \cs_generate_variant_aux:N #1
1812   \cs_split_function:NN #1 \cs_generate_variant_aux:nnNNn
1813   #1
1814 }

```

We discard the boolean `#3` and then set off a loop through the desired variant forms. The original function is retained as `#4` for efficiency.

```

1815 \cs_new:Npn \cs_generate_variant_aux:nnNNn #1#2#3#4#5
1816 { \cs_generate_variant_aux:Nnw #4 {#1}{#2} #5 , ? , \q_recursion_stop }

```

Next is the real work to be done. We now have 1: original function, 2: base name, 3: base signature, 4: beginning of variant signature. To construct the new `csname` and the `\exp_args:Ncc` form, we need the variant signature. In our example, we wanted to discard the first two letters of the base signature because the variant form started with `cc`. This is the same as putting first `cc` in the signature and then `\use_none:nn` followed by the base signature `NNn`. Depending on the number of characters in `#4`, the relevant `\use_none:n...n` is called.

Firstly though, we check whether to terminate the loop. Then build the variant function once, to avoid repeating this relatively expensive operation. Then recurse.

```

1817 \cs_new:Npn \cs_generate_variant_aux:Nnw #1#2#3#4 ,
1818 {
1819   \if:w ? #4
1820   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
1821   \fi:
1822   \exp_args:NNc \cs_generate_variant_aux:NNn
1823   #1
1824   {
1825     #2 : #4
1826     \exp_after:wN \use_i_delimit_by_q_stop:nw
1827     \use_none:nnnnnnnnn #4
1828     \use_none:nnnnnnnnn
1829     \use_none:nnnnnnnnn
1830     \use_none:nnnnnnnn
1831     \use_none:nnnnnn
1832     \use_none:nnnnn
1833     \use_none:nnnn
1834     \use_none:nnn
1835     \use_none:nn
1836     \use_none:n
1837     { }
1838     \q_stop
1839     #3

```

```

1840     }
1841     {#4}
1842     \cs_generate_variant_aux:Nnnw #1 {#2} {#3}
1843 }

```

Check if the variant form has already been defined. If not, then define it and then additionally check if the `\exp_args:N` form needed is defined. Otherwise tell that it was already defined.

```

1844 \cs_new:Npn \cs_generate_variant_aux:NNn #1 #2 #3
1845 {
1846   \cs_if_free:NTF #2
1847   {
1848     \cs_tmp:w #2 { \exp_not:c { exp_args:N #3 } \exp_not:N #1 }
1849     \cs_generate_internal_variant:n {#3}
1850   }
1851   {
1852     \iow_log:x
1853     {
1854       Variant~\token_to_str:N #2~%
1855       already~defined;~ not~ changing~ it~on~line~%
1856       \tex_the:D \tex_inputlineno:D
1857     }
1858   }
1859 }

```

(End definition for \cs_generate_variant:Nn. This function is documented on page ??.)

`\cs_generate_variant_aux:N` The idea here is to pick up protected parent functions, using the nature of the meaning
`\cs_generate_variant_aux:w` string that they generate. The test here is almost the same as `\tl_if_empty:nTF`, but
 has to be hard-coded as that function is not yet available and because it has to match
 both long and short macros.

```

1860 \group_begin:
1861 \tex_lccode:D '\Z = '\d \scan_stop:
1862 \tex_lccode:D '\? ='\ \ \scan_stop:
1863 \tex_catcode:D '\P = 12 \scan_stop:
1864 \tex_catcode:D '\R = 12 \scan_stop:
1865 \tex_catcode:D '\O = 12 \scan_stop:
1866 \tex_catcode:D '\T = 12 \scan_stop:
1867 \tex_catcode:D '\E = 12 \scan_stop:
1868 \tex_catcode:D '\C = 12 \scan_stop:
1869 \tex_catcode:D '\Z = 12 \scan_stop:
1870 \tex_lowercase:D
1871 {
1872   \group_end:
1873   \cs_new:Npn \cs_generate_variant_aux:N #1
1874   {
1875     \exp_after:wN \cs_generate_variant_aux:w
1876     \token_to_meaning:N #1
1877     \q_mark \cs_new_protected_nopar:Npx
1878     ? PROTECTEZ

```



```

1879         \q_mark \cs_new_nopar:Npx
1880         \q_stop
1881     }
1882     \cs_new:Npn \cs_generate_variant_aux:w
1883     #1 ? PROTECTEZ #2 \q_mark #3 #4 \q_stop
1884     {
1885         \cs_set_eq:NN \cs_tmp:w #3
1886     }
1887 }

```

(End definition for \cs_generate_variant_aux:N. This function is documented on page ??.)

\cs_generate_internal_variant:n Test if `exp_args:N #1` is already defined and if not define it via the `\::` commands using the chars in `#1`

```

1888 \cs_new_protected:Npn \cs_generate_internal_variant:n #1
1889 {
1890     \cs_if_free:cT { exp_args:N #1 }
1891     {
1892         \cs_new:cpx { exp_args:N #1 }
1893         { \cs_generate_internal_variant_aux:N #1 : }
1894     }
1895 }

```

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.

```

1896 \cs_new:Npn \cs_generate_internal_variant_aux:N #1
1897 {
1898     \exp_not:c { :: #1 }
1899     \if_meaning:w : #1
1900     \exp_after:wN \use_none:n
1901     \fi:
1902     \cs_generate_internal_variant_aux:N
1903 }

```

(End definition for \cs_generate_internal_variant:n. This function is documented on page ??.)

186.7 Variants which cannot be created earlier

\str_if_eq:Vn These cannot come earlier as they need \cs_generate_variant:Nn.

```

\str_if_eq:on 1904 \cs_generate_variant:Nn \str_if_eq_p:nn { V , o }
\str_if_eq:nV 1905 \cs_generate_variant:Nn \str_if_eq_p:nn { nV , no , VV }
\str_if_eq:no 1906 \cs_generate_variant:Nn \str_if_eq:nnT { V , o }
\str_if_eq:VV 1907 \cs_generate_variant:Nn \str_if_eq:nnT { nV , no , VV }
1908 \cs_generate_variant:Nn \str_if_eq:nnF { V , o }
1909 \cs_generate_variant:Nn \str_if_eq:nnF { nV , no , VV }
1910 \cs_generate_variant:Nn \str_if_eq:nnTF { V , o }
1911 \cs_generate_variant:Nn \str_if_eq:nnTF { nV , no , VV }

```

(End definition for \str_if_eq:Vn and others. These functions are documented on page ??.)

```

1912 </initex | package>

```

187 l3prg implementation

The following test files are used for this code: *m3prg001.lvt, m3prg002.lvt, m3prg003.lvt.*

```
1913 <*initex | package>
1914 <*package>
1915 \ProvidesExplPackage
1916   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
1917 \package_check_loaded_expl:
1918 </package>
```

187.1 Primitive conditionals

`\if_bool:N` Those two primitive TeX conditionals are synonyms. They should not be used outside the kernel code.

```
1919 \tex_let:D \if_bool:N          \tex_ifodd:D
1920 \tex_let:D \if_predicate:w      \tex_ifodd:D
      (End definition for \if_bool:N. This function is documented on page 41.)
```

187.2 Defining a set of conditional functions

These are all defined in `l3basics`, as they are needed “early”. This is just a reminder that that is the case!

(End definition for `\prg_set_conditional:Npnn` and others. These functions are documented on page ??.)

187.3 The boolean data type

Boolean variables have to be initiated when they are created. Other than that there is not much to say here.

```
1921 \cs_new_protected:Npn \bool_new:N #1 { \cs_new_eq:NN #1 \c_false_bool }
1922 \cs_generate_variant:Nn \bool_new:N { c }
      (End definition for \bool_new:N and \bool_new:c. These functions are documented on page ??.)
```

Setting is already pretty easy.

```
1923 \cs_new_protected:Npn \bool_set_true:N #1
1924   { \cs_set_eq:NN #1 \c_true_bool }
1925 \cs_new_protected:Npn \bool_set_false:N #1
1926   { \cs_set_eq:NN #1 \c_false_bool }
1927 \cs_new_protected:Npn \bool_gset_true:N #1
1928   { \cs_gset_eq:NN #1 \c_true_bool }
1929 \cs_new_protected:Npn \bool_gset_false:N #1
1930   { \cs_gset_eq:NN #1 \c_false_bool }
1931 \cs_generate_variant:Nn \bool_set_true:N { c }
1932 \cs_generate_variant:Nn \bool_set_false:N { c }
1933 \cs_generate_variant:Nn \bool_gset_true:N { c }
1934 \cs_generate_variant:Nn \bool_gset_false:N { c }
      (End definition for \bool_set_true:N and others. These functions are documented on page ??.)
```

`\bool_set_eq:NN` The usual copy code.

```

\bool_set_eq:cN 1935 \cs_new_eq:NN \bool_set_eq:NN \cs_set_eq:NN
\bool_set_eq:Nc 1936 \cs_new_eq:NN \bool_set_eq:Nc \cs_set_eq:Nc
\bool_set_eq:cc 1937 \cs_new_eq:NN \bool_set_eq:cN \cs_set_eq:cN
\bool_gset_eq:NN 1938 \cs_new_eq:NN \bool_set_eq:cc \cs_set_eq:cc
\bool_gset_eq:cN 1939 \cs_new_eq:NN \bool_gset_eq:NN \cs_gset_eq:NN
\bool_gset_eq:Nc 1940 \cs_new_eq:NN \bool_gset_eq:Nc \cs_gset_eq:Nc
\bool_gset_eq:cN 1941 \cs_new_eq:NN \bool_gset_eq:cN \cs_gset_eq:cN
\bool_gset_eq:cc 1942 \cs_new_eq:NN \bool_gset_eq:cc \cs_gset_eq:cc

```

(End definition for \bool_set_eq:NN and others. These functions are documented on page ??.)

`\bool_set:Nn` This function evaluates a boolean expression and assigns the first argument the meaning
`\bool_set:cn` `\c_true_bool` or `\c_false_bool`.

```

\bool_gset:Nn 1943 \cs_new:Npn \bool_set:Nn #1#2
\bool_gset:cn 1944 { \tex_chardef:D #1 = \bool_if_p:n {#2} }
1945 \cs_new:Npn \bool_gset:Nn #1#2
1946 { \tex_global:D \tex_chardef:D #1 = \bool_if_p:n {#2} }
1947 \cs_generate_variant:Nn \bool_set:Nn { c }
1948 \cs_generate_variant:Nn \bool_gset:Nn { c }

```

`\bool_if:N` Straight forward here. We could optimize here if we wanted to as the boolean can just
`\bool_if:c` be input directly.

```

1949 \prg_new_conditional:Npnn \bool_if:N #1 { p , T , F , TF }
1950 {
1951   \if_meaning:w \c_true_bool #1
1952   \prg_return_true:
1953   \else:
1954   \prg_return_false:
1955   \fi:
1956 }
1957 \cs_generate_variant:Nn \bool_if_p:N { c }
1958 \cs_generate_variant:Nn \bool_if:NT { c }
1959 \cs_generate_variant:Nn \bool_if:NF { c }
1960 \cs_generate_variant:Nn \bool_if:NTF { c }

```

(End definition for \bool_set:Nn and \bool_set:cn. These functions are documented on page ??.)

`\l_tmpa_bool` A few booleans just if you need them.
`\g_tmpa_bool`

```

1961 \bool_new:N \l_tmpa_bool
1962 \bool_new:N \g_tmpa_bool

```

(End definition for \l_tmpa_bool and \g_tmpa_bool. These functions are documented on page 36.)

187.4 Boolean expressions

`\bool_if: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 with (and) for grouping, ! for logical “Not”, && for logical “And” and || for logical “Or”. We shall use the terms Not, And, Or, Open and Close for these operations.

`\bool_!:w` Any expression is terminated by a Close operation. Evaluation happens from left to right in the following manner using a GetNext function:

- `\bool_Not:w`
 - If an Open is seen, start evaluating a new expression using the Eval function and call GetNext again.
- `\bool_(:w`
 - If a Not is seen, insert a negating function (if-even in this case) and call GetNext.
- `\bool_p:w`
 - If none of the above, start evaluating a new expression by reinserting the token found (this is supposed to be a predicate function) in front of Eval.

`\bool_8_1:w` The Eval function then contains a post-processing operation which grabs the instruction following the predicate. This is either And, Or or Close. In each case the truth value is used to determine where to go next. The following situations can arise:

`\bool_I_1:w` **<true>And** Current truth value is true, logical And seen, continue with GetNext to examine truth value of next boolean (sub-)expression.

`\bool_8_0:w` **<false>And** Current truth value is false, logical And seen, stop evaluating the predicates within this sub-expression and break to the nearest Close. Then return **<false>**.

`\bool_I_0:w` **<true>Or** Current truth value is true, logical Or seen, stop evaluating the predicates within this sub-expression and break to the nearest Close. Then return **<true>**.

`\bool_)_0:w` **<false>Or** Current truth value is false, logical Or seen, continue with GetNext to examine truth value of next boolean (sub-)expression.

`\bool_)_1:w` **<true>Close** Current truth value is true, Close seen, return **<true>**.

`\bool_S_0:w` **<false>Close** Current truth value is false, Close seen, return **<false>**.

`\bool_S_1:w` We introduce an additional Stop operation with the following semantics:

<true>Stop Current truth value is true, return **<true>**.

<false>Stop Current truth value is false, return **<false>**.

The reasons for this follow below.

Now for how these works in practice. The canonical true and false values have numerical values 1 and 0 respectively. We evaluate this using the primitive `\int_value:w:D` operation. First we issue a `\group_align_safe_begin:` as we are using && as syntax shorthand for the And operation and we need to hide it for T_EX. We also need to finish this special group before finally returning a `\c_true_bool` or `\c_false_bool` as there might otherwise be something left in front in the input stream. For this we call the Stop operation, denoted simply by a S following the last Close operation.

```

1963 \prg_new_conditional:Npnn \bool_if:n #1 { T , F , TF }
1964 {
1965   \if_predicate:w \bool_if_p:n {#1}
1966   \prg_return_true:
1967   \else:
1968     \prg_return_false:
1969   \fi:
1970 }
1971 \cs_new:Npn \bool_if_p:n #1
1972 {
1973   \group_align_safe_begin:
1974   \bool_get_next:N ( #1 ) S
1975 }

```

The GetNext operation. We make it a switch: If not a ! or (, we assume it is a predicate.

```

1976 \cs_new:Npn \bool_get_next:N #1
1977 {
1978   \use:c
1979   {
1980     bool_
1981     \if_meaning:w !#1 ! \else: \if_meaning:w (#1 ( \else: p \fi: \fi:
1982     :w
1983   }
1984   #1
1985 }

```

This variant gets called when a Not has just been entered. It (eventually) results in a reversal of the logic of the directly following material.

```

1986 \cs_new:Npn \bool_get_not_next:N #1
1987 {
1988   \use:c
1989   {
1990     bool_not_
1991     \if_meaning:w !#1 ! \else: \if_meaning:w (#1 ( \else: p \fi: \fi:
1992     :w
1993   }
1994   #1
1995 }

```

We need these later on to nullify the unity operation !!.

```

1996 \cs_new:Npn \bool_get_next:NN #1#2 { \bool_get_next:N #2 }
1997 \cs_new:Npn \bool_get_not_next:NN #1#2 { \bool_get_not_next:N #2 }

```

The Not operation. Discard the token read and reverse the truth value of the next expression if there are brackets; otherwise if we're coming up to a ! then we don't need to reverse anything (but we then want to continue scanning ahead in case some fool has written !(...)); otherwise we have a boolean that we can reverse here and now.

```

1998 \cs_new:cpn { bool_!:w } #1#2
1999 {
2000   \if_meaning:w ( #2
2001     \exp_after:wN \bool_Not:w

```

```

2002     \else:
2003         \if_meaning:w ! #2
2004         \exp_after:wN \exp_after:wN \exp_after:wN \bool_get_next:NN
2005     \else:
2006         \exp_after:wN \exp_after:wN \exp_after:wN \bool_Not:N
2007     \fi:
2008 \fi:
2009 #2
2010 }

```

Variant called when already inside a Not. Essentially the opposite of the above.

```

2011 \cs_new:cpn { bool_not_!:w } #1#2
2012 {
2013     \if_meaning:w ( #2
2014     \exp_after:wN \bool_not_Not:w
2015 \else:
2016     \if_meaning:w ! #2
2017     \exp_after:wN \exp_after:wN \exp_after:wN \bool_get_not_next:NN
2018 \else:
2019     \exp_after:wN \exp_after:wN \exp_after:wN \bool_not_Not:N
2020 \fi:
2021 \fi:
2022 #2
2023 }

```

These occur when processing !(...). The idea is to use a variant of \bool_get_next:N that finishes its parsing with a logic reversal. Of course, the double logic reversal gets us back to where we started.

```

2024 \cs_new:Npn \bool_Not:w { \exp_after:wN \int_value:w \bool_get_not_next:N }
2025 \cs_new:Npn \bool_not_Not:w { \exp_after:wN \int_value:w \bool_get_next:N }

```

These occur when processing !<bool> and can be evaluated directly.

```

2026 \cs_new:Npn \bool_Not:N #1
2027 {
2028     \exp_after:wN \bool_p:w
2029     \if_meaning:w #1 \c_true_bool
2030     \c_false_bool
2031 \else:
2032     \c_true_bool
2033 \fi:
2034 }
2035 \cs_new:Npn \bool_not_Not:N #1
2036 {
2037     \exp_after:wN \bool_p:w
2038     \if_meaning:w #1 \c_true_bool
2039     \c_true_bool
2040 \else:
2041     \c_false_bool
2042 \fi:
2043 }

```

The Open operation. Discard the token read and start a sub-expression. `\bool_get_next:N` continues building up the logical expressions as usual; `\bool_not_cleanup:N` is what reverses the logic if we're inside `!(...)`.

```

2044 \cs_new:cpn { bool_( :w } #1
2045 { \exp_after:wN \bool_cleanup:N \int_value:w \bool_get_next:N }
2046 \cs_new:cpn { bool_not_( :w } #1
2047 { \exp_after:wN \bool_not_cleanup:N \int_value:w \bool_get_next:N }

```

Otherwise just evaluate the predicate and look for And, Or or Close afterwards.

```

2048 \cs_new:cpn { bool_p:w } { \exp_after:wN \bool_cleanup:N \int_value:w }
2049 \cs_new:cpn { bool_not_p:w } { \exp_after:wN \bool_not_cleanup:N \int_value:w }

```

This cleanup function can be omitted once predicates return their true/false booleans outside the conditionals.

```

2050 \cs_new:Npn \bool_cleanup:N #1
2051 {
2052   \exp_after:wN \bool_choose:NN \exp_after:wN #1
2053   \int_to_roman:w - '\q
2054 }
2055 \cs_new:Npn \bool_not_cleanup:N #1
2056 {
2057   \exp_after:wN \bool_not_choose:NN \exp_after:wN #1
2058   \int_to_roman:w - '\q
2059 }

```

Branching the six way switch. Reversals should be reasonably straightforward.

```

2060 \cs_new:Npn \bool_choose:NN #1#2 { \use:c { bool_ #2 _ #1 :w } }
2061 \cs_new:Npn \bool_not_choose:NN #1#2 { \use:c { bool_not_ #2 _ #1 :w } }

```

Continues scanning. Must remove the second `&` or `|`.

```

2062 \cs_new_nopar:cpn { bool_&_1:w } & { \bool_get_next:N }
2063 \cs_new_nopar:cpn { bool_|_0:w } | { \bool_get_next:N }
2064 \cs_new_nopar:cpn { bool_not_&_0:w } & { \bool_get_next:N }
2065 \cs_new_nopar:cpn { bool_not_|_1:w } | { \bool_get_next:N }

```

Closing a group is just about returning the result. The Stop operation is similar except it closes the special alignment group before returning the boolean.

```

2066 \cs_new_nopar:cpn { bool_)_0:w } { \c_false_bool }
2067 \cs_new_nopar:cpn { bool_)_1:w } { \c_true_bool }
2068 \cs_new_nopar:cpn { bool_not_)_0:w } { \c_true_bool }
2069 \cs_new_nopar:cpn { bool_not_)_1:w } { \c_false_bool }
2070 \cs_new_nopar:cpn { bool_S_0:w } { \group_align_safe_end: \c_false_bool }
2071 \cs_new_nopar:cpn { bool_S_1:w } { \group_align_safe_end: \c_true_bool }

```

When the truth value has already been decided, we have to throw away the remainder of the current group as we are doing minimal evaluation. This is slightly tricky as there are no braces so we have to play match the `()` manually.

```

2072 \cs_new_nopar:cpn { bool_&_0:w } & { \bool_eval_skip_to_end:Nw \c_false_bool }
2073 \cs_new_nopar:cpn { bool_|_1:w } | { \bool_eval_skip_to_end:Nw \c_true_bool }
2074 \cs_new_nopar:cpn { bool_not_&_1:w } &
2075 { \bool_eval_skip_to_end:Nw \c_false_bool }

```

```

2076 \cs_new_nopar:cpn { bool_not_|_0:w } |
2077 { \bool_eval_skip_to_end:Nw \c_true_bool }

```

There is always at least one `)` waiting, namely the outer one. However, we are facing the problem that there may be more than one that need to be finished off and we have to detect the correct number of them. Here is a complicated example showing how this is done. After evaluating the following, we realize we must skip everything after the first `And`. Note the extra `Close` at the end.

```
\c_false_bool && ((abc) && xyz) && ((xyz) && (def)))
```

First read up to the first `Close`. This gives us the list we first read up until the first right parenthesis so we are looking at the token list

```
((abc
```

This contains two `Open` markers so we must remove two groups. Since no evaluation of the contents is to be carried out, it doesn't matter how we remove the groups as long as we wind up with the correct result. We therefore first remove a `()` pair and what preceded the `Open` – but leave the contents as it may contain `Open` tokens itself – leaving

```
(abc && xyz) && ((xyz) && (def)))
```

Another round of this gives us

```
(abc && xyz
```

which still contains an `Open` so we remove another `()` pair, giving us

```
abc && xyz && ((xyz) && (def)))
```

Again we read up to a `Close` and again find `Open` tokens:

```
abc && xyz && ((xyz
```

Further reduction gives us

```
(xyz && (def)))
```

and then

```
(xyz && (def
```

with reduction to

```
xyz && (def))
```

and ultimately we arrive at no `Open` tokens being skipped and we can finally close the group nicely.

```

2078 %% (
2079 \cs_new:Npn \bool_eval_skip_to_end:Nw #1#2 )
2080 {
2081   \bool_eval_skip_to_end_aux:Nw #1#2 ( % )
2082   \q_no_value \q_stop
2083   {#2}
2084 }

```


If no right parenthesis, then #3 is no_value and we are done, return the boolean #1. If there is, we need to grab a () pair and then recurse

```

2085 \cs_new:Npn \bool_eval_skip_to_end_aux:Nw #1#2 ( #3#4 \q_stop #5 % )
2086 {
2087   \quark_if_no_value:NTF #3
2088   {#1}
2089   { \bool_eval_skip_to_end_aux_ii:Nw #1 #5 }
2090 }

```

Keep the boolean, throw away anything up to the (as it is irrelevant, remove a () pair but remember to reinsert #3 as it may contain (tokens!

```

2091 \cs_new:Npn \bool_eval_skip_to_end_aux_ii:Nw #1#2 ( #3 )
2092 { % (
2093   \bool_eval_skip_to_end:Nw #1#3 )
2094 }

```

`\bool_not_p:n` The Not variant just reverses the outcome of `\bool_if_p:n`. Can be optimized but this is nice and simple and according to the implementation plan. Not even particularly useful to have it when the infix notation is easier to use.

```

2095 \cs_new:Npn \bool_not_p:n #1 { \bool_if_p:n { ! ( #1 ) } }

```

`\bool_xor_p:nn` Exclusive or. If the boolean expressions have same truth value, return false, otherwise return true.

```

2096 \cs_new:Npn \bool_xor_p:nn #1#2
2097 {
2098   \int_compare:nNnTF { \bool_if_p:n {#1} } = { \bool_if_p:n {#2} }
2099   \c_false_bool
2100   \c_true_bool
2101 }

```

187.5 Logical loops

`\bool_while_do:Nn` A while loop where the boolean is tested before executing the statement. The “while” version executes the code as long as the boolean is true; the “until” version executes the code as long as the boolean is false.

```

\bool_while_do:cn
\bool_until_do:Nn
\bool_until_do:cn
2102 \cs_new:Npn \bool_while_do:Nn #1#2
2103 { \bool_if:NT #1 { #2 \bool_while_do:Nn #1 {#2} } }
2104 \cs_new:Npn \bool_until_do:Nn #1#2
2105 { \bool_if:NF #1 { #2 \bool_until_do:Nn #1 {#2} } }
2106 \cs_generate_variant:Nn \bool_while_do:Nn { c }
2107 \cs_generate_variant:Nn \bool_until_do:Nn { c }

```

`\bool_do_while:Nn` A do-while loop where the body is performed at least once and the boolean is tested after executing the body. Otherwise identical to the above functions.

```

\bool_do_while:cn
\bool_do_until:Nn
\bool_do_until:cn
2108 \cs_new:Npn \bool_do_while:Nn #1#2
2109 { #2 \bool_if:NT #1 { \bool_do_while:Nn #1 {#2} } }
2110 \cs_new:Npn \bool_do_until:Nn #1#2
2111 { #2 \bool_if:NF #1 { \bool_do_until:Nn #1 {#2} } }

```

```

2112 \cs_generate_variant:Nn \bool_do_while:Nn { c }
2113 \cs_generate_variant:Nn \bool_do_until:Nn { c }

```

\bool_while_do:nn Loop functions with the test either before or after the first body expansion.

```

\bool_do_while:nn 2114 \cs_new:Npn \bool_while_do:nn #1#2
\bool_until_do:nn 2115 {
\bool_do_until:nn 2116   \bool_if:nT {#1}
2117   {
2118     #2
2119     \bool_while_do:nn {#1} {#2}
2120   }
2121 }
2122 \cs_new:Npn \bool_do_while:nn #1#2
2123 {
2124   #2
2125   \bool_if:nT {#1} { \bool_do_while:nn {#1} {#2} }
2126 }
2127 \cs_new:Npn \bool_until_do:nn #1#2
2128 {
2129   \bool_if:nF {#1}
2130   {
2131     #2
2132     \bool_until_do:nn {#1} {#2}
2133   }
2134 }
2135 \cs_new:Npn \bool_do_until:nn #1#2
2136 {
2137   #2
2138   \bool_if:nF {#1} { \bool_do_until:nn {#1} {#2} }
2139 }

```

187.6 Switching by case

A family of functions to select one case of a number: the same ideas are used for a number of different situations.

\prg_case_end:nw In all cases the end statement is the same. Here, #1 will be the code needed, #2 the other cases to throw away, including the “else” case. The \c_zero marker stops the expansion of \romannumeral which begins each \prg_case_... function.

```

2140 \cs_new:Npn \prg_case_end:nw #1 #2 \q_recursion_stop { \c_zero #1 }

```

\prg_case_int:nnn For integer cases, the first task is to fully expand the check condition. After that, a loop is started to compare each possible value and stop if the test is true. The tested value is put at the end to ensure that there is necessarily a match, which will fire the “else” pathway. The leading \romannumeral triggers an expansion which is then stopped in \prg_case_end:nw.

```

2141 \cs_new:Npn \prg_case_int:nnn #1
2142 {

```

```

2143 \tex_romannumeral:D
2144 \exp_args:Nf \prg_case_int_aux:nnn { \int_eval:n {#1} }
2145 }
2146 \cs_new:Npn \prg_case_int_aux:nnn #1 #2 #3
2147 { \prg_case_int_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop }
2148 \cs_new:Npn \prg_case_int_aux:nw #1#2#3
2149 {
2150 \int_compare:nNnTF {#1} = {#2}
2151 { \prg_case_end:nw {#3} }
2152 { \prg_case_int_aux:nw {#1} }
2153 }

```

\prg_case_dim:nnn The dimension function is the same, just a change of calculation method.
\prg_case_dim_aux:nnn
\prg_case_dim_aux:nw

```

2154 \cs_new:Npn \prg_case_dim:nnn #1
2155 {
2156 \tex_romannumeral:D
2157 \exp_args:Nf \prg_case_dim_aux:nnn { \dim_eval:n {#1} }
2158 }
2159 \cs_new:Npn \prg_case_dim_aux:nnn #1 #2 #3
2160 { \prg_case_dim_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop }
2161 \cs_new:Npn \prg_case_dim_aux:nw #1#2#3
2162 {
2163 \dim_compare:nNnTF {#1} = {#2}
2164 { \prg_case_end:nw {#3} }
2165 { \prg_case_dim_aux:nw {#1} }
2166 }

```

\prg_case_str:nnn No calculations for strings, otherwise no surprises.
\prg_case_str:onn
\prg_case_str:xxn
\prg_case_str_aux:nw
\prg_case_str_x_aux:nw

```

2167 \cs_new:Npn \prg_case_str:nnn #1#2#3
2168 {
2169 \tex_romannumeral:D
2170 \prg_case_str_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop
2171 }
2172 \cs_new:Npn \prg_case_str_aux:nw #1#2#3
2173 {
2174 \str_if_eq:nnTF {#1} {#2}
2175 { \prg_case_end:nw {#3} }
2176 { \prg_case_str_aux:nw {#1} }
2177 }
2178 \cs_generate_variant:Nn \prg_case_str:nnn { o }
2179 \cs_new:Npn \prg_case_str:xxn #1#2#3
2180 {
2181 \tex_romannumeral:D
2182 \prg_case_str_x_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop
2183 }
2184 \cs_new:Npn \prg_case_str_x_aux:nw #1#2#3
2185 {
2186 \str_if_eq:xxTF {#1} {#2}
2187 { \prg_case_end:nw {#3} }

```

```

2188     { \prg_case_str_x_aux:nw {#1} }
2189   }

```

Similar again, but this time with some variants.

```

\prg_case_tl:Nnn
\prg_case_tl:cnn
\prg_case_tl_aux:Nw
2190 \cs_new:Npn \prg_case_tl:Nnn #1#2#3
2191 {
2192   \tex_romannumeral:D
2193   \prg_case_tl_aux:Nw #1 #2 #1 {#3} \q_recursion_stop
2194 }
2195 \cs_new:Npn \prg_case_tl_aux:Nw #1#2#3
2196 {
2197   \tl_if_eq:NNTF #1 #2
2198   { \prg_case_end:nw {#3} }
2199   { \prg_case_tl_aux:Nw #1 }
2200 }
2201 \cs_generate_variant:Nn \prg_case_tl:Nnn { c }

```

187.7 Producing n copies

This function uses a cascading csname technique by David Kastrup (who else :-)

The idea is to make the input 25 result in first adding five, and then 20 copies of the 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. These functions also close the expansion of `\int_to_roman:w`, which ensures that `\prg_replicate:nn` only requires two steps of expansion.

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, and if necessary, it is possible to write `\prg_replicate:nn{1000}{\prg_replicate:nn{1000}{\code}}`. 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.

```

\prg_replicate:nn
\prg_replicate_aux:N
\prg_replicate_first_aux:N
\prg_replicate_
\prg_replicate_0:n
\prg_replicate_1:n
\prg_replicate_2:n
\prg_replicate_3:n
\prg_replicate_4:n
\prg_replicate_5:n
\prg_replicate_6:n
\prg_replicate_7:n
\prg_replicate_8:n
\prg_replicate_9:n
\prg_replicate_first_-:n
\prg_replicate_first_0:n
\prg_replicate_first_1:n
\prg_replicate_first_2:n
\prg_replicate_first_3:n
\prg_replicate_first_4:n
\prg_replicate_first_5:n
\prg_replicate_first_6:n
\prg_replicate_first_7:n
\prg_replicate_first_8:n
\prg_replicate_first_9:n
2202 \cs_new:Npn \prg_replicate:nn #1
2203 {
2204   \int_to_roman:w
2205   \exp_after:wN \prg_replicate_first_aux:N
2206   \int_value:w \int_eval:w #1 \int_eval_end:
2207   \cs_end:
2208 }
2209 \cs_new:Npn \prg_replicate_aux:N #1

```

```

2210 { \cs:w prg_replicate_#1 :n \prg_replicate_aux:N }
2211 \cs_new:Npn \prg_replicate_first_aux:N #1
2212 { \cs:w prg_replicate_first_#1 :n \prg_replicate_aux:N }

```

Then comes all the functions that do the hard work of inserting all the copies.

```

2213 \cs_new:Npn \prg_replicate_ :n #1 { \cs_end: }
2214 \cs_new:cpn { prg_replicate_0:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } }
2215 \cs_new:cpn { prg_replicate_1:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1 }
2216 \cs_new:cpn { prg_replicate_2:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1 }
2217 \cs_new:cpn { prg_replicate_3:n } #1
2218 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1 }
2219 \cs_new:cpn { prg_replicate_4:n } #1
2220 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1 }
2221 \cs_new:cpn { prg_replicate_5:n } #1
2222 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1 }
2223 \cs_new:cpn { prg_replicate_6:n } #1
2224 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1#1 }
2225 \cs_new:cpn { prg_replicate_7:n } #1
2226 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1#1#1 }
2227 \cs_new:cpn { prg_replicate_8:n } #1
2228 { \cs_end: {#1#1#1#1#1#1#1#1#1#1 } #1#1#1#1#1#1#1#1 }
2229 \cs_new:cpn { prg_replicate_9:n } #1
2230 { \cs_end: {#1#1#1#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...

```

2231 \cs_new:cpn { prg_replicate_first_--:n } #1
2232 { \c_zero \msg_expandable_kernel_error:nn { prg } { replicate-neg } }
2233 \cs_new:cpn { prg_replicate_first_0:n } #1 { \c_zero }
2234 \cs_new:cpn { prg_replicate_first_1:n } #1 { \c_zero #1 }
2235 \cs_new:cpn { prg_replicate_first_2:n } #1 { \c_zero #1#1 }
2236 \cs_new:cpn { prg_replicate_first_3:n } #1 { \c_zero #1#1#1 }
2237 \cs_new:cpn { prg_replicate_first_4:n } #1 { \c_zero #1#1#1#1 }
2238 \cs_new:cpn { prg_replicate_first_5:n } #1 { \c_zero #1#1#1#1#1 }
2239 \cs_new:cpn { prg_replicate_first_6:n } #1 { \c_zero #1#1#1#1#1#1 }
2240 \cs_new:cpn { prg_replicate_first_7:n } #1 { \c_zero #1#1#1#1#1#1#1 }
2241 \cs_new:cpn { prg_replicate_first_8:n } #1 { \c_zero #1#1#1#1#1#1#1#1 }
2242 \cs_new:cpn { prg_replicate_first_9:n } #1 { \c_zero #1#1#1#1#1#1#1#1#1 }

```

(End definition for \bool_if:n. This function is documented on page ??.)

```

\prg_stepwise_function:nnnN
\prg_stepwise_aux:nnnN
\prg_stepwise_aux:NnnnN

```

Repeating a function by steps first needs a check on the direction of the steps. After that, do the function for the start value then step and loop around. It would be more symmetrical to test for a step size of zero before checking the sign, but we optimize for the most frequent case (positive step).

```

2243 \cs_new:Npn \prg_stepwise_function:nnnN #1#2#3#4
2244 {
2245   \prg_stepwise_aux:nnnN {#1} {#2} {#3} #4
2246   \prg_break_point:n { }
2247 }
2248 \cs_new:Npn \prg_stepwise_aux:nnnN #1#2#3#4
2249 {

```

```

2250 \int_compare:nNnTF {#2} > \c_zero
2251 { \exp_args:NNf \prg_stepwise_aux:NnnnN > }
2252 {
2253   \int_compare:nNnTF {#2} = \c_zero
2254   {
2255     \msg_expandable_kernel_error:nnn { prg } { zero-step } {#4}
2256     \prg_map_break:
2257   }
2258   { \exp_args:NNf \prg_stepwise_aux:NnnnN < }
2259 }
2260 { \int_eval:n {#1} } {#2} {#3} #4
2261 }
2262 \cs_new:Npn \prg_stepwise_aux:NnnnN #1#2#3#4#5
2263 {
2264   \int_compare:nNnF {#2} #1 {#4}
2265   {
2266     #5 {#2}
2267     \exp_args:NNf \prg_stepwise_aux:NnnnN
2268     #1 { \int_eval:n { #2 + #3 } } {#3} {#4} #5
2269   }
2270 }

```

(End definition for \prg_stepwise_function:nnnN. This function is documented on page ??.)

\prg_stepwise_inline:nnnn The approach here is to build a function, with a global integer required to make the
\prg_stepwise_variable:nnnN nesting safe (as seen in other in line functions), and map that function using \prg_
\prg_stepwise_aux:NNnnnn stepwise_function:nnnN.

```

2271 \cs_new_protected:Npn \prg_stepwise_inline:nnnn
2272 {
2273   \exp_args:NNc \prg_stepwise_aux:NNnnnn
2274   \cs_gset_nopar:Npn
2275   { g_prg_stepwise_ \int_use:N \g_prg_map_int :n }
2276 }
2277 \cs_new_protected:Npn \prg_stepwise_variable:nnnN #1#2#3#4#5
2278 {
2279   \exp_args:NNc \prg_stepwise_aux:NNnnnn
2280   \cs_gset_nopar:Npx
2281   { g_prg_stepwise_ \int_use:N \g_prg_map_int :n }
2282   {#1}{#2}{#3}
2283   {
2284     \tl_set:Nn \exp_not:N #4 {##1}
2285     \exp_not:n {#5}
2286   }
2287 }
2288 \cs_new_protected:Npn \prg_stepwise_aux:NNnnnn #1#2#3#4#5#6
2289 {
2290   #1 #2 ##1 {#6}
2291   \int_gincr:N \g_prg_map_int
2292   \prg_stepwise_aux:nnnN {#3} {#4} {#5} #2
2293   \prg_break_point:n { \int_gdecr:N \g_prg_map_int }

```

```

2294 }
      (End definition for \prg_stepwise_inline:nnnn. This function is documented on page ??.)

```

187.8 Detecting T_EX's mode

`\mode_if_vertical`: For testing vertical mode. Strikes me here on the bus with David, that as long as we are just talking about returning true and false states, we can just use the primitive conditionals for this and gobbling the `\c_zero` in the input stream. However this requires knowledge of the implementation so we keep things nice and clean and use the return statements.

```

2295 \prg_new_conditional:Npnn \mode_if_vertical: { p , T , F , TF }
2296 { \if_mode_vertical: \prg_return_true: \else: \prg_return_false: \fi: }
      (End definition for \mode_if_vertical:. This function is documented on page ??.)

```

`\mode_if_horizontal`: For testing horizontal mode.

```

2297 \prg_new_conditional:Npnn \mode_if_horizontal: { p , T , F , TF }
2298 { \if_mode_horizontal: \prg_return_true: \else: \prg_return_false: \fi: }
      (End definition for \mode_if_horizontal:. This function is documented on page ??.)

```

`\mode_if_inner`: For testing inner mode.

```

2299 \prg_new_conditional:Npnn \mode_if_inner: { p , T , F , TF }
2300 { \if_mode_inner: \prg_return_true: \else: \prg_return_false: \fi: }
      (End definition for \mode_if_inner:. This function is documented on page ??.)

```

`\mode_if_math`: For testing math mode. At the beginning of an alignment cell, the programmer should insert `\scan_align_safe_stop`: before the test.

```

2301 \prg_new_conditional:Npnn \mode_if_math: { p , T , F , TF }
2302 { \if_mode_math: \prg_return_true: \else: \prg_return_false: \fi: }
      (End definition for \mode_if_math:. This function is documented on page ??.)

```

187.9 Internal programming functions

`\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 `\futurelet` 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*... We place the `\if_false: { \fi: }` part at that place so that the successive expansions of `\group_align_safe_begin/end`: are always brace balanced.

```

2303 \cs_new_nopar:Npn \group_align_safe_begin:

```

```

2304 { \if_int_compare:w \if_false: { \fi: ' } = \c_zero \fi: }
2305 \cs_new_nopar:Npn \group_align_safe_end:
2306 { \if_int_compare:w '{ = \c_zero } \fi: }

```

(End definition for `\group_align_safe_begin:` and `\group_align_safe_end:`. These functions are documented on page ??.)

`\scan_align_safe_stop:` When TeX 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 `\omit` or `\noalign` and hasn't looked at the preamble yet. Thus an `\ifmmode` test will always fail unless we insert `\scan_stop:` to stop TeX's scanning ahead. On the other hand we don't want to insert a `\scan_stop:` every time as that will destroy kerning between letters⁴ 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 if an only if 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. Thus an older definition here was

```

\cs_new_nopar:Npn \scan_align_safe_stop:
{
  \int_compare:nNnT \etex_currentgrouptype:D = \c_six
  {
    \int_compare:nNnF \etex_lastnodetype:D = \c_zero
    {
      \int_compare:nNnF \etex_lastnodetype:D = \c_seven
      { \scan_stop: }
    }
  }
}

```

However, this is not truly expandable, as there are places where the `\scan_stop:` ends up in the result. A simpler alternative, which can be used selectively, is therefore defined.

```

2307 \cs_new_protected_nopar:Npn \scan_align_safe_stop: { }

```

(End definition for `\scan_align_safe_stop:`. This function is documented on page ??.)

`\prg_variable_get_scope:N` Expandable functions to find the type of a variable, and to return `g` if the variable is global. The trick for `\prg_variable_get_scope:N` is the same as that in `\cs_split_function:NN`, but it can be simplified as the requirements here are less complex.

```

2308 \group_begin:
2309 \tex_lccode:D '\& = '\g \scan_stop:
2310 \tex_catcode:D '\& = \c_twelve
2311 \tl_to_lowercase:n
2312 {
2313   \group_end:
2314   \cs_new:Npn \prg_variable_get_scope:N #1
2315   {

```

⁴Unless we enforce an extra pass with an appropriate value of `\pretolerance`.


```

2316         \exp_after:wN \exp_after:wN
2317         \exp_after:wN \prg_variable_get_scope_aux:w
2318         \cs_to_str:N #1 \exp_stop_f: \q_stop
2319     }
2320     \cs_new:Npn \prg_variable_get_scope_aux:w #1#2 \q_stop
2321     { \token_if_eq_meaning:NNT & #1 { g } }
2322 }
2323 \group_begin:
2324     \tex_lccode:D '\& = '\_ \scan_stop:
2325     \tex_catcode:D '\& = \c_twelve
2326     \tl_to_lowercase:n
2327     {
2328         \group_end:
2329         \cs_new:Npn \prg_variable_get_type:N #1
2330         {
2331             \exp_after:wN \prg_variable_get_type_aux:w
2332             \token_to_str:N #1 & a \q_stop
2333         }
2334         \cs_new:Npn \prg_variable_get_type_aux:w #1 & #2#3 \q_stop
2335         {
2336             \token_if_eq_meaning:NNTF a #2
2337             {#1}
2338             { \prg_variable_get_type_aux:w #2#3 \q_stop }
2339         }
2340     }

```

(End definition for \prg_variable_get_scope:N. This function is documented on page ??.)

`\g_prg_map_int` A nesting counter for mapping.

```

2341 \int_new:N \g_prg_map_int

```

(End definition for \g_prg_map_int. This function is documented on page ??.)

`\prg_break_point:n` These are all defined in `l3basics`, as they are needed “early”. This is just a reminder that
`\prg_map_break:` that is the case!
`\prg_map_break:n` *(End definition for \prg_break_point:n. This function is documented on page 42.)*

187.10 Experimental programmings functions

`\prg_define_quicksort:nnn` #1 is the name, #2 and #3 are the tokens enclosing the argument. For the somewhat strange `<clist>` 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 variable, so one additionally has to define

\cs_set_nopar: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 initial list and then the braced list afterwards.

```

2342 \cs_new_protected:Npn \prg_define_quicksort:nnn #1#2#3 {
2343   \cs_set:cpx{#1_quicksort:n}##1{
2344     \exp_not:c{#1_quicksort_start_partition:w} ##1
2345     \exp_not:n{#2\q_nil#3\q_stop}
2346   }
2347   \cs_set:cpx{#1_quicksort_braced:n}##1{
2348     \exp_not:c{#1_quicksort_start_partition_braced:n} ##1
2349     \exp_not:N\q_nil\exp_not:N\q_stop
2350   }
2351   \cs_set:cpx {#1_quicksort_start_partition:w} #2 ##1 #3{
2352     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
2353     \exp_not:c{#1_quicksort_do_partition_i:nnnw} {##1}{-}{-}
2354   }
2355   \cs_set:cpx {#1_quicksort_start_partition_braced:n} ##1 {
2356     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
2357     \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn} {##1}{-}{-}
2358   }

```

Now for doing the partitions.

```

2359 \cs_set:cpx {#1_quicksort_do_partition_i:nnnw} ##1##2##3 #2 ##4 #3 {
2360   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2361   {
2362     \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
2363     \exp_not:c{#1_quicksort_partition_greater_ii:nnnn}
2364     \exp_not:c{#1_quicksort_partition_less_ii:nnnn}
2365   }
2366   {##1}{##2}{##3}{##4}
2367 }
2368 \cs_set:cpx {#1_quicksort_do_partition_i_braced:nnnn} ##1##2##3##4 {
2369   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2370   {
2371     \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
2372     \exp_not:c{#1_quicksort_partition_greater_ii_braced:nnnn}
2373     \exp_not:c{#1_quicksort_partition_less_ii_braced:nnnn}
2374   }
2375   {##1}{##2}{##3}{##4}
2376 }
2377 \cs_set:cpx {#1_quicksort_do_partition_ii:nnnw} ##1##2##3 #2 ##4 #3 {
2378   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2379   {
2380     \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
2381     \exp_not:c{#1_quicksort_partition_less_i:nnnn}
2382     \exp_not:c{#1_quicksort_partition_greater_i:nnnn}
2383   }
2384   {##1}{##2}{##3}{##4}

```

```

2385 }
2386 \cs_set:cpx {#1_quicksort_do_partition_ii_braced:nnnn} ##1##2##3##4 {
2387   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2388   {
2389     \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
2390     \exp_not:c{#1_quicksort_partition_less_i_braced:nnnn}
2391     \exp_not:c{#1_quicksort_partition_greater_i_braced:nnnn}
2392   }
2393   {##1}{##2}{##3}{##4}
2394 }

```

This part of the code handles the two branches in each sorting. Again we will also have to do it braced.

```

2395 \cs_set:cpx {#1_quicksort_partition_less_i:nnnn} ##1##2##3##4{
2396   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##4}{##3}}
2397 \cs_set:cpx {#1_quicksort_partition_less_ii:nnnn} ##1##2##3##4{
2398   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}}
2399 \cs_set:cpx {#1_quicksort_partition_greater_i:nnnn} ##1##2##3##4{
2400   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##4}{##2}{##3}}
2401 \cs_set:cpx {#1_quicksort_partition_greater_ii:nnnn} ##1##2##3##4{
2402   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##4}{##3}}
2403 \cs_set:cpx {#1_quicksort_partition_less_i_braced:nnnn} ##1##2##3##4{
2404   \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn}{##1}{##2}{##4}{##3}}
2405 \cs_set:cpx {#1_quicksort_partition_less_ii_braced:nnnn} ##1##2##3##4{
2406   \exp_not:c{#1_quicksort_do_partition_ii_braced:nnnn}{##1}{##2}{##3}{##4}}
2407 \cs_set:cpx {#1_quicksort_partition_greater_i_braced:nnnn} ##1##2##3##4{
2408   \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn}{##1}{##4}{##2}{##3}}
2409 \cs_set:cpx {#1_quicksort_partition_greater_ii_braced:nnnn} ##1##2##3##4{
2410   \exp_not:c{#1_quicksort_do_partition_ii_braced:nnnn}{##1}{##2}{##4}{##3}}

```

Finally, the big kahuna! This is where the sub-lists are sorted.

```

2411 \cs_set:cpx {#1_do_quicksort_braced:nnnnw} ##1##2##3##4\q_stop {
2412   \exp_not:c{#1_quicksort_braced:n}{##2}
2413   \exp_not:c{#1_quicksort_function:n}{##1}
2414   \exp_not:c{#1_quicksort_braced:n}{##3}
2415 }
2416 }

```

(End definition for \prg_define_quicksort:nnn. This function is documented on page ??.)

`\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.

```

2417 \prg_define_quicksort:nnn {prg}{-}{-}

```

(End definition for \prg_quicksort:n. This function is documented on page 42.)

`\prg_quicksort_function:n`
`\prg_quicksort_compare:nnTF`

```

2418 \cs_set:Npn \prg_quicksort_function:n {\ERROR}
2419 \cs_set:Npn \prg_quicksort_compare:nnTF {\ERROR}

```

(End definition for \prg_quicksort_function:n. This function is documented on page 42.)

187.11 Deprecated functions

These were deprecated on 2011-05-27 and will be removed entirely by 2011-08-31.

```
\prg_new_map_functions:Nn As we have restructured the structured variables, these are no longer needed.
\prg_set_map_functions:Nn
2420 <*deprecated>
2421 \cs_new_protected:Npn \prg_new_map_functions:Nn #1#2 { \deprecated }
2422 \cs_new_protected:Npn \prg_set_map_functions:Nn #1#2 { \deprecated }
2423 </deprecated>
      (End definition for \prg_new_map_functions:Nn. This function is documented on page ??.)
2424 </initex | package>
```

188 l3quark implementation

The following test files are used for this code: *m3quark001.lvt*.

```
2425 <*initex | package>
2426 <*package>
2427 \ProvidesExplPackage
2428   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
2429 \package_check_loaded_expl:
2430 </package>
```

188.1 Quarks

```
\quark_new:N Allocate a new quark.
2431 \cs_new_protected:Npn \quark_new:N #1 { \tl_const:Nn #1 {#1} }
      (End definition for \quark_new:N. This function is documented on page 43.)

\q_nil Some “public” quarks. \q_stop is an “end of argument” marker, \q_nil is a empty value
\q_mark and \q_no_value marks an empty argument.
\q_no_value
\q_stop
2432 \quark_new:N \q_nil
2433 \quark_new:N \q_mark
2434 \quark_new:N \q_no_value
2435 \quark_new:N \q_stop
      (End definition for \q_nil and others. These functions are documented on page 43.)

\q_recursion_tail Quarks for ending recursions. Only ever used there! \q_recursion_tail is appended to
\q_recursion_stop 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.
2436 \quark_new:N \q_recursion_tail
2437 \quark_new:N \q_recursion_stop
      (End definition for \q_recursion_tail and \q_recursion_stop. These functions are documented
on page 44.)
```

\quark_if_recursion_tail_stop:N
\quark_if_recursion_tail_stop_do:Nn

When doing recursions, it is easy to spend a lot of time testing if the end marker has been found. To avoid this, a dedicated end marker is used each time a recursion is set up. Thus if the marker is found everything can be wrapper up and finished off. The simple case is when the test can guarantee that only a single token is being tested. In this case, there is just a dedicated copy of the standard quark test. Both a gobbling version and one inserting end code are provided.

```

2438 \cs_new:Npn \quark_if_recursion_tail_stop:N #1
2439 {
2440   \if_meaning:w \q_recursion_tail #1
2441   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
2442   \fi:
2443 }
2444 \cs_new:Npn \quark_if_recursion_tail_stop_do:Nn #1
2445 {
2446   \if_meaning:w \q_recursion_tail #1
2447   \exp_after:wN \use_i_delimit_by_q_recursion_stop:nw
2448   \else:
2449   \exp_after:wN \use_none:n
2450   \fi:
2451 }

```

(End definition for \quark_if_recursion_tail_stop:N. This function is documented on page 45.)

\quark_if_recursion_tail_stop:n
\quark_if_recursion_tail_stop_o
\quark_if_recursion_tail_stop_do:nn
\quark_if_recursion_tail_stop_do:on

The same idea applies when testing multiple tokens, but here we just compare the token list to \q_recursion_tail as a string.

```

2452 \cs_new:Npn \quark_if_recursion_tail_stop:n #1
2453 {
2454   \if_int_compare:w \pdfTeX_strcmp:D
2455   { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2456   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
2457   \fi:
2458 }
2459 \cs_new:Npn \quark_if_recursion_tail_stop_do:nn #1
2460 {
2461   \if_int_compare:w \pdfTeX_strcmp:D
2462   { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2463   \exp_after:wN \use_i_delimit_by_q_recursion_stop:nw
2464   \else:
2465   \exp_after:wN \use_none:n
2466   \fi:
2467 }
2468 \cs_generate_variant:Nn \quark_if_recursion_tail_stop:n { o }
2469 \cs_generate_variant:Nn \quark_if_recursion_tail_stop_do:nn { o }

```

(End definition for \quark_if_recursion_tail_stop:n and \quark_if_recursion_tail_stop_o. These functions are documented on page ??.)

\quark_if_recursion_tail_break:N
\quark_if_recursion_tail_break:n

Analog of the \quark_if_recursion_tail_stop... functions. Break the mapping using \prg_map_break:.

```

2470 \cs_new:Npn \quark_if_recursion_tail_break:N #1

```

```

2471 {
2472   \if_meaning:w \q_recursion_tail #1
2473   \exp_after:wN \prg_map_break:
2474   \fi:
2475 }
2476 \cs_new:Npn \quark_if_recursion_tail_break:n #1
2477 {
2478   \if_int_compare:w \pdfTeX_strcmp:D
2479   { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2480   \exp_after:wN \prg_map_break:
2481   \fi:
2482 }

```

(End definition for \quark_if_recursion_tail_break:N. This function is documented on page ??.)

\quark_if_nil:N Here we test if we found a special quark as the first argument. We better start with
\quark_if_no_value:N. \q_no_value as the first argument since the whole thing may otherwise loop if #1 is
\quark_if_no_value:c wrongly given a string like aabc instead of a single token.⁵

```

2483 \prg_new_conditional:Nnn \quark_if_nil:N { p, T, F, TF }
2484 {
2485   \if_meaning:w \q_nil #1
2486   \prg_return_true:
2487   \else:
2488   \prg_return_false:
2489   \fi:
2490 }
2491 \prg_new_conditional:Nnn \quark_if_no_value:N { p, T, F, TF }
2492 {
2493   \if_meaning:w \q_no_value #1
2494   \prg_return_true:
2495   \else:
2496   \prg_return_false:
2497   \fi:
2498 }
2499 \cs_generate_variant:Nn \quark_if_no_value_p:N { c }
2500 \cs_generate_variant:Nn \quark_if_no_value_NT { c }
2501 \cs_generate_variant:Nn \quark_if_no_value_NF { c }
2502 \cs_generate_variant:Nn \quark_if_no_value_NTF { c }

```

(End definition for \quark_if_nil:N. This function is documented on page ??.)

\quark_if_nil:n These are essentially \str_if_eq:nn tests but done directly.
\quark_if_nil:V 2503 \prg_new_conditional:Nnn \quark_if_nil:n { p, T, F, TF }
\quark_if_nil:o 2504 {
\quark_if_no_value:n 2505 \if_int_compare:w \pdfTeX_strcmp:D
2506 { \exp_not:N \q_nil } { \exp_not:n {#1} } = \c_zero
2507 \prg_return_true:
2508 \else:
2509 \prg_return_false:

⁵It may still loop in special circumstances however!

```

2510     \fi:
2511   }
2512   \prg_new_conditional:Nnn \quark_if_no_value:n { p , T , F , TF }
2513   {
2514     \if_int_compare:w \pdfTeX_strcmp:D
2515       { \exp_not:N \q_no_value } { \exp_not:n {#1} } = \c_zero
2516     \prg_return_true:
2517   \else:
2518     \prg_return_false:
2519   \fi:
2520 }
2521 \cs_generate_variant:Nn \quark_if_nil_p:n { V , o }
2522 \cs_generate_variant:Nn \quark_if_nil:nTF { V , o }
2523 \cs_generate_variant:Nn \quark_if_nil:nT { V , o }
2524 \cs_generate_variant:Nn \quark_if_nil:nF { V , o }

```

(End definition for \quark_if_nil:n, \quark_if_nil:V, and \quark_if_nil:o. These functions are documented on page 44.)

`\q_tl_act_mark` These private quarks are needed by `l3tl`, but that is loaded before the quark module, hence their definition is deferred.

```

2525 \quark_new:N \q_tl_act_mark
2526 \quark_new:N \q_tl_act_stop

```

(End definition for \q_tl_act_mark and \q_tl_act_stop. These functions are documented on page 94.)

188.2 Scan marks

`\g_scan_marks_tl` The list of all scan marks currently declared.

```

2527 \tl_new:N \g_scan_marks_tl

```

(End definition for \g_scan_marks_tl. This function is documented on page ??.)

`\scan_new:N` Check whether the variable is already a scan mark, then declare it to be equal to `\scan_stop`: globally.

```

2528 \cs_new_protected:Npn \scan_new:N #1
2529 {
2530   \tl_if_in:NnTF \g_scan_marks_tl { #1 }
2531   {
2532     \msg_kernel_error:nxx { scan } { already-defined }
2533     { \token_to_str:N #1 }
2534   }
2535   {
2536     \tl_gput_right:Nn \g_scan_marks_tl {#1}
2537     \cs_new_eq:NN #1 \scan_stop:
2538   }
2539 }

```

(End definition for \scan_new:N. This function is documented on page 45.)

`\s_stop` We only declare one scan mark here, more can be defined by specific modules.

```

2540 \scan_new:N \s_stop

```

(End definition for `\s_stop`. This function is documented on page 45.)

`\use_none_delimit_by_s_stop:w` Similar to `\use_none_delimit_by_q_stop:w`.

2541 `\cs_new:Npn \use_none_delimit_by_s_stop:w #1 \s_stop { }`

(End definition for `\use_none_delimit_by_s_stop:w`. This function is documented on page 46.)

2542 `\initex | package)`

189 l3token implementation

2543 `*initex | package)`

2544 `*package)`

2545 `\ProvidesExplPackage`

2546 `{\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}`

2547 `\package_check_loaded_expl:`

2548 `\package)`

189.1 Character tokens

`\char_set_catcode:nn` Category code changes.

`\char_value_catcode:n`

`\char_show_value_catcode:n`

2549 `\cs_new_protected:Npn \char_set_catcode:nn #1#2`

2550 `{ \tex_catcode:D #1 = \int_eval:w #2 \int_eval_end: }`

2551 `\cs_new:Npn \char_value_catcode:n #1`

2552 `{ \tex_the:D \tex_catcode:D \int_eval:w #1 \int_eval_end: }`

2553 `\cs_new_protected:Npn \char_show_value_catcode:n #1`

2554 `{ \tex_showthe:D \tex_catcode:D \int_eval:w #1 \int_eval_end: }`

(End definition for `\char_set_catcode:nn`. This function is documented on page 49.)

`\char_set_catcode_escape:N`

`\char_set_catcode_group_begin:N`

`\char_set_catcode_group_end:N`

`\char_set_catcode_math_toggle:N`

`\char_set_catcode_alignment:N`

`\char_set_catcode_end_line:N`

`\char_set_catcode_parameter:N`

`\char_set_catcode_math_superscript:N`

`\char_set_catcode_math_subscript:N`

`\char_set_catcode_ignore:N`

`\char_set_catcode_space:N`

`\char_set_catcode_letter:N`

`\char_set_catcode_other:N`

`\char_set_catcode_active:N`

`\char_set_catcode_comment:N`

`\char_set_catcode_invalid:N`

2555 `\cs_new_protected:Npn \char_set_catcode_escape:N #1`

2556 `{ \char_set_catcode:nn { '#1 } \c_zero }`

2557 `\cs_new_protected:Npn \char_set_catcode_group_begin:N #1`

2558 `{ \char_set_catcode:nn { '#1 } \c_one }`

2559 `\cs_new_protected:Npn \char_set_catcode_group_end:N #1`

2560 `{ \char_set_catcode:nn { '#1 } \c_two }`

2561 `\cs_new_protected:Npn \char_set_catcode_math_toggle:N #1`

2562 `{ \char_set_catcode:nn { '#1 } \c_three }`

2563 `\cs_new_protected:Npn \char_set_catcode_alignment:N #1`

2564 `{ \char_set_catcode:nn { '#1 } \c_four }`

2565 `\cs_new_protected:Npn \char_set_catcode_end_line:N #1`

2566 `{ \char_set_catcode:nn { '#1 } \c_five }`

2567 `\cs_new_protected:Npn \char_set_catcode_parameter:N #1`

2568 `{ \char_set_catcode:nn { '#1 } \c_six }`

2569 `\cs_new_protected:Npn \char_set_catcode_math_superscript:N #1`

2570 `{ \char_set_catcode:nn { '#1 } \c_seven }`

2571 `\cs_new_protected:Npn \char_set_catcode_math_subscript:N #1`

2572 `{ \char_set_catcode:nn { '#1 } \c_eight }`

2573 `\cs_new_protected:Npn \char_set_catcode_ignore:N #1`

2574 `{ \char_set_catcode:nn { '#1 } \c_nine }`


```

2575 \cs_new_protected:Npn \char_set_catcode_space:N #1
2576 { \char_set_catcode:nn { '#1 } \c_ten }
2577 \cs_new_protected:Npn \char_set_catcode_letter:N #1
2578 { \char_set_catcode:nn { '#1 } \c_eleven }
2579 \cs_new_protected:Npn \char_set_catcode_other:N #1
2580 { \char_set_catcode:nn { '#1 } \c_twelve }
2581 \cs_new_protected:Npn \char_set_catcode_active:N #1
2582 { \char_set_catcode:nn { '#1 } \c_thirteen }
2583 \cs_new_protected:Npn \char_set_catcode_comment:N #1
2584 { \char_set_catcode:nn { '#1 } \c_fourteen }
2585 \cs_new_protected:Npn \char_set_catcode_invalid:N #1
2586 { \char_set_catcode:nn { '#1 } \c_fifteen }

```

(End definition for `\char_set_catcode_escape:N` and others. These functions are documented on page 48.)

```

\char_set_catcode_escape:n
  \char_set_catcode_group_begin:n
  \char_set_catcode_group_end:n
  \char_set_catcode_math_toggle:n
  \char_set_catcode_alignment:n
\char_set_catcode_end_line:n
  \char_set_catcode_parameter:n
  \char_set_catcode_math_superscript:n
  \char_set_catcode_math_subscript:n
\char_set_catcode_ignore:n
\char_set_catcode_space:n
\char_set_catcode_letter:n
\char_set_catcode_other:n
\char_set_catcode_active:n
\char_set_catcode_comment:n
\char_set_catcode_invalid:n
2587 \cs_new_protected:Npn \char_set_catcode_escape:n #1
2588 { \char_set_catcode:nn {#1} \c_zero }
2589 \cs_new_protected:Npn \char_set_catcode_group_begin:n #1
2590 { \char_set_catcode:nn {#1} \c_one }
2591 \cs_new_protected:Npn \char_set_catcode_group_end:n #1
2592 { \char_set_catcode:nn {#1} \c_two }
2593 \cs_new_protected:Npn \char_set_catcode_math_toggle:n #1
2594 { \char_set_catcode:nn {#1} \c_three }
2595 \cs_new_protected:Npn \char_set_catcode_alignment:n #1
2596 { \char_set_catcode:nn {#1} \c_four }
2597 \cs_new_protected:Npn \char_set_catcode_end_line:n #1
2598 { \char_set_catcode:nn {#1} \c_five }
2599 \cs_new_protected:Npn \char_set_catcode_parameter:n #1
2600 { \char_set_catcode:nn {#1} \c_six }
2601 \cs_new_protected:Npn \char_set_catcode_math_superscript:n #1
2602 { \char_set_catcode:nn {#1} \c_seven }
2603 \cs_new_protected:Npn \char_set_catcode_math_subscript:n #1
2604 { \char_set_catcode:nn {#1} \c_eight }
2605 \cs_new_protected:Npn \char_set_catcode_ignore:n #1
2606 { \char_set_catcode:nn {#1} \c_nine }
2607 \cs_new_protected:Npn \char_set_catcode_space:n #1
2608 { \char_set_catcode:nn {#1} \c_ten }
2609 \cs_new_protected:Npn \char_set_catcode_letter:n #1
2610 { \char_set_catcode:nn {#1} \c_eleven }
2611 \cs_new_protected:Npn \char_set_catcode_other:n #1
2612 { \char_set_catcode:nn {#1} \c_twelve }
2613 \cs_new_protected:Npn \char_set_catcode_active:n #1
2614 { \char_set_catcode:nn {#1} \c_thirteen }
2615 \cs_new_protected:Npn \char_set_catcode_comment:n #1
2616 { \char_set_catcode:nn {#1} \c_fourteen }
2617 \cs_new_protected:Npn \char_set_catcode_invalid:n #1
2618 { \char_set_catcode:nn {#1} \c_fifteen }

```

(End definition for `\char_set_catcode_escape:n` and others. These functions are documented on page 48.)

Pretty repetitive, but necessary!

```

\char_set_mathcode:nn
\char_value_mathcode:n
\char_show_value_mathcode:n
\char_set_lccode:nn
\char_value_lccode:n
\char_show_value_lccode:n
\char_set_uccode:nn
\char_value_uccode:n
\char_show_value_uccode:n
\char_set_sfcode:nn
\char_value_sfcode:n
\char_show_value_sfcode:n
2619 \cs_new_protected:Npn \char_set_mathcode:nn #1#2
2620 { \tex_mathcode:D #1 = \int_eval:w #2 \int_eval_end: }
2621 \cs_new:Npn \char_value_mathcode:n #1
2622 { \tex_the:D \tex_mathcode:D \int_eval:w #1\int_eval_end: }
2623 \cs_new_protected:Npn \char_show_value_mathcode:n #1
2624 { \tex_showthe:D \tex_mathcode:D \int_eval:w #1 \int_eval_end: }
2625 \cs_new_protected:Npn \char_set_lccode:nn #1#2
2626 { \tex_lccode:D #1 = \int_eval:w #2 \int_eval_end: }
2627 \cs_new:Npn \char_value_lccode:n #1
2628 { \tex_the:D \tex_lccode:D \int_eval:w #1\int_eval_end: }
2629 \cs_new_protected:Npn \char_show_value_lccode:n #1
2630 { \tex_showthe:D \tex_lccode:D \int_eval:w #1 \int_eval_end: }
2631 \cs_new_protected:Npn \char_set_uccode:nn #1#2
2632 { \tex_uccode:D #1 = \int_eval:w #2 \int_eval_end: }
2633 \cs_new:Npn \char_value_uccode:n #1
2634 { \tex_the:D \tex_uccode:D \int_eval:w #1\int_eval_end: }
2635 \cs_new_protected:Npn \char_show_value_uccode:n #1
2636 { \tex_showthe:D \tex_uccode:D \int_eval:w #1 \int_eval_end: }
2637 \cs_new_protected:Npn \char_set_sfcode:nn #1#2
2638 { \tex_sfcode:D #1 = \int_eval:w #2 \int_eval_end: }
2639 \cs_new:Npn \char_value_sfcode:n #1
2640 { \tex_the:D \tex_sfcode:D \int_eval:w #1\int_eval_end: }
2641 \cs_new_protected:Npn \char_show_value_sfcode:n #1
2642 { \tex_showthe:D \tex_sfcode:D \int_eval:w #1 \int_eval_end: }
(End definition for \char_set_mathcode:nn. This function is documented on page 51.)

```

189.2 Generic tokens

`\token_new:Nn` Creates a new token.

```

2643 \cs_new_protected:Npn \token_new:Nn #1#2 { \cs_new_eq:NN #1 #2 }
(End definition for \token_new:Nn. This function is documented on page 51.)

```

`\c_group_begin_token` `\c_group_end_token` We define these useful tokens. We have to do it by hand with the brace tokens for obvious reasons.

```

\c_math_toggle_token
\c_alignment_token
\c_parameter_token
\c_math_superscript_token
\c_math_subscript_token
\c_space_token
\c_catcode_letter_token
\c_catcode_other_token
2644 \cs_new_eq:NN \c_group_begin_token {
2645 \cs_new_eq:NN \c_group_end_token }
2646 \group_begin:
2647 \char_set_catcode_math_toggle:N \*
2648 \token_new:Nn \c_math_toggle_token { * }
2649 \char_set_catcode_alignment:N \*
2650 \token_new:Nn \c_alignment_token { * }
2651 \token_new:Nn \c_parameter_token { # }
2652 \token_new:Nn \c_math_superscript_token { ^ }
2653 \char_set_catcode_math_subscript:N \*
2654 \token_new:Nn \c_math_subscript_token { * }
2655 \token_new:Nn \c_space_token { ~ }
2656 \token_new:Nn \c_catcode_letter_token { a }

```

```

2657 \token_new:Nn \c_catcode_other_token { 1 }
2658 \group_end:
      (End definition for \c_group_begin_token and others. These functions are documented on page
51.)

```

`\c_catcode_active_tl` Not an implicit token!

```

2659 \group_begin:
2660 \char_set_catcode_active:N \*
2661 \tl_const:Nn \c_catcode_active_tl { \exp_not:N * }
2662 \group_end:
      (End definition for \c_catcode_active_tl. This function is documented on page 51.)

```

189.3 Token conditionals

`\token_if_group_begin:N` Check if token is a begin group token. We use the constant `\c_group_begin_token` for this.

```

2663 \prg_new_conditional:Npnn \token_if_group_begin:N #1 { p , T , F , TF }
2664 {
2665   \if_catcode:w \exp_not:N #1 \c_group_begin_token
2666   \prg_return_true: \else: \prg_return_false: \fi:
2667 }
      (End definition for \token_if_group_begin:N. This function is documented on page 52.)

```

`\token_if_group_end:N` Check if token is a end group token. We use the constant `\c_group_end_token` for this.

```

2668 \prg_new_conditional:Npnn \token_if_group_end:N #1 { p , T , F , TF }
2669 {
2670   \if_catcode:w \exp_not:N #1 \c_group_end_token
2671   \prg_return_true: \else: \prg_return_false: \fi:
2672 }
      (End definition for \token_if_group_end:N. This function is documented on page 52.)

```

`\token_if_math_toggle:N` Check if token is a math shift token. We use the constant `\c_math_toggle_token` for this.

```

2673 \prg_new_conditional:Npnn \token_if_math_toggle:N #1 { p , T , F , TF }
2674 {
2675   \if_catcode:w \exp_not:N #1 \c_math_toggle_token
2676   \prg_return_true: \else: \prg_return_false: \fi:
2677 }
      (End definition for \token_if_math_toggle:N. This function is documented on page 52.)

```

`\token_if_alignment:N` Check if token is an alignment tab token. We use the constant `\c_alignment_tab_token` for this.

```

2678 \prg_new_conditional:Npnn \token_if_alignment:N #1 { p , T , F , TF }
2679 {
2680   \if_catcode:w \exp_not:N #1 \c_alignment_token
2681   \prg_return_true: \else: \prg_return_false: \fi:
2682 }
      (End definition for \token_if_alignment:N. This function is documented on page 52.)

```

`\token_if_parameter:N` Check if token is a parameter token. We use the constant `\c_parameter_token` for this. We have to trick \TeX a bit to avoid an error message: within a group we prevent `\c_parameter_token` from behaving like a macro parameter character. The definitions of `\prg_new_conditional:Npnn` are global, so they will remain after the group.

```

2683 \group_begin:
2684 \cs_set_eq:NN \c_parameter_token \scan_stop:
2685 \prg_new_conditional:Npnn \token_if_parameter:N #1 { p , T , F , TF }
2686 {
2687     \if_catcode:w \exp_not:N #1 \c_parameter_token
2688     \prg_return_true: \else: \prg_return_false: \fi:
2689 }
2690 \group_end:

```

(End definition for `\token_if_parameter:N`. This function is documented on page 52.)

`\token_if_math_superscript:N` Check if token is a math superscript token. We use the constant `\c_superscript_token` for this.

```

2691 \prg_new_conditional:Npnn \token_if_math_superscript:N #1 { p , T , F , TF }
2692 {
2693     \if_catcode:w \exp_not:N #1 \c_math_superscript_token
2694     \prg_return_true: \else: \prg_return_false: \fi:
2695 }

```

(End definition for `\token_if_math_superscript:N`. This function is documented on page 52.)

`\token_if_math_subscript:N` Check if token is a math subscript token. We use the constant `\c_subscript_token` for this.

```

2696 \prg_new_conditional:Npnn \token_if_math_subscript:N #1 { p , T , F , TF }
2697 {
2698     \if_catcode:w \exp_not:N #1 \c_math_subscript_token
2699     \prg_return_true: \else: \prg_return_false: \fi:
2700 }

```

(End definition for `\token_if_math_subscript:N`. This function is documented on page 53.)

`\token_if_space:N` Check if token is a space token. We use the constant `\c_space_token` for this.

```

2701 \prg_new_conditional:Npnn \token_if_space:N #1 { p , T , F , TF }
2702 {
2703     \if_catcode:w \exp_not:N #1 \c_space_token
2704     \prg_return_true: \else: \prg_return_false: \fi:
2705 }

```

(End definition for `\token_if_space:N`. This function is documented on page 53.)

`\token_if_letter:N` Check if token is a letter token. We use the constant `\c_letter_token` for this.

```

2706 \prg_new_conditional:Npnn \token_if_letter:N #1 { p , T , F , TF }
2707 {
2708     \if_catcode:w \exp_not:N #1 \c_catcode_letter_token
2709     \prg_return_true: \else: \prg_return_false: \fi:
2710 }

```

(End definition for `\token_if_letter:N`. This function is documented on page 53.)

`\token_if_other:N` Check if token is an other char token. We use the constant `\c_other_char_token` for this.

```

2711 \prg_new_conditional:Npnn \token_if_other:N #1 { p , T , F , TF }
2712 {
2713   \if_catcode:w \exp_not:N #1 \c_catcode_other_token
2714   \prg_return_true: \else: \prg_return_false: \fi:
2715 }

```

(End definition for \token_if_other:N. This function is documented on page 53.)

`\token_if_active:N` Check if token is an active char token. We use the constant `\c_active_char_tl` for this. A technical point is that `\c_active_char_tl` is in fact a macro expanding to `\exp_not:N *`, where `*` is active.

```

2716 \prg_new_conditional:Npnn \token_if_active:N #1 { p , T , F , TF }
2717 {
2718   \if_catcode:w \exp_not:N #1 \c_catcode_active_tl
2719   \prg_return_true: \else: \prg_return_false: \fi:
2720 }

```

(End definition for \token_if_active:N. This function is documented on page 53.)

`\token_if_eq_meaning:NN` Check if the tokens #1 and #2 have same meaning.

```

2721 \prg_new_conditional:Npnn \token_if_eq_meaning:NN #1#2 { p , T , F , TF }
2722 {
2723   \if_meaning:w #1 #2
2724   \prg_return_true: \else: \prg_return_false: \fi:
2725 }

```

(End definition for \token_if_eq_meaning:NN. This function is documented on page 53.)

`\token_if_eq_catcode:NN` Check if the tokens #1 and #2 have same category code.

```

2726 \prg_new_conditional:Npnn \token_if_eq_catcode:NN #1#2 { p , T , F , TF }
2727 {
2728   \if_catcode:w \exp_not:N #1 \exp_not:N #2
2729   \prg_return_true: \else: \prg_return_false: \fi:
2730 }

```

(End definition for \token_if_eq_catcode:NN. This function is documented on page 53.)

`\token_if_eq_charcode:NN` Check if the tokens #1 and #2 have same character code.

```

2731 \prg_new_conditional:Npnn \token_if_eq_charcode:NN #1#2 { p , T , F , TF }
2732 {
2733   \if_charcode:w \exp_not:N #1 \exp_not:N #2
2734   \prg_return_true: \else: \prg_return_false: \fi:
2735 }

```

(End definition for \token_if_eq_charcode:NN. This function is documented on page 53.)

`\token_if_macro:N` When a token is a macro, `\token_to_meaning:N` will always output something like `\long macro:#1->#1` so we could naively check to see if the meaning contains `->`.
`\token_if_macro_p_aux:w` However, this can fail the five `\...mark` primitives, whose meaning has the form `\...mark:<user material>`. The problem is that the `<user material>` can contain `->`.

However, only characters, macros, and marks can contain the colon character. The idea is thus to grab until the first :, and analyse what is left. However, macros can have any combination of `\long`, `\protected` or `\outer` (not used in L^AT_EX3) before the string `macro:`. We thus only select the part of the meaning between the first `ma` and the first following `:`. If this string is `cro`, then we have a macro. If the string is `rk`, then we have a mark. The string can also be `cro parameter character` for a colon with a weird category code (namely the usual category code of #). Otherwise, it is empty.

This relies on the fact that `\long`, `\protected`, `\outer` cannot contain `ma`, regardless of the escape character, even if the escape character is `m...`.

Both `ma` and `:` must be of category code 12 (other), and we achieve using the standard lowercasing technique.

```

2736 \group_begin:
2737 \char_set_catcode_other:N \M
2738 \char_set_catcode_other:N \A
2739 \char_set_lccode:nn { '\; } { '\: }
2740 \char_set_lccode:nn { '\T } { '\T }
2741 \char_set_lccode:nn { '\F } { '\F }
2742 \tl_to_lowercase:n
2743 {
2744   \group_end:
2745   \prg_new_conditional:Npnn \token_if_macro:N #1 { p , T , F , TF }
2746   {
2747     \exp_after:wN \token_if_macro_p_aux:w
2748     \token_to_meaning:N #1 MA; \q_stop
2749   }
2750   \cs_new:Npn \token_if_macro_p_aux:w #1 MA #2 ; #3 \q_stop
2751   {
2752     \if_int_compare:w \pdfTeX_strcmp:D { #2 } { cro } = \c_zero
2753       \prg_return_true:
2754     \else:
2755       \prg_return_false:
2756     \fi:
2757   }
2758 }
```

(End definition for `\token_if_macro:N`. This function is documented on page ??.)

`\token_if_cs:N` Check if token has same catcode as a control sequence. This follows the same pattern as for `\token_if_letter:N` etc. We use `\scan_stop:` for this.

```

2759 \prg_new_conditional:Npnn \token_if_cs:N #1 { p , T , F , TF }
2760 {
2761   \if_catcode:w \exp_not:N #1 \scan_stop:
2762     \prg_return_true: \else: \prg_return_false: \fi:
2763 }
```

(End definition for `\token_if_cs:N`. This function is documented on page 53.)

`\token_if_expandable:N` Check if token is expandable. We use the fact that T_EX will temporarily convert `\exp_not:N` (*token*) into `\scan_stop:` if (*token*) is expandable.

```

2764 \prg_new_conditional:Npnn \token_if_expandable:N #1 { p , T , F , TF }
2765 {
2766   \cs_if_exist:NTF #1
2767   {
2768     \exp_after:wN \if_meaning:w \exp_not:N #1 #1
2769     \prg_return_false: \else: \prg_return_true: \fi:
2770   }
2771   { \prg_return_false: }
2772 }

```

(End definition for `\token_if_expandable:N`. This function is documented on page 54.)

`\token_if_chardef:N` Most of these functions have to check the meaning of the token in question so we need to do some checkups on which characters are output by `\token_to_meaning:N`. As usual, these characters have catcode 12 so we must do some serious substitutions in the code below...

```

\token_if_protected_macro:N
\token_if_protected_long_macro:N
\token_if_dim_register:N
\token_if_skip_register:N
\token_if_int_register:N
\token_if_toks_register:N
\token_if_chardef_p_aux:w
\token_if_mathchardef_p_aux:w
\token_if_int_register_p_aux:w
\token_if_skip_register_p_aux:w
\token_if_dim_register_p_aux:w
\token_if_toks_register_p_aux:w
\token_if_protected_macro_p_aux:w
\token_if_long_macro_p_aux:w
\token_if_protected_long_macro_p_aux:w
2773 \group_begin:
2774 \char_set_lccode:nn { '\T } { '\T }
2775 \char_set_lccode:nn { '\F } { '\F }
2776 \char_set_lccode:nn { '\X } { '\n }
2777 \char_set_lccode:nn { '\Y } { '\t }
2778 \char_set_lccode:nn { '\Z } { '\d }
2779 \char_set_lccode:nn { '\? } { '\ \ }
2780 \tl_map_inline:nn { \X \Y \Z \M \C \H \A \R \O \U \S \K \I \P \L \G \P \E }
2781 { \char_set_catcode:nn { '#1 } \c_twelve }

```

We convert the token list to lower case and restore the catcode and lowercase code changes.

```

2782 \tl_to_lowercase:n
2783 {
2784   \group_end:

```

First up is checking if something has been defined with `\chardef` or `\mathchardef`. This is easy since \TeX thinks of such tokens as hexadecimal so it stores them as `\char"<hex number>` or `\mathchar"<hex number>`.

```

2785 \prg_new_conditional:Npnn \token_if_chardef:N #1 { p , T , F , TF }
2786 {
2787   \exp_after:wN \token_if_chardef_aux:w
2788   \token_to_meaning:N #1 ?CHAR" \q_stop
2789 }
2790 \cs_new:Npn \token_if_chardef_aux:w #1 ?CHAR" #2 \q_stop
2791 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }
2792 \prg_new_conditional:Npnn \token_if_mathchardef:N #1 { p , T , F , TF }
2793 {
2794   \exp_after:wN \token_if_mathchardef_aux:w
2795   \token_to_meaning:N #1 ?MAYHCHAR" \q_stop
2796 }
2797 \cs_new:Npn \token_if_mathchardef_aux:w #1 ?MAYHCHAR" #2 \q_stop
2798 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

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.

```

2799 \prg_new_conditional:Npnn \token_if_int_register:N #1 { p , T , F , TF }
2800 {
2801   \if_meaning:w \tex_countdef:D #1
2802   \prg_return_false:
2803   \else:
2804     \exp_after:wN \token_if_int_register_aux:w
2805     \token_to_meaning:N #1 ?COUXY \q_stop
2806   \fi:
2807 }
2808 \cs_new:Npn \token_if_int_register_aux:w #1 ?COUXY #2 \q_stop
2809 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Skip registers are done the same way as the integer registers.

```

2810 \prg_new_conditional:Npnn \token_if_skip_register:N #1 { p , T , F , TF }
2811 {
2812   \if_meaning:w \tex_skipdef:D #1
2813   \prg_return_false:
2814   \else:
2815     \exp_after:wN \token_if_skip_register_aux:w
2816     \token_to_meaning:N #1 ?SKIP \q_stop
2817   \fi:
2818 }
2819 \cs_new:Npn \token_if_skip_register_aux:w #1 ?SKIP #2 \q_stop
2820 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Dim registers. No news here

```

2821 \prg_new_conditional:Npnn \token_if_dim_register:N #1 { p , T , F , TF }
2822 {
2823   \if_meaning:w \tex_dimendef:D #1
2824   \c_false_bool
2825   \else:
2826     \exp_after:wN \token_if_dim_register_aux:w
2827     \token_to_meaning:N #1 ?ZIMEX \q_stop
2828   \fi:
2829 }
2830 \cs_new:Npn \token_if_dim_register_aux:w #1 ?ZIMEX #2 \q_stop
2831 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

Toks registers.

```

2832 \prg_new_conditional:Npnn \token_if_toks_register:N #1 { p , T , F , TF }
2833 {
2834   \if_meaning:w \tex_toksdef:D #1
2835   \prg_return_false:
2836   \else:
2837     \exp_after:wN \token_if_toks_register_aux:w
2838     \token_to_meaning:N #1 ?YOKS \q_stop
2839   \fi:
2840 }
2841 \cs_new:Npn \token_if_toks_register_aux:w #1 ?YOKS #2 \q_stop

```



```
2842 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }
```

Protected macros.

```
2843 \prg_new_conditional:Npnn \token_if_protected_macro:N #1
2844 { p , T , F , TF }
2845 {
2846   \exp_after:wN \token_if_protected_macro_aux:w
2847   \token_to_meaning:N #1 ?PROYECYEZ~MACRO \q_stop
2848 }
2849 \cs_new:Npn \token_if_protected_macro_aux:w
2850 #1 ?PROYECYEZ~MACRO #2 \q_stop
2851 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }
```

Long macros.

```
2852 \prg_new_conditional:Npnn \token_if_long_macro:N #1 { p , T , F , TF }
2853 {
2854   \exp_after:wN \token_if_long_macro_aux:w
2855   \token_to_meaning:N #1 ?LOXG~MACRO \q_stop
2856 }
2857 \cs_new:Npn \token_if_long_macro_aux:w #1 ?LOXG~MACRO #2 \q_stop
2858 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }
```

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.

```
2859 \prg_new_conditional:Npnn \token_if_protected_long_macro:N #1
2860 { p , T , F , TF }
2861 {
2862   \exp_after:wN \token_if_protected_long_macro_aux:w
2863   \token_to_meaning:N #1 ?PROYECYEZ?LOXG~MACRO \q_stop
2864 }
2865 \cs_new:Npn \token_if_protected_long_macro_aux:w
2866 #1 ?PROYECYEZ?LOXG~MACRO #2 \q_stop
2867 { \tl_if_empty:nTF {#1} { \prg_return_true: } { \prg_return_false: } }
```

Finally the `\tl_to_lowercase:n` ends!

```
2868 }
```

(End definition for `\token_if_chardef:N` and others. These functions are documented on page ??.)

```
\token_if_primitive:N
\token_if_primitive_aux:NNw
\token_if_primitive_aux_space:w
\token_if_primitive_aux_nullfont:N
\token_if_primitive_aux_loop:N
\token_if_primitive_auxii:Nw
\token_if_primitive_aux_undefined:N
```

We filter out macros first, because they cause endless trouble later otherwise.

Primitives are almost distinguished by the fact that the result of `\token_to_meaning:N` is formed from letters only. Every other token has either a space (e.g., the letter A), a digit (e.g., `\count123`) or a double quote (e.g., `\char"A`).

Ten exceptions: on the one hand, `\c_undefined:D` is not a primitive, but its meaning is undefined, only letters; on the other hand, `\space`, `\italiccorr`, `\hyphen`, `\firstmark`, `\topmark`, `\botmark`, `\splitfirstmark`, `\splitbotmark`, and `\nullfont` are primitives, but have non-letters in their meaning.

We start by removing the two first (non-space) characters from the meaning. This removes the escape character (which may be inexistent depending on `\endlinechar`), and

takes care of three of the exceptions: `\space`, `\italiccorr` and `\hyphen`, whose meaning is at most two characters. This leaves a string terminated by some `:`, and `\q_stop`.

The meaning of each one of the five `\...mark` primitives has the form $\langle letters \rangle : \langle user material \rangle$. In other words, the first non-letter is a colon. We remove everything after the first colon.

We are now left with a string, which we must analyze. For primitives, it contains only letters. For non-primitives, it contains either `"`, or a space, or a digit. Two exceptions remain: `\c_undefined:D`, which is not a primitive, and `\nullfont`, which is a primitive.

Spaces cannot be grabbed in an undelimited way, so we check them separately. If there is a space, we test for `\nullfont`. Otherwise, we go through characters one by one, and stop at the first character less than `'A` (this is not quite a test for “only letters”, but is close enough to work in this context). If this first character is `:` then we have a primitive, or `\c_undefined:D`, and if it is `"` or a digit, then the token is not a primitive.

```

2869 \tex_chardef:D \c_token_A_int = 'A ~ %
2870 \group_begin:
2871 \char_set_catcode_other:N \;
2872 \char_set_lccode:nn { '\; } { '\: }
2873 \char_set_lccode:nn { '\T } { '\T }
2874 \char_set_lccode:nn { '\F } { '\F }
2875 \tl_to_lowercase:n {
2876   \group_end:
2877   \prg_new_conditional:Npnn \token_if_primitive:N #1 { p , T , F , TF }
2878   {
2879     \token_if_macro:NTF #1
2880     \prg_return_false:
2881     {
2882       \exp_after:wN \token_if_primitive_aux:NNw
2883       \token_to_meaning:N #1 ; ; ; \q_stop #1
2884     }
2885   }
2886   \cs_new:Npn \token_if_primitive_aux:NNw #1#2 #3 ; #4 \q_stop
2887   {
2888     \tl_if_empty:oTF { \token_if_primitive_aux_space:w #3 ~ }
2889     { \token_if_primitive_aux_loop:N #3 ; \q_stop }
2890     { \token_if_primitive_aux_nullfont:N }
2891   }
2892 }
2893 \cs_new:Npn \token_if_primitive_aux_space:w #1 ~ { }
2894 \cs_new:Npn \token_if_primitive_aux_nullfont:N #1
2895 {
2896   \if_meaning:w \tex_nullfont:D #1
2897   \prg_return_true:
2898   \else:
2899     \prg_return_false:
2900   \fi:
2901 }
2902 \cs_new:Npn \token_if_primitive_aux_loop:N #1
2903 {

```

```

2904     \if_num:w '#1 < \c_token_A_int %
2905         \exp_after:wN \token_if_primitive_auxii:Nw
2906         \exp_after:wN #1
2907     \else:
2908         \exp_after:wN \token_if_primitive_aux_loop:N
2909     \fi:
2910 }
2911 \cs_new:Npn \token_if_primitive_auxii:Nw #1 #2 \q_stop
2912 {
2913     \if:w : #1
2914         \exp_after:wN \token_if_primitive_aux_undefined:N
2915     \else:
2916         \prg_return_false:
2917         \exp_after:wN \use_none:n
2918     \fi:
2919 }
2920 \cs_new:Npn \token_if_primitive_aux_undefined:N #1
2921 {
2922     \if_cs_exist:N #1
2923         \prg_return_true:
2924     \else:
2925         \prg_return_false:
2926     \fi:
2927 }

```

(End definition for \token_if_primitive:N. This function is documented on page ??.)

189.4 Peeking ahead at the next token

Peeking ahead is implemented using a two part mechanism. The outer level provides a defined interface to the lower level material. This allows a large amount of code to be shared. There are four 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.

\l_peek_token Storage tokens which are publicly documented: the token peeked.
\g_peek_token

```

2928 \cs_new_eq:NN \l_peek_token ?
2929 \cs_new_eq:NN \g_peek_token ?

```

(End definition for \l_peek_token. This function is documented on page 55.)

\l_peek_search_token The token to search for as an implicit token: cf. \l_peek_search_tl.

```

2930 \cs_new_eq:NN \l_peek_search_token ?

```

(End definition for \l_peek_search_token. This function is documented on page ??.)

`\l_peek_search_tl` The token to search for as an explicit token: *cf.* `\l_peek_search_token`.

```

2931 \tl_new:N \l_peek_search_tl
      (End definition for \l_peek_search_tl. This function is documented on page ??.)

```

`\peek_true:w` Functions used by the branching and space-stripping code.

```

\peek_true_aux:w 2932 \cs_new_nopar:Npn \peek_true:w { }
\peek_false:w    2933 \cs_new_nopar:Npn \peek_true_aux:w { }
\peek_tmp:w      2934 \cs_new_nopar:Npn \peek_false:w { }
                  2935 \cs_new:Npn \peek_tmp:w { }
      (End definition for \peek_true:w and others. These functions are documented on page ??.)

```

`\peek_after:Nw` Simple wrappers for `\futurelet`: no arguments absorbed here.

```

\peek_after:Nw 2936 \cs_new_protected_nopar:Npn \peek_after:Nw
                2937 { \tex_futurelet:D \l_peek_token }
                2938 \cs_new_protected_nopar:Npn \peek_gafter:Nw
                2939 { \tex_global:D \tex_futurelet:D \g_peek_token }
      (End definition for \peek_after:Nw. This function is documented on page 55.)

```

`\peek_true_remove:w` A function to remove the next token and then regain control.

```

2940 \cs_new_protected:Npn \peek_true_remove:w
2941 {
2942   \group_align_safe_end:
2943   \tex_afterassignment:D \peek_true_aux:w
2944   \cs_set_eq:NN \peek_tmp:w
2945 }
      (End definition for \peek_true_remove:w. This function is documented on page ??.)

```

`\peek_token_generic:NN` The generic function stores the test token in both implicit and explicit modes, and the `true` and `false` code as token lists, more or less. The two branches have to be absorbed here as the input stream needs to be cleared for the peek function itself.

```

2946 \cs_new_protected:Npn \peek_token_generic:NNTF #1#2#3#4
2947 {
2948   \cs_set_eq:NN \l_peek_search_token #2
2949   \tl_set:Nn \l_peek_search_tl {#2}
2950   \cs_set_nopar:Npx \peek_true:w
2951   {
2952     \exp_not:N \group_align_safe_end:
2953     \exp_not:n {#3}
2954   }
2955   \cs_set_nopar:Npx \peek_false:w
2956   {
2957     \exp_not:N \group_align_safe_end:
2958     \exp_not:n {#4}
2959   }
2960   \group_align_safe_begin:
2961   \peek_after:Nw #1
2962 }
2963 \cs_new_protected:Npn \peek_token_generic:NNT #1#2#3

```

```

2964 { \peek_token_generic:NNTF #1 #2 {#3} { } }
2965 \cs_new_protected:Npn \peek_token_generic:NNTF #1#2#3
2966 { \peek_token_generic:NNTF #1 #2 { } {#3} }
      (End definition for \peek_token_generic:NN. This function is documented on page ??.)

```

\peek_token_remove_generic:NN For token removal there needs to be a call to the auxiliary function which does the work.

```

2967 \cs_new_protected:Npn \peek_token_remove_generic:NNTF #1#2#3#4
2968 {
2969   \cs_set_eq:NN \l_peek_search_token #2
2970   \tl_set:Nn \l_peek_search_tl {#2}
2971   \cs_set_eq:NN \peek_true:w \peek_true_remove:w
2972   \cs_set_nopar:Npx \peek_true_aux:w { \exp_not:n {#3} }
2973   \cs_set_nopar:Npx \peek_false:w
2974   {
2975     \exp_not:N \group_align_safe_end:
2976     \exp_not:n {#4}
2977   }
2978   \group_align_safe_begin:
2979   \peek_after:Nw #1
2980 }
2981 \cs_new_protected:Npn \peek_token_remove_generic:NNT #1#2#3
2982 { \peek_token_remove_generic:NNTF #1 #2 {#3} { } }
2983 \cs_new_protected:Npn \peek_token_remove_generic:NNTF #1#2#3
2984 { \peek_token_remove_generic:NNTF #1 #2 { } {#3} }
      (End definition for \peek_token_remove_generic:NN. This function is documented on page ??.)

```

\peek_execute_branches_catcode: The category code and meaning tests are straight forward.

```

\peek_execute_branches_meaning:
2985 \cs_new_nopar:Npn \peek_execute_branches_catcode:
2986 {
2987   \if_catcode:w
2988     \exp_not:N \l_peek_token \exp_not:N \l_peek_search_token
2989     \exp_after:wN \peek_true:w
2990   \else:
2991     \exp_after:wN \peek_false:w
2992   \fi:
2993 }
2994 \cs_new_nopar:Npn \peek_execute_branches_meaning:
2995 {
2996   \if_meaning:w \l_peek_token \l_peek_search_token
2997     \exp_after:wN \peek_true:w
2998   \else:
2999     \exp_after:wN \peek_false:w
3000   \fi:
3001 }
      (End definition for \peek_execute_branches_catcode: and \peek_execute_branches_meaning:.
      These functions are documented on page ??.)

```

`\peek_execute_branches_charcode:` First the character code test there is a need to worry about \TeX grabbing brace group or skipping spaces. These are all tested for using a category code check before grabbing what must be a real single token and doing the comparison.

```

3002 \cs_new_nopar:Npn \peek_execute_branches_charcode:
3003 {
3004   \bool_if:nTF
3005   {
3006     \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token
3007     || \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token
3008     || \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
3009   }
3010   { \peek_false:w }
3011   {
3012     \exp_after:wN \peek_execute_branches_charcode_aux:NN
3013     \l_peek_search_tl
3014   }
3015 }
3016 \cs_new:Npn \peek_execute_branches_charcode_aux:NN #1#2
3017 {
3018   \if:w \exp_not:N #1 \exp_not:N #2
3019   \exp_after:wN \peek_true:w
3020   \else:
3021   \exp_after:wN \peek_false:w
3022   \fi:
3023   #2
3024 }

```

(End definition for \peek_execute_branches_charcode:. This function is documented on page ??.)

`\peek_ignore_spaces_execute_branches:` This function removes one token at a time with a mechanism that can be applied to things other than spaces.

```

\peek_ignore_spaces_execute_branches_aux:
3025 \cs_new_protected_nopar:Npn \peek_ignore_spaces_execute_branches:
3026 {
3027   \token_if_eq_meaning:NNTF \l_peek_token \c_space_token
3028   {
3029     \tex_afterassignment:D \peek_ignore_spaces_execute_branches_aux:
3030     \cs_set_eq:NN \peek_tmp:w
3031   }
3032   { \peek_execute_branches: }
3033 }
3034 \cs_new_protected_nopar:Npn \peek_ignore_spaces_execute_branches_aux:
3035 { \peek_after:Nw \peek_ignore_spaces_execute_branches: }

```

(End definition for \peek_ignore_spaces_execute_branches:. This function is documented on page ??.)

`\peek_def:nnnn` The public functions themselves cannot be defined using `\prg_set_conditional:Npnn`
`\peek_def_aux:nnnnn` and so a couple of auxiliary functions are used. As a result, everything is done inside a group. As a result things are a bit complicated.

```

3036 \group_begin:
3037 \cs_set:Npn \peek_def:nnnn #1#2#3#4

```

```

3038 {
3039   \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { TF }
3040   \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { T }
3041   \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { F }
3042 }
3043 \cs_set:Npn \peek_def_aux:nnnnn #1#2#3#4#5
3044 {
3045   \cs_gset_nopar:cpx { #1 #5 }
3046   {
3047     \tl_if_empty:nF {#2}
3048     { \exp_not:n { \cs_set_eq:NN \peek_execute_branches: #2 } }
3049     \exp_not:c { #3 #5 }
3050     \exp_not:n {#4}
3051   }
3052 }

```

(End definition for \peek_def:nnnn. This function is documented on page ??.)

\peek_catcode:N With everything in place the definitions can take place. First for category codes.

```

\peek_catcode_ignore_spaces:N 3053 \peek_def:nnnn { peek_catcode:N }
\peek_catcode_remove:N         3054 { }
\peek_catcode_remove_ignore_spaces:N 3055 { peek_token_generic:NN }
                                3056 { \peek_execute_branches_catcode: }
                                3057 \peek_def:nnnn { peek_catcode_ignore_spaces:N }
                                3058 { \peek_execute_branches_catcode: }
                                3059 { peek_token_generic:NN }
                                3060 { \peek_ignore_spaces_execute_branches: }
                                3061 \peek_def:nnnn { peek_catcode_remove:N }
                                3062 { }
                                3063 { peek_token_remove_generic:NN }
                                3064 { \peek_execute_branches_catcode: }
                                3065 \peek_def:nnnn { peek_catcode_remove_ignore_spaces:N }
                                3066 { \peek_execute_branches_catcode: }
                                3067 { peek_token_remove_generic:NN }
                                3068 { \peek_ignore_spaces_execute_branches: }

```

(End definition for \peek_catcode:N and others. These functions are documented on page 56.)

\peek_charcode:N Then for character codes.

```

\peek_charcode_ignore_spaces:N 3069 \peek_def:nnnn { peek_charcode:N }
\peek_charcode_remove:N         3070 { }
\peek_charcode_remove_ignore_spaces:N 3071 { peek_token_generic:NN }
                                3072 { \peek_execute_branches_charcode: }
                                3073 \peek_def:nnnn { peek_charcode_ignore_spaces:N }
                                3074 { \peek_execute_branches_charcode: }
                                3075 { peek_token_generic:NN }
                                3076 { \peek_ignore_spaces_execute_branches: }
                                3077 \peek_def:nnnn { peek_charcode_remove:N }
                                3078 { }
                                3079 { peek_token_remove_generic:NN }
                                3080 { \peek_execute_branches_charcode: }

```

```

3081 \peek_def:nnnn { peek_charcode_remove_ignore_spaces:N }
3082 { \peek_execute_branches_charcode: }
3083 { peek_token_remove_generic:NN }
3084 { \peek_ignore_spaces_execute_branches: }

```

(End definition for \peek_charcode:N and others. These functions are documented on page 56.)

\peek_meaning:N

Finally for meaning, with the group closed to remove the temporary definition functions.

\peek_meaning_ignore_spaces:N

\peek_meaning_remove:N

\peek_meaning_remove_ignore_spaces:N

```

3085 \peek_def:nnnn { peek_meaning:N }
3086 { }
3087 { peek_token_generic:NN }
3088 { \peek_execute_branches_meaning: }
3089 \peek_def:nnnn { peek_meaning_ignore_spaces:N }
3090 { \peek_execute_branches_meaning: }
3091 { peek_token_generic:NN }
3092 { \peek_ignore_spaces_execute_branches: }
3093 \peek_def:nnnn { peek_meaning_remove:N }
3094 { }
3095 { peek_token_remove_generic:NN }
3096 { \peek_execute_branches_meaning: }
3097 \peek_def:nnnn { peek_meaning_remove_ignore_spaces:N }
3098 { \peek_execute_branches_meaning: }
3099 { peek_token_remove_generic:NN }
3100 { \peek_ignore_spaces_execute_branches: }
3101 \group_end:

```

(End definition for \peek_meaning:N and others. These functions are documented on page 57.)

189.5 Decomposing a macro definition

\token_get_prefix_spec:N

\token_get_arg_spec:N

\token_get_replacement_spec:N

\token_get_prefix_arg_replacement_aux:wN

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.

```

3102 \exp_args:Nno \use:nn
3103 { \cs_new:Npn \token_get_prefix_arg_replacement_aux:wN #1 }
3104 { \tl_to_str:n { macro : } #2 -> #3 \q_stop #4 }
3105 { #4 {#1} {#2} {#3} }
3106 \cs_new:Npn \token_get_prefix_spec:N #1
3107 {
3108   \token_if_macro:NTF #1
3109   {
3110     \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3111     \token_to_meaning:N #1 \q_stop \use_i:nnn
3112   }
3113   { \scan_stop: }
3114 }
3115 \cs_new:Npn \token_get_arg_spec:N #1

```



```

3116 {
3117   \token_if_macro:NTF #1
3118   {
3119     \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3120     \token_to_meaning:N #1 \q_stop \use_ii:nnn
3121   }
3122   { \scan_stop: }
3123 }
3124 \cs_new:Npn \token_get_replacement_spec:N #1
3125 {
3126   \token_if_macro:NTF #1
3127   {
3128     \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3129     \token_to_meaning:N #1 \q_stop \use_iii:nnn
3130   }
3131   { \scan_stop: }
3132 }

```

(End definition for \token_get_prefix_spec:N. This function is documented on page ??.)

189.6 Experimental token functions

```

\char_set_active:Npn
\char_set_active:Npx
\char_set_active:Npn
\char_set_active:Npx
\char_set_active_eq:NN
\char_gset_active_eq:NN

```

```

3133 \group_begin:
3134   \char_set_catcode_active:N ^^@
3135   \cs_set:Npn \char_tmp:NN #1#2
3136   {
3137     \cs_new:Npn #1 ##1
3138     {
3139       \char_set_catcode_active:n { '##1 }
3140       \group_begin:
3141       \char_set_lccode:nn { '\^^@ } { '##1 }
3142       \tl_to_lowercase:n { \group_end: #2 ^^@ }
3143     }
3144   }
3145   \char_tmp:NN \char_set_active:Npn   \cs_set:Npn
3146   \char_tmp:NN \char_set_active:Npx   \cs_set:Npx
3147   \char_tmp:NN \char_gset_active:Npn   \cs_gset:Npn
3148   \char_tmp:NN \char_gset_active:Npx   \cs_gset:Npx
3149   \char_tmp:NN \char_set_active_eq:NN  \cs_set_eq:NN
3150   \char_tmp:NN \char_gset_active_eq:NN \cs_gset_eq:NN
3151 \group_end:

```

(End definition for \char_set_active:Npn and \char_set_active:Npx. These functions are documented on page 59.)

\peek_N_type: The next token is normal if it is neither a begin-group token, nor an end-group token, nor a charcode-32 space token. Note that implicit begin-group tokens, end-group tokens, and spaces are also recognized as non-N-type. Here, there is no *search token*, so we feed a dummy \scan_stop: to the \peek_token_generic::NN functions.

```

3152 \cs_new_protected_nopar:Npn \peek_execute_branches_N_type:

```

```

3153 {
3154   \bool_if:nTF
3155   {
3156     \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token ||
3157     \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token   ||
3158     \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
3159   }
3160   { \peek_false:w }
3161   { \peek_true:w }
3162 }
3163 \cs_new_protected_nopar:Npn \peek_N_type:TF
3164 { \peek_token_generic:NNTF \peek_execute_branches_N_type: \scan_stop: }
3165 \cs_new_protected_nopar:Npn \peek_N_type:T
3166 { \peek_token_generic:NNT \peek_execute_branches_N_type: \scan_stop: }
3167 \cs_new_protected_nopar:Npn \peek_N_type:F
3168 { \peek_token_generic:NNTF \peek_execute_branches_N_type: \scan_stop: }

```

(End definition for \peek_N_type:. This function is documented on page ??.)

189.7 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

Primitives renamed.

```

\char_set_catcode:w
\char_set_mathcode:w
\char_set_lccode:w
\char_set_uccode:w
\char_set_sfcode:w

```

```

3169 <deprecated>
3170 \cs_new_eq:NN \char_set_catcode:w \tex_catcode:D
3171 \cs_new_eq:NN \char_set_mathcode:w \tex_mathcode:D
3172 \cs_new_eq:NN \char_set_lccode:w \tex_lccode:D
3173 \cs_new_eq:NN \char_set_uccode:w \tex_uccode:D
3174 \cs_new_eq:NN \char_set_sfcode:w \tex_sfcode:D
3175 </deprecated>

```

(End definition for \char_set_catcode:w. This function is documented on page ??.)

More w functions we should not have.

```

\char_value_catcode:w
\char_show_value_catcode:w
\char_value_mathcode:w
\char_show_value_mathcode:w
\char_value_lccode:w
\char_show_value_lccode:w
\char_value_uccode:w
\char_show_value_uccode:w
\char_value_sfcode:w
\char_show_value_sfcode:w

```

```

3176 <deprecated>
3177 \cs_new_nopar:Npn \char_value_catcode:w { \tex_the:D \char_set_catcode:w }
3178 \cs_new_nopar:Npn \char_show_value_catcode:w
3179 { \tex_showthe:D \char_set_catcode:w }
3180 \cs_new_nopar:Npn \char_value_mathcode:w { \tex_the:D \char_set_mathcode:w }
3181 \cs_new_nopar:Npn \char_show_value_mathcode:w
3182 { \tex_showthe:D \char_set_mathcode:w }
3183 \cs_new_nopar:Npn \char_value_lccode:w { \tex_the:D \char_set_lccode:w }
3184 \cs_new_nopar:Npn \char_show_value_lccode:w
3185 { \tex_showthe:D \char_set_lccode:w }
3186 \cs_new_nopar:Npn \char_value_uccode:w { \tex_the:D \char_set_uccode:w }
3187 \cs_new_nopar:Npn \char_show_value_uccode:w
3188 { \tex_showthe:D \char_set_uccode:w }
3189 \cs_new_nopar:Npn \char_value_sfcode:w { \tex_the:D \char_set_sfcode:w }
3190 \cs_new_nopar:Npn \char_show_value_sfcode:w
3191 { \tex_showthe:D \char_set_sfcode:w }
3192 </deprecated>

```

(End definition for `\char_value_catcode:w`. This function is documented on page ??.)

`\peek_after:NN`
`\peek_gafter:NN`

The second argument here must be `w`.

```
3193 \*deprecated)
3194 \cs_new_eq:NN \peek_after:NN \peek_after:Nw
3195 \cs_new_eq:NN \peek_gafter:NN \peek_gafter:Nw
3196 \end{deprecated}
```

(End definition for `\peek_after:NN`. This function is documented on page ??.)

Functions deprecated 2011-05-28 for removal by 2011-08-31.

`\c_alignment_tab_token`
`\c_math_shift_token`
`\c_letter_token`
`\c_other_char_token`

```
3197 \*deprecated)
3198 \cs_new_eq:NN \c_alignment_tab_token \c_alignment_token
3199 \cs_new_eq:NN \c_math_shift_token \c_math_toggle_token
3200 \cs_new_eq:NN \c_letter_token \c_catcode_letter_token
3201 \cs_new_eq:NN \c_other_char_token \c_catcode_other_token
3202 \end{deprecated}
```

(End definition for `\c_alignment_tab_token`. This function is documented on page ??.)

`\c_active_char_token`

An odd one: this was never a token!

```
3203 \*deprecated)
3204 \cs_new_eq:NN \c_active_char_token \c_catcode_active_tl
3205 \end{deprecated}
```

(End definition for `\c_active_char_token`. This function is documented on page ??.)

`\char_make_escape:N`

Two renames in one block!

`\char_make_group_begin:N`
`\char_make_group_end:N`
`\char_make_math_toggle:N`
`\char_make_alignment:N`
`\char_make_end_line:N`
`\char_make_parameter:N`
`\char_make_math_superscript:N`
`\char_make_math_subscript:N`
`\char_make_ignore:N`
`\char_make_space:N`
`\char_make_letter:N`
`\char_make_other:N`
`\char_make_active:N`
`\char_make_comment:N`
`\char_make_invalid:N`
`\char_make_escape:n`
`\char_make_group_begin:n`
`\char_make_group_end:n`
`\char_make_math_toggle:n`
`\char_make_alignment:n`
`\char_make_end_line:n`
`\char_make_parameter:n`
`\char_make_math_superscript:n`

```
3206 \*deprecated)
3207 \cs_new_eq:NN \char_make_escape:N \char_set_catcode_escape:N
3208 \cs_new_eq:NN \char_make_begin_group:N \char_set_catcode_group_begin:N
3209 \cs_new_eq:NN \char_make_end_group:N \char_set_catcode_group_end:N
3210 \cs_new_eq:NN \char_make_math_shift:N \char_set_catcode_math_toggle:N
3211 \cs_new_eq:NN \char_make_alignment_tab:N \char_set_catcode_alignment:N
3212 \cs_new_eq:NN \char_make_end_line:N \char_set_catcode_end_line:N
3213 \cs_new_eq:NN \char_make_parameter:N \char_set_catcode_parameter:N
3214 \cs_new_eq:NN \char_make_math_superscript:N \char_set_catcode_math_superscript:N
3215 \char_set_catcode_math_superscript:N
3216 \cs_new_eq:NN \char_make_math_subscript:N \char_set_catcode_math_subscript:N
3217 \char_set_catcode_math_subscript:N
3218 \cs_new_eq:NN \char_make_ignore:N \char_set_catcode_ignore:N
3219 \cs_new_eq:NN \char_make_space:N \char_set_catcode_space:N
3220 \cs_new_eq:NN \char_make_letter:N \char_set_catcode_letter:N
3221 \cs_new_eq:NN \char_make_other:N \char_set_catcode_other:N
3222 \cs_new_eq:NN \char_make_active:N \char_set_catcode_active:N
3223 \cs_new_eq:NN \char_make_comment:N \char_set_catcode_comment:N
3224 \cs_new_eq:NN \char_make_invalid:N \char_set_catcode_invalid:N
3225 \cs_new_eq:NN \char_make_escape:n \char_set_catcode_escape:n
3226 \cs_new_eq:NN \char_make_begin_group:n \char_set_catcode_group_begin:n
3227 \cs_new_eq:NN \char_make_end_group:n \char_set_catcode_group_end:n
3228 \cs_new_eq:NN \char_make_math_shift:n \char_set_catcode_math_toggle:n
```

`\char_make_math_subscript:n`
`\char_make_ignore:n`
`\char_make_space:n`
`\char_make_letter:n`
`\char_make_other:n`
`\char_make_active:n`
`\char_make_comment:n`
`\char_make_invalid:n`

```

3229 \cs_new_eq:NN \char_make_alignment_tab:n \char_set_catcode_alignment:n
3230 \cs_new_eq:NN \char_make_end_line:n \char_set_catcode_end_line:n
3231 \cs_new_eq:NN \char_make_parameter:n \char_set_catcode_parameter:n
3232 \cs_new_eq:NN \char_make_math_superscript:n
3233 \char_set_catcode_math_superscript:n
3234 \cs_new_eq:NN \char_make_math_subscript:n
3235 \char_set_catcode_math_subscript:n
3236 \cs_new_eq:NN \char_make_ignore:n \char_set_catcode_ignore:n
3237 \cs_new_eq:NN \char_make_space:n \char_set_catcode_space:n
3238 \cs_new_eq:NN \char_make_letter:n \char_set_catcode_letter:n
3239 \cs_new_eq:NN \char_make_other:n \char_set_catcode_other:n
3240 \cs_new_eq:NN \char_make_active:n \char_set_catcode_active:n
3241 \cs_new_eq:NN \char_make_comment:n \char_set_catcode_comment:n
3242 \cs_new_eq:NN \char_make_invalid:n \char_set_catcode_invalid:n
3243 \</deprecated>

```

(End definition for `\char_make_escape:N` and others. These functions are documented on page ??.)

```

\token_if_alignment_tab:N
\token_if_math_shift:N
\token_if_other_char:N
\token_if_active_char:N

```

```

3244 \*deprecated>
3245 \cs_new_eq:NN \token_if_alignment_tab_p:N \token_if_alignment_p:N
3246 \cs_new_eq:NN \token_if_alignment_tab:NT \token_if_alignment:NT
3247 \cs_new_eq:NN \token_if_alignment_tab:NF \token_if_alignment:NF
3248 \cs_new_eq:NN \token_if_alignment_tab:NTF \token_if_alignment:NTF
3249 \cs_new_eq:NN \token_if_math_shift_p:N \token_if_math_toggle_p:N
3250 \cs_new_eq:NN \token_if_math_shift:NT \token_if_math_toggle:NT
3251 \cs_new_eq:NN \token_if_math_shift:NF \token_if_math_toggle:NF
3252 \cs_new_eq:NN \token_if_math_shift:NTF \token_if_math_toggle:NTF
3253 \cs_new_eq:NN \token_if_other_char_p:N \token_if_other_p:N
3254 \cs_new_eq:NN \token_if_other_char:NT \token_if_other:NT
3255 \cs_new_eq:NN \token_if_other_char:NF \token_if_other:NF
3256 \cs_new_eq:NN \token_if_other_char:NTF \token_if_other:NTF
3257 \cs_new_eq:NN \token_if_active_char_p:N \token_if_active_p:N
3258 \cs_new_eq:NN \token_if_active_char:NT \token_if_active:NT
3259 \cs_new_eq:NN \token_if_active_char:NF \token_if_active:NF
3260 \cs_new_eq:NN \token_if_active_char:NTF \token_if_active:NTF
3261 \</deprecated>

```

(End definition for `\token_if_alignment_tab:N`. This function is documented on page ??.)

```

3262 \</initex | package>

```

190 l3int implementation

```

3263 \*initex | package>

```

The following test files are used for this code: `m3int001,m3int002,m3int03`.

```

3264 \*package>
3265 \ProvidesExplPackage
3266 {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
3267 \package_check_loaded_expl:

```

```

3268 </package>

\int_to_roman:w Done in l3basics.
\if_int_compare:w (End definition for \int_to_roman:w. This function is documented on page 70.)

\int_value:w Here are the remaining primitives for number comparisons and expressions.
\int_eval:w 3269 \cs_new_eq:NN \int_value:w \tex_number:D
\int_eval_end: 3270 \cs_new_eq:NN \int_eval:w \etex_numexpr:D
\if_num:w 3271 \cs_new_eq:NN \int_eval_end: \tex_relax:D
\if_int_odd:w 3272 \cs_new_eq:NN \if_num:w \tex_ifnum:D
\if_case:w 3273 \cs_new_eq:NN \if_int_odd:w \tex_ifodd:D
3274 \cs_new_eq:NN \if_case:w \tex_ifcase:D
(End definition for \int_value:w. This function is documented on page 70.)

```

190.1 Integer expressions

`\int_eval:n` Wrapper for `\int_eval:w`. Can be used in an integer expression or directly in the input stream. In format mode, there is already a definition in `l3alloc` for bookstrapping, which is therefore corrected to the “real” version here.

```

3275 <*initex>
3276 \cs_set:Npn \int_eval:n #1 { \int_value:w \int_eval:w #1 \int_eval_end: }
3277 </initex>
3278 <*package>
3279 \cs_new:Npn \int_eval:n #1 { \int_value:w \int_eval:w #1 \int_eval_end: }
3280 </package>
(End definition for \int_eval:n. This function is documented on page 60.)

```

`\int_max:nn` Functions for min, max, and absolute value.

```

\int_min:nn 3281 \cs_new:Npn \int_abs:n #1
\int_abs:n 3282 {
3283   \int_value:w
3284   \if_int_compare:w \int_eval:w #1 < \c_zero
3285   -
3286   \fi:
3287   \int_eval:w #1 \int_eval_end:
3288 }
3289 \cs_new:Npn \int_max:nn #1#2
3290 {
3291   \int_value:w \int_eval:w
3292   \if_int_compare:w
3293     \int_eval:w #1 > \int_eval:w #2 \int_eval_end:
3294     #1
3295   \else:
3296     #2
3297   \fi:
3298   \int_eval_end:
3299 }
3300 \cs_new:Npn \int_min:nn #1#2
3301 {

```

```

3302     \int_value:w \int_eval:w
3303     \if_int_compare:w
3304         \int_eval:w #1 < \int_eval:w #2 \int_eval_end:
3305         #1
3306     \else:
3307         #2
3308     \fi:
3309     \int_eval_end:
3310 }

```

(End definition for `\int_max:nn`. This function is documented on page 60.)

`\int_div_truncate:nn` As `\int_eval:w` rounds the result of a division we also provide a version that truncates the result. This version is thanks to Heiko Oberdiek: getting things right in all cases is not so easy.

```

3311 \cs_new:Npn \int_div_truncate:nn #1#2
3312 {
3313     \int_value:w \int_eval:w
3314     \if_int_compare:w \int_eval:w #1 = \c_zero
3315     0
3316     \else:
3317         ( #1 % )
3318         \if_int_compare:w \int_eval:w #1 < \c_zero
3319         \if_int_compare:w \int_eval:w #2 < \c_zero
3320             - ( #2 + % )
3321         \else:
3322             + ( #2 - % )
3323         \fi:
3324     \else:
3325         \if_int_compare:w \int_eval:w #2 < \c_zero
3326             + ( #2 + % )
3327         \else:
3328             - ( #2 - % )
3329         \fi:
3330     \fi: % ( (
3331     1 ) / 2 )
3332     \fi:
3333     / ( #2 )
3334     \int_eval_end:
3335 }

```

For the sake of completeness:

```

3336 \cs_new:Npn \int_div_round:nn #1#2 { \int_eval:n { ( #1 ) / ( #2 ) } }

```

Finally there's the modulus operation.

```

3337 \cs_new:Npn \int_mod:nn #1#2
3338 {
3339     \int_value:w \int_eval:w
3340     #1 - \int_div_truncate:nn {#1} {#2} * ( #2 )
3341     \int_eval_end:
3342 }

```

(End definition for `\int_div_truncate:nn`. This function is documented on page 61.)

190.2 Creating and initialising integers

`\int_new:N` Two ways to do this: one for the format and one for the L^AT_EX 2_ε package.

```

\int_new:c 3343 <*package>
3344 \cs_new_protected:Npn \int_new:N #1
3345 {
3346   \chk_if_free_cs:N #1
3347   \newcount #1
3348 }
3349 </package>
3350 \cs_generate_variant:Nn \int_new:N { c }
      (End definition for \int_new:N and \int_new:c. These functions are documented on page ??.)

```

`\int_const:Nn` As stated, most constants can be defined as `\chardef` or `\mathchardef` but that's engine dependent. As a result, there is some set up code to determine what can be done.

```

\int_const:cn
\int_constdef:Nw 3351 \cs_new_protected:Npn \int_const:Nn #1#2
\c_max_const_int 3352 {
3353   \int_compare:nNnTF {#2} > \c_minus_one
3354   {
3355     \int_compare:nNnTF {#2} > \c_max_const_int
3356     {
3357       \int_new:N #1
3358       \int_gset:Nn #1 {#2}
3359     }
3360     {
3361       \chk_if_free_cs:N #1
3362       \tex_global:D \int_constdef:Nw #1 =
3363       \int_eval:w #2 \int_eval_end:
3364     }
3365   }
3366   {
3367     \int_new:N #1
3368     \int_gset:Nn #1 {#2}
3369   }
3370 }
3371 \cs_generate_variant:Nn \int_const:Nn { c }
3372 \pdfTeX_if_engine:TF
3373 {
3374   \cs_new_eq:NN \int_constdef:Nw \tex_mathchardef:D
3375   \tex_mathchardef:D \c_max_const_int 32 767 ~
3376 }
3377 {
3378   \cs_new_eq:NN \int_constdef:Nw \tex_chardef:D
3379   \tex_chardef:D \c_max_const_int 1 114 111 ~
3380 }
      (End definition for \int_const:Nn and \int_const:cn. These functions are documented on page ??.)

```

`\int_zero:N` Functions that reset an *<integer>* register to zero.

```

\int_zero:c
\int_gzero:N
\int_gzero:c

```

```

3381 \cs_new_protected:Npn \int_zero:N #1 { #1 = \c_zero }
3382 \cs_new_protected:Npn \int_gzero:N #1 { \tex_global:D #1 = \c_zero }
3383 \cs_generate_variant:Nn \int_zero:N { c }
3384 \cs_generate_variant:Nn \int_gzero:N { c }

```

(End definition for `\int_zero:N` and `\int_zero:c`. These functions are documented on page ??.)

`\int_zero_new:N` Create a register if needed, otherwise clear it.

```

\int_zero_new:c
\int_gzero_new:N
\int_gzero_new:c

```

```

3385 \cs_new_protected:Npn \int_zero_new:N #1
3386 { \cs_if_exist:NTF #1 { \int_zero:N #1 } { \int_new:N #1 } }
3387 \cs_new_protected:Npn \int_gzero_new:N #1
3388 { \cs_if_exist:NTF #1 { \int_gzero:N #1 } { \int_new:N #1 } }
3389 \cs_generate_variant:Nn \int_zero_new:N { c }
3390 \cs_generate_variant:Nn \int_gzero_new:N { c }

```

(End definition for `\int_zero_new:N` and others. These functions are documented on page ??.)

`\int_set_eq:NN` Setting equal means using one integer inside the set function of another.

```

\int_set_eq:cN
\int_set_eq:Nc
\int_set_eq:cc
\int_gset_eq:NN
\int_gset_eq:cN
\int_gset_eq:Nc
\int_gset_eq:cc

```

```

3391 \cs_new_protected:Npn \int_set_eq:NN #1#2 { #1 = #2 }
3392 \cs_generate_variant:Nn \int_set_eq:NN { c }
3393 \cs_generate_variant:Nn \int_set_eq:NN { Nc , cc }
3394 \cs_new_protected:Npn \int_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
3395 \cs_generate_variant:Nn \int_gset_eq:NN { c }
3396 \cs_generate_variant:Nn \int_gset_eq:NN { Nc , cc }

```

(End definition for `\int_set_eq:NN` and others. These functions are documented on page ??.)

190.3 Setting and incrementing integers

`\int_add:Nn` Adding and subtracting to and from a counter ...

```

\int_add:cn
\int_gadd:Nn
\int_gadd:cn
\int_sub:Nn
\int_sub:cn
\int_gsub:Nn
\int_gsub:cn

```

```

3397 \cs_new_protected:Npn \int_add:Nn #1#2
3398 { \tex_advance:D #1 by \int_eval:w #2 \int_eval_end: }
3399 \cs_new_protected:Npn \int_sub:Nn #1#2
3400 { \tex_advance:D #1 by - \int_eval:w #2 \int_eval_end: }
3401 \cs_new_protected_nopar:Npn \int_gadd:Nn
3402 { \tex_global:D \int_add:Nn }
3403 \cs_new_protected_nopar:Npn \int_gsub:Nn
3404 { \tex_global:D \int_sub:Nn }
3405 \cs_generate_variant:Nn \int_add:Nn { c }
3406 \cs_generate_variant:Nn \int_gadd:Nn { c }
3407 \cs_generate_variant:Nn \int_sub:Nn { c }
3408 \cs_generate_variant:Nn \int_gsub:Nn { c }

```

(End definition for `\int_add:Nn` and `\int_add:cn`. These functions are documented on page ??.)

`\int_incr:N` Incrementing and decrementing of integer registers is done with the following functions.

```

\int_incr:c
\int_gincr:N
\int_gincr:c
\int_decr:N
\int_decr:c
\int_gdecr:N
\int_gdecr:c

```

```

3409 \cs_new_protected:Npn \int_incr:N #1
3410 { \tex_advance:D #1 \c_one }
3411 \cs_new_protected:Npn \int_decr:N #1
3412 { \tex_advance:D #1 \c_minus_one }
3413 \cs_new_protected_nopar:Npn \int_gincr:N
3414 { \tex_global:D \int_incr:N }

```



```

3415 \cs_new_protected_nopar:Npn \int_gdecr:N
3416 { \tex_global:D \int_decr:N }
3417 \cs_generate_variant:Nn \int_incr:N { c }
3418 \cs_generate_variant:Nn \int_decr:N { c }
3419 \cs_generate_variant:Nn \int_gincr:N { c }
3420 \cs_generate_variant:Nn \int_gdecr:N { c }

```

(End definition for `\int_incr:N` and `\int_incr:c`. These functions are documented on page ??.)

`\int_set:Nn` As integers are register-based TeX will issue an error if they are not defined. Thus there is no need for the checking code seen with token list variables.

```

\int_set:cn
\int_gset:Nn
\int_gset:cn
3421 \cs_new_protected:Npn \int_set:Nn #1#2
3422 { #1 ~ \int_eval:w #2\int_eval_end: }
3423 \cs_new_protected_nopar:Npn \int_gset:Nn { \tex_global:D \int_set:Nn }
3424 \cs_generate_variant:Nn \int_set:Nn { c }
3425 \cs_generate_variant:Nn \int_gset:Nn { c }

```

(End definition for `\int_set:Nn` and `\int_set:cn`. These functions are documented on page ??.)

190.4 Using integers

`\int_use:N` Here is how counters are accessed:

```

\int_use:c
3426 \cs_new_eq:NN \int_use:N \tex_the:D
3427 \cs_new:Npn \int_use:c #1 { \int_use:N \cs:w #1 \cs_end: }

```

(End definition for `\int_use:N` and `\int_use:c`. These functions are documented on page ??.)

190.5 Integer expression conditionals

`\int_compare:n` Comparison tests using a simple syntax where only one set of braces is required and additional operators such as `!=` and `>=` are supported. First some notes on the idea behind this. We wish to support writing code like

```

\int_compare_p:n { 5 + \l_tmpa_int != 4 - \l_tmpb_int }

```

In other words, we want to somehow add the missing `\int_eval:w` where required. We can start evaluating from the left using `\int_eval:w`, and we know that since the relation symbols `<`, `>`, `=` and `!` are not allowed in such expressions, they will terminate the expression. Therefore, we first let TeX evaluate this left hand side of the (in)equality.

```

3428 \prg_new_conditional:Npnn \int_compare:n #1 { p , T , F , TF }
3429 { \exp_after:wN \int_compare_aux:nw \int_value:w \int_eval:w #1 \q_stop }

```

Then the next step is to figure out which relation we should use, so we have to somehow get rid of the first evaluation so that we can see what stopped it. `\int_to_roman:w` is handy here since its expansion given a non-positive number is `<null>`. We therefore simply check if the first token of the left hand side evaluation is a minus. If not, we insert it and issue `\int_to_roman:w`, thereby ridding us of the left hand side evaluation. We do however save it for later.

```

3430 \cs_new:Npn \int_compare_aux:nw #1#2 \q_stop
3431 {
3432   \exp_after:wN \int_compare_aux:Nw

```

```

3433 \int_to_roman:w
3434 \if:w #1 -
3435 \else:
3436 -
3437 \fi:
3438 #1#2 \q_mark #1#2 \q_stop
3439 }

```

This leaves the first relation symbol in front and assuming the right hand side has been input, at least one other token as well. We support the following forms: =, <, > and the extended !=, ==, <= and >=. All the extended forms have an extra = so we check if that is present as well. Then use specific function to perform the test.

```

3440 \cs_new:Npn \int_compare_aux:Nw #1#2#3 \q_mark
3441 { \use:c { int_compare_ #1 \if_meaning:w = #2 = \fi: :w } }

```

The actual comparisons are then simple function calls, using the relation as delimiter for a delimited argument. Equality is easy:

```

3442 \cs_new:cpn { int_compare_=:w } #1 = #2 \q_stop
3443 {
3444 \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3445 \prg_return_true:
3446 \else:
3447 \prg_return_false:
3448 \fi:
3449 }

```

So is the one using == we just have to use == in the parameter text.

```

3450 \cs_new:cpn { int_compare_==:w } #1 == #2 \q_stop
3451 {
3452 \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3453 \prg_return_true:
3454 \else:
3455 \prg_return_false:
3456 \fi:
3457 }

```

Not equal is just about reversing the truth value.

```

3458 \cs_new:cpn { int_compare_!=:w } #1 != #2 \q_stop
3459 {
3460 \if_int_compare:w #1 = \int_eval:w #2 \int_eval_end:
3461 \prg_return_false:
3462 \else:
3463 \prg_return_true:
3464 \fi:
3465 }

```

Less than and greater than are also straight forward.

```

3466 \cs_new:cpn { int_compare_<:w } #1 < #2 \q_stop
3467 {
3468 \if_int_compare:w #1 < \int_eval:w #2 \int_eval_end:
3469 \prg_return_true:

```

```

3470     \else:
3471         \prg_return_false:
3472     \fi:
3473 }
3474 \cs_new:cpn { int_compare_>:w } #1 > #2 \q_stop
3475 {
3476     \if_int_compare:w #1 > \int_eval:w #2 \int_eval_end:
3477         \prg_return_true:
3478     \else:
3479         \prg_return_false:
3480     \fi:
3481 }

```

The less than or equal operation is just the opposite of the greater than operation. *Vice versa* for less than or equal.

```

3482 \cs_new:cpn { int_compare_<=:w } #1 <= #2 \q_stop
3483 {
3484     \if_int_compare:w #1 > \int_eval:w #2 \int_eval_end:
3485         \prg_return_false:
3486     \else:
3487         \prg_return_true:
3488     \fi:
3489 }
3490 \cs_new:cpn { int_compare_>=:w } #1 >= #2 \q_stop
3491 {
3492     \if_int_compare:w #1 < \int_eval:w #2 \int_eval_end:
3493         \prg_return_false:
3494     \else:
3495         \prg_return_true:
3496     \fi:
3497 }

```

(End definition for `\int_compare:n`. This function is documented on page ??.)

`\int_compare:nNn` More efficient but less natural in typing.

```

3498 \prg_new_conditional:Npnn \int_compare:nNn #1#2#3 { p , T , F , TF}
3499 {
3500     \if_int_compare:w \int_eval:w #1 #2 \int_eval:w #3 \int_eval_end:
3501         \prg_return_true:
3502     \else:
3503         \prg_return_false:
3504     \fi:
3505 }

```

(End definition for `\int_compare:nNn`. This function is documented on page 63.)

`\int_if_odd:n` A predicate function.

```

\int_if_even:n 3506 \prg_new_conditional:Npnn \int_if_odd:n #1 { p , T , F , TF}
3507 {
3508     \if_int_odd:w \int_eval:w #1 \int_eval_end:
3509         \prg_return_true:

```

```

3510     \else:
3511         \prg_return_false:
3512     \fi:
3513 }
3514 \prg_new_conditional:Npnn \int_if_even:n #1 { p , T , F , TF}
3515 {
3516     \if_int_odd:w \int_eval:w #1 \int_eval_end:
3517         \prg_return_false:
3518     \else:
3519         \prg_return_true:
3520     \fi:
3521 }

```

(End definition for \int_if_odd:n. This function is documented on page 63.)

190.6 Integer expression loops

\int_while_do:nn These are quite easy given the above functions. The while versions test first and then execute the body. The do_while does it the other way round.

```

\int_while_do:nn
\int_until_do:nn
\int_do_while:nn
\int_do_until:nn
3522 \cs_new:Npn \int_while_do:nn #1#2
3523 {
3524     \int_compare:nT {#1}
3525     {
3526         #2
3527         \int_while_do:nn {#1} {#2}
3528     }
3529 }
3530 \cs_new:Npn \int_until_do:nn #1#2
3531 {
3532     \int_compare:nF {#1}
3533     {
3534         #2
3535         \int_until_do:nn {#1} {#2}
3536     }
3537 }
3538 \cs_new:Npn \int_do_while:nn #1#2
3539 {
3540     #2
3541     \int_compare:nT {#1}
3542     { \int_do_while:nn {#1} {#2} }
3543 }
3544 \cs_new:Npn \int_do_until:nn #1#2
3545 {
3546     #2
3547     \int_compare:nF {#1}
3548     { \int_do_until:nn {#1} {#2} }
3549 }

```

(End definition for \int_while_do:nn. This function is documented on page 64.)

\int_while_do:nNnn As above but not using the more natural syntax.
\int_until_do:nNnn
\int_do_while:nNnn
\int_do_until:nNnn

```

3550 \cs_new:Npn \int_while_do:nNnn #1#2#3#4
3551 {
3552   \int_compare:nNnT {#1} #2 {#3}
3553   {
3554     #4
3555     \int_while_do:nNnn {#1} #2 {#3} {#4}
3556   }
3557 }
3558 \cs_new:Npn \int_until_do:nNnn #1#2#3#4
3559 {
3560   \int_compare:nNnF {#1} #2 {#3}
3561   {
3562     #4
3563     \int_until_do:nNnn {#1} #2 {#3} {#4}
3564   }
3565 }
3566 \cs_new:Npn \int_do_while:nNnn #1#2#3#4
3567 {
3568   #4
3569   \int_compare:nNnT {#1} #2 {#3}
3570   { \int_do_while:nNnn {#1} #2 {#3} {#4} }
3571 }
3572 \cs_new:Npn \int_do_until:nNnn #1#2#3#4
3573 {
3574   #4
3575   \int_compare:nNnF {#1} #2 {#3}
3576   { \int_do_until:nNnn {#1} #2 {#3} {#4} }
3577 }

```

(End definition for `\int_while_do:nNnn`. This function is documented on page 64.)

190.7 Formatting integers

`\int_to_arabic:n` Nothing exciting here.

```

3578 \cs_new:Npn \int_to_arabic:n #1 { \int_eval:n {#1} }

```

(End definition for `\int_to_arabic:n`. This function is documented on page 65.)

`\int_to_symbols:nnn` For conversion of integers to arbitrary symbols the method is in general as follows. The input number (#1) is compared to the total number of symbols available at each place (#2). If the input is larger than the total number of symbols available then the modulus is needed, with one added so that the positions don't have to number from zero. Using an f-type expansion, this is done so that the system is recursive. The actual conversion function therefore gets a 'nice' number at each stage. Of course, if the initial input was small enough then there is no problem and everything is easy.

```

3579 \cs_new:Npn \int_to_symbols:nnn #1#2#3
3580 {
3581   \int_compare:nNnTF {#1} > {#2}
3582   {
3583     \exp_args:NNo \exp_args:No \int_to_symbols_aux:nnnn

```

```

3584     {
3585         \prg_case_int:nnn
3586         { 1 + \int_mod:nn { #1 - 1 } {#2} }
3587         {#3} { }
3588     }
3589     {#1} {#2} {#3}
3590 }
3591 { \prg_case_int:nnn {#1} {#3} { } }
3592 }
3593 \cs_new:Npn \int_to_symbols_aux:nnnn #1#2#3#4
3594 {
3595     \exp_args:Nf \int_to_symbols:nnn
3596     { \int_div_truncate:nn { #2 - 1 } {#3} } {#3} {#4}
3597     #1
3598 }

```

(End definition for `\int_to_symbols:nnn`. This function is documented on page 66.)

`\int_to_alph:n` These both use the above function with input functions that make sense for the alphabet
`\int_to_Alph:n` in English.

```

3599 \cs_new:Npn \int_to_alph:n #1
3600 {
3601     \int_to_symbols:nnn {#1} { 26 }
3602     {
3603         { 1 } { a }
3604         { 2 } { b }
3605         { 3 } { c }
3606         { 4 } { d }
3607         { 5 } { e }
3608         { 6 } { f }
3609         { 7 } { g }
3610         { 8 } { h }
3611         { 9 } { i }
3612         { 10 } { j }
3613         { 11 } { k }
3614         { 12 } { l }
3615         { 13 } { m }
3616         { 14 } { n }
3617         { 15 } { o }
3618         { 16 } { p }
3619         { 17 } { q }
3620         { 18 } { r }
3621         { 19 } { s }
3622         { 20 } { t }
3623         { 21 } { u }
3624         { 22 } { v }
3625         { 23 } { w }
3626         { 24 } { x }
3627         { 25 } { y }
3628         { 26 } { z }

```

```

3629     }
3630   }
3631   \cs_new:Npn \int_to_Alph:n #1
3632   {
3633     \int_to_symbols:nnn {#1} { 26 }
3634     {
3635       { 1 } { A }
3636       { 2 } { B }
3637       { 3 } { C }
3638       { 4 } { D }
3639       { 5 } { E }
3640       { 6 } { F }
3641       { 7 } { G }
3642       { 8 } { H }
3643       { 9 } { I }
3644       { 10 } { J }
3645       { 11 } { K }
3646       { 12 } { L }
3647       { 13 } { M }
3648       { 14 } { N }
3649       { 15 } { O }
3650       { 16 } { P }
3651       { 17 } { Q }
3652       { 18 } { R }
3653       { 19 } { S }
3654       { 20 } { T }
3655       { 21 } { U }
3656       { 22 } { V }
3657       { 23 } { W }
3658       { 24 } { X }
3659       { 25 } { Y }
3660       { 26 } { Z }
3661     }
3662   }

```

(End definition for `\int_to_alph:n` and `\int_to_Alph:n`. These functions are documented on page 65.)

<pre> \int_to_base:nn \int_to_base_aux_i:nn \int_to_base_aux_ii:nnN \int_to_base_aux_iii:nnnN \int_to_letter:n </pre>	<p>Converting from base ten (#1) to a second base (#2) starts with computing #1: if it is a complicated calculation, we shouldn't perform it twice. Then check the sign, store it, either - or <code>\c_empty_tl</code>, and feed the absolute value to the next auxiliary function.</p> <pre> 3663 \cs_new:Npn \int_to_base:nn #1 3664 { \exp_args:Nf \int_to_base_aux_i:nn { \int_eval:n {#1} } } 3665 \cs_new:Npn \int_to_base_aux_i:nn #1#2 3666 { 3667 \int_compare:nNnTF {#1} < \c_zero 3668 { \exp_args:No \int_to_base_aux_ii:nnN { \use_none:n #1 } {#2} - } 3669 { \int_to_base_aux_ii:nnN {#1} {#2} \c_empty_tl } 3670 } </pre>
---	---

Here, the idea is to provide a recursive system to deal with the input. The output is built up after the end of the function. At each pass, the value in #1 is checked to see if it is less than the new base (#2). If it is, then it is converted directly, putting the sign back in front. On the other hand, if the value to convert is greater than or equal to the new base then the modulus and remainder values are found. The modulus is converted to a symbol and put on the right, and the remainder is carried forward to the next round.

```

3671 \cs_new:Npn \int_to_base_aux_ii:nnN #1#2#3
3672 {
3673   \int_compare:nNnTF {#1} < {#2}
3674   { \exp_last_unbraced:Nf #3 { \int_to_letter:n {#1} } }
3675   {
3676     \exp_args:Nf \int_to_base_aux_iii:nnnN
3677     { \int_to_letter:n { \int_mod:nn {#1} {#2} } }
3678     {#1}
3679     {#2}
3680     #3
3681   }
3682 }
3683 \cs_new:Npn \int_to_base_aux_iii:nnnN #1#2#3#4
3684 {
3685   \exp_args:Nf \int_to_base_aux_ii:nnN
3686   { \int_div_truncate:nn {#2} {#3} }
3687   {#3}
3688   #4
3689   #1
3690 }

```

Convert to a letter only if necessary, otherwise simply return the value unchanged. It would be cleaner to use `\prg_case_int:nnn`, but in our case, the cases are contiguous, so it is forty times faster to use the `\if_case:w` primitive. The first `\exp_after:wN` expands the conditional, jumping to the correct case, the second one expands after the resulting character to close the conditional. Since #1 might be an expression, and not directly a single digit, we need to evaluate it properly, and expand the trailing `\fi:`.

```

3691 \cs_new:Npn \int_to_letter:n #1
3692 {
3693   \exp_after:wN \exp_after:wN
3694   \if_case:w \int_eval:w #1 - \c_ten \int_eval_end:
3695   A
3696   \or: B
3697   \or: C
3698   \or: D
3699   \or: E
3700   \or: F
3701   \or: G
3702   \or: H
3703   \or: I
3704   \or: J
3705   \or: K
3706   \or: L

```



```

3707     \or: M
3708     \or: N
3709     \or: O
3710     \or: P
3711     \or: Q
3712     \or: R
3713     \or: S
3714     \or: T
3715     \or: U
3716     \or: V
3717     \or: W
3718     \or: X
3719     \or: Y
3720     \or: Z
3721     \else: \int_value:w \int_eval:w #1 \exp_after:wN \int_eval_end:
3722     \fi:
3723 }

```

(End definition for `\int_to_base:nn`. This function is documented on page 69.)

```

\int_to_binary:n  Wrappers around the generic function.
\int_to_hexadecimal:n 3724 \cs_new:Npn \int_to_binary:n #1
\int_to_octal:n    3725 { \int_to_base:nn {#1} { 2 } }
                   3726 \cs_new:Npn \int_to_hexadecimal:n #1
                   3727 { \int_to_base:nn {#1} { 16 } }
                   3728 \cs_new:Npn \int_to_octal:n #1
                   3729 { \int_to_base:nn {#1} { 8 } }

```

(End definition for `\int_to_binary:n`, `\int_to_hexadecimal:n`, and `\int_to_octal:n`. These functions are documented on page 66.)

`\int_to_roman:n` The `\int_to_roman:w` primitive creates tokens of category code 12 (other). Usually, what is actually wanted is letters. The approach here is to convert the output of the primitive into letters using appropriate control sequence names. That keeps everything expandable. The loop will be terminated by the conversion of the Q.

```

\int_to_roman:n 3730 \cs_new:Npn \int_to_roman:n #1
\int_to_Roman:n 3731 {
\int_to_roman_aux:N 3732   \exp_after:wN \int_to_roman_aux:N
\int_to_roman_l:w 3733   \int_to_roman:w \int_eval:n {#1} Q
\int_to_roman_c:w 3734 }
\int_to_roman_d:w 3735 \cs_new:Npn \int_to_roman_aux:N #1
\int_to_roman_m:w 3736 {
\int_to_roman_Q:w 3737   \use:c { int_to_roman_ #1 :w }
\int_to_Roman_i:w 3738   \int_to_roman_aux:N
\int_to_Roman_v:w 3739 }
\int_to_Roman_x:w 3740 \cs_new:Npn \int_to_Roman:n #1
\int_to_Roman_l:w 3741 {
\int_to_Roman_c:w 3742   \exp_after:wN \int_to_Roman_aux:N
\int_to_Roman_d:w 3743   \int_to_roman:w \int_eval:n {#1} Q
\int_to_Roman_m:w 3744 }
\int_to_Roman_Q:w 3745 \cs_new:Npn \int_to_Roman_aux:N #1

```

```

3746 {
3747   \use:c { int_to_Roman_ #1 :w }
3748   \int_to_Roman_aux:N
3749 }
3750 \cs_new_nopar:Npn \int_to_roman_i:w { i }
3751 \cs_new_nopar:Npn \int_to_roman_v:w { v }
3752 \cs_new_nopar:Npn \int_to_roman_x:w { x }
3753 \cs_new_nopar:Npn \int_to_roman_l:w { l }
3754 \cs_new_nopar:Npn \int_to_roman_c:w { c }
3755 \cs_new_nopar:Npn \int_to_roman_d:w { d }
3756 \cs_new_nopar:Npn \int_to_roman_m:w { m }
3757 \cs_new_nopar:Npn \int_to_roman_Q:w #1 { }
3758 \cs_new_nopar:Npn \int_to_Roman_i:w { I }
3759 \cs_new_nopar:Npn \int_to_Roman_v:w { V }
3760 \cs_new_nopar:Npn \int_to_Roman_x:w { X }
3761 \cs_new_nopar:Npn \int_to_Roman_l:w { L }
3762 \cs_new_nopar:Npn \int_to_Roman_c:w { C }
3763 \cs_new_nopar:Npn \int_to_Roman_d:w { D }
3764 \cs_new_nopar:Npn \int_to_Roman_m:w { M }
3765 \cs_new:Npn \int_to_Roman_Q:w #1 { }

```

(End definition for `\int_to_roman:n` and `\int_to_Roman:n`. These functions are documented on page ??.)

190.8 Converting from other formats to integers

`\int_get_sign:n` Finding a number and its sign requires dealing with an arbitrary list of + and - symbols. This is done by working through token by token until there is something else at the start of the input. The sign of the input is tracked by the first Boolean used by the auxiliary function.

`\int_get_digits:n`

`\int_get_sign_and_digits_aux:nNNN`

`\int_get_sign_and_digits_aux:oNNN`

```

3766 \cs_new:Npn \int_get_sign:n #1
3767 {
3768   \int_get_sign_and_digits_aux:nNNN {#1}
3769   \c_true_bool \c_true_bool \c_false_bool
3770 }
3771 \cs_new:Npn \int_get_digits:n #1
3772 {
3773   \int_get_sign_and_digits_aux:nNNN {#1}
3774   \c_true_bool \c_false_bool \c_true_bool
3775 }

```

The auxiliary loops through, finding sign tokens and removing them. The sign itself is carried through as a flag.

```

3776 \cs_new:Npn \int_get_sign_and_digits_aux:nNNN #1#2#3#4
3777 {
3778   \exp_args:Nf \tl_if_head_eq_charcode:nNTF {#1} -
3779   {
3780     \bool_if:NTF #2
3781     {
3782       \int_get_sign_and_digits_aux:oNNN

```

```

3783         { \use_none:n #1 } \c_false_bool #3#4
3784     }
3785     {
3786         \int_get_sign_and_digits_aux:oNNN
3787         { \use_none:n #1 } \c_true_bool #3#4
3788     }
3789 }
3790 {
3791     \exp_args:Nf \tl_if_head_eq_charcode:nNTF {#1} +
3792     { \int_get_sign_and_digits_aux:oNNN { \use_none:n #1 } #2#3#4 }
3793     {
3794         \bool_if:NT #3 { \bool_if:NF #2 - }
3795         \bool_if:NT #4 {#1}
3796     }
3797 }
3798 }
3799 \cs_generate_variant:Nn \int_get_sign_and_digits_aux:nNNN { o }

```

(End definition for \int_get_sign:n. This function is documented on page ??.)

\int_from_alph:n The aim here is to iterate through the input, converting one letter at a time to a number.
\int_from_alph_aux:n The same approach is also used for base conversion, but this needs a different final
\int_from_alph_aux:nN auxiliary.
\int_from_alph_aux:N

```

3800 \cs_new:Npn \int_from_alph:n #1
3801 {
3802     \int_eval:n
3803     {
3804         \int_get_sign:n {#1}
3805         \exp_args:Nf \int_from_alph_aux:n { \int_get_digits:n {#1} }
3806     }
3807 }
3808 \cs_new:Npn \int_from_alph_aux:n #1
3809 { \int_from_alph_aux:nN { 0 } #1 \q_nil }
3810 \cs_new:Npn \int_from_alph_aux:nN #1#2
3811 {
3812     \quark_if_nil:NTF #2
3813     {#1}
3814     {
3815         \exp_args:Nf \int_from_alph_aux:nN
3816         { \int_eval:n { #1 * 26 + \int_from_alph_aux:N #2 } }
3817     }
3818 }
3819 \cs_new:Npn \int_from_alph_aux:N #1
3820 { \int_eval:n { '#1 - \int_compare:nNnTF { '#1 } < { 91 } { 64 } { 96 } } }

```

(End definition for \int_from_alph:n. This function is documented on page ??.)

\int_from_base:nn Conversion to base ten means stripping off the sign then iterating through the input one
\int_from_base_aux:nn token at a time. The total number is then added up as the code loops.
\int_from_base_aux:nnN
\int_from_base_aux:N

```

3821 \cs_new:Npn \int_from_base:nn #1#2
3822 {

```

```

3823 \int_eval:n
3824 {
3825   \int_get_sign:n {#1}
3826   \exp_args:Nf \int_from_base_aux:nn
3827   { \int_get_digits:n {#1} } {#2}
3828 }
3829 }
3830 \cs_new:Npn \int_from_base_aux:nn #1#2
3831 { \int_from_base_aux:nnN { 0 } { #2 } #1 \q_nil }
3832 \cs_new:Npn \int_from_base_aux:nnN #1#2#3
3833 {
3834   \quark_if_nil:NTF #3
3835   {#1}
3836   {
3837     \exp_args:Nf \int_from_base_aux:nnN
3838     { \int_eval:n { #1 * #2 + \int_from_base_aux:N #3 } }
3839     {#2}
3840   }
3841 }

```

The conversion here will take lower or upper case letters and turn them into the appropriate number, hence the two-part nature of the function.

```

3842 \cs_new:Npn \int_from_base_aux:N #1
3843 {
3844   \int_compare:nNnTF { '#1 } < { 58 }
3845   {#1}
3846   {
3847     \int_eval:n
3848     { '#1 - \int_compare:nNnTF { '#1 } < { 91 } { 55 } { 87 } }
3849   }
3850 }

```

(End definition for \int_from_base:nn. This function is documented on page ??.)

```

\int_from_binary:n
\int_from_hexadecimal:n
\int_from_octal:n

```

Wrappers around the generic function.

```

3851 \cs_new:Npn \int_from_binary:n #1
3852 { \int_from_base:nn {#1} \c_two }
3853 \cs_new:Npn \int_from_hexadecimal:n #1
3854 { \int_from_base:nn {#1} \c_sixteen }
3855 \cs_new:Npn \int_from_octal:n #1
3856 { \int_from_base:nn {#1} \c_eight }

```

(End definition for \int_from_binary:n, \int_from_hexadecimal:n, and \int_from_octal:n. These functions are documented on page 67.)

```

\c_int_from_roman_i_int
\c_int_from_roman_v_int
\c_int_from_roman_x_int
\c_int_from_roman_l_int
\c_int_from_roman_c_int
\c_int_from_roman_d_int
\c_int_from_roman_m_int
\c_int_from_roman_I_int
\c_int_from_roman_V_int
\c_int_from_roman_X_int
\c_int_from_roman_L_int
\c_int_from_roman_C_int
\c_int_from_roman_D_int
\c_int_from_roman_M_int

```

Constants used to convert from Roman numerals to integers.

```

3857 \int_const:cn { c_int_from_roman_i_int } { 1 }
3858 \int_const:cn { c_int_from_roman_v_int } { 5 }
3859 \int_const:cn { c_int_from_roman_x_int } { 10 }
3860 \int_const:cn { c_int_from_roman_l_int } { 50 }
3861 \int_const:cn { c_int_from_roman_c_int } { 100 }
3862 \int_const:cn { c_int_from_roman_d_int } { 500 }

```

```

3863 \int_const:cn { c_int_from_roman_m_int } { 1000 }
3864 \int_const:cn { c_int_from_roman_I_int } { 1 }
3865 \int_const:cn { c_int_from_roman_V_int } { 5 }
3866 \int_const:cn { c_int_from_roman_X_int } { 10 }
3867 \int_const:cn { c_int_from_roman_L_int } { 50 }
3868 \int_const:cn { c_int_from_roman_C_int } { 100 }
3869 \int_const:cn { c_int_from_roman_D_int } { 500 }
3870 \int_const:cn { c_int_from_roman_M_int } { 1000 }

```

(End definition for `\c_int_from_roman_i_int` and others. These functions are documented on page ??.)

```

\int_from_roman:n
\int_from_roman_aux:NN
\int_from_roman_end:w
\int_from_roman_clean_up:w

```

The method here is to iterate through the input, finding the appropriate value for each letter and building up a sum. This is then evaluated by `TEX`.

```

3871 \cs_new:Npn \int_from_roman:n #1
3872 {
3873   \tl_if_blank:nF {#1}
3874   {
3875     \exp_after:wN \int_from_roman_end:w
3876     \int_value:w \int_eval:w
3877     \int_from_roman_aux:NN #1 Q \q_stop
3878   }
3879 }
3880 \cs_new:Npn \int_from_roman_aux:NN #1#2
3881 {
3882   \str_if_eq:nnTF {#1} { Q }
3883   {#1#2}
3884   {
3885     \str_if_eq:nnTF {#2} { Q }
3886     {
3887       \cs_if_exist:cF { c_int_from_roman_ #1 _int }
3888       { \int_from_roman_clean_up:w }
3889       +
3890       \use:c { c_int_from_roman_ #1 _int }
3891       #2
3892     }
3893     {
3894       \cs_if_exist:cF { c_int_from_roman_ #1 _int }
3895       { \int_from_roman_clean_up:w }
3896       \cs_if_exist:cF { c_int_from_roman_ #2 _int }
3897       { \int_from_roman_clean_up:w }
3898       \int_compare:nNnTF
3899       { \use:c { c_int_from_roman_ #1 _int } }
3900       <
3901       { \use:c { c_int_from_roman_ #2 _int } }
3902       {
3903         + \use:c { c_int_from_roman_ #2 _int }
3904         - \use:c { c_int_from_roman_ #1 _int }
3905         \int_from_roman_aux:NN
3906       }
3907     }

```

```

3908         + \use:c { c_int_from_roman_ #1 _int }
3909         \int_from_roman_aux:NN #2
3910     }
3911 }
3912 }
3913 }
3914 \cs_new:Npn \int_from_roman_end:w #1 Q #2 \q_stop
3915 { \tl_if_empty:nTF {#2} {#1} {#2} }
3916 \cs_new:Npn \int_from_roman_clean_up:w #1 Q { + 0 Q -1 }
    (End definition for \int_from_roman:n. This function is documented on page ??.)

```

190.9 Viewing integer

```

\int_show:N
\int_show:c
3917 \cs_new_eq:NN \int_show:N \kernel_register_show:N
3918 \cs_new_eq:NN \int_show:c \kernel_register_show:c
    (End definition for \int_show:N and \int_show:c. These functions are documented on page ??.)

\int_show:n
3919 \cs_new_protected:Npn \int_show:n #1
3920 { \tex_showthe:D \int_eval:w #1 \int_eval_end: }
    (End definition for \int_show:n. This function is documented on page 68.)

```

190.10 Constant integers

```

\c_minus_one This is needed early, and so is in l3basics
    (End definition for \c_minus_one. This function is documented on page 68.)

\c_zero Again, one in l3basics for obvious reasons.
    (End definition for \c_zero. This function is documented on page 68.)

\c_six Once again, in l3basics.
\c_seven (End definition for \c_six and \c_seven. These functions are documented on page 68.)
\c_twelve
\c_one Low-number values not previously defined.
\c_sixteen
\c_two
3921 \int_const:Nn \c_one { 1 }
\c_three
3922 \int_const:Nn \c_two { 2 }
\c_four
3923 \int_const:Nn \c_three { 3 }
\c_five
3924 \int_const:Nn \c_four { 4 }
\c_eight
3925 \int_const:Nn \c_five { 5 }
\c_nine
3926 \int_const:Nn \c_eight { 8 }
\c_ten
3927 \int_const:Nn \c_nine { 9 }
\c_eleven
3928 \int_const:Nn \c_ten { 10 }
\c_thirteen
3929 \int_const:Nn \c_eleven { 11 }
\c_fourteen
3930 \int_const:Nn \c_thirteen { 13 }
\c_fifteen
3931 \int_const:Nn \c_fourteen { 14 }
3932 \int_const:Nn \c_fifteen { 15 }
    (End definition for \c_one and others. These functions are documented on page 68.)

```

`\c_thirty_two` One middling value.

```
3933 \int_const:Nn \c_thirty_two { 32 }
```

(End definition for `\c_thirty_two`. This function is documented on page 68.)

`\c_two_hundred_fifty_five` `\c_two_hundred_fifty_six` Two classic mid-range integer constants.

```
3934 \int_const:Nn \c_two_hundred_fifty_five { 255 }
3935 \int_const:Nn \c_two_hundred_fifty_six { 256 }
```

(End definition for `\c_two_hundred_fifty_five` and `\c_two_hundred_fifty_six`. These functions are documented on page 68.)

`\c_one_hundred` `\c_one_thousand` `\c_ten_thousand` Simple runs of powers of ten.

```
3936 \int_const:Nn \c_one_hundred { 100 }
3937 \int_const:Nn \c_one_thousand { 1000 }
3938 \int_const:Nn \c_ten_thousand { 10000 }
```

(End definition for `\c_one_hundred`, `\c_one_thousand`, and `\c_ten_thousand`. These functions are documented on page 68.)

`\c_max_int` The largest number allowed is $2^{31} - 1$

```
3939 \int_const:Nn \c_max_int { 2 147 483 647 }
```

(End definition for `\c_max_int`. This function is documented on page 68.)

190.11 Scratch integers

`\l_tmpa_int` `\l_tmpb_int` `\l_tmpc_int` `\g_tmpa_int` `\g_tmpb_int` We provide four local and two global scratch counters, maybe we need more or less.

```
3940 \int_new:N \l_tmpa_int
3941 \int_new:N \l_tmpb_int
3942 \int_new:N \l_tmpc_int
3943 \int_new:N \g_tmpa_int
3944 \int_new:N \g_tmpb_int
```

(End definition for `\l_tmpa_int`, `\l_tmpb_int`, and `\l_tmpc_int`. These functions are documented on page 69.)

190.12 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\int_convert_from_base_ten:nn` `\int_convert_to_symbols:nnn` `\int_convert_to_base_ten:nn` Some simple renames.

```
3945 <*/deprecated>
3946 \cs_new_eq:NN \int_convert_from_base_ten:nn \int_to_base:nn
3947 \cs_new_eq:NN \int_convert_to_symbols:nnn \int_to_symbols:nnn
3948 \cs_new_eq:NN \int_convert_to_base_ten:nn \int_from_base:nn
3949 </deprecated>
```

(End definition for `\int_convert_from_base_ten:nn`. This function is documented on page ??.)

```

\int_to_symbol:n This is rather too tied to LATEX 2ε.
\int_to_symbol_math:n 3950 <*deprecated>
\int_to_symbol_text:n 3951 \cs_new_nopar:Npn \int_to_symbol:n
3952 {
3953   \scan_align_safe_stop:
3954   \mode_if_math:TF
3955     { \int_to_symbol_math:n }
3956     { \int_to_symbol_text:n }
3957 }
3958 \cs_new:Npn \int_to_symbol_math:n #1
3959 {
3960   \int_to_symbols:nnn {#1} { 9 }
3961   {
3962     { 1 } { * }
3963     { 2 } { \dagger }
3964     { 3 } { \ddagger }
3965     { 4 } { \mathsection }
3966     { 5 } { \mathparagraph }
3967     { 6 } { \| }
3968     { 7 } { ** }
3969     { 8 } { \dagger \dagger }
3970     { 9 } { \ddagger \ddagger }
3971   }
3972 }
3973 \cs_new:Npn \int_to_symbol_text:n #1
3974 {
3975   \int_to_symbols:nnn {#1} { 9 }
3976   {
3977     { 1 } { \textasteriskcentered }
3978     { 2 } { \textdagger }
3979     { 3 } { \textdaggerdbl }
3980     { 4 } { \textsection }
3981     { 5 } { \textparagraph }
3982     { 6 } { \textbardbl }
3983     { 7 } { \textasteriskcentered \textasteriskcentered }
3984     { 8 } { \textdagger \textdagger }
3985     { 9 } { \textdaggerdbl \textdaggerdbl }
3986   }
3987 }
3988 </deprecated>
      (End definition for \int_to_symbol:n. This function is documented on page ??.)
3989 </initex | package>

```

191 l3skip implementation

```

3990 <*initex | package>
3991 <*package>
3992 \ProvidesExplPackage

```



```

3993   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
3994   \package_check_loaded_expl:
3995   \</package>

```

191.1 Length primitives renamed

```

\if_dim:w Primitives renamed.
\dim_eval:w 3996 \cs_new_eq:NN \if_dim:w \tex_ifdim:D
\dim_eval_end: 3997 \cs_new_eq:NN \dim_eval:w \etex_dimexpr:D
3998 \cs_new_eq:NN \dim_eval_end: \tex_relax:D
              (End definition for \if_dim:w. This function is documented on page ??.)

```

191.2 Creating and initialising dim variables

```

\dim_new:N Allocating <dim> registers ...
\dim_new:c 3999 <*package>
4000 \cs_new_protected:Npn \dim_new:N #1
4001 {
4002   \chk_if_free_cs:N #1
4003   \newdimen #1
4004 }
4005 \</package>
4006 \cs_generate_variant:Nn \dim_new:N { c }
              (End definition for \dim_new:N and \dim_new:c. These functions are documented on page ??.)

\dim_zero:N Reset the register to zero.
\dim_zero:c 4007 \cs_new_protected:Npn \dim_zero:N #1 { #1 \c_zero_dim }
\dim_gzero:N 4008 \cs_new_protected:Npn \dim_gzero:N { \tex_global:D \dim_zero:N }
\dim_gzero:c 4009 \cs_generate_variant:Nn \dim_zero:N { c }
4010 \cs_generate_variant:Nn \dim_gzero:N { c }
              (End definition for \dim_zero:N and \dim_zero:c. These functions are documented on page ??.)

```

191.3 Setting dim variables

```

\dim_set:Nn Setting dimensions is easy enough.
\dim_set:cn 4011 \cs_new_protected:Npn \dim_set:Nn #1#2
\dim_gset:Nn 4012 { #1 ~ \dim_eval:w #2 \dim_eval_end: }
\dim_gset:cn 4013 \cs_new_protected:Npn \dim_gset:Nn { \tex_global:D \dim_set:Nn }
4014 \cs_generate_variant:Nn \dim_set:Nn { c }
4015 \cs_generate_variant:Nn \dim_gset:Nn { c }
              (End definition for \dim_set:Nn and \dim_set:cn. These functions are documented on page ??.)

\dim_set_eq:NN All straightforward.
\dim_set_eq:cN 4016 \cs_new_protected:Npn \dim_set_eq:NN #1#2 { #1 = #2 }
\dim_set_eq:Nc 4017 \cs_generate_variant:Nn \dim_set_eq:NN { c }
\dim_set_eq:cc 4018 \cs_generate_variant:Nn \dim_set_eq:NN { Nc , cc }
\dim_gset_eq:NN 4019 \cs_new_protected:Npn \dim_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\dim_gset_eq:cN 4020 \cs_generate_variant:Nn \dim_gset_eq:NN { c }
\dim_gset_eq:Nc 4021 \cs_generate_variant:Nn \dim_gset_eq:NN { Nc , cc }
\dim_gset_eq:cc

```

(End definition for `\dim_set_eq:Nn` and others. These functions are documented on page ??.)

`\dim_set_max:Nn` Setting maximum and minimum values is simply a case of so build-in comparison. This
`\dim_set_max:cn` only applies to dimensions as skips are not ordered.

```

\dim_set_min:Nn 4022 \cs_new_protected:Npn \dim_set_max:Nn #1#2
\dim_set_min:cn 4023 { \dim_compare:nNnT {#1} < {#2} { \dim_set:Nn #1 {#2} } }
\dim_gset_max:Nn 4024 \cs_new_protected:Npn \dim_gset_max:Nn
\dim_gset_max:cn 4025 { \tex_global:D \dim_set_max:Nn }
\dim_gset_min:Nn 4026 \cs_new_protected:Npn \dim_set_min:Nn #1#2
\dim_gset_min:cn 4027 { \dim_compare:nNnT {#1} > {#2} { \dim_set:Nn #1 {#2} } }
4028 \cs_new_protected:Npn \dim_gset_min:Nn
4029 { \tex_global:D \dim_set_min:Nn }
4030 \cs_generate_variant:Nn \dim_set_max:Nn { c }
4031 \cs_generate_variant:Nn \dim_gset_max:Nn { c }
4032 \cs_generate_variant:Nn \dim_set_min:Nn { c }
4033 \cs_generate_variant:Nn \dim_gset_min:Nn { c }

```

(End definition for `\dim_set_max:Nn` and `\dim_set_max:cn`. These functions are documented on page ??.)

`\dim_add:Nn` Using by here deals with the (incorrect) case `\dimen123`.

```

\dim_add:cn 4034 \cs_new_protected:Npn \dim_add:Nn #1#2
\dim_gadd:Nn 4035 { \tex_advance:D #1 by \dim_eval:w #2 \dim_eval_end: }
\dim_gadd:cn 4036 \cs_new_protected:Npn \dim_gadd:Nn { \tex_global:D \dim_add:Nn }
\dim_sub:Nn 4037 \cs_generate_variant:Nn \dim_add:Nn { c }
\dim_sub:cn 4038 \cs_generate_variant:Nn \dim_gadd:Nn { c }
\dim_gsub:Nn 4039 \cs_new_protected:Npn \dim_sub:Nn #1#2
\dim_gsub:cn 4040 { \tex_advance:D #1 by - \dim_eval:w #2 \dim_eval_end: }
4041 \cs_new_protected:Npn \dim_gsub:Nn { \tex_global:D \dim_sub:Nn }
4042 \cs_generate_variant:Nn \dim_sub:Nn { c }
4043 \cs_generate_variant:Nn \dim_gsub:Nn { c }

```

(End definition for `\dim_add:Nn` and `\dim_add:cn`. These functions are documented on page ??.)

191.4 Utilities for dimension calculations

`\dim_abs:n` Similar to the `\int_abs:n` function, but here an additional $\langle dimexpr \rangle$ is needed as TeX won't simply tidy up an additional – for us.

```

4044 \cs_new:Npn \dim_abs:n #1
4045 {
4046   \dim_use:N
4047   \dim_eval:w
4048   \if_dim:w \dim_eval:w #1 < \c_zero_dim
4049   -
4050   \fi:
4051   \dim_eval:w #1 \dim_eval_end:
4052   \dim_eval_end:
4053 }

```

(End definition for `\dim_abs:n`. This function is documented on page 72.)

`\dim_ratio:nn` With dimension expressions, something like `10 pt * (5 pt / 10 pt)` will not work. `\dim_ratio_aux:n` Instead, the ratio part needs to be converted to an integer expression. Using `\int_value:w` forces everything into `sp`, avoiding any decimal parts.

```

4054 \cs_new:Npn \dim_ratio:nn #1#2
4055 { \dim_ratio_aux:n {#1} / \dim_ratio_aux:n {#2} }
4056 \cs_new:Npn \dim_ratio_aux:n #1
4057 { \int_value:w \dim_eval:w #1 \dim_eval_end: }

```

(End definition for \dim_ratio:nn. This function is documented on page ??.)

191.5 Dimension expression conditionals

```

\dim_compare_p:nNn
\dim_compare:nNn

```

```

4058 \prg_new_conditional:Npnn \dim_compare:nNn #1#2#3 { p , T , F , TF }
4059 {
4060   \if_dim:w \dim_eval:w #1 #2 \dim_eval:w #3 \dim_eval_end:
4061   \prg_return_true: \else: \prg_return_false: \fi:
4062 }

```

(End definition for \dim_compare_p:nNn. This function is documented on page 73.)

`\dim_compare:n` [This code plus comments are adapted from the `\int_compare:nTF` function.] Comparison tests using a simple syntax where only one set of braces is required and additional operators such as `!=` and `>=` are supported. First some notes on the idea behind this. We wish to support writing code like

```

\dim_compare_p:n { 5mm + \l_tmpa_dim >= 4pt - \l_tmpb_dim }

```

`\dim_compare_aux:wNN`
`\dim_compare_<:nw`
`\dim_compare_=:nw`
`\dim_compare_>:nw`
`\dim_compare_==:nw`
`\dim_compare_<=:nw`
`\dim_compare_!=:nw`
`\dim_compare_>=:nw`

In other words, we want to somehow add the missing `\dim_eval:w` where required. We can start evaluating from the left using `\dim_use:N \dim_eval:w`, and we know that since the relation symbols `<`, `>`, `=` and `!` are not allowed in such expressions, they will terminate the expression. Therefore, we first let `TeX` evaluate this left hand side of the (in)equality.

Eventually, we will convert the relation symbol to the appropriate version of `\if_dim:w`, and add `\dim_eval:w` after it. We optimize by placing the end-code already here: this avoids needless grabbing of arguments later.

```

4063 \prg_new_conditional:Npnn \dim_compare:n #1 { p , T , F , TF }
4064 {
4065   \exp_after:wN \dim_compare_aux:wNN \dim_use:N \dim_eval:w #1
4066   \dim_eval_end:
4067   \prg_return_true:
4068   \else:
4069   \prg_return_false:
4070   \fi:
4071 }

```

Contrarily to the case of integers, where we have to remove the result in order to access the relation, `\dim_use:N` nicely produces a result which ends in `pt`. We can thus use a delimited argument to find the relation. `\tl_to_str:n` is needed to convert `pt` to “other” characters.

The relation might be one character, #2, or two characters #2#3. We support the following forms: =, <, > and the extended !=, ==, <= and >=. All the extended forms have an extra = so we check if that is present as well. Then use specific function to perform the (unbalanced) test.

```

4072 \exp_args:Nno \use:nn
4073 { \cs_new:Npn \dim_compare_aux:wNN #1 }
4074 { \tl_to_str:n { pt } }
4075 #2 #3
4076 {
4077   \use:c
4078   {
4079     dim_compare_ #2
4080     \if_meaning:w = #3 = \fi:
4081     :nw
4082   }
4083   { #1 pt } #3
4084 }

```

Here, \dim_eval:w will begin the right hand side of a dimension comparison (with \if_dim:w), closed cleanly by the trailing tokens we put in the definition of \dim_compare:n.

The actual comparisons take as a first argument the left-hand side of the comparison (a length). In the case of normal comparisons, just place the relevant \if_dim:w, with a trailing \dim_eval:w to evaluate the right hand side. For extended comparisons, remove the trailing = that we left, before evaluating with \dim_eval:w. In both cases, the expansion of \dim_eval:w is stopped properly, and the conditional ended correctly by the tokens we put in the definition of \dim_compare:n.

Equal, less than and greater than are straightforward.

```

4085 \cs_new:cpn { dim_compare_<:nw } #1 { \if_dim:w #1 < \dim_eval:w }
4086 \cs_new:cpn { dim_compare_=:nw } #1 { \if_dim:w #1 = \dim_eval:w }
4087 \cs_new:cpn { dim_compare_>:nw } #1 { \if_dim:w #1 > \dim_eval:w }

```

For the extended syntax ==, we remove #2, trailing = sign, and otherwise act as for =.

```

4088 \cs_new:cpn {dim_compare_==:nw} #1#2 { \if_dim:w #1 = \dim_eval:w }

```

Not equal, greater than or equal, less than or equal follow the same scheme as the extended equality syntax, with an additional \reverse_if:N to get the opposite of their “simple” analog.

```

4089 \cs_new:cpn {dim_compare_<=:nw} #1#2 {\reverse_if:N \if_dim:w #1 > \dim_eval:w}
4090 \cs_new:cpn {dim_compare_!=:nw} #1#2 {\reverse_if:N \if_dim:w #1 = \dim_eval:w}
4091 \cs_new:cpn {dim_compare_>=:nw} #1#2 {\reverse_if:N \if_dim:w #1 < \dim_eval:w}

```

(End definition for \dim_compare:n. This function is documented on page ??.)

191.6 Dimension expression loops

\dim_while_do:nn while_do and do_while functions for dimensions. Same as for the int type only the names have changed.

```

\dim_do_while:nn
\dim_do_until:nn
4092 \cs_set:Npn \dim_while_do:nn #1#2
4093 {
4094   \dim_compare:nT {#1}

```

```

4095     {
4096         #2
4097         \dim_while_do:nn {#1} {#2}
4098     }
4099 }
4100 \cs_set:Npn \dim_until_do:nn #1#2
4101 {
4102     \dim_compare:nF {#1}
4103     {
4104         #2
4105         \dim_until_do:nn {#1} {#2}
4106     }
4107 }
4108 \cs_set:Npn \dim_do_while:nn #1#2
4109 {
4110     #2
4111     \dim_compare:nT {#1}
4112     { \dim_do_while:nNnn {#1} {#2} }
4113 }
4114 \cs_set:Npn \dim_do_until:nn #1#2
4115 {
4116     #2
4117     \dim_compare:nF {#1}
4118     { \dim_do_until:nn {#1} {#2} }
4119 }

```

(End definition for \dim_while_do:nn. This function is documented on page 74.)

\dim_while_do:nNnn while_do and do_while functions for dimensions. Same as for the int type only the names have changed.

```

\dim_until_do:nNnn
\dim_do_while:nNnn
\dim_do_until:nNnn
4120 \cs_set:Npn \dim_while_do:nNnn #1#2#3#4
4121 {
4122     \dim_compare:nNnT {#1} #2 {#3}
4123     {
4124         #4
4125         \dim_while_do:nNnn {#1} #2 {#3} {#4}
4126     }
4127 }
4128 \cs_set:Npn \dim_until_do:nNnn #1#2#3#4
4129 {
4130     \dim_compare:nNnF {#1} #2 {#3}
4131     {
4132         #4
4133         \dim_until_do:nNnn {#1} #2 {#3} {#4}
4134     }
4135 }
4136 \cs_set:Npn \dim_do_while:nNnn #1#2#3#4
4137 {
4138     #4
4139     \dim_compare:nNnT {#1} #2 {#3}

```

```

4140     { \dim_do_while:nNnn {#1} #2 {#3} {#4} }
4141   }
4142   \cs_set:Npn \dim_do_until:nNnn #1#2#3#4
4143   {
4144     #4
4145     \dim_compare:nNnF {#1} #2 {#3}
4146     { \dim_do_until:nNnn {#1} #2 {#3} {#4} }
4147   }

```

(End definition for `\dim_while_do:nNnn`. This function is documented on page 73.)

191.7 Using dim expressions and variables

`\dim_eval:n` Evaluating a dimension expression expandably.

```

4148   \cs_new:Npn \dim_eval:n #1
4149   { \dim_use:N \dim_eval:w #1 \dim_eval_end: }

```

(End definition for `\dim_eval:n`. This function is documented on page 75.)

`\dim_strip_bp:n`

```

4150   \cs_new:Npn \dim_strip_bp:n #1
4151   { \dim_strip_pt:n { 0.996 26 \dim_eval:w #1 \dim_eval_end: } }

```

(End definition for `\dim_strip_bp:n`. This function is documented on page 81.)

`\dim_strip_pt:n` A function which comes up often enough to deserve a place in the kernel. The idea here is that the input is assumed to be in `pt`, but can be given in other units, while the output is the value of the dimension in `pt` but with no units given. This is used a lot by low-level manipulations.

`\dim_strip_pt:w`

```

4152   \cs_new:Npn \dim_strip_pt:n #1
4153   {
4154     \exp_after:wN
4155     \dim_strip_pt:w \dim_use:N \dim_eval:w #1 \dim_eval_end: \q_stop
4156   }
4157   \use:x
4158   {
4159     \cs_new:Npn \exp_not:N \dim_strip_pt:w
4160     ##1 . ##2 \tl_to_str:n { pt } ##3 \exp_not:N \q_stop
4161     {
4162       ##1
4163       \exp_not:N \int_compare:nNnT {##2} > \c_zero
4164       { . ##2 }
4165     }
4166   }

```

(End definition for `\dim_strip_pt:n`. This function is documented on page ??.)

`\dim_use:N` Accessing a $\langle dim \rangle$.

`\dim_use:c`

```

4167   \cs_new_eq:NN \dim_use:N \tex_the:D
4168   \cs_generate_variant:Nn \dim_use:N { c }

```

(End definition for `\dim_use:N` and `\dim_use:c`. These functions are documented on page ??.)

191.8 Viewing dim variables

`\dim_show:N` Diagnostics.

`\dim_show:c` 4169 `\cs_new_eq:NN \dim_show:N \kernel_register_show:N`
4170 `\cs_generate_variant:Nn \dim_show:N { c }`
(End definition for `\dim_show:N` and `\dim_show:c`. These functions are documented on page ??.)

`\dim_show:n` Diagnostics.

4171 `\cs_new_protected:Npn \dim_show:n #1`
4172 `{ \tex_showthe:D \dim_eval:w #1 \dim_eval_end: }`
(End definition for `\dim_show:n`. This function is documented on page 75.)

191.9 Constant dimensions

`\c_zero_dim` The source for these depends on whether we are in package mode.

`\c_max_dim` 4173 `\<initex>`
4174 `\dim_new:N \c_zero_dim`
4175 `\dim_new:N \c_max_dim`
4176 `\dim_set:Nn \c_max_dim { 16383.99999 pt }`
4177 `\</initex>`
4178 `\<*package>`
4179 `\cs_new_eq:NN \c_zero_dim \z@`
4180 `\cs_new_eq:NN \c_max_dim \maxdimen`
4181 `\</package>`
(End definition for `\c_zero_dim`. This function is documented on page 75.)

191.10 Scratch dimensions

`\l_tmpa_dim` We provide three local and two global scratch registers, maybe we need more or less.

`\l_tmpp_dim` 4182 `\dim_new:N \l_tmpa_dim`
`\l_tmppc_dim` 4183 `\dim_new:N \l_tmppb_dim`
`\g_tmpa_dim` 4184 `\dim_new:N \l_tmppc_dim`
`\g_tmpp_dim` 4185 `\dim_new:N \g_tmpa_dim`
4186 `\dim_new:N \g_tmppb_dim`
(End definition for `\l_tmpa_dim`, `\l_tmppb_dim`, and `\l_tmppc_dim`. These functions are documented on page 76.)

191.11 Creating and initialising skip variables

`\skip_new:N` Allocation of a new internal registers.

`\skip_new:c` 4187 `\<*package>`
4188 `\cs_new_protected:Npn \skip_new:N #1`
4189 `{`
4190 `\chk_if_free_cs:N #1`
4191 `\newskip #1`
4192 `}`
4193 `\</package>`
4194 `\cs_generate_variant:Nn \skip_new:N { c }`

(End definition for `\skip_new:N` and `\skip_new:c`. These functions are documented on page ??.)

```

\skip_zero:N Reset the register to zero.
\skip_zero:c 4195 \cs_new_protected:Npn \skip_zero:N #1 { #1 \c_zero_skip }
\skip_gzero:N 4196 \cs_new_protected:Npn \skip_gzero:N { \tex_global:D \skip_zero:N }
\skip_gzero:c 4197 \cs_generate_variant:Nn \skip_zero:N { c }
               4198 \cs_generate_variant:Nn \skip_gzero:N { c }

```

(End definition for `\skip_zero:N` and `\skip_zero:c`. These functions are documented on page ??.)

191.12 Setting skip variables

```

\skip_set:Nn Much the same as for dimensions.
\skip_set:cn 4199 \cs_new_protected:Npn \skip_set:Nn #1#2
\skip_gset:Nn 4200 { #1 ~ \etex_glueexpr:D #2 \scan_stop: }
\skip_gset:cn 4201 \cs_new_protected:Npn \skip_gset:Nn { \tex_global:D \skip_set:Nn }
               4202 \cs_generate_variant:Nn \skip_set:Nn { c }
               4203 \cs_generate_variant:Nn \skip_gset:Nn { c }

```

(End definition for `\skip_set:Nn` and `\skip_set:cn`. These functions are documented on page ??.)

```

\skip_set_eq:NN All straightforward.
\skip_set_eq:cN 4204 \cs_new_protected:Npn \skip_set_eq:NN #1#2 { #1 = #2 }
\skip_set_eq:Nc 4205 \cs_generate_variant:Nn \skip_set_eq:NN { c }
\skip_set_eq:cc 4206 \cs_generate_variant:Nn \skip_set_eq:NN { Nc , cc }
\skip_gset_eq:NN 4207 \cs_new_protected:Npn \skip_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\skip_gset_eq:cN 4208 \cs_generate_variant:Nn \skip_gset_eq:NN { c }
\skip_gset_eq:Nc 4209 \cs_generate_variant:Nn \skip_gset_eq:NN { Nc , cc }
\skip_gset_eq:cc (End definition for \skip_set_eq:NN and others. These functions are documented on page ??.)

```

```

\skip_add:Nn Using by here deals with the (incorrect) case \skip123.
\skip_add:cn 4210 \cs_new_protected:Npn \skip_add:Nn #1#2
\skip_gadd:Nn 4211 { \tex_advance:D #1 by \etex_glueexpr:D #2 \scan_stop: }
\skip_gadd:cn 4212 \cs_new_protected:Npn \skip_gadd:Nn { \tex_global:D \skip_add:Nn }
\skip_sub:Nn 4213 \cs_generate_variant:Nn \skip_add:Nn { c }
\skip_sub:cn 4214 \cs_generate_variant:Nn \skip_gadd:Nn { c }
\skip_gsub:Nn 4215 \cs_new_protected:Npn \skip_sub:Nn #1#2
\skip_gsub:cn 4216 { \tex_advance:D #1 by - \etex_glueexpr:D #2 \scan_stop: }
               4217 \cs_new_protected:Npn \skip_gsub:Nn { \tex_global:D \skip_sub:Nn }
               4218 \cs_generate_variant:Nn \skip_sub:Nn { c }
               4219 \cs_generate_variant:Nn \skip_gsub:Nn { c }

```

(End definition for `\skip_add:Nn` and `\skip_add:cn`. These functions are documented on page ??.)

191.13 Skip expression conditionals

`\skip_if_eq:n` Comparing skips means doing two expansions to make strings, and then testing them. As a result, only equality is tested.

```

4220 \prg_new_conditional:Npnn \skip_if_eq:n #1#2 { p , T , F , TF }
4221 {
4222   \if_int_compare:w
4223     \pdfTeX_strcmp:D { \skip_eval:n { #1 } } { \skip_eval:n { #2 } }
4224     = \c_zero
4225     \prg_return_true:
4226   \else:
4227     \prg_return_false:
4228   \fi:
4229 }
```

(End definition for \skip_if_eq:n. This function is documented on page 77.)

`\skip_if_infinite_glue:n` 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. `csskip_if_infinite_glue:nTF` tests it directly by looking at the stretch and shrink order. If either of the predicate functions return $\langle true \rangle$, `\bool_if:nTF` will return $\langle true \rangle$ and the logic test will take the true branch.

```

4230 \prg_new_conditional:Npnn \skip_if_infinite_glue:n #1 { p , T , F , TF }
4231 {
4232   \bool_if:nTF
4233   {
4234     \int_compare_p:nNn { \etex_gluestretchorder:D #1 } > \c_zero ||
4235     \int_compare_p:nNn { \etex_glueshrinkorder:D #1 } > \c_zero
4236   }
4237   { \prg_return_true: }
4238   { \prg_return_false: }
4239 }
```

(End definition for \skip_if_infinite_glue:n. This function is documented on page 77.)

191.14 Using skip expressions and variables

`\skip_eval:n` Evaluating a skip expression expandably.

```

4240 \cs_new:Npn \skip_eval:n #1
4241 { \skip_use:N \etex_glueexpr:D #1 \scan_stop: }
(End definition for \skip_eval:n. This function is documented on page 77.)
```

`\skip_use:N` Accessing a $\langle skip \rangle$.

```

4242 \cs_new_eq:NN \skip_use:N \tex_the:D
4243 \cs_generate_variant:Nn \skip_use:N { c }
(End definition for \skip_use:N and \skip_use:c. These functions are documented on page ??.)
```

191.15 Inserting skips into the output

`\skip_horizontal:N` Inserting skips.

```

\skip_horizontal:c 4244 \cs_new_eq:NN \skip_horizontal:N \tex_hskip:D
\skip_horizontal:n 4245 \cs_new:Npn \skip_horizontal:n #1
                    4246 { \skip_horizontal:N \etex_glueexpr:D #1 \scan_stop: }
\skip_vertical:N    4247 \cs_new_eq:NN \skip_vertical:N \tex_vskip:D
\skip_vertical:c    4248 \cs_new:Npn \skip_vertical:n #1
                    4249 { \skip_vertical:N \etex_glueexpr:D #1 \scan_stop: }
\skip_vertical:n    4250 \cs_generate_variant:Nn \skip_horizontal:N { c }
                    4251 \cs_generate_variant:Nn \skip_vertical:N { c }

```

(End definition for `\skip_horizontal:N`, `\skip_horizontal:c`, and `\skip_horizontal:n`. These functions are documented on page ??.)

191.16 Viewing skip variables

`\skip_show:N` Diagnostics.

```

\skip_show:c 4252 \cs_new_eq:NN \skip_show:N \kernel_register_show:N
              4253 \cs_generate_variant:Nn \skip_show:N { c }

```

(End definition for `\skip_show:N` and `\skip_show:c`. These functions are documented on page ??.)

`\skip_show:n` Diagnostics.

```

              4254 \cs_new_protected:Npn \skip_show:n #1
              4255 { \tex_showthe:D \etex_glueexpr:D #1 \scan_stop: }

```

(End definition for `\skip_show:n`. This function is documented on page 78.)

191.17 Constant skips

`\c_zero_skip` Skips with no rubber component are just dimensions

```

\c_max_skip 4256 \cs_new_eq:NN \c_zero_skip \c_zero_dim
              4257 \cs_new_eq:NN \c_max_skip \c_max_dim

```

(End definition for `\c_zero_skip`. This function is documented on page 78.)

191.18 Scratch skips

`\l_tmpa_skip` We provide three local and two global scratch registers, maybe we need more or less.

```

\l_tmpb_skip 4258 \skip_new:N \l_tmpa_skip
\l_tmpc_skip 4259 \skip_new:N \l_tmpb_skip
\g_tmpa_skip 4260 \skip_new:N \l_tmpc_skip
\g_tmpb_skip 4261 \skip_new:N \g_tmpa_skip
              4262 \skip_new:N \g_tmpb_skip

```

(End definition for `\l_tmpa_skip`, `\l_tmpb_skip`, and `\l_tmpc_skip`. These functions are documented on page 78.)

191.19 Creating and initialising muskip variables

`\muskip_new:N` And then we add muskips.

```
\muskip_new:c 4263 \*package>
4264 \cs_new_protected:Npn \muskip_new:N #1
4265 {
4266     \chk_if_free_cs:N #1
4267     \newmuskip #1
4268 }
4269 \</package>
4270 \cs_generate_variant:Nn \muskip_new:N { c }
(End definition for \muskip_new:N and \muskip_new:c. These functions are documented on page ??.)
```

`\muskip_zero:N` Reset the register to zero.

```
\muskip_zero:c 4271 \cs_new_protected:Npn \muskip_zero:N #1
\muskip_gzero:N 4272 { #1 \c_zero_muskip }
\muskip_gzero:c 4273 \cs_new_protected:Npn \muskip_gzero:N { \tex_global:D \muskip_zero:N }
4274 \cs_generate_variant:Nn \muskip_zero:N { c }
4275 \cs_generate_variant:Nn \muskip_gzero:N { c }
(End definition for \muskip_zero:N and \muskip_zero:c. These functions are documented on page ??.)
```

191.20 Setting muskip variables

`\muskip_set:Nn` This should be pretty familiar.

```
\muskip_set:cn 4276 \cs_new_protected:Npn \muskip_set:Nn #1#2
\muskip_gset:Nn 4277 { #1 ~ \etex_muexpr:D #2 \scan_stop: }
\muskip_gset:cn 4278 \cs_new_protected:Npn \muskip_gset:Nn { \tex_global:D \muskip_set:Nn }
4279 \cs_generate_variant:Nn \muskip_set:Nn { c }
4280 \cs_generate_variant:Nn \muskip_gset:Nn { c }
(End definition for \muskip_set:Nn and \muskip_set:cn. These functions are documented on page ??.)
```

`\muskip_set_eq:NN` All straightforward.

```
\muskip_set_eq:cN 4281 \cs_new_protected:Npn \muskip_set_eq:NN #1#2 { #1 = #2 }
\muskip_set_eq:Nc 4282 \cs_generate_variant:Nn \muskip_set_eq:NN { c }
\muskip_set_eq:cc 4283 \cs_generate_variant:Nn \muskip_set_eq:NN { Nc , cc }
\muskip_gset_eq:NN 4284 \cs_new_protected:Npn \muskip_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\muskip_gset_eq:cN 4285 \cs_generate_variant:Nn \muskip_gset_eq:NN { c }
\muskip_gset_eq:Nc 4286 \cs_generate_variant:Nn \muskip_gset_eq:NN { Nc , cc }
\muskip_gset_eq:cc (End definition for \muskip_set_eq:NN and others. These functions are documented on page ??.)
```

`\muskip_add:Nn` Using by here deals with the (incorrect) case `\muskip123`.

```
\muskip_add:cn 4287 \cs_new_protected:Npn \muskip_add:Nn #1#2
\muskip_gadd:Nn 4288 { \tex_advance:D #1 by \etex_muexpr:D #2 \scan_stop: }
\muskip_gadd:cn 4289 \cs_new_protected:Npn \muskip_gadd:Nn { \tex_global:D \muskip_add:Nn }
\muskip_sub:Nn 4290 \cs_generate_variant:Nn \muskip_add:Nn { c }
\muskip_sub:cn 4291 \cs_generate_variant:Nn \muskip_gadd:Nn { c }
\muskip_gsub:Nn
\muskip_gsub:cn
```

```

4292 \cs_new_protected:Npn \muskip_sub:Nn #1#2
4293 { \tex_advance:D #1 by - \etex_muexpr:D #2 \scan_stop: }
4294 \cs_new_protected:Npn \muskip_gsub:Nn { \tex_global:D \muskip_sub:Nn }
4295 \cs_generate_variant:Nn \muskip_sub:Nn { c }
4296 \cs_generate_variant:Nn \muskip_gsub:Nn { c }
      (End definition for \muskip_add:Nn and \muskip_add:cn. These functions are documented on
page ??.)

```

191.21 Using muskip expressions and variables

`\muskip_eval:n` Evaluating a muskip expression expandably.

```

4297 \cs_new:Npn \muskip_eval:n #1
4298 { \muskip_use:N \etex_muexpr:D #1 \scan_stop: }
      (End definition for \muskip_eval:n. This function is documented on page 79.)

```

`\muskip_use:N` Accessing a $\langle muskip \rangle$.

```

\muskip_use:c
4299 \cs_new_eq:NN \muskip_use:N \tex_the:D
4300 \cs_generate_variant:Nn \muskip_use:N { c }
      (End definition for \muskip_use:N and \muskip_use:c. These functions are documented on page
??.)

```

191.22 Viewing muskip variables

`\muskip_show:N` Diagnostics.

```

\muskip_show:c
4301 \cs_new_eq:NN \muskip_show:N \kernel_register_show:N
4302 \cs_generate_variant:Nn \muskip_show:N { c }
      (End definition for \muskip_show:N and \muskip_show:c. These functions are documented on
page ??.)

```

`\muskip_show:n` Diagnostics.

```

4303 \cs_new_protected:Npn \muskip_show:n #1
4304 { \tex_showthe:D \etex_muexpr:D #1 \scan_stop: }
      (End definition for \muskip_show:n. This function is documented on page 80.)

```

191.23 Experimental skip functions

`\skip_split_finite_else_action:nnNN` This macro is useful when performing error checking in certain circumstances. If the $\langle skip \rangle$ 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.

```

4305 \cs_new:Npn \skip_split_finite_else_action:nnNN #1#2#3#4
4306 {
4307   \skip_if_infinite_glue:nTF {#1}
4308   {
4309     #3 = \c_zero_skip
4310     #4 = \c_zero_skip
4311     #2

```

```

4312     }
4313     {
4314         #3 = \etex_gluestretch:D #1 \scan_stop:
4315         #4 = \etex_glueshrink:D #1 \scan_stop:
4316     }
4317 }

```

(End definition for `\skip_split_finite_else_action:nnNN`. This function is documented on page 81.)

```

4318 </initex | package>

```

192 l3tl implementation

```

4319 <*initex | package>
4320 <*package>
4321 \ProvidesExplPackage
4322   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
4323   \package_check_loaded_expl:
4324 </package>

```

A token list variable is a \TeX macro that holds tokens. By using the $\varepsilon\text{-TeX}$ primitive `\unexpanded` inside a \TeX `\edef` it is possible to store any tokens, including `#`, in this way.

192.1 Functions

`\tl_new:N` Creating new token list variables is a case of checking for an existing definition and if free doing the definition.

```

4325 \cs_new_protected:Npn \tl_new:N #1
4326 {
4327     \chk_if_free_cs:N #1
4328     \cs_gset_eq:NN #1 \c_empty_tl
4329 }
4330 \cs_generate_variant:Nn \tl_new:N { c }

```

(End definition for `\tl_new:N` and `\tl_new:c`. These functions are documented on page ??.)

`\tl_const:Nn` Constants are also easy to generate.

```

\tl_const:Nx 4331 \cs_new_protected:Npn \tl_const:Nn #1#2
\tl_const:cn 4332 {
\tl_const:cx 4333     \chk_if_free_cs:N #1
4334     \cs_gset_nopar:Npx #1 { \exp_not:n {#2} }
4335 }
4336 \cs_new_protected:Npn \tl_const:Nx #1#2
4337 {
4338     \chk_if_free_cs:N #1
4339     \cs_gset_nopar:Npx #1 {#2}
4340 }
4341 \cs_generate_variant:Nn \tl_const:Nn { c }
4342 \cs_generate_variant:Nn \tl_const:Nx { c }

```

(End definition for `\tl_const:Nn` and others. These functions are documented on page ??.)

`\c_empty_tl` Never full. We need to define that constant early for `\tl_new:N` to work properly.

```

4343 \tl_const:Nn \c_empty_tl { }
      (End definition for \c_empty_tl. This function is documented on page 93.)

```

`\tl_clear:N` Clearing a token list variable means setting it to an empty value. Error checking will be sorted out by the parent function.

`\tl_clear:c`

`\tl_gclear:N`

`\tl_gclear:c`

```

4344 \cs_new_protected:Npn \tl_clear:N #1
4345 { \tl_set_eq:NN #1 \c_empty_tl }
4346 \cs_new_protected:Npn \tl_gclear:N #1
4347 { \tl_gset_eq:NN #1 \c_empty_tl }
4348 \cs_generate_variant:Nn \tl_clear:N { c }
4349 \cs_generate_variant:Nn \tl_gclear:N { c }
      (End definition for \tl_clear:N and \tl_clear:c. These functions are documented on page ??.)

```

`\tl_clear_new:N` Clearing a token list variable means setting it to an empty value. Error checking will be sorted out by the parent function.

`\tl_clear_new:c`

`\tl_gclear_new:N`

`\tl_gclear_new:c`

```

4350 \cs_new_protected:Npn \tl_clear_new:N #1
4351 { \cs_if_exist:NTF #1 { \tl_clear:N #1 } { \tl_new:N #1 } }
4352 \cs_new_protected:Npn \tl_gclear_new:N #1
4353 { \cs_if_exist:NTF #1 { \tl_gclear:N #1 } { \tl_new:N #1 } }
4354 \cs_generate_variant:Nn \tl_clear_new:N { c }
4355 \cs_generate_variant:Nn \tl_gclear_new:N { c }
      (End definition for \tl_clear_new:N and \tl_clear_new:c. These functions are documented on page ??.)

```

`\tl_set_eq:NN` For setting token list variables equal to each other.

`\tl_set_eq:Nc`

`\tl_set_eq:cN`

`\tl_set_eq:Nc`

`\tl_set_eq:cc`

`\tl_gset_eq:NN`

`\tl_gset_eq:Nc`

`\tl_gset_eq:cN`

`\tl_gset_eq:Nc`

`\tl_gset_eq:cc`

```

4356 \cs_new_eq:NN \tl_set_eq:NN \cs_set_eq:NN
4357 \cs_new_eq:NN \tl_set_eq:cN \cs_set_eq:cN
4358 \cs_new_eq:NN \tl_set_eq:Nc \cs_set_eq:Nc
4359 \cs_new_eq:NN \tl_set_eq:cc \cs_set_eq:cc
4360 \cs_new_eq:NN \tl_gset_eq:NN \cs_gset_eq:NN
4361 \cs_new_eq:NN \tl_gset_eq:cN \cs_gset_eq:cN
4362 \cs_new_eq:NN \tl_gset_eq:Nc \cs_gset_eq:Nc
4363 \cs_new_eq:NN \tl_gset_eq:cc \cs_gset_eq:cc
      (End definition for \tl_set_eq:NN and others. These functions are documented on page ??.)

```

192.2 Adding to token list variables

`\tl_set:Nn` By using `\exp_not:n` token list variables can contain `#` tokens, which makes the token list registers provided by T_EX more or less redundant. The `\tl_set:No` version is done “by hand” as it is used quite a lot.

```

4364 \cs_new_protected:Npn \tl_set:Nn #1#2
4365 { \cs_set_nopar:Npx #1 { \exp_not:n {#2} } }
4366 \cs_new_protected:Npn \tl_set:No #1#2
4367 { \cs_set_nopar:Npx #1 { \exp_not:o {#2} } }
4368 \cs_new_protected:Npn \tl_set:Nx #1#2
4369 { \cs_set_nopar:Npx #1 {#2} }
4370 \cs_new_protected:Npn \tl_gset:Nn #1#2

```

`\tl_set:cf`

`\tl_set:cx`

`\tl_gset:Nn`

`\tl_gset:Nv`

`\tl_gset:Nv`

`\tl_gset:No`

`\tl_gset:Nf`

`\tl_gset:Nx`

`\tl_gset:cn`

`\tl_gset:cn`

```

4371 { \cs_gset_nopar:Npx #1 { \exp_not:n {#2} } }
4372 \cs_new_protected:Npn \tl_gset:No #1#2
4373 { \cs_gset_nopar:Npx #1 { \exp_not:o {#2} } }
4374 \cs_new_protected:Npn \tl_gset:Nx #1#2
4375 { \cs_gset_nopar:Npx #1 {#2} }
4376 \cs_generate_variant:Nn \tl_set:Nn { NV , Nv , Nf }
4377 \cs_generate_variant:Nn \tl_set:Nx { c }
4378 \cs_generate_variant:Nn \tl_set:Nn { c , co , cV , cv , cf }
4379 \cs_generate_variant:Nn \tl_gset:Nn { NV , Nv , Nf }
4380 \cs_generate_variant:Nn \tl_gset:Nx { c }
4381 \cs_generate_variant:Nn \tl_gset:Nn { c , co , cV , cv , cf }

```

(End definition for \tl_set:Nn and others. These functions are documented on page ??.)

\tl_put_left:Nn Adding to the left is done directly to gain a little performance.

```

\tl_put_left:NV
\tl_put_left:No
\tl_put_left:Nx
\tl_put_left:cn
\tl_put_left:cV
\tl_put_left:co
\tl_put_left:cx
\tl_gput_left:Nn
\tl_gput_left:NV
\tl_gput_left:No
\tl_gput_left:Nx
\tl_gput_left:cn
\tl_gput_left:cV
\tl_gput_left:co
\tl_gput_left:cx

```

```

4382 \cs_new_protected:Npn \tl_put_left:Nn #1#2
4383 { \cs_set_nopar:Npx #1 { \exp_not:n {#2} \exp_not:o #1 } }
4384 \cs_new_protected:Npn \tl_put_left:NV #1#2
4385 { \cs_set_nopar:Npx #1 { \exp_not:V #2 \exp_not:o #1 } }
4386 \cs_new_protected:Npn \tl_put_left:No #1#2
4387 { \cs_set_nopar:Npx #1 { \exp_not:o {#2} \exp_not:o #1 } }
4388 \cs_new_protected:Npn \tl_put_left:Nx #1#2
4389 { \cs_set_nopar:Npx #1 { #2 \exp_not:o #1 } }
4390 \cs_new_protected:Npn \tl_gput_left:Nn #1#2
4391 { \cs_gset_nopar:Npx #1 { \exp_not:n {#2} \exp_not:o #1 } }
4392 \cs_new_protected:Npn \tl_gput_left:NV #1#2
4393 { \cs_gset_nopar:Npx #1 { \exp_not:V #2 \exp_not:o #1 } }
4394 \cs_new_protected:Npn \tl_gput_left:No #1#2
4395 { \cs_gset_nopar:Npx #1 { \exp_not:o {#2} \exp_not:o #1 } }
4396 \cs_new_protected:Npn \tl_gput_left:Nx #1#2
4397 { \cs_gset_nopar:Npx #1 { #2 \exp_not:o {#1} } }
4398 \cs_generate_variant:Nn \tl_put_left:Nn { c }
4399 \cs_generate_variant:Nn \tl_put_left:NV { c }
4400 \cs_generate_variant:Nn \tl_put_left:No { c }
4401 \cs_generate_variant:Nn \tl_put_left:Nx { c }
4402 \cs_generate_variant:Nn \tl_gput_left:Nn { c }
4403 \cs_generate_variant:Nn \tl_gput_left:NV { c }
4404 \cs_generate_variant:Nn \tl_gput_left:No { c }
4405 \cs_generate_variant:Nn \tl_gput_left:Nx { c }

```

(End definition for \tl_put_left:Nn and others. These functions are documented on page ??.)

\tl_put_right:Nn The same on the right.

```

\tl_put_right:NV
\tl_put_right:No
\tl_put_right:Nx
\tl_put_right:cn
\tl_put_right:cV
\tl_put_right:co
\tl_put_right:cx
\tl_gput_right:Nn
\tl_gput_right:NV
\tl_gput_right:No
\tl_gput_right:Nx
\tl_gput_right:cn
\tl_gput_right:cV
\tl_gput_right:co
\tl_gput_right:cx

```

```

4406 \cs_new_protected:Npn \tl_put_right:Nn #1#2
4407 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:n {#2} } }
4408 \cs_new_protected:Npn \tl_put_right:NV #1#2
4409 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:V #2 } }
4410 \cs_new_protected:Npn \tl_put_right:No #1#2
4411 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:o {#2} } }
4412 \cs_new_protected:Npn \tl_put_right:Nx #1#2
4413 { \cs_set_nopar:Npx #1 { \exp_not:o #1 #2 } }

```

```

4414 \cs_new_protected:Npn \tl_gput_right:Nn #1#2
4415 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:n {#2} } }
4416 \cs_new_protected:Npn \tl_gput_right:NV #1#2
4417 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:V #2 } }
4418 \cs_new_protected:Npn \tl_gput_right:No #1#2
4419 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:o {#2} } }
4420 \cs_new_protected:Npn \tl_gput_right:Nx #1#2
4421 { \cs_gset_nopar:Npx #1 { \exp_not:o {#1} #2 } }
4422 \cs_generate_variant:Nn \tl_put_right:Nn { c }
4423 \cs_generate_variant:Nn \tl_put_right:NV { c }
4424 \cs_generate_variant:Nn \tl_put_right:No { c }
4425 \cs_generate_variant:Nn \tl_put_right:Nx { c }
4426 \cs_generate_variant:Nn \tl_gput_right:Nn { c }
4427 \cs_generate_variant:Nn \tl_gput_right:NV { c }
4428 \cs_generate_variant:Nn \tl_gput_right:No { c }
4429 \cs_generate_variant:Nn \tl_gput_right:Nx { c }

```

(End definition for `\tl_put_right:Nn` and others. These functions are documented on page ??.)

192.3 Reassigning token list category codes

`\c_tl_rescan_marker_tl` The rescanning code needs a special token list containing the same character with two different category codes. This is set up here, while the detail is described below.

```

4430 \group_begin:
4431 \tex_lccode:D '\A = '\@ \scan_stop:
4432 \tex_lccode:D '\B = '\@ \scan_stop:
4433 \tex_catcode:D '\A = 8 \scan_stop:
4434 \tex_catcode:D '\B = 3 \scan_stop:
4435 \tex_lowercase:D
4436 {
4437   \group_end:
4438   \tl_const:Nn \c_tl_rescan_marker_tl { A B }
4439 }

```

(End definition for `\c_tl_rescan_marker_tl`. This function is documented on page ??.)

`\tl_set_rescan:Nnn` The idea here is to deal cleanly with the problem that `\scantokens` treats the argument as a file, and without the correct settings a T_EX error occurs:

`\tl_set_rescan:Nno` ! File ended while scanning definition of ...

`\tl_set_rescan:Nnx` When expanding a token list this can be handled using `\exp_not:N` but this fails if the token list is not being expanded. So instead a delimited argument is used with an end marker which cannot appear within the token list which is scanned: two @ symbols with different category codes. The rescanned token list cannot contain the end marker, because all @ present in the token list are read with the same category code. As every character with charcode `\newlinechar` is replaced by the `\endlinechar`, and an extra `\endlinechar` is added at the end, we need to set both of those to -1, “unprintable”.

```

4440 \cs_new_protected_nopar:Npn \tl_set_rescan:Nnn
4441 { \tl_set_rescan_aux:NNnn \tl_set:Nn }
4442 \cs_new_protected_nopar:Npn \tl_gset_rescan:Nnn

```

`\tl_set_rescan:cno`

`\tl_set_rescan:cnx`

`\tl_gset_rescan:Nnn`

`\tl_gset_rescan:Nno`

`\tl_gset_rescan:Nnx`

`\tl_gset_rescan:cnn`

`\tl_gset_rescan:cno`

`\tl_gset_rescan:cnx`

`\tl_rescan:nn`

`\tl_set_rescan_aux:NNnn`

`\tl_rescan_aux:w`


```

4443 { \tl_set_rescan_aux:NNnn \tl_gset:Nn }
4444 \cs_new_protected_nopar:Npn \tl_rescan:nn
4445 { \tl_set_rescan_aux:NNnn \prg_do_nothing: \use:n }
4446 \cs_new_protected:Npn \tl_set_rescan_aux:NNnn #1#2#3#4
4447 {
4448   \group_begin:
4449   \exp_args:No \etex_everyeof:D { \c_tl_rescan_marker_tl \exp_not:N }
4450   \tex_endlinechar:D \c_minus_one
4451   \tex_newlinechar:D \c_minus_one
4452   #3
4453   \use:x
4454   {
4455     \group_end:
4456     #1 \exp_not:N #2
4457     {
4458       \exp_after:wN \tl_rescan_aux:w
4459       \exp_after:wN \prg_do_nothing:
4460       \etex_scantokens:D {#4}
4461     }
4462   }
4463 }
4464 \use:x
4465 {
4466   \cs_new:Npn \exp_not:N \tl_rescan_aux:w ##1
4467   \c_tl_rescan_marker_tl
4468   { \exp_not:N \exp_not:o { ##1 } }
4469 }
4470 \cs_generate_variant:Nn \tl_set_rescan:Nnn { Nno , Nnx }
4471 \cs_generate_variant:Nn \tl_set_rescan:Nnn { c , cno , cnx }
4472 \cs_generate_variant:Nn \tl_gset_rescan:Nnn { Nno , Nnx }
4473 \cs_generate_variant:Nn \tl_gset_rescan:Nnn { c , cno }

```

(End definition for `\tl_set_rescan:Nnn` and others. These functions are documented on page ??.)

192.4 Reassigning token list character codes

`\tl_to_lowercase:n` Just some names for a few primitives.
`\tl_to_uppercase:n`

```

4474 \cs_new_eq:NN \tl_to_lowercase:n \tex_lowercase:D
4475 \cs_new_eq:NN \tl_to_uppercase:n \tex_uppercase:D

```

(End definition for `\tl_to_lowercase:n`. This function is documented on page 85.)

192.5 Modifying token list variables

`\l_tl_replace_tl` A scratch variable for doing token replacement.

```

4476 \tl_new:N \l_tl_replace_tl

```

(End definition for `\l_tl_replace_tl`. This function is documented on page ??.)

`\tl_replace_all:Nnn` All of the replace functions are based on `\tl_replace_aux:NNNnn`, whose arguments are:
`\tl_replace_all:cnn` $\langle function \rangle$, $\langle tl _ (g) set:Nx \rangle$, $\langle tl _ var \rangle$, $\langle search \ tokens \rangle$, $\langle replacement \ tokens \rangle$.
`\tl_greplace_all:Nnn`
`\tl_greplace_all:cnn`
`\tl_replace_once:Nnn`
`\tl_replace_once:cnn`
`\tl_greplace_once:Nnn`
`\tl_greplace_once:cnn`
`\tl_replace_aux:NNNnn`
`\tl_replace_aux_ii:w`
`\tl_replace_all_aux:`
`\tl_replace_once_aux:`

```

4477 \cs_new_protected_nopar:Npn \tl_replace_once:Nnn
4478 { \tl_replace_aux:NNNnn \tl_replace_once_aux: \tl_set:Nx }
4479 \cs_new_protected_nopar:Npn \tl_greplace_once:Nnn
4480 { \tl_replace_aux:NNNnn \tl_replace_once_aux: \tl_gset:Nx }
4481 \cs_new_protected_nopar:Npn \tl_replace_all:Nnn
4482 { \tl_replace_aux:NNNnn \tl_replace_all_aux: \tl_set:Nx }
4483 \cs_new_protected_nopar:Npn \tl_greplace_all:Nnn
4484 { \tl_replace_aux:NNNnn \tl_replace_all_aux: \tl_gset:Nx }
4485 \cs_generate_variant:Nn \tl_replace_once:Nnn { c }
4486 \cs_generate_variant:Nn \tl_greplace_once:Nnn { c }
4487 \cs_generate_variant:Nn \tl_replace_all:Nnn { c }
4488 \cs_generate_variant:Nn \tl_greplace_all:Nnn { c }

```

The idea is easier to understand by considering the case of `\tl_replace_all:Nnn`. The replacement happens within an x-type expansion. We use an auxiliary function `\tl_tmp:w`, which essentially replaces the next *<search tokens>* by *<replacement tokens>*. To avoid runaway arguments, we expand something like `\tl_tmp:w <token list> \q_mark <search tokens> \q_stop`, repeating until the end. How do we detect that we have reached the last occurrence of *<search tokens>*? The last replacement is characterized by the fact that the argument of `\tl_tmp:w` contains `\q_mark`. In the code below, `\tl_replace_aux_ii:w` takes an argument delimited by `\q_mark`, and removes the following token. Before we reach the end, this gobbles `\q_mark \use_none_delimit_by_q_stop:w` which appear in the definition of `\tl_tmp:w`, and leaves the *<replacement tokens>*, passed to `\exp_not:n`, to be included in the x-expanding definition. At the end, the first `\q_mark` is within the argument of `\tl_tmp:w`, and `\tl_replace_aux_ii:w` gobbles the second `\q_mark` as well, leaving `\use_none_delimit_by_q_stop:w`, which ends the recursion cleanly.

```

4489 \cs_new_protected:Npn \tl_replace_aux:NNNnn #1#2#3#4#5
4490 {
4491   \tl_if_empty:nTF {#4}
4492   {
4493     \msg_kernel_error:nxx { tl } { empty-search-pattern }
4494     { \tl_to_str:n {#5} }
4495   }
4496   {
4497     \group_align_safe_begin:
4498     \cs_set:Npx \tl_tmp:w ##1##2 #4
4499     {
4500       ##2
4501       \exp_not:N \q_mark
4502       \exp_not:N \use_none_delimit_by_q_stop:w
4503       \exp_not:n { \exp_not:n {#5} }
4504       ##1
4505     }
4506     \group_align_safe_end:
4507     #2 #3
4508     {
4509       \exp_after:wN #1
4510       #3 \q_mark #4 \q_stop

```

```

4511     }
4512   }
4513 }
4514 \cs_new:Npn \tl_replace_aux_ii:w #1 \q_mark #2 { \exp_not:o {#1} }

```

The first argument of `\tl_tmp:w` is responsible for repeating the replacement in the case of `replace_all`, and stopping it early for `replace_once`. Note also that we build `\tl_tmp:w` within an `x`-expansion so that the *replacement tokens* can contain `#`. The second `\exp_not:n` ensures that the *replacement tokens* are not expanded by `\tl_(g)set:Nx`.

Now on to the difference between “once” and “all”. The `\prg_do_nothing:` and accompanying `o`-expansion ensure that we don’t lose braces in case the tokens between two occurrences of the *search tokens* form a brace group.

```

4515 \cs_new:Npn \tl_replace_all_aux:
4516 {
4517   \exp_after:wN \tl_replace_aux_ii:w
4518   \tl_tmp:w \tl_replace_all_aux: \prg_do_nothing:
4519 }
4520 \cs_new_nopar:Npn \tl_replace_once_aux:
4521 {
4522   \exp_after:wN \tl_replace_aux_ii:w
4523   \tl_tmp:w { \tl_replace_once_aux_end:w \prg_do_nothing: } \prg_do_nothing:
4524 }
4525 \cs_new:Npn \tl_replace_once_aux_end:w #1 \q_mark #2 \q_stop
4526 { \exp_not:o {#1} }

```

(End definition for `\tl_replace_all:Nnn` and `\tl_replace_all:cnn`. These functions are documented on page ??.)

```

\tl_remove_once:Nn Removal is just a special case of replacement.
\tl_remove_once:cn 4527 \cs_new_protected:Npn \tl_remove_once:Nn #1#2
\tl_gremove_once:Nn 4528 { \tl_replace_once:Nnn #1 {#2} { } }
\tl_gremove_once:cn 4529 \cs_new_protected:Npn \tl_gremove_once:Nn #1#2
4530 { \tl_greplace_once:Nnn #1 {#2} { } }
4531 \cs_generate_variant:Nn \tl_remove_once:Nn { c }
4532 \cs_generate_variant:Nn \tl_gremove_once:Nn { c }

```

(End definition for `\tl_remove_once:Nn` and `\tl_remove_once:cn`. These functions are documented on page ??.)

```

\tl_remove_all:Nn Removal is just a special case of replacement.
\tl_remove_all:cn 4533 \cs_new_protected:Npn \tl_remove_all:Nn #1#2
\tl_gremove_all:Nn 4534 { \tl_replace_all:Nnn #1 {#2} { } }
\tl_gremove_all:cn 4535 \cs_new_protected:Npn \tl_gremove_all:Nn #1#2
4536 { \tl_greplace_all:Nnn #1 {#2} { } }
4537 \cs_generate_variant:Nn \tl_remove_all:Nn { c }
4538 \cs_generate_variant:Nn \tl_gremove_all:Nn { c }

```

192.6 Token list conditionals

`\tl_if_blank:n` TeX skips spaces when reading a non-delimited arguments. Thus, a *token list* is blank if and only if `\use_none:n <token list> ?` is empty. For performance reasons, we hard-code the emptiness test done in `\tl_if_empty:n(TF)`: convert to harmless characters with `\tl_to_str:n`, and then use `\if_meaning:w \q_nil ... \q_nil`. Note that converting to a string is done after reading the delimited argument for `\use_none:n`. The similar construction `\exp_after:wN \use_none:n \tl_to_str:n {<token list>} ?` would fail if the token list contains the control sequence `\`, while `\escapechar` is a space or is unprintable.

```

4539 \prg_new_conditional:Npnn \tl_if_blank:n #1 { p , T , F , TF }
4540 { \tl_if_empty_return:o { \use_none:n #1 ? } }
4541 \cs_generate_variant:Nn \tl_if_blank_p:n { V }
4542 \cs_generate_variant:Nn \tl_if_blank:nT { V }
4543 \cs_generate_variant:Nn \tl_if_blank:nF { V }
4544 \cs_generate_variant:Nn \tl_if_blank:nTF { V }
4545 \cs_generate_variant:Nn \tl_if_blank_p:n { o }
4546 \cs_generate_variant:Nn \tl_if_blank:nT { o }
4547 \cs_generate_variant:Nn \tl_if_blank:nF { o }
4548 \cs_generate_variant:Nn \tl_if_blank:nTF { o }
      (End definition for \tl_remove_all:Nn and \tl_remove_all:cn. These functions are documented
      on page ??.)

```

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

`\tl_if_empty:c`

```

4549 \prg_set_conditional:Npnn \tl_if_empty:N #1 { p , T , F , TF }
4550 {
4551   \if_meaning:w #1 \c_empty_tl
4552   \prg_return_true:
4553   \else:
4554     \prg_return_false:
4555   \fi:
4556 }
4557 \cs_generate_variant:Nn \tl_if_empty_p:N { c }
4558 \cs_generate_variant:Nn \tl_if_empty:NT { c }
4559 \cs_generate_variant:Nn \tl_if_empty:NF { c }
4560 \cs_generate_variant:Nn \tl_if_empty:NTF { c }
      (End definition for \tl_if_empty:N and \tl_if_empty:c. These functions are documented on
      page ??.)

```

`\tl_if_empty:n` It would be tempting to just use `\if_meaning:w \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:w`, 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.

```

4561 \prg_new_conditional:Npnn \tl_if_empty:n #1 { p , TF , T , F }
4562 {
4563   \exp_after:wN \if_meaning:w \exp_after:wN \q_nil \tl_to_str:n {#1} \q_nil
4564   \prg_return_true:
4565   \else:
4566     \prg_return_false:
4567   \fi:
4568 }
4569 \cs_generate_variant:Nn \tl_if_empty_p:n { V }
4570 \cs_generate_variant:Nn \tl_if_empty:nTF { V }
4571 \cs_generate_variant:Nn \tl_if_empty:nT { V }
4572 \cs_generate_variant:Nn \tl_if_empty:nF { V }

```

(End definition for `\tl_if_empty:n` and `\tl_if_empty:V`. These functions are documented on page ??.)

`\tl_if_empty:o` The auxiliary function `\tl_if_empty_return:o` is for use in conditionals on token lists, which mostly reduce to testing if a given token list is empty after applying a simple function to it. The test for emptiness is based on `\tl_if_empty:n(TF)`, but the expansion is hard-coded for efficiency, as this auxiliary function is used in many places. Note that this works because `\tl_to_str:n` expands tokens that follow until reading a catcode 1 (begin-group) token.

`\tl_if_empty_return:o`

```

4573 \cs_new:Npn \tl_if_empty_return:o #1
4574 {
4575   \exp_after:wN \if_meaning:w \exp_after:wN \q_nil
4576   \tl_to_str:n \exp_after:wN {#1} \q_nil
4577   \prg_return_true:
4578   \else:
4579     \prg_return_false:
4580   \fi:
4581 }
4582 \prg_new_conditional:Npnn \tl_if_empty:o #1 { p , TF , T , F }
4583 { \tl_if_empty_return:o {#1} }

```

(End definition for `\tl_if_empty:o`. This function is documented on page ??.)

`\tl_if_eq:NN` Returns `\c_true_bool` if and only if the two token list variables are equal.

```

4584 \prg_new_conditional:Npnn \tl_if_eq:NN #1#2 { p , T , F , TF }
4585 {
4586   \if_meaning:w #1 #2
4587   \prg_return_true:
4588   \else:
4589     \prg_return_false:
4590   \fi:
4591 }
4592 \cs_generate_variant:Nn \tl_if_eq_p:NN { Nc , c , cc }
4593 \cs_generate_variant:Nn \tl_if_eq:NNTF { Nc , c , cc }
4594 \cs_generate_variant:Nn \tl_if_eq:NNT { Nc , c , cc }
4595 \cs_generate_variant:Nn \tl_if_eq:NNF { Nc , c , cc }

```

(End definition for `\tl_if_eq:NN` and others. These functions are documented on page ??.)

`\tl_if_eq:nn` A simple store and compare routine.

`\l_tl_tmpa_tl` 4596 `\prg_new_protected_conditional:Npnn \tl_if_eq:nn #1#2 { T , F , TF }`

`\l_tl_tmppb_tl` 4597 `{`

4598 `\group_begin:`

4599 `\tl_set:Nn \l_tl_tmpa_tl {#1}`

4600 `\tl_set:Nn \l_tl_tmppb_tl {#2}`

4601 `\if_meaning:w \l_tl_tmpa_tl \l_tl_tmppb_tl`

4602 `\group_end:`

4603 `\prg_return_true:`

4604 `\else:`

4605 `\group_end:`

4606 `\prg_return_false:`

4607 `\fi:`

4608 `}`

4609 `\tl_new:N \l_tl_tmpa_tl`

4610 `\tl_new:N \l_tl_tmppb_tl`

(End definition for \tl_if_eq:nn. This function is documented on page ??.)

`\tl_if_in:Nn` See `\tl_if_in:nn(TF)` for further comments. Here we simply expand the token list

`\tl_if_in:cn` variable and pass it to `\tl_if_in:nn(TF)`.

4611 `\cs_new_protected_nopar:Npn \tl_if_in:NnT { \exp_args:No \tl_if_in:nnT }`

4612 `\cs_new_protected_nopar:Npn \tl_if_in:NnF { \exp_args:No \tl_if_in:nnF }`

4613 `\cs_new_protected_nopar:Npn \tl_if_in:NnTF { \exp_args:No \tl_if_in:nnTF }`

4614 `\cs_generate_variant:Nn \tl_if_in:NnT { c }`

4615 `\cs_generate_variant:Nn \tl_if_in:NnF { c }`

4616 `\cs_generate_variant:Nn \tl_if_in:NnTF { c }`

(End definition for \tl_if_in:Nn and \tl_if_in:cn. These functions are documented on page ??.)

`\tl_if_in:nn` Once more, the test relies on `\tl_to_str:n` for robustness. The function `\tl_tmp:w`

`\tl_if_in:Vn` removes tokens until the first occurrence of #2. If this does not appear in #1, then the

`\tl_if_in:on` final #2 is removed, leaving an empty token list. Otherwise some tokens remain, and the

`\tl_if_in:no` test is false. See `\tl_if_empty:n(TF)` for details on the emptiness test.

Special care is needed to treat correctly cases like `\tl_if_in:nnTF {a state}{states}`, where #1#2 contains #2 before the end. To cater for this case, we insert `{ }{ }` between the two token lists. This marker may not appear in #2 because of TeX limitations on what can delimit a parameter, hence we are safe. Using two brace groups makes the test work also for empty arguments.

4617 `\prg_new_protected_conditional:Npnn \tl_if_in:nn #1#2 { T , F , TF }`

4618 `{`

4619 `\cs_set:Npn \tl_tmp:w ##1 #2 { }`

4620 `\tl_if_empty:oTF { \tl_tmp:w #1 {} {} } #2 }`

4621 `{ \prg_return_false: } { \prg_return_true: }`

4622 `}`

4623 `\cs_generate_variant:Nn \tl_if_in:nnT { V , o , no }`

4624 `\cs_generate_variant:Nn \tl_if_in:nnF { V , o , no }`

4625 `\cs_generate_variant:Nn \tl_if_in:nnTF { V , o , no }`

(End definition for \tl_if_in:nn and others. These functions are documented on page ??.)

192.7 Mapping to token lists

`\tl_map_function:nN` Expandable loop macro for token lists. These have the advantage of not needing to test if the argument is empty, because if it is, the stop marker will be read immediately and the loop terminated.

```
\tl_map_function:cN
\tl_map_function_aux:Nn
4626 \cs_new:Npn \tl_map_function:nN #1#2
4627 {
4628   \tl_map_function_aux:Nn #2 #1
4629   \q_recursion_tail
4630   \prg_break_point:n { }
4631 }
4632 \cs_new_nopar:Npn \tl_map_function:NN
4633 { \exp_args:No \tl_map_function:nN }
4634 \cs_new:Npn \tl_map_function_aux:Nn #1#2
4635 {
4636   \quark_if_recursion_tail_break:n {#2}
4637   #1 {#2} \tl_map_function_aux:Nn #1
4638 }
4639 \cs_generate_variant:Nn \tl_map_function:NN { c }
(End definition for \tl_map_function:nN. This function is documented on page ??.)
```

`\tl_map_inline:nn` The inline functions are straight forward by now. We use a little trick with the counter `\g_prg_map_int` to make them nestable. We can also make use of `\tl_map_function_aux:Nn` from before.

```
4640 \cs_new_protected:Npn \tl_map_inline:nn #1#2
4641 {
4642   \int_gincr:N \g_prg_map_int
4643   \cs_gset:cpn { tl_map_inline_ \int_use:N \g_prg_map_int :n }
4644   ##1 {#2}
4645   \exp_args:Nc \tl_map_function_aux:Nn
4646   { tl_map_inline_ \int_use:N \g_prg_map_int :n }
4647   #1 \q_recursion_tail
4648   \prg_break_point:n { \int_gdecr:N \g_prg_map_int }
4649 }
4650 \cs_new_protected:Npn \tl_map_inline:Nn
4651 { \exp_args:No \tl_map_inline:nn }
4652 \cs_generate_variant:Nn \tl_map_inline:Nn { c }
(End definition for \tl_map_inline:nn. This function is documented on page ??.)
```

`\tl_map_variable:nNn` `\tl_map_variable:nNn` $\langle token\ list \rangle$ $\langle temp \rangle$ $\langle action \rangle$ assigns $\langle temp \rangle$ to each element and executes $\langle action \rangle$.

```
\tl_map_variable:NNn
\tl_map_variable:cNn
\tl_map_variable_aux:Nnn
4653 \cs_new_protected:Npn \tl_map_variable:nNn #1#2#3
4654 {
4655   \tl_map_variable_aux:Nnn #2 {#3} #1
4656   \q_recursion_tail
4657   \prg_break_point:n { }
4658 }
4659 \cs_new_protected_nopar:Npn \tl_map_variable:NNn
4660 { \exp_args:No \tl_map_variable:nNn }
```

```

4661 \cs_new_protected:Npn \tl_map_variable_aux:Nnn #1#2#3
4662 {
4663   \tl_set:Nn #1 {#3}
4664   \quark_if_recursion_tail_break:N #1
4665   \use:n {#2}
4666   \tl_map_variable_aux:Nnn #1 {#2}
4667 }
4668 \cs_generate_variant:Nn \tl_map_variable:NNn { c }

```

(End definition for \tl_map_variable:nNn. This function is documented on page ??.)

`\tl_map_break:` The break statements are simply copies.

`\tl_map_break:n`

```

4669 \cs_new_eq:NN \tl_map_break: \prg_map_break:
4670 \cs_new_eq:NN \tl_map_break:n \prg_map_break:n

```

(End definition for \tl_map_break:. This function is documented on page ??.)

192.8 Using token lists

`\tl_to_str:n` Another name for a primitive.

```

4671 \cs_new_eq:NN \tl_to_str:n \etex_detokenize:D

```

(End definition for \tl_to_str:n. This function is documented on page 88.)

`\tl_to_str:N` These functions return the replacement text of a token list as a string.

`\tl_to_str:c`

```

4672 \cs_new:Npn \tl_to_str:N #1 { \etex_detokenize:D \exp_after:wN {#1} }
4673 \cs_generate_variant:Nn \tl_to_str:N { c }

```

(End definition for \tl_to_str:N and \tl_to_str:c. These functions are documented on page ??.)

`\tl_use:N` Token lists which are simply not defined will give a clear T_EX error here. No such luck

`\tl_use:c` for ones equal to `\scan_stop:` so instead a test is made and if there is an issue an error is forced.

```

4674 \cs_new:Npn \tl_use:N #1
4675 {
4676   \cs_if_exist:NTF #1 {#1}
4677   { \msg_expandable_kernel_error:nnn { kernel } { bad-var } {#1} }
4678 }
4679 \cs_generate_variant:Nn \tl_use:N { c }

```

(End definition for \tl_use:N and \tl_use:c. These functions are documented on page ??.)

192.9 Working with the contents of token lists

`\tl_length:n` Count number of elements within a token list or token list variable. Brace groups within

`\tl_length:V` the list are read as a single element. Spaces are ignored. `\tl_length_aux:n` grabs the

`\tl_length:o` element and replaces it by +1. The 0 to ensure it works on an empty list.

`\tl_length:N`

```

4680 \cs_new:Npn \tl_length:n #1

```

`\tl_length:c`

```

4681 {

```

`\tl_length_aux:n`

```

4682   \int_eval:n
4683   { 0 \tl_map_function:nN {#1} \tl_length_aux:n }

```



```

4684 }
4685 \cs_new:Npn \tl_length:N #1
4686 {
4687   \int_eval:n
4688     { 0 \tl_map_function:NN #1 \tl_length_aux:n }
4689 }
4690 \cs_new:Npn \tl_length_aux:n #1 { + \c_one }
4691 \cs_generate_variant:Nn \tl_length:n { V , o }
4692 \cs_generate_variant:Nn \tl_length:N { c }

```

(End definition for `\tl_length:n`, `\tl_length:V`, and `\tl_length:o`. These functions are documented on page ??.)

`\tl_reverse_items:n` Reversal of a token list is done by taking one item at a time and putting it after `\q_-`
`\tl_reverse_items_aux:nwNwn` recursion_stop.
`\tl_reverse_items_aux:wn`

```

4693 \cs_new:Npn \tl_reverse_items:n #1
4694 {
4695   \tl_reverse_items_aux:nwNwn #1 ?
4696   \q_mark \tl_reverse_items_aux:nwNwn
4697   \q_mark \tl_reverse_items_aux:wn
4698   \q_stop { }
4699 }
4700 \cs_new:Npn \tl_reverse_items_aux:nwNwn #1 #2 \q_mark #3 #4 \q_stop #5
4701 {
4702   #3 #2
4703   \q_mark \tl_reverse_items_aux:nwNwn
4704   \q_mark \tl_reverse_items_aux:wn
4705   \q_stop { {#1} #5 }
4706 }
4707 \cs_new:Npn \tl_reverse_items_aux:wn #1 \q_stop #2 { \use_none:nn #2 }

```

(End definition for `\tl_reverse_items:n`. This function is documented on page ??.)

`\tl_trim_spaces:n` Trimming spaces from around the input is done using delimited arguments and quarks,
`\tl_trim_spaces:N` and to get spaces at odd places in the definitions, we nest those in `\tl_tmp:w`, which
`\tl_trim_spaces:c` then receives a single space as its argument: `#1` is `␣`. Removing leading spaces is done
`\tl_gtrim_spaces:N` with `\tl_trim_spaces_aux_i:w`, which loops until `\q_mark␣` matches the end of the
`\tl_gtrim_spaces:c` token list: then `##1` is the token list and `##3` is `\tl_trim_spaces_aux_ii:w`. This hands
`\tl_trim_spaces_aux_i:w` the relevant tokens to the loop `\tl_trim_spaces_aux_iii:w`, responsible for trimming
`\tl_trim_spaces_aux_ii:w` trailing spaces. The end is reached when `␣ \q_nil` matches the one present in the
`\tl_trim_spaces_aux_iii:w` definition of `\tl_trim_spaces:n`. Then `\tl_trim_spaces_aux_iv:w` puts the token list
`\tl_trim_spaces_aux_iv:w` into a group, as the argument of the initial `\unexpanded`. The `\unexpanded` here is used
so that space trimming will behave correctly within an x-type expansion.

Some of the auxiliaries used in this code are also used in the `l3clist` module. Change with care.

```

4708 \cs_set:Npn \tl_tmp:w #1
4709 {
4710   \cs_new:Npn \tl_trim_spaces:n ##1
4711   {
4712     \etex_unexpanded:D

```

```

4713         \tl_trim_spaces_aux_i:w
4714         \q_mark
4715         ##1
4716         \q_nil
4717         \q_mark #1 { }
4718         \q_mark \tl_trim_spaces_aux_ii:w
4719         \tl_trim_spaces_aux_iii:w
4720         #1 \q_nil
4721         \tl_trim_spaces_aux_iv:w
4722         \q_stop
4723     }
4724 \cs_new:Npn \tl_trim_spaces_aux_i:w ##1 \q_mark #1 ##2 \q_mark ##3
4725 {
4726     ##3
4727     \tl_trim_spaces_aux_i:w
4728     \q_mark
4729     ##2
4730     \q_mark #1 {##1}
4731 }
4732 \cs_new:Npn \tl_trim_spaces_aux_ii:w ##1 \q_mark \q_mark ##2
4733 {
4734     \tl_trim_spaces_aux_iii:w
4735     ##2
4736 }
4737 \cs_new:Npn \tl_trim_spaces_aux_iii:w ##1 #1 \q_nil ##2
4738 {
4739     ##2
4740     ##1 \q_nil
4741     \tl_trim_spaces_aux_iii:w
4742 }
4743 \cs_new:Npn \tl_trim_spaces_aux_iv:w ##1 \q_nil ##2 \q_stop
4744 { \exp_after:wN { \use_none:n ##1 } }
4745 }
4746 \tl_tmp:w { ~ }
4747 \cs_new_protected:Npn \tl_trim_spaces:N #1
4748 { \tl_set:Nx #1 { \exp_after:wN \tl_trim_spaces:n \exp_after:wN {#1} } }
4749 \cs_new_protected:Npn \tl_gtrim_spaces:N #1
4750 { \tl_gset:Nx #1 { \exp_after:wN \tl_trim_spaces:n \exp_after:wN {#1} } }
4751 \cs_generate_variant:Nn \tl_trim_spaces:N { c }
4752 \cs_generate_variant:Nn \tl_gtrim_spaces:N { c }

```

(End definition for `\tl_trim_spaces:n`. This function is documented on page ??.)

192.10 The first token from a token list

`\tl_head:n` These functions pick up either the head or the tail of a list. The empty brace groups in `\tl_head:n` and `\tl_tail:n` ensure that a blank argument gives an empty result.

```

\tl_head:V
\tl_head:v 4753 \cs_new:Npn \tl_head:w #1#2 \q_stop {#1}
\tl_head:f 4754 \cs_new:Npn \tl_tail:w #1#2 \q_stop {#2}
\tl_head:w 4755 \cs_new:Npn \tl_head:n #1
\tl_tail:n
\tl_tail:V
\tl_tail:v
\tl_tail:f
\tl_tail:w

```

```

4756 { \tl_head:w #1 { } \q_stop }
4757 \cs_new:Npn \tl_tail:n #1
4758 { \tl_tail_aux:w #1 \q_mark { } \q_mark \q_stop }
4759 \cs_new:Npn \tl_tail_aux:w #1 #2 \q_mark #3 \q_stop { #2 }
4760 \cs_generate_variant:Nn \tl_head:n { V , v , f }
4761 \cs_generate_variant:Nn \tl_tail:n { V , v , f }

```

(End definition for `\tl_head:n` and others. These functions are documented on page 90.)

`\str_head:n` After `\tl_to_str:n`, we have a list of character tokens, all with category code 12, except
`\str_tail:n` the space, which has category code 10. Directly using `\tl_head:w` would thus lose leading
`\str_head_aux:w` spaces. Instead, we take an argument delimited by an explicit space, and then only use
`\str_tail_aux:w` `\tl_head:w`. If the string started with a space, then the argument of `\str_head_aux:w`
is empty, and the function correctly returns a space character. Otherwise, it returns the
first token of #1, which is the first token of the string. If the string is empty, we return
an empty result.

To remove the first character of `\tl_to_str:n {#1}`, we test it using `\if_charcode:w \scan_stop:`, always false for characters. If the argument was non-empty, then `\str_tail_aux:w` returns everything until the first X (with category code letter, no risk of confusing with the user input). If the argument was empty, the first X is taken by `\if_charcode:w`, and nothing is returned. We use X as a *marker*, rather than a quark because the test `\if_charcode:w \scan_stop: <marker>` has to be false.

```

4762 \cs_new:Npn \str_head:n #1
4763 {
4764   \exp_after:wN \str_head_aux:w
4765   \tl_to_str:n {#1}
4766   { { } } ~ \q_stop
4767 }
4768 \cs_new:Npn \str_head_aux:w #1 ~ %
4769 { \tl_head:w #1 { ~ } }
4770 \cs_new:Npn \str_tail:n #1
4771 {
4772   \exp_after:wN \str_tail_aux:w
4773   \reverse_if:N \if_charcode:w
4774   \scan_stop: \tl_to_str:n {#1} X X \q_stop
4775 }
4776 \cs_new:Npn \str_tail_aux:w #1 X #2 \q_stop { \fi: #1 }

```

(End definition for `\str_head:n` and `\str_tail:n`. These functions are documented on page ??.)

`\tl_if_head_eq_meaning:nN` Accessing the first token of a token list is tricky in two cases: when it has category code 1
`\tl_if_head_eq_charcode:nN` (begin-group token), or when it is an explicit space, with category code 10 and character
`\tl_if_head_eq_charcode:fN` code 32.
`\tl_if_head_eq_catcode:nN`

Forgetting temporarily about this issue we would use the following test in `\tl_if_head_eq_charcode:nN`. Here, an empty #1 argument yields `\q_nil`, otherwise the first token of the token list.

```

\tl_if_charcode:w
  \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop
  \exp_not:N #2

```

The special cases are detected using `\tl_if_head_N_type:n` (the extra ? takes care of empty arguments). In those cases, the first token is a character, and since we only care about its character code, we can use `\str_head:n` to access it (this works even if it is a space character).

```

4777 \prg_new_conditional:Npnn \tl_if_head_eq_charcode:nN #1#2 { p , T , F , TF }
4778 {
4779   \if_charcode:w
4780     \exp_not:N #2
4781     \tl_if_head_N_type:nTF { #1 ? }
4782     { \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop }
4783     { \str_head:n {#1} }
4784     \prg_return_true:
4785   \else:
4786     \prg_return_false:
4787   \fi:
4788 }
4789 \cs_generate_variant:Nn \tl_if_head_eq_charcode_p:nN { f }
4790 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNTF { f }
4791 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNT { f }
4792 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNF { f }

```

For `\tl_if_head_eq_catcode:nN`, again we detect special cases with a `\tl_if_head_N_type`. Then we need to test if the first token is a begin-group token or an explicit space token, and produce the relevant token, either `\c_group_begin_token` or `\c_space_token`.

```

4793 \prg_new_conditional:Npnn \tl_if_head_eq_catcode:nN #1 #2 { p , T , F , TF }
4794 {
4795   \if_catcode:w
4796     \exp_not:N #2
4797     \tl_if_head_N_type:nTF { #1 ? }
4798     { \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop }
4799     {
4800       \tl_if_head_group:nTF {#1}
4801       { \c_group_begin_token }
4802       { \c_space_token }
4803     }
4804     \prg_return_true:
4805   \else:
4806     \prg_return_false:
4807   \fi:
4808 }

```

For `\tl_if_head_eq_meaning:nN`, again, detect special cases. In the normal case, use `\tl_head:w`, with no `\exp_not:N` this time, since `\if_meaning:w` causes no expansion. In the special cases, we know that the first token is a character, hence `\if_charcode:w` and `\if_catcode:w` together are enough. We combine them in some order, hopefully faster than the reverse.

```

4809 \prg_new_conditional:Npnn \tl_if_head_eq_meaning:nN #1#2 { p , T , F , TF }
4810 {

```

```

4811 \tl_if_head_N_type:nTF { #1 ? }
4812 { \tl_if_head_eq_meaning_aux_normal:nN }
4813 { \tl_if_head_eq_meaning_aux_special:nN }
4814 {#1} #2
4815 }
4816 \cs_new:Npn \tl_if_head_eq_meaning_aux_normal:nN #1 #2
4817 {
4818 \exp_after:wN \if_meaning:w \tl_head:w #1 \q_nil \q_stop #2
4819 \prg_return_true:
4820 \else:
4821 \prg_return_false:
4822 \fi:
4823 }
4824 \cs_new:Npn \tl_if_head_eq_meaning_aux_special:nN #1 #2
4825 {
4826 \if_charcode:w \str_head:n {#1} \exp_not:N #2
4827 \exp_after:wN \use:n
4828 \else:
4829 \prg_return_false:
4830 \exp_after:wN \use_none:n
4831 \fi:
4832 {
4833 \if_catcode:w \exp_not:N #2
4834 \tl_if_head_group:nTF {#1}
4835 { \c_group_begin_token }
4836 { \c_space_token }
4837 \prg_return_true:
4838 \else:
4839 \prg_return_false:
4840 \fi:
4841 }
4842 }

```

(End definition for `\tl_if_head_eq_meaning:nN`. This function is documented on page 91.)

`\tl_if_head_N_type:n` The first token of a token list can be either an N-type argument, a begin-group token (catcode 1), or an explicit space token (catcode 10 and charcode 32). These two cases are characterized by the fact that `\use:n` removes some tokens from `#1`, hence changing its string representation (no token can have an empty string representation). The extra brace group covers the case of an empty argument, whose head is not “normal”.

```

4843 \prg_new_conditional:Npnn \tl_if_head_N_type:n #1 { p , T , F , TF }
4844 {
4845 \str_if_eq_return:xx
4846 { \exp_not:o { \use:n #1 { } } }
4847 { \exp_not:n { #1 { } } }
4848 }

```

(End definition for `\tl_if_head_N_type:n`. This function is documented on page 92.)

`\tl_if_head_group:n` Pass the first token of `#1` through `\token_to_str:N`, then check for the brace balance.

The extra ? caters for an empty argument.⁶

```

4849 \prg_new_conditional:Npnn \tl_if_head_group:n #1 { p , T , F , TF }
4850 {
4851   \if_catcode:w *
4852     \exp_after:wN \use_none:n
4853     \exp_after:wN {
4854       \exp_after:wN {
4855         \token_to_str:N #1 ?
4856       }
4857     }
4858   *
4859   \prg_return_false:
4860 \else:
4861   \prg_return_true:
4862 \fi:
4863 }

```

(End definition for \tl_if_head_group:n. This function is documented on page 91.)

`\tl_if_head_space:n` If the first token of the token list is an explicit space, i.e., a character token with character code 32 and category code 10, then this test will be *<true>*. It is *<false>* if the token list is empty, if the first token is an implicit space token, such as `\c_space_token`, or any token other than an explicit space. The slightly convoluted approach with `\romannumeral` ensures that each expansion step gives a balanced token list.

```

4864 \prg_new_conditional:Npnn \tl_if_head_space:n #1 { p , T , F , TF }
4865 {
4866   \tex_romannumeral:D \if_false: { \fi:
4867     \tl_if_head_space_aux:w ? #1 ? ~ }
4868 }
4869 \cs_new:Npn \tl_if_head_space_aux:w #1 ~
4870 {
4871   \tl_if_empty:oTF { \use_none:n #1 }
4872   { \exp_after:wN \c_zero \exp_after:wN \prg_return_true: }
4873   { \exp_after:wN \c_zero \exp_after:wN \prg_return_false: }
4874   \exp_after:wN \use_none:n \exp_after:wN { \if_false: } \fi:
4875 }

```

(End definition for \tl_if_head_space:n. This function is documented on page ??.)

192.11 Viewing token lists

`\tl_show:N` Showing token list variables is done directly: at the moment do not worry if they are defined.
`\tl_show:c`

```

4876 \cs_new_protected:Npn \tl_show:N #1 { \cs_show:N #1 }
4877 \cs_generate_variant:Nn \tl_show:N { c }

```

(End definition for \tl_show:N and \tl_show:c. These functions are documented on page ??.)

⁶Bruno: this could be made faster, but we don't: if we hope to ever have an e-type argument, we need all brace "tricks" to happen in one step of expansion, keeping the token list brace balanced at all times.

`\tl_show:n` For literal token lists, life is easy.

```
4878 \cs_new_eq:NN \tl_show:n \etex_showtokens:D
      (End definition for \tl_show:n. This function is documented on page 92.)
```

192.12 Constant token lists

`\c_job_name_tl` Inherited from the L^AT_EX3 name for the primitive: this needs to actually contain the text of the job name rather than the name of the primitive, of course. Lua_TE_X does not quote file names containing spaces, whereas pdf_TE_X and X_TE_X do. So there may be a correction to make in the Lua_TE_X case.

```
4879 <*initex>
4880 \tex_everyjob:D \exp_after:wN
4881 {
4882   \tex_the:D \tex_everyjob:D
4883   \luatex_if_engine:T
4884   {
4885     \lua_now:x
4886     { dofile ( assert ( kpse.find_file ("lualatexquotejobname.lua" ) ) ) }
4887   }
4888 }
4889 </initex>
4890 \tl_const:Nx \c_job_name_tl { \tex_jobname:D }
      (End definition for \c_job_name_tl. This function is documented on page 92.)
```

`\c_space_tl` A space as a token list (as opposed to as a character).

```
4891 \tl_const:Nn \c_space_tl { ~ }
      (End definition for \c_space_tl. This function is documented on page 93.)
```

192.13 Scratch token lists

`\g_tmpa_tl` Global temporary token list variables. 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.

```
4892 \tl_new:N \g_tmpa_tl
4893 \tl_new:N \g_tmpb_tl
      (End definition for \g_tmpa_tl and \g_tmpb_tl. These functions are documented on page 93.)
```

`\l_tmpa_tl` These are local temporary token list variables. Be sure not to assume that the value you put into them will survive for long—see discussion above.

```
4894 \tl_new:N \l_tmpa_tl
4895 \tl_new:N \l_tmpb_tl
      (End definition for \l_tmpa_tl and \l_tmpb_tl. These functions are documented on page 93.)
```

192.14 Experimental functions

`\str_if_eq_return:xx` It turns out that we often need to compare a token list with the result of applying some function to it, and return with `\prg_return_true/false:.` This test is similar to `\str_if_eq:nnTF`, but hard-coded for speed.

```

4896 \cs_new:Npn \str_if_eq_return:xx #1 #2
4897 {
4898   \if_int_compare:w \pdfTeX_strcmp:D {#1} {#2} = \c_zero
4899     \prg_return_true:
4900   \else:
4901     \prg_return_false:
4902   \fi:
4903 }
```

(End definition for \str_if_eq_return:xx. This function is documented on page ??.)

`\tl_if_single:N` Expand the token list and feed it to `\tl_if_single:n`.

```

4904 \cs_new:Npn \tl_if_single_p:N { \exp_args:No \tl_if_single_p:n }
4905 \cs_new:Npn \tl_if_single:NT { \exp_args:No \tl_if_single:nT }
4906 \cs_new:Npn \tl_if_single:NF { \exp_args:No \tl_if_single:nF }
4907 \cs_new:Npn \tl_if_single:NTF { \exp_args:No \tl_if_single:nTF }
```

(End definition for \tl_if_single:N. This function is documented on page 86.)

`\tl_if_single:n` A token list has exactly one item if it is either a single token surrounded by optional explicit spaces, or a single brace group surrounded by optional explicit spaces. The naive version of this test would do `\use_none:n #1`, and test if the result is empty. However, this will fail when the token list is empty. Furthermore, it does not allow optional trailing spaces.

```

4908 \prg_new_conditional:Npnn \tl_if_single:n #1 { p , T , F , TF }
4909 { \str_if_eq_return:xx { \exp_not:o { \use_none:nn #1 ?? } } { ? } }
```

(End definition for \tl_if_single:n. This function is documented on page 86.)

`\tl_if_single_token:n` There are four cases: empty token list, token list starting with a normal token, with a brace group, or with a space token. If the token list starts with a normal token, remove it and check for emptiness. Otherwise, compare with a single space, only case where we have a single token.

```

4910 \prg_new_conditional:Npnn \tl_if_single_token:n #1 { p , T , F , TF }
4911 {
4912   \tl_if_head_N_type:nTF {#1}
4913   { \str_if_eq_return:xx { \exp_not:o { \use_none:n #1 } } { } }
4914   { \str_if_eq_return:xx { \exp_not:n {#1} } { ~ } }
4915 }
```

(End definition for \tl_if_single_token:n. This function is documented on page 86.)

`\q_tl_act_mark` The `\tl_act` functions may be applied to any token list. Hence, we use two private quarks, to allow any token, even quarks, in the token list. Only `\q_tl_act_mark` and `\q_tl_act_stop` may not appear in the token lists manipulated by `\tl_act` functions. The quarks are effectively defined in `l3quark`.

(End definition for \q_tl_act_mark and \q_tl_act_stop. These functions are documented on page 94.)

`\tl_act:NNNnn` To help control the expansion, `\tl_act:NNNnn` starts with `\romannumeral` and ends by
`\tl_act_aux:NNNnn` producing `\c_zero` once the result has been obtained. Then loop over tokens, groups,
`\tl_act_output:n` and spaces in #5. The marker `\q_tl_act_mark` is used both to avoid losing outer braces
`\tl_act_reverse_output:n` and to detect the end of the token list more easily. The result is stored as an argument
`\tl_act_group_recurse:Nnn` for the dummy function `\tl_act_result:n`.

```

4916 \cs_new:Npn \tl_act:NNNnn { \tex_romannumeral:D \tl_act_aux:NNNnn }
4917 \cs_new:Npn \tl_act_aux:NNNnn #1 #2 #3 #4 #5
4918 {
4919   \group_align_safe_begin:
4920   \tl_act_loop:w #5 \q_tl_act_mark \q_tl_act_stop
4921   {#4} #1 #2 #3
4922   \tl_act_result:n { }
4923 }

```

In the loop, we check how the token list begins and act accordingly. In the “normal” case, we may have reached `\q_tl_act_mark`, the end of the list. Then leave `\c_zero` and the result in the input stream, to terminate the expansion of `\romannumeral`. Otherwise, apply the relevant function to the “arguments”, #3 and to the head of the token list. Then repeat the loop. The scheme is the same if the token list starts with a group or with a space. Some extra work is needed to make `\tl_act_space:wnnNNN` gobble the space.

```

4924 \cs_new:Npn \tl_act_loop:w #1 \q_tl_act_stop
4925 {
4926   \tl_if_head_N_type:nTF {#1}
4927   { \tl_act_normal:NnnNNN }
4928   {
4929     \tl_if_head_group:nTF {#1}
4930     { \tl_act_group:wnnNNN }
4931     { \tl_act_space:wnnNNN }
4932   }
4933   #1 \q_tl_act_stop
4934 }
4935 \cs_new:Npn \tl_act_normal:NnnNNN #1 #2 \q_tl_act_stop #3#4
4936 {
4937   \if_meaning:w \q_tl_act_mark #1
4938   \exp_after:wN \tl_act_end:wn
4939   \fi:
4940   #4 {#3} #1
4941   \tl_act_loop:w #2 \q_tl_act_stop
4942   {#3} #4
4943 }
4944 \cs_new:Npn \tl_act_end:wn #1 \tl_act_result:n #2
4945 { \group_align_safe_end: \c_zero #2 }
4946 \cs_new:Npn \tl_act_group:wnnNNN #1 #2 \q_tl_act_stop #3#4#5
4947 {
4948   #5 {#3} {#1}
4949   \tl_act_loop:w #2 \q_tl_act_stop
4950   {#3} #4 #5
4951 }

```

```

4952 \exp_last_unbraced:NNo
4953 \cs_new:Npn \tl_act_space:wwnNNN \c_space_tl #1 \q_tl_act_stop #2#3#4#5
4954 {
4955     #5 {#2}
4956     \tl_act_loop:w #1 \q_tl_act_stop
4957     {#2} #3 #4 #5
4958 }

```

Typically, the output is done to the right of what was already output, using `\tl_act_output:n`, but for the `\tl_act_reverse` functions, it should be done to the left.

```

4959 \cs_new:Npn \tl_act_output:n #1 #2 \tl_act_result:n #3
4960 { #2 \tl_act_result:n { #3 #1 } }
4961 \cs_new:Npn \tl_act_reverse_output:n #1 #2 \tl_act_result:n #3
4962 { #2 \tl_act_result:n { #1 #3 } }

```

In many applications of `\tl_act:NNNnn`, we need to recursively apply some transformation within brace groups, then output. In this code, `#1` is the output function, `#2` is the transformation, which should expand in two steps, and `#3` is the group.

```

4963 \cs_new:Npn \tl_act_group_recurse:Nnn #1#2#3
4964 {
4965     \exp_args:Nf #1
4966     { \exp_after:wN \exp_after:wN \exp_after:wN { #2 {#3} } }
4967 }

```

(End definition for `\tl_act:NNNnn` and `\tl_act_aux:NNNnn`. These functions are documented on page ??.)

```

\tl_reverse_tokens:n
\tl_act_reverse_normal:nN
\tl_act_reverse_group:nn
\tl_act_reverse_space:n

```

The goal is to reverse a token list. This is done by feeding `\tl_act_aux:NNNnn` three functions, an empty fourth argument (we don't use it for `\tl_act_reverse_tokens:n`), and as a fifth argument the token list to be reversed. Spaces and normal tokens are output to the left of the current output. For groups, we must recursively apply `\tl_act_reverse_tokens:n` to the group, and output, still on the left. Note that in all three cases, we throw one argument away: this *parameter* is where for instance the upper/lowercasing action stores the information of whether it is uppercasing or lowercasing.

```

4968 \cs_new:Npn \tl_reverse_tokens:n
4969 {
4970     \tex_romannumeral:D
4971     \tl_act_aux:NNNnn
4972     \tl_act_reverse_normal:nN
4973     \tl_act_reverse_group:nn
4974     \tl_act_reverse_space:n
4975     { }
4976 }
4977 \cs_new:Npn \tl_act_reverse_space:n #1
4978 { \tl_act_reverse_output:n {~} }
4979 \cs_new:Npn \tl_act_reverse_normal:nN #1 #2
4980 { \tl_act_reverse_output:n {#2} }
4981 \cs_new:Npn \tl_act_reverse_group:nn #1
4982 {
4983     \tl_act_group_recurse:Nnn
4984     \tl_act_reverse_output:n

```

```

4985     { \tl_reverse_tokens:n }
4986   }

```

(End definition for `\tl_reverse_tokens:n`. This function is documented on page ??.)

`\tl_reverse:n` The goal here is to reverse without losing spaces nor braces. The only difference with
`\tl_reverse:o` `\tl_reverse_tokens:n` is that we now simply output groups without entering them.
`\tl_reverse:V`
`\tl_reverse_group_preserve:nn`

```

4987 \cs_new:Npn \tl_reverse:n
4988 {
4989   \tex_romannumeral:D
4990   \tl_act_aux:NNNnn
4991   \tl_act_reverse_normal:nN
4992   \tl_act_reverse_group_preserve:nn
4993   \tl_act_reverse_space:n
4994   { }
4995 }
4996 \cs_new:Npn \tl_act_reverse_group_preserve:nn #1 #2
4997 { \tl_act_reverse_output:n { {#2} } }
4998 \cs_generate_variant:Nn \tl_reverse:n { o , V }

```

(End definition for `\tl_reverse:n`, `\tl_reverse:o`, and `\tl_reverse:V`. These functions are documented on page ??.)

`\tl_reverse:N` This reverses the list, leaving `\exp_stop_f:` in front, which stops the f-expansion.
`\tl_reverse:c`
`\tl_greverse:N`
`\tl_greverse:c`

```

4999 \cs_new_protected:Npn \tl_reverse:N #1
5000 { \tl_set:Nf #1 { \tl_reverse:o { #1 \exp_stop_f: } } }
5001 \cs_new_protected:Npn \tl_greverse:N #1
5002 { \tl_gset:Nf #1 { \tl_reverse:o { #1 \exp_stop_f: } } }
5003 \cs_generate_variant:Nn \tl_reverse:N { c }
5004 \cs_generate_variant:Nn \tl_greverse:N { c }

```

(End definition for `\tl_reverse:N` and others. These functions are documented on page ??.)

`\tl_length_tokens:n` The length is computed through an `\int_eval:n` construction. Each 1+ is output to
`\tl_act_length_normal:nN` the left, into the integer expression, and the sum is ended by the `\c_zero` inserted by
`\tl_act_length_group:nn` `\tl_act_end:wn`. Somewhat a hack.
`\tl_act_length_space:n`

```

5005 \cs_new:Npn \tl_length_tokens:n #1
5006 {
5007   \int_eval:n
5008   {
5009     \tl_act_aux:NNNnn
5010     \tl_act_length_normal:nN
5011     \tl_act_length_group:nn
5012     \tl_act_length_space:n
5013     { }
5014     {#1}
5015   }
5016 }
5017 \cs_new:Npn \tl_act_length_normal:nN #1 #2 { 1 + }
5018 \cs_new:Npn \tl_act_length_space:n #1 { 1 + }
5019 \cs_new:Npn \tl_act_length_group:nn #1 #2
5020 { 2 + \tl_length_tokens:n {#2} + }

```

(End definition for \tl_length_tokens:n. This function is documented on page ??.)

\c_tl_act_uppercase_tl These constants contain the correspondance between lowercase and uppercase letters, in the form aAbBcC... and AaBbCc... respectively.

```

5021 \tl_const:Nn \c_tl_act_uppercase_tl
5022 {
5023   aA bB cC dD eE fF gG hH iI jJ kK lL mM
5024   nN oO pP qQ rR sS tT uU vV wW xX yY zZ
5025 }
5026 \tl_const:Nn \c_tl_act_lowercase_tl
5027 {
5028   Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm
5029   Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz
5030 }

```

(End definition for \c_tl_act_uppercase_tl and \c_tl_act_lowercase_tl. These functions are documented on page ??.)

\tl_expandable_uppercase:n The only difference between uppercasing and lowercasing is the table of correspondance that is used. As for other token list actions, we feed \tl_act_aux:NNNnn three functions, \tl_act_case_normal:nN and this time, we use the <parameters> argument to carry which case-changing we are applying. A space is simply output. A normal token is compared to each letter in the alphabet using \str_if_eq:nn tests, and converted if necessary to upper/lowercase, before being output. For a group, we must perform the conversion within the group (the \exp_after:wN trigger \romannumeral, which expands fully to give the converted group), then output.

```

5031 \cs_new:Npn \tl_expandable_uppercase:n
5032 { \tex_romannumeral:D \tl_act_case_aux:nn { \c_tl_act_uppercase_tl } }
5033 \cs_new:Npn \tl_expandable_lowercase:n
5034 { \tex_romannumeral:D \tl_act_case_aux:nn { \c_tl_act_lowercase_tl } }
5035 \cs_new:Npn \tl_act_case_aux:nn
5036 {
5037   \tl_act_aux:NNNnn
5038   \tl_act_case_normal:nN
5039   \tl_act_case_group:nn
5040   \tl_act_case_space:n
5041 }
5042 \cs_new:Npn \tl_act_case_space:n #1 { \tl_act_output:n {~} }
5043 \cs_new:Npn \tl_act_case_normal:nN #1 #2
5044 {
5045   \exp_args:Nf \tl_act_output:n
5046   {
5047     \exp_args:NNo \prg_case_str:nnn #2 {#1}
5048     { \exp_stop_f: #2 }
5049   }
5050 }
5051 \cs_new:Npn \tl_act_case_group:nn #1 #2
5052 {
5053   \exp_after:wN \tl_act_output:n \exp_after:wN
5054   { \exp_after:wN { \tex_romannumeral:D \tl_act_case_aux:nn {#1} {#2} } }

```

5055 }

(End definition for `\tl_expandable_uppercase:n` and `\tl_expandable_lowercase:n`. These functions are documented on page ??.)

`\tl_item:nn` The idea here is to find the offset of the item from the left, then use a loop to grab
`\tl_item:Nn` the correct item. If the resulting offset is too large, then `\quark_if_recursion_tail_`
`\tl_item:cn` `stop:n` terminates the loop, and returns nothing at all.
`\tl_item_aux:nn`

```
5056 \cs_new:Npn \tl_item:nn #1#2
5057 {
5058   \exp_args:Nf \tl_item_aux:nn
5059   {
5060     \int_eval:n
5061     {
5062       \int_compare:nNnT {#2} < \c_zero
5063       { \tl_length:n {#1} + }
5064       #2
5065     }
5066   }
5067   #1
5068   \q_recursion_tail
5069   \prg_break_point:n { }
5070 }
5071 \cs_new:Npn \tl_item_aux:nn #1#2
5072 {
5073   \quark_if_recursion_tail_break:n {#2}
5074   \int_compare:nNnTF {#1} = \c_zero
5075   { \tl_map_break:n {#2} }
5076   { \exp_args:Nf \tl_item_aux:nn { \int_eval:n { #1 - 1 } } }
5077 }
5078 \cs_new_nopar:Npn \tl_item:Nn { \exp_args:No \tl_item:nn }
5079 \cs_generate_variant:Nn \tl_item:Nn { c }
```

(End definition for `\tl_item:nn`, `\tl_item:Nn`, and `\tl_item:cn`. These functions are documented on page ??.)

`\tl_if_empty:x` We can test expandably the emptiness of an expanded token list thanks to the primitive `\pdfstrcmp` which expands its argument: a token list is empty if and only if its string representation is empty.

```
5080 \prg_new_conditional:Npnn \tl_if_empty:x #1 { p , T , F , TF }
5081 { \str_if_eq_return:xx { } {#1} }
```

(End definition for `\tl_if_empty:x`. This function is documented on page ??.)

192.15 Deprecated functions

`\tl_new:Nn` Use either `\tl_const:Nn` or `\tl_new:N`.

`\tl_new:cn` 5082 `{*deprecated}`

`\tl_new:Nx` 5083 `\cs_new_protected:Npn \tl_new:Nn #1#2`

```
5084 {
5085   \tl_new:N #1
```

```

5086 \tl_gset:Nn #1 {#2}
5087 }
5088 \cs_generate_variant:Nn \tl_new:Nn { c }
5089 \cs_generate_variant:Nn \tl_new:Nn { Nx }
5090 </deprecated>

```

(End definition for \tl_new:Nn, \tl_new:cn, and \tl_new:Nx. These functions are documented on page ??.)

\tl_gset:Nc This was useful once, but nowadays does not make much sense.
\tl_set:Nc

```

5091 <*deprecated>
5092 \cs_new_protected_nopar:Npn \tl_gset:Nc
5093 { \tex_global:D \tl_set:Nc }
5094 \cs_new_protected:Npn \tl_set:Nc #1#2
5095 { \tl_set:No #1 { \cs:w #2 \cs_end: } }
5096 </deprecated>

```

(End definition for \tl_gset:Nc. This function is documented on page ??.)

\tl_replace_in:Nnn These are renamed.
\tl_replace_in:cnn
\tl_greplace_in:Nnn
\tl_greplace_in:cnn
\tl_replace_all_in:Nnn
\tl_replace_all_in:cnn
\tl_greplace_all_in:Nnn
\tl_greplace_all_in:cnn

```

5097 <*deprecated>
5098 \cs_new_eq:NN \tl_replace_in:Nnn \tl_replace_once:Nnn
5099 \cs_new_eq:NN \tl_replace_in:cnn \tl_replace_once:cnn
5100 \cs_new_eq:NN \tl_greplace_in:Nnn \tl_greplace_once:Nnn
5101 \cs_new_eq:NN \tl_greplace_in:cnn \tl_greplace_once:cnn
5102 \cs_new_eq:NN \tl_replace_all_in:Nnn \tl_replace_all:Nnn
5103 \cs_new_eq:NN \tl_replace_all_in:cnn \tl_replace_all:cnn
5104 \cs_new_eq:NN \tl_greplace_all_in:Nnn \tl_greplace_all:Nnn
5105 \cs_new_eq:NN \tl_greplace_all_in:cnn \tl_greplace_all:cnn
5106 </deprecated>

```

(End definition for \tl_replace_in:Nnn and \tl_replace_in:cnn. These functions are documented on page ??.)

\tl_remove_in:Nn Also renamed.
\tl_remove_in:cn
\tl_gremove_in:Nn
\tl_gremove_in:cn
\tl_remove_all_in:Nn
\tl_remove_all_in:cn
\tl_gremove_all_in:Nn
\tl_gremove_all_in:cn

```

5107 <*deprecated>
5108 \cs_new_eq:NN \tl_remove_in:Nn \tl_remove_once:Nn
5109 \cs_new_eq:NN \tl_remove_in:cn \tl_remove_once:cn
5110 \cs_new_eq:NN \tl_gremove_in:Nn \tl_gremove_once:Nn
5111 \cs_new_eq:NN \tl_gremove_in:cn \tl_gremove_once:cn
5112 \cs_new_eq:NN \tl_remove_all_in:Nn \tl_remove_all:Nn
5113 \cs_new_eq:NN \tl_remove_all_in:cn \tl_remove_all:cn
5114 \cs_new_eq:NN \tl_gremove_all_in:Nn \tl_gremove_all:Nn
5115 \cs_new_eq:NN \tl_gremove_all_in:cn \tl_gremove_all:cn
5116 </deprecated>

```

(End definition for \tl_remove_in:Nn and \tl_remove_in:cn. These functions are documented on page ??.)

\tl_elt_count:n Another renaming job.
\tl_elt_count:V
\tl_elt_count:o
\tl_elt_count:N
\tl_elt_count:c

```

5117 <*deprecated>
5118 \cs_new_eq:NN \tl_elt_count:n \tl_length:n
5119 \cs_new_eq:NN \tl_elt_count:V \tl_length:V

```

```

5120 \cs_new_eq:NN \tl_elt_count:o \tl_length:o
5121 \cs_new_eq:NN \tl_elt_count:N \tl_length:N
5122 \cs_new_eq:NN \tl_elt_count:c \tl_length:c
5123 </deprecated>
      (End definition for \tl_elt_count:n, \tl_elt_count:V, and \tl_elt_count:o. These functions
are documented on page ??.)

```

```

\tl_head_i:n Two renames, and a few that are rather too specialised.
\tl_head_i:w 5124 <*deprecated>
\tl_head_iii:n 5125 \cs_new_eq:NN \tl_head_i:n \tl_head:n
\tl_head_iii:f 5126 \cs_new_eq:NN \tl_head_i:w \tl_head:w
\tl_head_iii:w 5127 \cs_new:Npn \tl_head_iii:n #1 { \tl_head_iii:w #1 \q_stop }
5128 \cs_generate_variant:Nn \tl_head_iii:n { f }
5129 \cs_new:Npn \tl_head_iii:w #1#2#3#4 \q_stop {#1#2#3}
5130 </deprecated>
      (End definition for \tl_head_i:n. This function is documented on page ??.)
5131 </initex | package>

```

193 l3seq implementation

The following test files are used for this code: *m3seq002*, *m3seq003*.

```

5132 <*initex | package>
5133 <*package>
5134 \ProvidesExplPackage
5135   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
5136 \package_check_loaded_expl:
5137 </package>

```

A sequence is a control sequence whose top-level expansion is of the form “`\seq_item:n {⟨item0⟩} ... \seq_item:n {⟨itemn-1⟩}`”. An earlier implementation used the structure “`\seq_elt:w ⟨item1⟩ \seq_elt_end: ... \seq_elt:w ⟨itemn⟩ \seq_elt_end:`”. This allows rapid searching using a delimited function, but is not suitable for items containing `{`, `}` and `#` tokens, and also leads to the loss of surrounding braces around items.

`\seq_item:n` The delimiter is always defined, but when used incorrectly simply removes its argument and hits an undefined control sequence to raise an error.

```

5138 \cs_new:Npn \seq_item:n
5139   {
5140     \msg_expandable_kernel_error:nn { seq } { misused }
5141     \use_none:n
5142   }
      (End definition for \seq_item:n. This function is documented on page 102.)

```

`\l_seq_tmpa_tl` Scratch space for various internal uses.

```

\l_seq_tmpb_tl 5143 \tl_new:N \l_seq_tmpa_tl
5144 \tl_new:N \l_seq_tmpb_tl
      (End definition for \l_seq_tmpa_tl and \l_seq_tmpb_tl. These functions are documented on
page ??.)

```

193.1 Allocation and initialisation

`\seq_new:N` Internally, sequences are just token lists.

`\seq_new:c` 5145 `\cs_new_eq:NN \seq_new:N \tl_new:N`

5146 `\cs_new_eq:NN \seq_new:c \tl_new:c`

(End definition for `\seq_new:N` and `\seq_new:c`. These functions are documented on page ??.)

`\seq_clear:N` Clearing sequences is just the same as clearing token lists.

`\seq_clear:c` 5147 `\cs_new_eq:NN \seq_clear:N \tl_clear:N`

`\seq_gclear:N` 5148 `\cs_new_eq:NN \seq_clear:c \tl_clear:c`

`\seq_gclear:c` 5149 `\cs_new_eq:NN \seq_gclear:N \tl_gclear:N`

5150 `\cs_new_eq:NN \seq_gclear:c \tl_gclear:c`

(End definition for `\seq_clear:N` and `\seq_clear:c`. These functions are documented on page ??.)

`\seq_clear_new:N` Once again a copy from the token list functions.

`\seq_clear_new:c` 5151 `\cs_new_eq:NN \seq_clear_new:N \tl_clear_new:N`

`\seq_gclear_new:N` 5152 `\cs_new_eq:NN \seq_clear_new:c \tl_clear_new:c`

`\seq_gclear_new:c` 5153 `\cs_new_eq:NN \seq_gclear_new:N \tl_gclear_new:N`

5154 `\cs_new_eq:NN \seq_gclear_new:c \tl_gclear_new:c`

(End definition for `\seq_clear_new:N` and `\seq_clear_new:c`. These functions are documented on page ??.)

`\seq_set_eq:NN` Once again, these are simple copies from the token list functions.

`\seq_set_eq:cN` 5155 `\cs_new_eq:NN \seq_set_eq:NN \tl_set_eq:NN`

`\seq_set_eq:Nc` 5156 `\cs_new_eq:NN \seq_set_eq:Nc \tl_set_eq:Nc`

`\seq_set_eq:cc` 5157 `\cs_new_eq:NN \seq_set_eq:cN \tl_set_eq:cN`

`\seq_gset_eq:NN` 5158 `\cs_new_eq:NN \seq_set_eq:cc \tl_set_eq:cc`

`\seq_gset_eq:cN` 5159 `\cs_new_eq:NN \seq_gset_eq:NN \tl_gset_eq:NN`

`\seq_gset_eq:Nc` 5160 `\cs_new_eq:NN \seq_gset_eq:Nc \tl_gset_eq:Nc`

`\seq_gset_eq:cN` 5161 `\cs_new_eq:NN \seq_gset_eq:cN \tl_gset_eq:cN`

5162 `\cs_new_eq:NN \seq_gset_eq:cc \tl_gset_eq:cc`

(End definition for `\seq_set_eq:NN` and others. These functions are documented on page ??.)

`\seq_set_split:Nnn` The goal is to split a given token list at a marker, strip spaces from each item, and
`\seq_gset_split:Nnn` remove one set of outer braces if after removing leading and trailing spaces the item
`\seq_set_split_aux:Nnnn` is enclosed within braces. After `\tl_replace_all:Nnn`, the token list `\l_seq_tmpa_tl`
`\seq_set_split_aux_i:w` is a repetition of the pattern `\seq_set_split_aux_i:w \prg_do_nothing: <item with`
`\seq_set_split_aux_ii:w` spaces> `\seq_set_split_aux_end:.` Then, x-expansion causes `\seq_set_split_aux_`
`\seq_set_split_aux_end:` `i:w` to trim spaces, and leaves its result as `\seq_set_split_aux_ii:w <trimmed item>`
`\seq_set_split_aux_end:.` This is then converted to the l3seq internal structure by
another x-expansion. In the first step, we insert `\prg_do_nothing:` to avoid losing
braces too early: that would cause space trimming to act within those lost braces. The
second step is solely there to strip braces which are outermost after space trimming.

5163 `\cs_new_protected_nopar:Npn \seq_set_split:Nnn`

5164 `{ \seq_set_split_aux:Nnnn \tl_set:Nx }`

5165 `\cs_new_protected_nopar:Npn \seq_gset_split:Nnn`

5166 `{ \seq_set_split_aux:Nnnn \tl_gset:Nx }`


```

5167 \cs_new_protected:Npn \seq_set_split_aux:NNnn #1 #2 #3 #4
5168 {
5169   \tl_if_empty:nTF {#3}
5170   { #1 #2 { \tl_map_function:nN {#4} \seq_wrap_item:n } }
5171   {
5172     \tl_set:Nn \l_seq_tmpa_tl
5173     {
5174       \seq_set_split_aux_i:w \prg_do_nothing:
5175       #4
5176       \seq_set_split_aux_end:
5177     }
5178     \tl_replace_all:Nnn \l_seq_tmpa_tl { #3 }
5179     {
5180       \seq_set_split_aux_end:
5181       \seq_set_split_aux_i:w \prg_do_nothing:
5182     }
5183     \tl_set:Nx \l_seq_tmpa_tl { \l_seq_tmpa_tl }
5184     #1 #2 { \l_seq_tmpa_tl }
5185   }
5186 }
5187 \cs_new:Npn \seq_set_split_aux_i:w #1 \seq_set_split_aux_end:
5188 {
5189   \exp_not:N \seq_set_split_aux_ii:w
5190   \exp_args:No \tl_trim_spaces:n {#1}
5191   \exp_not:N \seq_set_split_aux_end:
5192 }
5193 \cs_new:Npn \seq_set_split_aux_ii:w #1 \seq_set_split_aux_end:
5194 { \seq_wrap_item:n {#1} }

```

(End definition for \seq_set_split:Nnn and \seq_gset_split:Nnn. These functions are documented on page ??.)

```

\seq_concat:NNN Concatenating sequences is easy.
\seq_concat:ccc 5195 \cs_new_protected:Npn \seq_concat:NNN #1#2#3
\seq_gconcat:NNN 5196 { \tl_set:Nx #1 { \exp_not:o {#2} \exp_not:o {#3} } }
\seq_gconcat:ccc 5197 \cs_new_protected:Npn \seq_gconcat:NNN #1#2#3
5198 { \tl_gset:Nx #1 { \exp_not:o {#2} \exp_not:o {#3} } }
5199 \cs_generate_variant:Nn \seq_concat:NNN { ccc }
5200 \cs_generate_variant:Nn \seq_gconcat:NNN { ccc }

```

(End definition for \seq_concat:NNN and \seq_concat:ccc. These functions are documented on page ??.)

193.2 Appending data to either end

```

\seq_put_left:Nn The code here is just a wrapper for adding to token lists.
\seq_put_left:NV 5201 \cs_new_protected:Npn \seq_put_left:Nn #1#2
\seq_put_left:Nv 5202 { \tl_put_left:Nn #1 { \seq_item:n {#2} } }
\seq_put_left:No 5203 \cs_new_protected:Npn \seq_put_right:Nn #1#2
\seq_put_left:Nx 5204 { \tl_put_right:Nn #1 { \seq_item:n {#2} } }
\seq_put_left:cn 5205 \cs_generate_variant:Nn \seq_put_left:Nn { NV , Nv , No , Nx }
\seq_put_left:cV
\seq_put_left:cv
\seq_put_left:co
\seq_put_left:cx
\seq_put_right:Nn
\seq_put_right:NV
\seq_put_right:Nv
\seq_put_right:No
\seq_put_right:Nx
\seq_put_right:cn

```

```

5206 \cs_generate_variant:Nn \seq_put_left:Nn { c , cV , cv , co , cx }
5207 \cs_generate_variant:Nn \seq_put_right:Nn { NV , Nv , No , Nx }
5208 \cs_generate_variant:Nn \seq_put_right:Nn { c , cV , cv , co , cx }
(End definition for \seq_put_left:Nn and others. These functions are documented on page ??.)

```

The same for global addition.

```

\seq_gput_left:Nn
\seq_gput_left:NV
\seq_gput_left:Nv
\seq_gput_left:No
\seq_gput_left:Nx
\seq_gput_left:cn
\seq_gput_left:cV
\seq_gput_left:cv
\seq_gput_left:co
\seq_gput_left:cx
\seq_gput_right:Nn
\seq_gput_right:NV
\seq_gput_right:Nv
\seq_gput_right:No
\seq_gput_right:Nx
\seq_gput_right:cn
\seq_gput_right:cV
\seq_gput_right:cv
\seq_gput_right:co
\seq_gput_right:cx

```

```

5209 \cs_new_protected:Npn \seq_gput_left:Nn #1#2
5210 { \tl_gput_left:Nn #1 { \seq_item:n {#2} } }
5211 \cs_new_protected:Npn \seq_gput_right:Nn #1#2
5212 { \tl_gput_right:Nn #1 { \seq_item:n {#2} } }
5213 \cs_generate_variant:Nn \seq_gput_left:Nn { NV , Nv , No , Nx }
5214 \cs_generate_variant:Nn \seq_gput_left:Nn { c , cV , cv , co , cx }
5215 \cs_generate_variant:Nn \seq_gput_right:Nn { NV , Nv , No , Nx }
5216 \cs_generate_variant:Nn \seq_gput_right:Nn { c , cV , cv , co , cx }
(End definition for \seq_gput_left:Nn and others. These functions are documented on page ??.)

```

193.3 Modifying sequences

This function converts its argument to a proper sequence item in an x-expansion context.

```

5217 \cs_new:Npn \seq_wrap_item:n #1 { \exp_not:n { \seq_item:n {#1} } }
(End definition for \seq_wrap_item:n. This function is documented on page ??.)

```

An internal sequence for the removal routines.

```

5218 \seq_new:N \l_seq_remove_seq
(End definition for \l_seq_remove_seq. This function is documented on page ??.)

```

Removing duplicates means making a new list then copying it.

```

\seq_remove_duplicates:N
\seq_remove_duplicates:c
\seq_gremove_duplicates:N
\seq_gremove_duplicates:c
\seq_remove_duplicates_aux:NN

```

```

5219 \cs_new_protected:Npn \seq_remove_duplicates:N
5220 { \seq_remove_duplicates_aux:NN \seq_set_eq:NN }
5221 \cs_new_protected:Npn \seq_gremove_duplicates:N
5222 { \seq_remove_duplicates_aux:NN \seq_gset_eq:NN }
5223 \cs_new_protected:Npn \seq_remove_duplicates_aux:NN #1#2
5224 {
5225   \seq_clear:N \l_seq_remove_seq
5226   \seq_map_inline:Nn #2
5227   {
5228     \seq_if_in:NnF \l_seq_remove_seq {##1}
5229     { \seq_put_right:Nn \l_seq_remove_seq {##1} }
5230   }
5231   #1 #2 \l_seq_remove_seq
5232 }
5233 \cs_generate_variant:Nn \seq_remove_duplicates:N { c }
5234 \cs_generate_variant:Nn \seq_gremove_duplicates:N { c }
(End definition for \seq_remove_duplicates:N and \seq_remove_duplicates:c. These functions are documented on page ??.)

```

`\seq_remove_all:Nn` The idea of the code here is to avoid a relatively expensive addition of items one at a time
`\seq_remove_all:cn` to an intermediate sequence. The approach taken is therefore similar to that in `\seq_`
`\seq_gremove_all:Nn` `pop_right_aux_ii:NNN`, using a “flexible” x-type expansion to do most of the work.
`\seq_gremove_all:cn` As `\tl_if_eq:nnT` is not expandable, a two-part strategy is needed. First, the x-type
`\seq_remove_all_aux:NNn` expansion uses `\str_if_eq:nnT` to find potential matches. If one is found, the expansion
is halted and the necessary set up takes place to use the `\tl_if_eq:NNT` test. The x-type
is started again, including all of the items copied already. This will happen repeatedly
until the entire sequence has been scanned. The code is set up to avoid needing and
intermediate scratch list: the lead-off x-type expansion (`#1 #2 {#2}`) will ensure that
nothing is lost.

```

5235 \cs_new_protected:Npn \seq_remove_all:Nn
5236 { \seq_remove_all_aux:NNn \tl_set:Nx }
5237 \cs_new_protected:Npn \seq_gremove_all:Nn
5238 { \seq_remove_all_aux:NNn \tl_gset:Nx }
5239 \cs_new_protected:Npn \seq_remove_all_aux:NNn #1#2#3
5240 {
5241   \seq_push_item_def:n
5242   {
5243     \str_if_eq:nnT {##1} {#3}
5244     {
5245       \if_false: { \fi: }
5246       \tl_set:Nn \l_seq_tmpb_tl {##1}
5247       #1 #2
5248       { \if_false: } \fi:
5249       \exp_not:o {#2}
5250       \tl_if_eq:NNT \l_seq_tmpa_tl \l_seq_tmpb_tl
5251       { \use_none:nn }
5252     }
5253     \seq_wrap_item:n {##1}
5254   }
5255   \tl_set:Nn \l_seq_tmpa_tl {#3}
5256   #1 #2 {#2}
5257   \seq_pop_item_def:
5258 }
5259 \cs_generate_variant:Nn \seq_remove_all:Nn { c }
5260 \cs_generate_variant:Nn \seq_gremove_all:Nn { c }

```

(End definition for `\seq_remove_all:Nn` and `\seq_remove_all:cn`. These functions are documented on page ??.)

193.4 Sequence conditionals

`\seq_if_empty:N` Simple copies from the token list variable material.

```

\seq_if_empty:c
5261 \prg_new_eq_conditional:NNn \seq_if_empty:N \tl_if_empty:N
5262 { p , T , F , TF }
5263 \prg_new_eq_conditional:NNn \seq_if_empty:c \tl_if_empty:c
5264 { p , T , F , TF }

```

(End definition for `\seq_if_empty:N` and `\seq_if_empty:c`. These functions are documented on page ??.)

`\seq_if_in:Nn` The approach here is to define `\seq_item:n` to compare its argument with the test
`\seq_if_in:NV` sequence. If the two items are equal, the mapping is terminated and `\prg_return_-`
`\seq_if_in:Nv` `true:` is inserted. On the other hand, if there is no match then the loop will break
`\seq_if_in:No` returning `\prg_return_false:`. In either case, `\prg_break_point:n` ensures that the
`\seq_if_in:Nx` group ends before the logical value is returned. Everything is inside a group so that
`\seq_if_in:cn` `\seq_item:n` is preserved in nested situations.
`\seq_if_in:cV` 5265 `\prg_new_protected_conditional:Npnn \seq_if_in:Nn #1#2`
`\seq_if_in:cv` 5266 `{ T , F , TF }`
`\seq_if_in:co` 5267 `{`
`\seq_if_in:cx` 5268 `\group_begin:`
`\seq_if_in_aux:` 5269 `\tl_set:Nn \l_seq_tmpa_tl {#2}`
5270 `\cs_set_protected:Npn \seq_item:n ##1`
5271 `{`
5272 `\tl_set:Nn \l_seq_tmpb_tl {##1}`
5273 `\if_meaning:w \l_seq_tmpa_tl \l_seq_tmpb_tl`
5274 `\exp_after:wN \seq_if_in_aux:`
5275 `\fi:`
5276 `}`
5277 `#1`
5278 `\seq_break:n { \prg_return_false: }`
5279 `\prg_break_point:n { \group_end: }`
5280 `}`
5281 `\cs_new_nopar:Npn \seq_if_in_aux: { \seq_break:n { \prg_return_true: } }`
5282 `\cs_generate_variant:Nn \seq_if_in:NnT { NV , Nv , No , Nx }`
5283 `\cs_generate_variant:Nn \seq_if_in:NnT { c , cV , cv , co , cx }`
5284 `\cs_generate_variant:Nn \seq_if_in:NnF { NV , Nv , No , Nx }`
5285 `\cs_generate_variant:Nn \seq_if_in:NnF { c , cV , cv , co , cx }`
5286 `\cs_generate_variant:Nn \seq_if_in:NnTF { NV , Nv , No , Nx }`
5287 `\cs_generate_variant:Nn \seq_if_in:NnTF { c , cV , cv , co , cx }`
(End definition for \seq_if_in:Nn and others. These functions are documented on page ??.)

193.5 Recovering data from sequences

`\seq_get_left:NN` Getting an item from the left of a sequence is pretty easy: just trim off the first item
`\seq_get_left:cN` after removing the `\seq_item:n` at the start.
`\seq_get_left_aux:NnwN` 5288 `\cs_new_protected:Npn \seq_get_left:NN #1#2`
5289 `{`
5290 `\seq_if_empty_err_break:N #1`
5291 `\exp_after:wN \seq_get_left_aux:NnwN #1 \q_stop #2`
5292 `\prg_break_point:n { }`
5293 `}`
5294 `\cs_new_protected:Npn \seq_get_left_aux:NnwN \seq_item:n #1#2 \q_stop #3`
5295 `{ \tl_set:Nn #3 {#1} }`
5296 `\cs_generate_variant:Nn \seq_get_left:NN { c }`
(End definition for \seq_get_left:NN and \seq_get_left:cN. These functions are documented on page ??.)

`\seq_pop_left:NN` The approach to popping an item is pretty similar to that to get an item, with the only
`\seq_pop_left:cN` difference being that the sequence itself has to be redefined. This makes it more sensible
`\seq_gpop_left:NN` to use an auxiliary function for the local and global cases.
`\seq_gpop_left:cN`
`\seq_pop_left_aux:NNN`
`\seq_pop_left_aux:NnwNNN`

```

5297 \cs_new_protected_nopar:Npn \seq_pop_left:NN
5298 { \seq_pop_left_aux:NNN \tl_set:Nn }
5299 \cs_new_protected_nopar:Npn \seq_gpop_left:NN
5300 { \seq_pop_left_aux:NNN \tl_gset:Nn }
5301 \cs_new_protected:Npn \seq_pop_left_aux:NNN #1#2#3
5302 {
5303   \seq_if_empty_err_break:N #2
5304   \exp_after:wN \seq_pop_left_aux:NnwNNN #2 \q_stop #1#2#3
5305   \prg_break_point:n { }
5306 }
5307 \cs_new_protected:Npn \seq_pop_left_aux:NnwNNN \seq_item:n #1#2 \q_stop #3#4#5
5308 {
5309   #3 #4 {#2}
5310   \tl_set:Nn #5 {#1}
5311 }
5312 \cs_generate_variant:Nn \seq_pop_left:NN { c }
5313 \cs_generate_variant:Nn \seq_gpop_left:NN { c }

```

(End definition for `\seq_pop_left:NN` and `\seq_pop_left:cN`. These functions are documented on page ??.)

`\seq_get_right:NN` The idea here is to remove the very first `\seq_item:n` from the sequence, leaving a token
`\seq_get_right:cN` list starting with the first braced entry. Two arguments at a time are then grabbed: apart
`\seq_get_right_aux:NN` from the right-hand end of the sequence, this will be a brace group followed by `\seq_`
`\seq_get_right_loop:nn` `item:n`. The set up code means that these all disappear. At the end of the sequence, the
assignment is placed in front of the very last entry in the sequence, before a tidying-up
step takes place to remove the loop and reset the meaning of `\seq_item:n`.

```

5314 \cs_new_protected:Npn \seq_get_right:NN #1#2
5315 {
5316   \seq_if_empty_err_break:N #1
5317   \seq_get_right_aux:NN #1#2
5318   \prg_break_point:n { }
5319 }
5320 \cs_new_protected:Npn \seq_get_right_aux:NN #1#2
5321 {
5322   \seq_push_item_def:n { }
5323   \exp_after:wN \exp_after:wN \exp_after:wN \seq_get_right_loop:nn
5324   \exp_after:wN \use_none:n #1
5325   { \tl_set:Nn #2 }
5326   { }
5327   {
5328     \seq_pop_item_def:
5329     \seq_break:
5330   }
5331 }
5332 \cs_new:Npn \seq_get_right_loop:nn #1#2
5333 {

```

```

5334     #2 {#1}
5335     \seq_get_right_loop:nn
5336   }
5337   \cs_generate_variant:Nn \seq_get_right:NN { c }

```

(End definition for `\seq_get_right:NN` and `\seq_get_right:cN`. These functions are documented on page ??.)

```

\seq_pop_right:NN
\seq_pop_right:cN
\seq_gpop_right:NN
\seq_gpop_right:cN
\seq_pop_right_aux:NNN
\seq_pop_right_aux_ii:NNN

```

The approach to popping from the right is a bit more involved, but does use some of the same ideas as getting from the right. What is needed is a “flexible length” way to set a token list variable. This is supplied by the `{ \if_false: } \fi: ... \if_false: { \fi: }` construct. Using an x-type expansion and a “non-expanding” definition for `\seq_item:n`, the left-most $n - 1$ entries in a sequence of n items will be stored back in the sequence. That needs a loop of unknown length, hence using the strange `\if_false:` way of including brackets. When the last item of the sequence is reached, the closing bracket for the assignment is inserted, and `\tl_set:Nn #3` is inserted in front of the final entry. This therefore does the pop assignment, then a final loop clears up the code.

```

5338   \cs_new_protected_nopar:Npn \seq_pop_right:NN
5339     { \seq_pop_right_aux:NNN \tl_set:Nx }
5340   \cs_new_protected_nopar:Npn \seq_gpop_right:NN
5341     { \seq_pop_right_aux:NNN \tl_gset:Nx }
5342   \cs_new_protected:Npn \seq_pop_right_aux:NNN #1#2#3
5343     {
5344       \seq_if_empty_err_break:N #2
5345       \seq_pop_right_aux_ii:NNN #1 #2 #3
5346       \prg_break_point:n { }
5347     }
5348   \cs_new_protected:Npn \seq_pop_right_aux_ii:NNN #1#2#3
5349     {
5350       \seq_push_item_def:n { \seq_wrap_item:n {##1} }
5351       #1 #2 { \if_false: } \fi:
5352       \exp_after:wN \exp_after:wN \exp_after:wN \seq_get_right_loop:nn
5353       \exp_after:wN \use_none:n #2
5354       {
5355         \if_false: { \fi: }
5356         \tl_set:Nn #3
5357       }
5358       { }
5359       {
5360         \seq_pop_item_def:
5361         \seq_break:
5362       }
5363     }
5364   \cs_generate_variant:Nn \seq_pop_right:NN { c }
5365   \cs_generate_variant:Nn \seq_gpop_right:NN { c }

```

(End definition for `\seq_pop_right:NN` and `\seq_pop_right:cN`. These functions are documented on page ??.)

193.6 Mapping to sequences

`\seq_map_break:` To break a function, the special token `\prg_break_point:n` is used to find the end of the code. Any ending code is then inserted before the return value of `\seq_map_break:n` is inserted. Semantically-logical copies of the break functions for use inside mappings.

```
\seq_break:
\seq_break:N
5366 \cs_new_eq:NN \seq_break: \prg_map_break:
5367 \cs_new_eq:NN \seq_break:n \prg_map_break:n
5368 \cs_new_eq:NN \seq_map_break: \prg_map_break:
5369 \cs_new_eq:NN \seq_map_break:n \prg_map_break:n
```

(End definition for \seq_map_break:. This function is documented on page 103.)

`\seq_if_empty_err_break:N` A function to check that sequences really have some content. This is optimised for speed, hence the direct primitive use.

```
5370 \cs_new_protected:Npn \seq_if_empty_err_break:N #1
5371 {
5372   \if_meaning:w #1 \c_empty_tl
5373   \msg_kernel_error:nxx { seq } { empty-sequence } { \token_to_str:N #1 }
5374   \exp_after:wN \seq_break:
5375   \fi:
5376 }
```

(End definition for \seq_if_empty_err_break:N. This function is documented on page 102.)

`\seq_map_function:NN` The idea here is to apply the code of #2 to each item in the sequence without altering the definition of `\seq_item:n`. This is done as by noting that every odd token in the sequence must be `\seq_item:n`, which can be gobbled by `\use_none:n`. At the end of the loop, #2 is instead `\seq_map_break:`, which therefore breaks the loop without needing to do a (relatively-expensive) quark test.

```
5377 \cs_new:Npn \seq_map_function:NN #1#2
5378 {
5379   \exp_after:wN \seq_map_function_aux:NNn \exp_after:wN #2 #1
5380   { ? \seq_map_break: } { }
5381   \prg_break_point:n { }
5382 }
5383 \cs_new:Npn \seq_map_function_aux:NNn #1#2#3
5384 {
5385   \use_none:n #2
5386   #1 {#3}
5387   \seq_map_function_aux:NNn #1
5388 }
5389 \cs_generate_variant:Nn \seq_map_function:NN { c }
```

(End definition for \seq_map_function:NN and \seq_map_function:cN. These functions are documented on page ??.)

`\seq_push_item_def:n` The definition of `\seq_item:n` needs to be saved and restored at various points within the mapping and manipulation code. That is handled here: as always, this approach uses global assignments.

```
\seq_push_item_def:x
\seq_push_item_def_aux:
\seq_pop_item_def:
5390 \cs_new_protected:Npn \seq_push_item_def:n
5391 {
```

```

5392 \seq_push_item_def_aux:
5393 \cs_gset:Npn \seq_item:n ##1
5394 }
5395 \cs_new_protected:Npn \seq_push_item_def:x
5396 {
5397 \seq_push_item_def_aux:
5398 \cs_gset:Npx \seq_item:n ##1
5399 }
5400 \cs_new_protected:Npn \seq_push_item_def_aux:
5401 {
5402 \cs_gset_eq:cN { seq_item_ \int_use:N \g_prg_map_int :n }
5403 \seq_item:n
5404 \int_gincr:N \g_prg_map_int
5405 }
5406 \cs_new_protected_nopar:Npn \seq_pop_item_def:
5407 {
5408 \int_gdecr:N \g_prg_map_int
5409 \cs_gset_eq:Nc \seq_item:n
5410 { seq_item_ \int_use:N \g_prg_map_int :n }
5411 }

```

(End definition for \seq_push_item_def:n and \seq_push_item_def:x. These functions are documented on page ??.)

`\seq_map_inline:Nn` The idea here is that `\seq_item:n` is already “applied” to each item in a sequence, and
`\seq_map_inline:cn` so an in-line mapping is just a case of redefining `\seq_item:n`.

```

5412 \cs_new_protected:Npn \seq_map_inline:Nn #1#2
5413 {
5414 \seq_push_item_def:n {#2}
5415 #1
5416 \prg_break_point:n { \seq_pop_item_def: }
5417 }
5418 \cs_generate_variant:Nn \seq_map_inline:Nn { c }

```

(End definition for \seq_map_inline:Nn and \seq_map_inline:cn. These functions are documented on page ??.)

`\seq_map_variable:NNn` This is just a specialised version of the in-line mapping function, using an `x`-type expansion for the code set up so that the number of `#` tokens required is as expected.
`\seq_map_variable:Ncn`
`\seq_map_variable:cNn`
`\seq_map_variable:ccn`

```

5419 \cs_new_protected:Npn \seq_map_variable:NNn #1#2#3
5420 {
5421 \seq_push_item_def:x
5422 {
5423 \tl_set:Nn \exp_not:N #2 {##1}
5424 \exp_not:n {#3}
5425 }
5426 #1
5427 \prg_break_point:n { \seq_pop_item_def: }
5428 }
5429 \cs_generate_variant:Nn \seq_map_variable:NNn { Nc }
5430 \cs_generate_variant:Nn \seq_map_variable:NNn { c , cc }

```


(End definition for `\seq_map_variable:NNn` and others. These functions are documented on page ??.)

193.7 Sequence stacks

The same functions as for sequences, but with the correct naming.

`\seq_push:Nn` Pushing to a sequence is the same as adding on the left.

<code>\seq_push:NV</code>	5431	<code>\cs_new_eq:NN \seq_push:Nn \seq_put_left:Nn</code>
<code>\seq_push:Nv</code>	5432	<code>\cs_new_eq:NN \seq_push:NV \seq_put_left:Nv</code>
<code>\seq_push:No</code>	5433	<code>\cs_new_eq:NN \seq_push:Nv \seq_put_left:Nv</code>
<code>\seq_push:Nx</code>	5434	<code>\cs_new_eq:NN \seq_push:No \seq_put_left:No</code>
<code>\seq_push:cn</code>	5435	<code>\cs_new_eq:NN \seq_push:Nx \seq_put_left:Nx</code>
<code>\seq_push:cV</code>	5436	<code>\cs_new_eq:NN \seq_push:cn \seq_put_left:cn</code>
<code>\seq_push:cV</code>	5437	<code>\cs_new_eq:NN \seq_push:cV \seq_put_left:cV</code>
<code>\seq_push:co</code>	5438	<code>\cs_new_eq:NN \seq_push:cV \seq_put_left:cv</code>
<code>\seq_push:co</code>	5439	<code>\cs_new_eq:NN \seq_push:co \seq_put_left:co</code>
<code>\seq_push:cx</code>	5440	<code>\cs_new_eq:NN \seq_push:cx \seq_put_left:cx</code>
<code>\seq_gpush:Nn</code>	5441	<code>\cs_new_eq:NN \seq_gpush:Nn \seq_gput_left:Nn</code>
<code>\seq_gpush:NV</code>	5442	<code>\cs_new_eq:NN \seq_gpush:Nv \seq_gput_left:Nv</code>
<code>\seq_gpush:Nv</code>	5443	<code>\cs_new_eq:NN \seq_gpush:No \seq_gput_left:No</code>
<code>\seq_gpush:No</code>	5444	<code>\cs_new_eq:NN \seq_gpush:Nx \seq_gput_left:Nx</code>
<code>\seq_gpush:Nx</code>	5445	<code>\cs_new_eq:NN \seq_gpush:cn \seq_gput_left:cn</code>
<code>\seq_gpush:cn</code>	5446	<code>\cs_new_eq:NN \seq_gpush:cV \seq_gput_left:cV</code>
<code>\seq_gpush:cV</code>	5447	<code>\cs_new_eq:NN \seq_gpush:cV \seq_gput_left:cv</code>
<code>\seq_gpush:cv</code>	5448	<code>\cs_new_eq:NN \seq_gpush:co \seq_gput_left:co</code>
<code>\seq_gpush:co</code>	5449	<code>\cs_new_eq:NN \seq_gpush:cx \seq_gput_left:cx</code>
<code>\seq_gpush:cx</code>	5450	

(End definition for `\seq_push:Nn` and others. These functions are documented on page ??.)

`\seq_get:NN` In most cases, getting items from the stack does not need to specify that this is from the left. So alias are provided.

<code>\seq_get:cN</code>	
<code>\seq_pop:NN</code>	5451 <code>\cs_new_eq:NN \seq_get:NN \seq_get_left:NN</code>
<code>\seq_pop:cN</code>	5452 <code>\cs_new_eq:NN \seq_get:cN \seq_get_left:cN</code>
<code>\seq_gpop:NN</code>	5453 <code>\cs_new_eq:NN \seq_pop:NN \seq_pop_left:NN</code>
<code>\seq_gpop:cN</code>	5454 <code>\cs_new_eq:NN \seq_pop:cN \seq_pop_left:cN</code>
	5455 <code>\cs_new_eq:NN \seq_gpop:NN \seq_gpop_left:NN</code>
	5456 <code>\cs_new_eq:NN \seq_gpop:cN \seq_gpop_left:cN</code>

(End definition for `\seq_get:NN` and `\seq_get:cN`. These functions are documented on page ??.)

193.8 Viewing sequences

`\seq_show:N` Apply the general `\msg_aux_show:Nnx`.

<code>\seq_show:c</code>	5457	<code>\cs_new_protected:Npn \seq_show:N #1</code>
	5458	<code>{</code>
	5459	<code>\msg_aux_show:Nnx</code>
	5460	<code>#1</code>
	5461	<code>{ seq }</code>

```

5462         { \seq_map_function:NN #1 \msg_aux_show:n }
5463     }
5464 \cs_generate_variant:Nn \seq_show:N { c }
      (End definition for \seq_show:N and \seq_show:c. These functions are documented on page ??.)

```

193.9 Experimental functions

`\seq_if_empty_break_return_false:N` The name says it all: of the sequence is empty, returns logical false.

```

5465 \cs_new:Npn \seq_if_empty_break_return_false:N #1
5466 {
5467     \if_meaning:w #1 \c_empty_tl
5468     \prg_return_false:
5469     \exp_after:wN \seq_break:
5470     \fi:
5471 }
      (End definition for \seq_if_empty_break_return_false:N. This function is documented on page ??.)

```

`\seq_get_left:NN` Getting from the left or right with a check on the results.

```

\seq_get_left:cN 5472 \prg_new_protected_conditional:Npnn \seq_get_left:NN #1 #2 { T , F , TF }
\seq_get_right:NN 5473 {
\seq_get_right:cN 5474     \seq_if_empty_break_return_false:N #1
5475     \exp_after:wN \seq_get_left_aux:Nw #1 \q_stop #2
5476     \prg_return_true:
5477     \seq_break:
5478     \prg_break_point:n { }
5479 }
5480 \prg_new_protected_conditional:Npnn \seq_get_right:NN #1#2 { T , F , TF }
5481 {
5482     \seq_if_empty_break_return_false:N #1
5483     \seq_get_right_aux:NN #1#2
5484     \prg_return_true: \seq_break:
5485     \prg_break_point:n { }
5486 }
5487 \cs_generate_variant:Nn \seq_get_left:NN { c }
5488 \cs_generate_variant:Nn \seq_get_left:NNF { c }
5489 \cs_generate_variant:Nn \seq_get_left:NNTF { c }
5490 \cs_generate_variant:Nn \seq_get_right:NN { c }
5491 \cs_generate_variant:Nn \seq_get_right:NNF { c }
5492 \cs_generate_variant:Nn \seq_get_right:NNTF { c }

```

(End definition for `\seq_get_left:NN` and `\seq_get_left:cN`. These functions are documented on page ??.)

`\seq_pop_left:NN` More or less the same for popping.

```

\seq_pop_left:cN 5493 \prg_new_protected_conditional:Npnn \seq_pop_left:NN #1#2 { T , F , TF }
\seq_gpop_left:NN 5494 {
\seq_gpop_left:cN 5495     \seq_if_empty_break_return_false:N #1
\seq_pop_right:NN 5496     \exp_after:wN \seq_pop_left_aux:NwNNN #1 \q_stop \tl_set:Nn #1#2
\seq_pop_right:cN 5497     \prg_return_true: \seq_break:
\seq_gpop_right:NN
\seq_gpop_right:cN

```

```

5498     \prg_break_point:n { }
5499   }
5500 \prg_new_protected_conditional:Npnn \seq_gpop_left:NN #1#2 { T , F , TF }
5501 {
5502   \seq_if_empty_break_return_false:N #1
5503   \exp_after:wN \seq_pop_left_aux:NnwNNN #1 \q_stop \tl_gset:Nn #1#2
5504   \prg_return_true: \seq_break:
5505   \prg_break_point:n { }
5506 }
5507 \prg_new_protected_conditional:Npnn \seq_pop_right:NN #1#2 { T , F , TF }
5508 {
5509   \seq_if_empty_break_return_false:N #1
5510   \seq_pop_right_aux_ii:NNN \tl_set:Nx #1 #2
5511   \prg_return_true: \seq_break:
5512   \prg_break_point:n { }
5513 }
5514 \prg_new_protected_conditional:Npnn \seq_gpop_right:NN #1#2 { T , F , TF }
5515 {
5516   \seq_if_empty_break_return_false:N #1
5517   \seq_pop_right_aux_ii:NNN \tl_gset:Nx #1 #2
5518   \prg_return_true: \seq_break:
5519   \prg_break_point:n { }
5520 }
5521 \cs_generate_variant:Nn \seq_pop_left:NNT { c }
5522 \cs_generate_variant:Nn \seq_pop_left:NNF { c }
5523 \cs_generate_variant:Nn \seq_pop_left:NNTF { c }
5524 \cs_generate_variant:Nn \seq_gpop_left:NNT { c }
5525 \cs_generate_variant:Nn \seq_gpop_left:NNF { c }
5526 \cs_generate_variant:Nn \seq_gpop_left:NNTF { c }
5527 \cs_generate_variant:Nn \seq_pop_right:NNT { c }
5528 \cs_generate_variant:Nn \seq_pop_right:NNF { c }
5529 \cs_generate_variant:Nn \seq_pop_right:NNTF { c }
5530 \cs_generate_variant:Nn \seq_gpop_right:NNT { c }
5531 \cs_generate_variant:Nn \seq_gpop_right:NNF { c }
5532 \cs_generate_variant:Nn \seq_gpop_right:NNTF { c }

```

(End definition for `\seq_pop_left:NN` and `\seq_pop_left:cN`. These functions are documented on page ??.)

`\seq_length:N` Counting the items in a sequence is done using the same approach as for other length functions: turn each entry into a +1 then use integer evaluation to actually do the mathematics.

`\seq_length:c`
`\seq_length_aux:n`

```

5533 \cs_new:Npn \seq_length:N #1
5534 {
5535   \int_eval:n
5536   {
5537     0
5538     \seq_map_function:NN #1 \seq_length_aux:n
5539   }
5540 }

```

```

5541 \cs_new:Npn \seq_length_aux:n #1 { +1 }
5542 \cs_generate_variant:Nn \seq_length:N { c }
      (End definition for \seq_length:N and \seq_length:c. These functions are documented on page
??.)

```

`\seq_item:Nn` The idea here is to find the offset of the item from the left, then use a loop to grab the correct item. If the resulting offset is too large, then the stop code `{ ? \seq_break: } { }`
`\seq_item:cn` will be used by the auxiliary, terminating the loop and returning nothing at all.
`\seq_item_aux:nnn`

```

5543 \cs_new:Npn \seq_item:Nn #1#2
5544 {
5545   \exp_last_unbraced:Nfo \seq_item_aux:nnn
5546   {
5547     \int_eval:n
5548     {
5549       \int_compare:nNnT {#2} < \c_zero
5550       { \seq_length:N #1 + }
5551       #2
5552     }
5553   }
5554   #1
5555   { ? \seq_break: }
5556   { }
5557   \prg_break_point:n { }
5558 }
5559 \cs_new:Npn \seq_item_aux:nnn #1#2#3
5560 {
5561   \use_none:n #2
5562   \int_compare:nNnTF {#1} = \c_zero
5563   { \seq_break:n {#3} }
5564   { \exp_args:Nf \seq_item_aux:nnn { \int_eval:n { #1 - 1 } } }
5565 }
5566 \cs_generate_variant:Nn \seq_item:Nn { c }
      (End definition for \seq_item:Nn and \seq_item:cn. These functions are documented on page
??.)

```

`\seq_use:N` A simple short cut for a mapping.

```

\seq_use:c      5567 \cs_new:Npn \seq_use:N #1 { \seq_map_function:NN #1 \use:n }
                5568 \cs_generate_variant:Nn \seq_use:N { c }
                (End definition for \seq_use:N and \seq_use:c. These functions are documented on page ??.)

```

`\seq_mapthread_function:NNN` The idea here is to first expand both of the sequences, adding the usual `{ ? \seq_break: } { }`
`\seq_mapthread_function:NcN` to the end of each on. This is most conveniently done in two steps using an auxiliary
`\seq_mapthread_function:cNN` function. The mapping then throws away the first token of #2 and #5, which for items
`\seq_mapthread_function:ccN` in the sequences will both be `\seq_item:n`. The function to be mapped will then be
`\seq_mapthread_function_aux:NN` applied to the two entries. When the code hits the end of one of the sequences, the break
`\seq_mapthread_function_aux:Nnnwnn` material will stop the entire loop and tidy up. This avoids needing to find the length of
the two sequences, or worrying about which is longer.

```

5569 \cs_new:Npn \seq_mapthread_function:NNN #1#2#3

```

```

5570 {
5571   \exp_after:wN \seq_mapthread_function_aux:NN
5572   \exp_after:wN #3
5573   \exp_after:wN #1
5574   #2
5575   { ? \seq_break: } { }
5576   \prg_break_point:n { }
5577 }
5578 \cs_new:Npn \seq_mapthread_function_aux:NN #1#2
5579 {
5580   \exp_after:wN \seq_mapthread_function_aux:Nnnwnn
5581   \exp_after:wN #1
5582   #2
5583   { ? \seq_break: } { }
5584   \q_stop
5585 }
5586 \cs_new:Npn \seq_mapthread_function_aux:Nnnwnn #1#2#3#4 \q_stop #5#6
5587 {
5588   \use_none:n #2
5589   \use_none:n #5
5590   #1 {#3} {#6}
5591   \seq_mapthread_function_aux:Nnnwnn #1 #4 \q_stop
5592 }
5593 \cs_generate_variant:Nn \seq_mapthread_function:NNN { Nc }
5594 \cs_generate_variant:Nn \seq_mapthread_function:NNN { c , cc }

```

(End definition for \seq_mapthread_function:NNN and others. These functions are documented on page ??.)

\seq_set_from_clist:NN Setting a sequence from a comma-separated list is done using a simple mapping.

```

\seq_set_from_clist:cN 5595 \cs_new_protected:Npn \seq_set_from_clist:NN #1#2
\seq_set_from_clist:Nc 5596 {
\seq_set_from_clist:cc 5597   \tl_set:Nx #1
\seq_set_from_clist:Nn 5598   { \clist_map_function:NN #2 \seq_wrap_item:n }
\seq_set_from_clist:cn 5599 }
\seq_gset_from_clist:NN 5600 \cs_new_protected:Npn \seq_gset_from_clist:Nn #1#2
\seq_gset_from_clist:cN 5601 {
\seq_gset_from_clist:Nc 5602   \tl_set:Nx #1
\seq_gset_from_clist:cc 5603   { \clist_map_function:nN {#2} \seq_wrap_item:n }
\seq_gset_from_clist:Nn 5604 }
\seq_gset_from_clist:cn 5605 \cs_new_protected:Npn \seq_gset_from_clist:NN #1#2
5606 {
5607   \tl_gset:Nx #1
5608   { \clist_map_function:NN #2 \seq_wrap_item:n }
5609 }
5610 \cs_new_protected:Npn \seq_gset_from_clist:Nn #1#2
5611 {
5612   \tl_gset:Nx #1
5613   { \clist_map_function:nN {#2} \seq_wrap_item:n }
5614 }
5615 \cs_generate_variant:Nn \seq_set_from_clist:NN { Nc }

```

```

5616 \cs_generate_variant:Nn \seq_set_from_clist:NN { c , cc }
5617 \cs_generate_variant:Nn \seq_set_from_clist:Nn { c      }
5618 \cs_generate_variant:Nn \seq_gset_from_clist:NN {      Nc }
5619 \cs_generate_variant:Nn \seq_gset_from_clist:NN { c , cc }
5620 \cs_generate_variant:Nn \seq_gset_from_clist:Nn { c      }

```

(End definition for `\seq_set_from_clist:NN` and others. These functions are documented on page ??.)

```

\seq_reverse:N Previously, \seq_reverse:N was coded by collecting the items in reverse order after an
\seq_reverse:c \exp_stop_f: marker.
\seq_greverse:N
\seq_greverse:c
\seq_reverse_aux:NN
\seq_reverse_aux_item:nwn
\cs_new_protected:Npn \seq_reverse:N #1
{
  \cs_set_eq:NN \seq_item:n \seq_reverse_aux_item:nw
  \tl_set:Nf #2 { #2 \exp_stop_f: }
}
\cs_new:Npn \seq_reverse_aux_item:nw #1 #2 \exp_stop_f:
{
  #2 \exp_stop_f:
  \seq_item:n {#1}
}

```

At first, this seems optimal, since we can forget about each item as soon as it is placed after `\exp_stop_f:`. Unfortunately, \TeX 's usual tail recursion does not take place in this case: since the following `\seq_reverse_aux_item:nw` only reads tokens until `\exp_stop_f:`, and never reads the `\seq_item:n {#1}` left by the previous call, \TeX cannot remove that previous call from the stack, and in particular must retain the various macro parameters in memory, until the end of the replacement text is reached. The stack is thus only flushed after all the `\seq_reverse_aux_item:nw` are expanded. Keeping track of the arguments of all those calls uses up a memory quadratic in the length of the sequence. \TeX can then not cope with more than a few thousand items.

Instead, we collect the items in the argument of `\exp_not:n`. The previous calls are cleanly removed from the stack, and the memory consumption becomes linear.

```

5621 \cs_new_protected_nopar:Npn \seq_tmp:w { }
5622 \cs_new_protected_nopar:Npn \seq_reverse:N
5623 { \seq_reverse_aux:NN \tl_set:Nx }
5624 \cs_new_protected_nopar:Npn \seq_greverse:N
5625 { \seq_reverse_aux:NN \tl_gset:Nx }
5626 \cs_new_protected:Npn \seq_reverse_aux:NN #1 #2
5627 {
5628   \cs_set_eq:NN \seq_tmp:w \seq_item:n
5629   \cs_set_eq:NN \seq_item:n \seq_reverse_aux_item:nwn
5630   #1 #2 { #2 \exp_not:n { } }
5631   \cs_set_eq:NN \seq_item:n \seq_tmp:w
5632 }
5633 \cs_new:Npn \seq_reverse_aux_item:nwn #1 #2 \exp_not:n #3
5634 {
5635   #2

```

```

5636 \exp_not:n { \seq_item:n {#1} #3 }
5637 }

```

```

5638 \cs_generate_variant:Nn \seq_reverse:N { c }
5639 \cs_generate_variant:Nn \seq_greverse:N { c }

```

(End definition for \seq_reverse:N and others. These functions are documented on page ??.)

`\seq_set_filter:NNn` Similar to `\seq_map_inline:Nn`, without a `\prg_break_point:n` because the user's code is performed within the evaluation of a boolean expression, and skipping out of that would break horribly. The `\seq_wrap_item:n` function inserts the relevant `\seq_item:n` without expansion in the input stream, hence in the x-expanding assignment.

```

5640 \cs_new_protected_nopar:Npn \seq_set_filter:NNn
5641 { \seq_set_filter_aux:NNNn \tl_set:Nx }
5642 \cs_new_protected_nopar:Npn \seq_gset_filter:NNn
5643 { \seq_set_filter_aux:NNNn \tl_gset:Nx }
5644 \cs_new_protected:Npn \seq_set_filter_aux:NNNn #1#2#3#4
5645 {
5646   \seq_push_item_def:n { \bool_if:nT {#4} { \seq_wrap_item:n {##1} } }
5647   #1 #2 { #3 \prg_break_point:n { } }
5648   \seq_pop_item_def:
5649 }

```

(End definition for \seq_set_filter:NNn and \seq_gset_filter:NNn. These functions are documented on page ??.)

`\seq_set_map:NNn` Very similar to `\seq_set_filter:NNn`. We could actually merge the two within a single function, but it would have weird semantics.

```

\seq_gset_map:NNn
\seq_set_map_aux:NNNn
5650 \cs_new_protected_nopar:Npn \seq_set_map:NNn
5651 { \seq_set_map_aux:NNNn \tl_set:Nx }
5652 \cs_new_protected_nopar:Npn \seq_gset_map:NNn
5653 { \seq_set_map_aux:NNNn \tl_gset:Nx }
5654 \cs_new_protected:Npn \seq_set_map_aux:NNNn #1#2#3#4
5655 {
5656   \seq_push_item_def:n { \exp_not:N \seq_item:n {#4} }
5657   #1 #2 { #3 }
5658   \seq_pop_item_def:
5659 }

```

(End definition for \seq_set_map:NNn and \seq_gset_map:NNn. These functions are documented on page ??.)

193.10 Deprecated interfaces

A few functions which are no longer documented: these were moved here on or before 2011-04-20, and will be removed entirely by 2011-07-20.

`\seq_top:NN` These are old stack functions.

```

\seq_top:cN
5660 {*deprecated}
5661 \cs_new_eq:NN \seq_top:NN \seq_get_left:NN
5662 \cs_new_eq:NN \seq_top:cN \seq_get_left:cN
5663 {/deprecated}

```

(End definition for \seq_top:NN and \seq_top:cN. These functions are documented on page ??.)

\seq_display:N An older name for \seq_show:N.

\seq_display:c 5664 <*deprecated>
5665 \cs_new_eq:NN \seq_display:N \seq_show:N
5666 \cs_new_eq:NN \seq_display:c \seq_show:c
5667 </deprecated>

(End definition for \seq_display:N and \seq_display:c. These functions are documented on page ??.)

5668 </initex | package>

194 l3clist implementation

The following test files are used for this code: m3clist002.

5669 <*initex | package>
5670 <*package>
5671 \ProvidesExplPackage
5672 {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
5673 \package_check_loaded_expl:
5674 </package>

\l_clist_tmpa_clist Scratch space for various internal uses. This comma list variable cannot be declared as such because it comes before \clist_new:N

5675 \tl_new:N \l_clist_tmpa_clist
(End definition for \l_clist_tmpa_clist. This function is documented on page ??.)

\clist_tmp:w A temporary function for various purposes.

5676 \cs_new_protected:Npn \clist_tmp:w { }
(End definition for \clist_tmp:w. This function is documented on page ??.)

194.1 Allocation and initialisation

\clist_new:N Internally, comma lists are just token lists.

\clist_new:c 5677 \cs_new_eq:NN \clist_new:N \tl_new:N
5678 \cs_new_eq:NN \clist_new:c \tl_new:c
(End definition for \clist_new:N and \clist_new:c. These functions are documented on page ??.)

\clist_clear:N Clearing comma lists is just the same as clearing token lists.

\clist_clear:c 5679 \cs_new_eq:NN \clist_clear:N \tl_clear:N
\clist_gclear:N 5680 \cs_new_eq:NN \clist_clear:c \tl_clear:c
5681 \cs_new_eq:NN \clist_gclear:N \tl_gclear:N
5682 \cs_new_eq:NN \clist_gclear:c \tl_gclear:c
(End definition for \clist_clear:N and \clist_clear:c. These functions are documented on page ??.)

`\clist_clear_new:N` Once again a copy from the token list functions.

```

\clist_clear_new:c 5683 \cs_new_eq:NN \clist_clear_new:N \tl_clear_new:N
\clist_gclear_new:N 5684 \cs_new_eq:NN \clist_clear_new:c \tl_clear_new:c
\clist_gclear_new:c 5685 \cs_new_eq:NN \clist_gclear_new:N \tl_gclear_new:N
5686 \cs_new_eq:NN \clist_gclear_new:c \tl_gclear_new:c

```

(End definition for \clist_clear_new:N and \clist_clear_new:c. These functions are documented on page ??.)

`\clist_set_eq:NN` Once again, these are simple copies from the token list functions.

```

\clist_set_eq:cN 5687 \cs_new_eq:NN \clist_set_eq:NN \tl_set_eq:NN
\clist_set_eq:Nc 5688 \cs_new_eq:NN \clist_set_eq:Nc \tl_set_eq:Nc
\clist_set_eq:cc 5689 \cs_new_eq:NN \clist_set_eq:cN \tl_set_eq:cN
\clist_gset_eq:NN 5690 \cs_new_eq:NN \clist_set_eq:cc \tl_set_eq:cc
\clist_gset_eq:cN 5691 \cs_new_eq:NN \clist_gset_eq:NN \tl_gset_eq:NN
\clist_gset_eq:Nc 5692 \cs_new_eq:NN \clist_gset_eq:Nc \tl_gset_eq:Nc
\clist_gset_eq:cN 5693 \cs_new_eq:NN \clist_gset_eq:cN \tl_gset_eq:cN
\clist_gset_eq:cc 5694 \cs_new_eq:NN \clist_gset_eq:cc \tl_gset_eq:cc

```

(End definition for \clist_set_eq:NN and others. These functions are documented on page ??.)

`\clist_concat:NNN` Concatenating sequences is not quite as easy as it seems, as there needs to be the correct addition of a comma to the output. So a little work to do.

```

\clist_concat:ccc 5695 \cs_new_protected_nopar:Npn \clist_concat:NNN
\clist_gconcat:NNN 5696 { \clist_concat_aux:NNNN \tl_set:Nx }
\clist_concat_aux:NNNN 5697 \cs_new_protected_nopar:Npn \clist_gconcat:NNN
5698 { \clist_concat_aux:NNNN \tl_gset:Nx }
5699 \cs_new_protected:Npn \clist_concat_aux:NNNN #1#2#3#4
5700 {
5701   #1 #2
5702   {
5703     \exp_not:o #3
5704     \clist_if_empty:NF #3 { \clist_if_empty:NF #4 { , } }
5705     \exp_not:o #4
5706   }
5707 }
5708 \cs_generate_variant:Nn \clist_concat:NNN { ccc }
5709 \cs_generate_variant:Nn \clist_gconcat:NNN { ccc }

```

(End definition for \clist_concat:NNN and \clist_concat:ccc. These functions are documented on page ??.)

194.2 Removing spaces around items

`\clist_trim_spaces_generic:nw` Used as ‘`\clist_trim_spaces_generic:nw {<code>} \q_mark <item> ,`’ (including the comma). This expands to the `<code>`, followed by a brace group containing the `<item>`, with leading and trailing spaces removed. The calling function is responsible for inserting `\q_mark` in front of the `<item>`, as well as testing for the end of the list. See `\tl_trim_spaces:n` for a partial explanation of what is happening here. We changed `\tl_trim_spaces_aux_iv:w` into `\clist_trim_spaces_generic_aux:w` compared to `\tl_trim_spaces:n`, and dropped a `\q_mark`, which is already included in the argument `##2`.

```

5710 \cs_set:Npn \clist_tmp:w #1
5711 {
5712   \cs_new:Npn \clist_trim_spaces_generic:nw ##1 ##2 ,
5713   {
5714     \tl_trim_spaces_aux_i:w
5715     ##2
5716     \q_nil
5717     \q_mark #1 { }
5718     \q_mark \tl_trim_spaces_aux_ii:w
5719     \tl_trim_spaces_aux_iii:w
5720     #1 \q_nil
5721     \clist_trim_spaces_generic_aux:w
5722     \q_stop
5723     {##1}
5724   }
5725 }
5726 \clist_tmp:w {~}
5727 \cs_new:Npn \clist_trim_spaces_generic_aux:w #1 \q_nil #2 \q_stop
5728 { \exp_args:No \clist_trim_spaces_generic_aux_ii:nn { \use_none:n #1 } }
5729 \cs_new:Npn \clist_trim_spaces_generic_aux_ii:nn #1 #2 { #2 {#1} }

```

(End definition for \clist_trim_spaces_generic:nw. This function is documented on page ??.)

`\clist_trim_spaces:n` The first argument of `\clist_trim_spaces_aux:nn` is initially empty, and later a comma, namely, as soon as we have added an item to the resulting list. The auxiliary tests for the end of the list, and also prevents empty arguments from finding their way into the output.

```

5730 \cs_new:Npn \clist_trim_spaces:n #1
5731 {
5732   \clist_trim_spaces_generic:nw
5733   { \clist_trim_spaces_aux:nn { } }
5734   \q_mark #1 ,
5735   \q_recursion_tail, \q_recursion_stop
5736 }
5737 \cs_new:Npn \clist_trim_spaces_aux:nn #1 #2
5738 {
5739   \quark_if_recursion_tail_stop:n {#2}
5740   \tl_if_empty:nTF {#2}
5741   {
5742     \clist_trim_spaces_generic:nw
5743     { \clist_trim_spaces_aux:nn {#1} } \q_mark
5744   }
5745   {
5746     #1 \exp_not:n {#2}
5747     \clist_trim_spaces_generic:nw
5748     { \clist_trim_spaces_aux:nn { , } } \q_mark
5749   }
5750 }

```

(End definition for \clist_trim_spaces:n. This function is documented on page ??.)

194.3 Adding data to comma lists

```

\clist_set:Nn
\clist_set:NV
\clist_set:No
\clist_set:Nx
\clist_set:cn
\clist_set:cV
\clist_set:co
\clist_set:cx
\clist_gset:Nn
\clist_put_left:Nn
\clist_gset:NV
\clist_put_left:NV
\clist_gset:No
\clist_put_left:No
\clist_gset:Nx
\clist_put_left:Nx
\clist_gset:cn
\clist_put_left:cn
\clist_gset:cV
\clist_put_left:cV
\clist_gset:co
\clist_put_left:co
\clist_gset:cx
\clist_put_left:cx
\clist_gput_left:Nn
\clist_gput_left:NV
\clist_gput_left:No
\clist_gput_left:Nx
\clist_gput_left:cn
\clist_gput_left:cV
\clist_gput_left:co
\clist_gput_left:cx
\clist_put_right:Nn
\clist_put_right:NV
\clist_put_right:No
\clist_put_right:Nx
\clist_put_right:cn
\clist_put_right:cV
\clist_put_right:co
\clist_put_right:cx
\clist_gput_right:Nn
\clist_gput_right:NV
\clist_gput_right:No
\clist_gput_right:Nx
\clist_gput_right:cn
\clist_gput_right:cV
\clist_gput_right:co
\clist_gput_right:cx
\clist_put_right_aux:NNNn
\clist_gput_right_aux:NNNn

```

5751 \cs_new_protected:Npn \clist_set:Nn #1#2
5752 { \tl_set:Nx #1 { \clist_trim_spaces:n {#2} } }
5753 \cs_new_protected:Npn \clist_gset:Nn #1#2
5754 { \tl_gset:Nx #1 { \clist_trim_spaces:n {#2} } }
5755 \cs_generate_variant:Nn \clist_set:Nn { NV , No , Nx , c , cV , co , cx }
5756 \cs_generate_variant:Nn \clist_gset:Nn { NV , No , Nx , c , cV , co , cx }
(End definition for \clist_set:Nn and others. These functions are documented on page ??.)

Comma lists cannot hold empty values: there are therefore a couple of sanity checks to avoid accumulating commas.

5757 \cs_new_protected_nopar:Npn \clist_put_left:Nn
5758 { \clist_put_left_aux:NNNn \clist_concat:NNN \clist_set:Nn }
5759 \cs_new_protected_nopar:Npn \clist_gput_left:Nn
5760 { \clist_put_left_aux:NNNn \clist_gconcat:NNN \clist_set:Nn }
5761 \cs_new_protected:Npn \clist_put_left_aux:NNNn #1#2#3#4
5762 {
5763 #2 \l_clist_tmpa_clist {#4}
5764 #1 #3 \l_clist_tmpa_clist #3
5765 }
5766 \cs_generate_variant:Nn \clist_put_left:Nn { NV , No , Nx }
5767 \cs_generate_variant:Nn \clist_put_left:Nn { c , cV , co , cx }
5768 \cs_generate_variant:Nn \clist_gput_left:Nn { NV , No , Nx }
5769 \cs_generate_variant:Nn \clist_gput_left:Nn { c , cV , co , cx }
(End definition for \clist_put_left:Nn and others. These functions are documented on page ??.)

5770 \cs_new_protected_nopar:Npn \clist_put_right:Nn
5771 { \clist_put_right_aux:NNNn \clist_concat:NNN \clist_set:Nn }
5772 \cs_new_protected_nopar:Npn \clist_gput_right:Nn
5773 { \clist_put_right_aux:NNNn \clist_gconcat:NNN \clist_gset:Nn }
5774 \cs_new_protected:Npn \clist_put_right_aux:NNNn #1#2#3#4
5775 {
5776 #2 \l_clist_tmpa_clist {#4}
5777 #1 #3 #3 \l_clist_tmpa_clist
5778 }
5779 \cs_generate_variant:Nn \clist_put_right:Nn { NV , No , Nx }
5780 \cs_generate_variant:Nn \clist_put_right:Nn { c , cV , co , cx }
5781 \cs_generate_variant:Nn \clist_gput_right:Nn { NV , No , Nx }
5782 \cs_generate_variant:Nn \clist_gput_right:Nn { c , cV , co , cx }
(End definition for \clist_put_right:Nn and others. These functions are documented on page ??.)

194.4 Comma lists as stacks

`\clist_get:NN` Getting an item from the left of a comma list is pretty easy: just trim off the first item
`\clist_get:cN` using the comma.
`\clist_get_aux:wN`

```
5783 \cs_new_protected:Npn \clist_get:NN #1#2
5784 { \exp_after:wN \clist_get_aux:wN #1 , \q_stop #2 }
5785 \cs_new_protected:Npn \clist_get_aux:wN #1 , #2 \q_stop #3
5786 { \tl_set:Nn #3 {#1} }
5787 \cs_generate_variant:Nn \clist_get:NN { c }
```

(End definition for `\clist_get:NN` and `\clist_get:cN`. These functions are documented on page ??.)

`\clist_pop:NN` The aim here is to get the popped item as #1 in the auxiliary, with #2 containing either
`\clist_pop:cN` the remainder of the list or `\q_nil` if there were insufficient items. That keeps the
`\clist_gpop:NN` number of auxiliary functions down.
`\clist_gpop:cN`

```
5788 \cs_new_protected_nopar:Npn \clist_pop:NN
5789 { \clist_pop_aux:NNN \tl_set:Nf }
5790 \cs_new_protected_nopar:Npn \clist_gpop:NN
5791 { \clist_pop_aux:NNN \tl_gset:Nf }
5792 \cs_new_protected:Npn \clist_pop_aux:NNN #1#2#3
5793 {
5794   \exp_after:wN \clist_pop_aux:wNNN #2 , \q_nil \q_stop #1#2#3
5795 }
5796 \cs_new_protected:Npn \clist_pop_aux:wNNN #1 , #2 \q_stop #3#4#5
5797 {
5798   \tl_set:Nn #5 {#1}
5799   \quark_if_nil:nTF {#2}
5800   { #3 #4 { } }
5801   { #3 #4 { \clist_pop_aux:w \exp_stop_f: #2 } }
5802 }
5803 \cs_new_protected:Npn \clist_pop_aux:w #1 , \q_nil {#1}
5804 \cs_generate_variant:Nn \clist_pop:NN { c }
5805 \cs_generate_variant:Nn \clist_gpop:NN { c }
```

(End definition for `\clist_pop:NN` and `\clist_pop:cN`. These functions are documented on page ??.)

`\clist_push:Nn` Pushing to a sequence is the same as adding on the left.

```
5806 \cs_new_eq:NN \clist_push:Nn \clist_put_left:Nn
5807 \cs_new_eq:NN \clist_push:NV \clist_put_left:NV
5808 \cs_new_eq:NN \clist_push:No \clist_put_left:No
5809 \cs_new_eq:NN \clist_push:Nx \clist_put_left:Nx
5810 \cs_new_eq:NN \clist_push:cn \clist_put_left:cn
5811 \cs_new_eq:NN \clist_push:cV \clist_put_left:cV
5812 \cs_new_eq:NN \clist_push:co \clist_put_left:co
5813 \cs_new_eq:NN \clist_push:cx \clist_put_left:cx
5814 \cs_new_eq:NN \clist_gpush:Nn \clist_gput_left:Nn
5815 \cs_new_eq:NN \clist_gpush:NV \clist_gput_left:NV
5816 \cs_new_eq:NN \clist_gpush:No \clist_gput_left:No
5817 \cs_new_eq:NN \clist_gpush:Nx \clist_gput_left:Nx
```

```
\clist_gpush:cn
\clist_gpush:cV
\clist_gpush:co
\clist_gpush:cx
```

```

5818 \cs_new_eq:NN \clist_gpush:cn \clist_gput_left:cn
5819 \cs_new_eq:NN \clist_gpush:cV \clist_gput_left:cV
5820 \cs_new_eq:NN \clist_gpush:co \clist_gput_left:co
5821 \cs_new_eq:NN \clist_gpush:cx \clist_gput_left:cx

```

(End definition for \clist_push:Nn and others. These functions are documented on page ??.)

194.5 Using comma lists

\clist_use:N The approach is the same as for \tl_use:N.

```

\clist_use:c 5822 \cs_new_eq:NN \clist_use:N \tl_use:N
5823 \cs_new_eq:NN \clist_use:c \tl_use:c

```

(End definition for \clist_use:N and \clist_use:c. These functions are documented on page ??.)

194.6 Modifying comma lists

\l_clist_remove_clist An internal comma list for the removal routines.

```

5824 \clist_new:N \l_clist_remove_clist

```

(End definition for \l_clist_remove_clist. This function is documented on page ??.)

\clist_remove_duplicates:N Removing duplicates means making a new list then copying it.

```

\clist_remove_duplicates:c 5825 \cs_new_protected:Npn \clist_remove_duplicates:N
\clist_gremove_duplicates:N 5826 { \clist_remove_duplicates_aux:NN \clist_set_eq:NN }
\clist_gremove_duplicates:c 5827 \cs_new_protected:Npn \clist_gremove_duplicates:N
\clist_remove_duplicates_aux:NN 5828 { \clist_remove_duplicates_aux:NN \clist_gset_eq:NN }
5829 \cs_new_protected:Npn \clist_remove_duplicates_aux:NN #1#2
5830 {
5831   \clist_clear:N \l_clist_remove_clist
5832   \clist_map_inline:Nn #2
5833   {
5834     \clist_if_in:NnF \l_clist_remove_clist {##1}
5835     { \clist_put_right:Nn \l_clist_remove_clist {##1} }
5836   }
5837   #1 #2 \l_clist_remove_clist
5838 }
5839 \cs_generate_variant:Nn \clist_remove_duplicates:N { c }
5840 \cs_generate_variant:Nn \clist_gremove_duplicates:N { c }

```

(End definition for \clist_remove_duplicates:N and \clist_remove_duplicates:c. These functions are documented on page ??.)

\clist_remove_all:Nn The method used here is very similar to \tl_replace_all:Nnn. Build a function delimited by the *<item>* that should be removed, surrounded with commas, and call that function followed by the expanded comma list, and another copy of the *<item>*. The loop is controlled by the argument grabbed by \clist_remove_all_aux:w: when the item was found, the \q_mark delimiter used is the one inserted by \clist_tmp:w, and \use_none_delimit_by_q_stop:w is deleted. At the end, the final *<item>* is grabbed, and the argument of \clist_tmp:w contains \q_mark: in that case, \clist_remove_all_aux:w

removes the second `\q_mark` (inserted by `\clist_tmp:w`), and lets `\use_none_delimit_by_q_stop:w` act.

No brace is lost because items are always grabbed with a leading comma. The result of the first assignment has an extra leading comma, which we remove in a second assignment. Two exceptions: if the clist lost all of its elements, the result is empty, and we shouldn't remove anything; if the clist started up empty, the first step happens to turn it into a single comma, and the second step removes it.

```

5841 \cs_new_protected:Npn \clist_remove_all:Nn
5842 { \clist_remove_all_aux:NNn \tl_set:Nx }
5843 \cs_new_protected:Npn \clist_gremove_all:Nn
5844 { \clist_remove_all_aux:NNn \tl_gset:Nx }
5845 \cs_new_protected:Npn \clist_remove_all_aux:NNn #1#2#3
5846 {
5847   \cs_set:Npn \clist_tmp:w ##1 , #3 ,
5848   {
5849     ##1
5850     , \q_mark , \use_none_delimit_by_q_stop:w ,
5851     \clist_remove_all_aux:
5852   }
5853   #1 #2
5854   {
5855     \exp_after:wN \clist_remove_all_aux:
5856     #2 , \q_mark , #3 , \q_stop
5857   }
5858   \clist_if_empty:NF #2
5859   {
5860     #1 #2
5861     {
5862       \exp_args:No \exp_not:o
5863       { \exp_after:wN \use_none:n #2 }
5864     }
5865   }
5866 }
5867 \cs_new:Npn \clist_remove_all_aux:
5868 { \exp_after:wN \clist_remove_all_aux:w \clist_tmp:w , }
5869 \cs_new:Npn \clist_remove_all_aux:w #1 , \q_mark , #2 , { \exp_not:n {#1} }
5870 \cs_generate_variant:Nn \clist_remove_all:Nn { c }
5871 \cs_generate_variant:Nn \clist_gremove_all:Nn { c }

```

(End definition for `\clist_remove_all:Nn` and `\clist_remove_all:cn`. These functions are documented on page ??.)

194.7 Comma list conditionals

`\l_clist_if_in_clist` An internal comma list for `\clist_if_in:nn` conditionals.

```

5872 \clist_new:N \l_clist_if_in_clist

```

(End definition for `\l_clist_if_in_clist`. This function is documented on page ??.)

`\clist_if_empty:N` Simple copies from the token list variable material.
`\clist_if_empty:c`

```

5873 \prg_new_eq_conditional:NNn \clist_if_empty:N \tl_if_empty:N { p , T , F , TF }
5874 \prg_new_eq_conditional:NNn \clist_if_empty:c \tl_if_empty:c { p , T , F , TF }
    (End definition for \clist_if_empty:N and \clist_if_empty:c. These functions are documented
    on page ??.)

```

\clist_if_eq:NN Simple copies from the token list variable material.

```

\clist_if_eq:Nc 5875 \prg_new_eq_conditional:NNn \clist_if_eq:NN \tl_if_eq:NN { p , T , F , TF }
\clist_if_eq:cN 5876 \prg_new_eq_conditional:NNn \clist_if_eq:Nc \tl_if_eq:Nc { p , T , F , TF }
\clist_if_eq:cc 5877 \prg_new_eq_conditional:NNn \clist_if_eq:cN \tl_if_eq:cN { p , T , F , TF }
5878 \prg_new_eq_conditional:NNn \clist_if_eq:cc \tl_if_eq:cc { p , T , F , TF }
    (End definition for \clist_if_eq:NN and others. These functions are documented on page ??.)

```

\clist_if_in:NN See description of the \tl_if_in:NN function for details. We simply surround the comma list, and the item, with commas.

```

\clist_if_in:Nv 5879 \prg_new_protected_conditional:Npnn \clist_if_in:NN #1#2 { T , F , TF }
\clist_if_in:No 5880 {
\clist_if_in:cn 5881   \exp_args:No \clist_if_in_return:nn #1 {#2}
\clist_if_in:cV 5882 }
\clist_if_in:co 5883 \prg_new_protected_conditional:Npnn \clist_if_in:nn #1#2 { T , F , TF }
\clist_if_in:nn 5884 {
\clist_if_in:nV 5885   \clist_set:Nn \l_clist_if_in_clist {#1}
\clist_if_in:no 5886   \exp_args:No \clist_if_in_return:nn \l_clist_if_in_clist {#2}
\clist_if_in_return:nn 5887 }
5888 \cs_new_protected:Npn \clist_if_in_return:nn #1#2
5889 {
5890   \cs_set:Npn \clist_tmp:w ##1 ,#2, { }
5891   \tl_if_empty:oTF
5892     { \clist_tmp:w ,#1, {} {} } ,#2, {
5893     { \prg_return_false: } { \prg_return_true: }
5894   }
5895 \cs_generate_variant:Nn \clist_if_in:NnT { NV , No }
5896 \cs_generate_variant:Nn \clist_if_in:NnT { c , cV , co }
5897 \cs_generate_variant:Nn \clist_if_in:NnF { NV , No }
5898 \cs_generate_variant:Nn \clist_if_in:NnF { c , cV , co }
5899 \cs_generate_variant:Nn \clist_if_in:NnTF { NV , No }
5900 \cs_generate_variant:Nn \clist_if_in:NnTF { c , cV , co }
5901 \cs_generate_variant:Nn \clist_if_in:nnT { nV , no }
5902 \cs_generate_variant:Nn \clist_if_in:nnF { nV , no }
5903 \cs_generate_variant:Nn \clist_if_in:nnTF { nV , no }
    (End definition for \clist_if_in:NN and others. These functions are documented on page ??.)

```

194.8 Mapping to comma lists

\clist_map_function:NN If the variable is empty, the mapping is skipped (otherwise, that comma-list would be seen as consisting of one empty item). Then loop over the comma-list, grabbing one comma-delimited item at a time. The end is marked by \q_recursion_tail. The auxiliary function \clist_map_function_aux:Nw is used directly in \clist_map_inline:Nn. Change with care.

```

5904 \cs_new:Npn \clist_map_function:NN #1#2
5905 {
5906   \clist_if_empty:NF #1
5907   {
5908     \exp_last_unbraced:NNo \clist_map_function_aux:Nw #2 #1
5909     , \q_recursion_tail ,
5910     \prg_break_point:n { }
5911   }
5912 }
5913 \cs_new:Npn \clist_map_function_aux:Nw #1#2 ,
5914 {
5915   \quark_if_recursion_tail_break:n {#2}
5916   #1 {#2}
5917   \clist_map_function_aux:Nw #1
5918 }
5919 \cs_generate_variant:Nn \clist_map_function:NN { c }

```

(End definition for \clist_map_function:NN and \clist_map_function:cN. These functions are documented on page ??.)

\clist_map_function:nN The n-type mapping function is a bit more awkward, since spaces must be trimmed from
\clist_map_function_n_aux:Nn each item. Space trimming is again based on \clist_trim_spaces_generic:nw. The
\clist_map_aux_unbrace:Nw auxiliary \clist_map_function_n_aux:Nn receives as arguments the function, and the
result of removing leading and trailing spaces from the item which lies until the next
comma. Empty items are ignored, then one level of braces is removed by \clist_map_
aux_unbrace:Nw.

```

5920 \cs_new:Npn \clist_map_function:nN #1#2
5921 {
5922   \clist_trim_spaces_generic:nw { \clist_map_function_n_aux:Nn #2 }
5923   \q_mark #1, \q_recursion_tail,
5924   \prg_break_point:n { }
5925 }
5926 \cs_new:Npn \clist_map_function_n_aux:Nn #1 #2
5927 {
5928   \quark_if_recursion_tail_break:n {#2}
5929   \tl_if_empty:nF {#2} { \clist_map_aux_unbrace:Nw #1 #2, }
5930   \clist_trim_spaces_generic:nw { \clist_map_function_n_aux:Nn #1 }
5931   \q_mark
5932 }
5933 \cs_new:Npn \clist_map_aux_unbrace:Nw #1 #2, { #1 {#2} }

```

(End definition for \clist_map_function:nN. This function is documented on page ??.)

\clist_map_inline:Nn Inline mapping is done by creating a suitable function “on the fly”: this is done globally
\clist_map_inline:cn to avoid any issues with TeX’s groups. We use a different function for each level of
\clist_map_inline:nn nesting.

Since the mapping is non-expandable, we can perform the space-trimming needed by the n version simply by storing the comma-list in a variable. We don’t need a different comma-list for each nesting level: the comma-list is expanded before the mapping starts.

```

5934 \cs_new_protected:Npn \clist_map_inline:Nn #1#2

```



```

5935 {
5936   \clist_if_empty:NF #1
5937   {
5938     \int_gincr:N \g_prg_map_int
5939     \cs_gset:cpn { \clist_map_ \int_use:N \g_prg_map_int :n } ##1 {#2}
5940     \exp_last_unbraced:Nco \clist_map_function_aux:Nw
5941     { \clist_map_ \int_use:N \g_prg_map_int :n }
5942     #1 , \q_recursion_tail ,
5943     \prg_break_point:n { \int_gdecr:N \g_prg_map_int }
5944   }
5945 }
5946 \cs_new_protected:Npn \clist_map_inline:nn #1
5947 {
5948   \clist_set:Nn \l_clist_tmpa_clist {#1}
5949   \clist_map_inline:Nn \l_clist_tmpa_clist
5950 }
5951 \cs_generate_variant:Nn \clist_map_inline:Nn { c }

```

(End definition for \clist_map_inline:Nn and \clist_map_inline:cn. These functions are documented on page ??.)

\clist_map_variable:NNn As for other comma-list mappings, filter out the case of an empty list. Same approach
 \clist_map_variable:cNn as \clist_map_function:Nn, additionally we store each item in the given variable. As
 \clist_map_variable:nNn for inline mappings, space trimming for the n variant is done by storing the comma list
 \clist_map_variable_aux:Nnw in a variable.

```

5952 \cs_new_protected:Npn \clist_map_variable:NNn #1#2#3
5953 {
5954   \clist_if_empty:NF #1
5955   {
5956     \exp_args:Nno \use:nn
5957     { \clist_map_variable_aux:Nnw #2 {#3} }
5958     #1
5959     , \q_recursion_tail , \q_recursion_stop
5960     \prg_break_point:n { }
5961   }
5962 }
5963 \cs_new_protected:Npn \clist_map_variable:nNn #1
5964 {
5965   \clist_set:Nn \l_clist_tmpa_clist {#1}
5966   \clist_map_variable:NNn \l_clist_tmpa_clist
5967 }
5968 \cs_new_protected:Npn \clist_map_variable_aux:Nnw #1#2#3,
5969 {
5970   \tl_set:Nn #1 {#3}
5971   \quark_if_recursion_tail_stop:N #1
5972   \use:n {#2}
5973   \clist_map_variable_aux:Nnw #1 {#2}
5974 }
5975 \cs_generate_variant:Nn \clist_map_variable:NNn { c }

```

(End definition for \clist_map_variable:NNn and \clist_map_variable:cNn. These functions are documented on page ??.)

`\clist_map_break:` The break statements are simply copies.
`\clist_map_break:n` 5976 `\cs_new_eq:NN \clist_map_break: \prg_map_break:`
5977 `\cs_new_eq:NN \clist_map_break:n \prg_map_break:n`
(End definition for `\clist_map_break:`. This function is documented on page 109.)

194.9 Viewing comma lists

`\clist_show:N` Apply the general `\msg_aux_show:Nnx`. In the case of an n-type comma-list, first store it
`\clist_show:c` in a scratch variable, then show that variable, omitting its name from the 4-th argument.
`\clist_show:n` 5978 `\cs_new_protected:Npn \clist_show:N #1`
5979 `{`
5980 `\msg_aux_show:Nnx`
5981 `#1`
5982 `{ clist }`
5983 `{ \clist_map_function:NN #1 \msg_aux_show:n }`
5984 `}`
5985 `\cs_new_protected:Npn \clist_show:n #1`
5986 `{`
5987 `\clist_set:Nn \l_clist_tmpa_clist {#1}`
5988 `\msg_aux_show:Nnx`
5989 `\l_clist_tmpa_clist`
5990 `{ clist }`
5991 `{ \clist_map_function:NN \l_clist_tmpa_clist \msg_aux_show:n }`
5992 `}`
5993 `\cs_generate_variant:Nn \clist_show:N { c }`
(End definition for `\clist_show:N` and `\clist_show:c`. These functions are documented on page 110.)

194.10 Scratch comma lists

`\l_tmpa_clist` Temporary comma list variables.
`\l_tmpb_clist` 5994 `\clist_new:N \l_tmpa_clist`
`\g_tmpa_clist` 5995 `\clist_new:N \l_tmpb_clist`
`\g_tmpb_clist` 5996 `\clist_new:N \g_tmpa_clist`
5997 `\clist_new:N \g_tmpb_clist`
(End definition for `\l_tmpa_clist` and `\l_tmpb_clist`. These functions are documented on page 110.)

194.11 Experimental functions

`\clist_length:N` Counting the items in a comma list is done using the same approach as for other length
`\clist_length:c` functions: turn each entry into a +1 then use integer evaluation to actually do the math-
`\clist_length:n` ematics. In the case of an n-type comma-list, we could of course use `\clist_map_-`
`\clist_length_aux:w` `function:nN`, but that is very slow, because it carefully removes spaces. Instead, we
loop manually, and skip blank items (but not `{}`, hence the extra spaces).
5998 `\cs_new:Npn \clist_length:N #1`
5999 `{`

```

6000     \int_eval:n
6001     {
6002         0
6003         \clist_map_function:NN #1 \clist_length_aux:n
6004     }
6005 }
6006 \cs_new:Npn \clist_length_aux:n #1 { +1 }
6007 \cs_new:Npx \clist_length:n #1
6008 {
6009     \exp_not:N \int_eval:n
6010     {
6011         0
6012         \exp_not:N \clist_length_n_aux:w \c_space_tl
6013         #1 \exp_not:n { , \q_recursion_tail , \q_recursion_stop }
6014     }
6015 }
6016 \cs_new:Npx \clist_length_n_aux:w #1 ,
6017 {
6018     \exp_not:n { \exp_args:Nf \quark_if_recursion_tail_stop:n } {#1}
6019     \exp_not:N \tl_if_blank:nF {#1} { + \c_one }
6020     \exp_not:N \clist_length_n_aux:w \c_space_tl
6021 }
6022 \cs_generate_variant:Nn \clist_length:N { c }

```

(End definition for `\clist_length:N` and `\clist_length:c`. These functions are documented on page ??.)

`\clist_item:Nn` To avoid needing to test the end of the list at each step, we first compute the $\langle length \rangle$ of
`\clist_item:cn` the list. If the item number is less than $-\langle length \rangle$ or more than $\langle length \rangle - 1$, the result is
`\clist_item_aux:nnNn` empty. If it is negative, but not less than $-\langle length \rangle$, add the $\langle length \rangle$ to the item number
`\clist_item_N_loop:nw` before performing the loop. The loop itself is very simple, return the item if the counter
reached zero, otherwise, decrease the counter and repeat.

```

6023 \cs_new:Npn \clist_item:Nn #1#2
6024 {
6025     \exp_args:Nfo \clist_item_aux:nnNn
6026     { \clist_length:N #1 }
6027     #1
6028     \clist_item_N_loop:nw
6029     {#2}
6030 }
6031 \cs_new:Npn \clist_item_aux:nnNn #1#2#3#4
6032 {
6033     \int_compare:nNnTF {#4} < \c_zero
6034     {
6035         \int_compare:nNnTF {#4} < { - #1 }
6036         { \use_none_delimit_by_q_stop:w }
6037         { \exp_args:Nf #3 { \int_eval:n { #4 + #1 } } }
6038     }
6039     {
6040         \int_compare:nNnTF {#4} < {#1}

```

```

6041         { #3 {#4} }
6042         { \use_none_delimit_by_q_stop:w }
6043     }
6044     #2, \q_stop
6045 }
6046 \cs_new:Npn \clist_item_N_loop:nw #1 #2,
6047 {
6048     \int_compare:nNnTF {#1} = \c_zero
6049     { \use_i_delimit_by_q_stop:nw {#2} }
6050     { \exp_args:Nf \clist_item_N_loop:nw { \int_eval:n { #1 - 1 } } }
6051 }
6052 \cs_generate_variant:Nn \clist_item:Nn { c }

```

(End definition for \clist_item:Nn and \clist_item:cn. These functions are documented on page ??.)

\clist_item:nn This starts in the same way as \clist_item:Nn by checking the length of the comma list.
\clist_item_n_aux:nw The final item should be space-trimmed before being brace-stripped, hence we insert a
\clist_item_n_loop:nw couple of odd-looking \prg_do_nothing: to avoid losing braces. Blank items are ignored.
\clist_item_n_end:n
\clist_item_n_strip:w

```

6053 \cs_new:Npn \clist_item:nn #1#2
6054 {
6055     \exp_args:Nf \clist_item_aux:nnNn
6056     { \clist_length:n {#1} }
6057     {#1}
6058     \clist_item_n_aux:nw
6059     {#2}
6060 }
6061 \cs_new:Npn \clist_item_n_aux:nw #1
6062 { \clist_item_n_loop:nw {#1} \prg_do_nothing: }
6063 \cs_new:Npn \clist_item_n_loop:nw #1 #2,
6064 {
6065     \exp_args:No \tl_if_blank:nTF {#2}
6066     { \clist_item_n_loop:nw {#1} \prg_do_nothing: }
6067     {
6068         \int_compare:nNnTF {#1} = \c_zero
6069         { \exp_args:No \clist_item_n_end:n {#2} }
6070         {
6071             \exp_args:Nf \clist_item_n_loop:nw
6072             { \int_eval:n { #1 - 1 } }
6073             \prg_do_nothing:
6074         }
6075     }
6076 }
6077 \cs_new:Npn \clist_item_n_end:n #1 #2 \q_stop
6078 {
6079     \exp_after:wN \exp_after:wN \exp_after:wN \clist_item_n_strip:w
6080     \tl_trim_spaces:n {#1} ,
6081 }
6082 \cs_new:Npn \clist_item_n_strip:w #1 , {#1}

```

(End definition for \clist_item:nn. This function is documented on page ??.)

`\clist_set_from_seq:NN` Setting a comma list from a comma-separated list is done using a simple mapping. We
`\clist_set_from_seq:cN` wrap most items with `\exp_not:n`, and a comma. Items which contain a comma or a
`\clist_set_from_seq:Nc` space are surrounded by an extra set of braces. The first comma must be removed, except
`\clist_set_from_seq:cc` in the case of an empty comma-list.

```

6083 \cs_new_protected:Npn \clist_set_from_seq:NN
6084 { \clist_set_from_seq_aux:NNNN \clist_clear:N \tl_set:Nx }
6085 \cs_new_protected:Npn \clist_gset_from_seq:NN
6086 { \clist_set_from_seq_aux:NNNN \clist_gclear:N \tl_gset:Nx }
6087 \cs_new_protected:Npn \clist_set_from_seq_aux:NNNN #1#2#3#4
6088 {
6089   \seq_if_empty:NTF #4
6090   { #1 #3 }
6091   {
6092     \seq_push_item_def:n
6093     {
6094       ,
6095       \tl_if_empty:oTF { \clist_set_from_seq_aux:w ##1 ~ , ##1 ~ }
6096       { \exp_not:n {##1} }
6097       { \exp_not:n { {##1} } }
6098     }
6099     #2 #3 { \exp_last_unbraced:Nf \use_none:n #4 }
6100     \seq_pop_item_def:
6101   }
6102 }
6103 \cs_new:Npn \clist_set_from_seq_aux:w #1 , #2 ~ { }
6104 \cs_generate_variant:Nn \clist_set_from_seq:NN { Nc }
6105 \cs_generate_variant:Nn \clist_set_from_seq:NN { c , cc }
6106 \cs_generate_variant:Nn \clist_gset_from_seq:NN { Nc }
6107 \cs_generate_variant:Nn \clist_gset_from_seq:NN { c , cc }

```

(End definition for `\clist_set_from_seq:NN` and others. These functions are documented on page ??.)

`\clist_const:Nn` Creating and initializing a constant comma list is done in a way similar to `\clist_set:Nn`
`\clist_const:cn` and `\clist_gset:Nn`, being careful to strip spaces.

```

6108 \cs_new_protected:Npn \clist_const:Nn #1#2
6109 { \tl_const:Nx #1 { \clist_trim_spaces:n {#2} } }
6110 \cs_generate_variant:Nn \clist_const:Nn { c , Nx , cx }

```

(End definition for `\clist_const:Nn` and others. These functions are documented on page ??.)

`\clist_if_empty:n` As usual, we insert a token (here ?) before grabbing any argument: this avoids losing
`\clist_if_empty_n_aux:w` braces. The argument of `\tl_if_empty:oTF` is empty if #1 is ? followed by blank spaces
`\clist_if_empty_n_aux:wNw` (besides, this particular variant of the emptiness test is optimized). If the item of the
comma list is blank, grab the next one. As soon as one item is non-blank, exit: the second
auxiliary will grab `\prg_return_false:` as #2, unless every item in the comma list was
blank and the loop actually got broken by the trailing `\q_mark \prg_return_false:`
item.

```

6111 \prg_new_conditional:Npnn \clist_if_empty:n #1 { p , T , F , TF }
6112 {

```

```

6113     \clist_if_empty_n_aux:w ? #1
6114     , \q_mark \prg_return_false:
6115     , \q_mark \prg_return_true:
6116     \q_stop
6117   }
6118   \cs_new:Npn \clist_if_empty_n_aux:w #1 ,
6119   {
6120     \tl_if_empty:oTF { \use_none:nn #1 ? }
6121     { \clist_if_empty_n_aux:w ? }
6122     { \clist_if_empty_n_aux:wNw }
6123   }
6124   \cs_new:Npn \clist_if_empty_n_aux:wNw #1 \q_mark #2#3 \q_stop {#2}
  
```

(End definition for \clist_if_empty:n. This function is documented on page ??.)

194.12 Deprecated interfaces

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\clist_top:NN` These are old stack functions.

```

\clist_top:cN 6125 {*deprecated}
6126 \cs_new_eq:NN \clist_top:NN \clist_get:NN
6127 \cs_new_eq:NN \clist_top:cN \clist_get:cN
6128 </deprecated>
  
```

(End definition for \clist_top:NN and \clist_top:cN. These functions are documented on page ??.)

`\clist_remove_element:Nn` An older name for `\clist_remove_all:Nn`.

```

\clist_gremove_element:Nn 6129 {*deprecated}
6130 \cs_new_eq:NN \clist_remove_element:Nn \clist_remove_all:Nn
6131 \cs_new_eq:NN \clist_gremove_element:Nn \clist_gremove_all:Nn
6132 </deprecated>
  
```

(End definition for \clist_remove_element:Nn and \clist_gremove_element:Nn. These functions are documented on page ??.)

`\clist_display:N` An older name for `\clist_show:N`.

```

\clist_display:c 6133 {*deprecated}
6134 \cs_new_eq:NN \clist_display:N \clist_show:N
6135 \cs_new_eq:NN \clist_display:c \clist_show:c
6136 </deprecated>
  
```

(End definition for \clist_display:N and \clist_display:c. These functions are documented on page ??.)

Deprecated on 2011-09-05, for removal by 2011-12-32.

`\clist_trim_spaces:N` Since clist items are now always stripped from their surrounding spaces, it is redundant to provide these functions. The `\clist_trim_spaces:n` function is now internal, deprecated for use outside the kernel.

```

\clist_trim_spaces:c 6137 \cs_new_protected:Npn \clist_trim_spaces:N #1 { \clist_set:No #1 {#1} }
6138 \cs_new_protected:Npn \clist_gtrim_spaces:N #1 { \clist_gset:No #1 {#1} }
6139 \cs_generate_variant:Nn \clist_trim_spaces:N { c }
6140 \cs_generate_variant:Nn \clist_gtrim_spaces:N { c }
  
```

(End definition for `\clist_trim_spaces:N` and others. These functions are documented on page ??.)

```
6141 </initex | package>
```

195 l3prop implementation

The following test files are used for this code: `m3prop001`.

```
6142 <*initex | package>
6143 <*package>
6144 \ProvidesExplPackage
6145   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
6146   \package_check_loaded_expl:
6147 </package>
```

A property list is a macro whose top-level expansion is for the form “`\q_prop <key0> \q_prop {<value0>} \q_prop ... \q_prop <keyn-1> \q_prop {<valuen-1>} \q_prop`”. The trailing `\q_prop` is always present for performance reasons: this means that empty property lists are not actually empty.

`\q_prop` A private quark is used as a marker between entries.

```
6148 \quark_new:N \q_prop
(End definition for \q_prop. This function is documented on page 117.)
```

`\c_empty_prop` An empty prop contains exactly one `\q_prop`.

```
6149 \tl_const:Nn \c_empty_prop { \q_prop }
(End definition for \c_empty_prop. This function is documented on page 117.)
```

195.1 Allocation and initialisation

`\prop_new:N` Internally, property lists are token lists, but an empty prop is not an empty tl, so we need to do things by hand.

```
6150 \cs_new_protected:Npn \prop_new:N #1 { \cs_new_eq:NN #1 \c_empty_prop }
6151 \cs_new_protected:Npn \prop_new:c #1 { \cs_new_eq:cN {#1} \c_empty_prop }
(End definition for \prop_new:N and \prop_new:c. These functions are documented on page ??.)
```

`\prop_clear:N` The same idea for clearing

```

\prop_clear:c 6152 \cs_new_protected:Npn \prop_clear:N #1 { \cs_set_eq:NN #1 \c_empty_prop }
\prop_gclear:N 6153 \cs_new_protected:Npn \prop_clear:c #1 { \cs_set_eq:cN {#1} \c_empty_prop }
\prop_gclear:c 6154 \cs_new_protected:Npn \prop_gclear:N #1 { \cs_gset_eq:NN #1 \c_empty_prop }
6155 \cs_new_protected:Npn \prop_gclear:c #1 { \cs_gset_eq:cN {#1} \c_empty_prop }
(End definition for \prop_clear:N and \prop_clear:c. These functions are documented on page ??.)
```

```

\prop_clear_new:N      Once again a simple copy from the token list functions.
\prop_clear_new:c      6156 \cs_new_protected:Npn \prop_clear_new:N #1
\prop_gclear_new:N     6157 { \cs_if_exist:NTF #1 { \prop_clear:N #1 } { \prop_new:N #1 } }
\prop_gclear_new:c     6158 \cs_generate_variant:Nn \prop_clear_new:N { c }
                      6159 \cs_new_protected:Npn \prop_gclear_new:N #1
                      6160 { \cs_if_exist:NTF #1 { \prop_gclear:N #1 } { \prop_new:N #1 } }
                      6161 \cs_generate_variant:Nn \prop_gclear_new:N { c }

```

(End definition for `\prop_clear_new:N` and `\prop_clear_new:c`. These functions are documented on page ??.)

```

\prop_set_eq:NN      Once again, these are simply copies from the token list functions.
\prop_set_eq:cN      6162 \cs_new_eq:NN \prop_set_eq:NN \tl_set_eq:NN
\prop_set_eq:Nc      6163 \cs_new_eq:NN \prop_set_eq:Nc \tl_set_eq:Nc
\prop_set_eq:cc      6164 \cs_new_eq:NN \prop_set_eq:cN \tl_set_eq:cN
\prop_gset_eq:NN     6165 \cs_new_eq:NN \prop_set_eq:cc \tl_set_eq:cc
\prop_gset_eq:cN     6166 \cs_new_eq:NN \prop_gset_eq:NN \tl_gset_eq:NN
\prop_gset_eq:Nc     6167 \cs_new_eq:NN \prop_gset_eq:Nc \tl_gset_eq:Nc
\prop_gset_eq:cN     6168 \cs_new_eq:NN \prop_gset_eq:cN \tl_gset_eq:cN
\prop_gset_eq:cc     6169 \cs_new_eq:NN \prop_gset_eq:cc \tl_gset_eq:cc

```

(End definition for `\prop_set_eq:NN` and others. These functions are documented on page ??.)

195.2 Accessing data in property lists

```

\prop_split:NnTF      This function is used by most of the module, and hence must be fast. The aim here is
\prop_split_aux:NnTF  to split a property list at a given key into the part before the key–value pair, the value
\prop_split_aux:nnnn  associated with the key and the part after the key–value pair. To do this, the key is first
\prop_split_aux:w      detokenized (to avoid repeatedly doing this), then a delimited function is constructed
                      to match the key. It will match \q_prop <detokenized key> \q_prop {<value>} (extra
                      argument), effectively separating an <extract1> before the key in the property list and an
                      <extract2> after the key.

```

If the key is present in the property list, then *<extra argument>* is simply `\q_prop`, and `\prop_split_aux:nnnn` will gobble this and the false branch (#4), leaving the correct code on the input stream. More precisely, it leaves the user code (true branch), followed by three groups, `{<extract1>}{<value>}{<extract2>}`. In order for *<extract1>**<extract2>* to be a well-formed property list, *<extract1>* has a leading and trailing `\q_prop`, retaining exactly the structure of a property list, while *<extract2>* omits the leading `\q_prop`.

If the key is not there, then *<extra argument>* is `? \use_ii:nn { }`, and `\prop_split_aux:nnnn ? \u` removes the three brace groups that just follow. Then `\use_ii:nn` removes the true branch, leaving the false branch, with no trailing material.

```

6170 \cs_new_protected:Npn \prop_split:NnTF #1#2
6171 { \exp_args:NNo \prop_split_aux:NnTF #1 { \tl_to_str:n {#2} } }
6172 \cs_new_protected:Npn \prop_split_aux:NnTF #1#2
6173 {
6174   \cs_set_protected:Npn \prop_split_aux:w
6175     ##1 \q_prop #2 \q_prop ##2 ##3 ##4 \q_mark ##5 \q_stop
6176     { \prop_split_aux:nnnn ##3 { {##1 \q_prop } {##2} {##4} } }
6177   \exp_after:wN \prop_split_aux:w #1 \q_mark

```



```

6178         \q_prop #2 \q_prop { } { ? \use_ii:nn { } } \q_mark \q_stop
6179     }
6180 \cs_new:Npn \prop_split_aux:nnnn #1#2#3#4 { #3 #2 }
6181 \cs_new_protected:Npn \prop_split_aux:w { }
        (End definition for \prop_split:NnTF. This function is documented on page ??.)

```

\prop_split:Nnn The goal here is to provide a common interface for both true and false branches of **\prop_split:NnTF**. In both cases, the code given by the user will be placed in front of three brace groups, $\{\langle extract_1 \rangle\} \{\langle value \rangle\} \{\langle extract_2 \rangle\}$. If the key was missing from the property list, then $\langle extract_1 \rangle$ is the full property list, $\langle value \rangle$ is **\q_no_value**, and $\langle extract_2 \rangle$ is empty. Otherwise, $\langle extract_1 \rangle$ is the part of the property list before the $\langle key \rangle$, and has the structure of a property list, $\langle value \rangle$ is the value corresponding to the $\langle key \rangle$, and $\langle extract_2 \rangle$ (the part after the $\langle key \rangle$) is missing the leading **\q_prop**.

```

6182 \cs_new_protected:Npn \prop_split:Nnn #1#2#3
6183 {
6184     \prop_split:NnTF #1 {#2}
6185     {#3}
6186     { \exp_args:Nno \use:n {#3} {#1} { \q_no_value } { } }
6187 }
        (End definition for \prop_split:Nnn. This function is documented on page 117.)

```

\prop_del:Nn Deleting from a property starts by splitting the list. If the key is present in the property list, the returned value is ignored. If the key is missing, nothing happens.

```

\prop_del:NV
\prop_del:cn
\prop_del:cV
\prop_gdel:Nn
\prop_gdel:NV
\prop_gdel:cn
\prop_gdel:cV
\prop_del_aux:NNnnn
6188 \cs_new_protected:Npn \prop_del:Nn #1#2
6189 { \prop_split:NnTF #1 {#2} { \prop_del_aux:NNnnn \tl_set:Nn #1 } { } }
6190 \cs_new_protected:Npn \prop_gdel:Nn #1#2
6191 { \prop_split:NnTF #1 {#2} { \prop_del_aux:NNnnn \tl_gset:Nn #1 } { } }
6192 \cs_new_protected:Npn \prop_del_aux:NNnnn #1#2#3#4#5
6193 { #1 #2 { #3 #5 } }
6194 \cs_generate_variant:Nn \prop_del:Nn { NV }
6195 \cs_generate_variant:Nn \prop_del:Nn { c , cV }
6196 \cs_generate_variant:Nn \prop_gdel:Nn { NV }
6197 \cs_generate_variant:Nn \prop_gdel:Nn { c , cV }
        (End definition for \prop_del:Nn and others. These functions are documented on page ??.)

```

\prop_get:NnN Getting an item from a list is very easy: after splitting, if the key is in the property list, just set the token list variable to the return value, otherwise to **\q_no_value**.

```

\prop_get:NVN
\prop_get:NoN
\prop_get:cnN
\prop_get:cVN
\prop_get:NoN
\prop_get_aux:NNnn
6198 \cs_new_protected:Npn \prop_get:NnN #1#2#3
6199 {
6200     \prop_split:NnTF #1 {#2}
6201     { \prop_get_aux:NNnn #3 }
6202     { \tl_set:Nn #3 { \q_no_value } }
6203 }
6204 \cs_new_protected:Npn \prop_get_aux:NNnn #1#2#3#4
6205 { \tl_set:Nn #1 {#3} }
6206 \cs_generate_variant:Nn \prop_get:NnN { NV , No }
6207 \cs_generate_variant:Nn \prop_get:NnN { c , cV , co }
        (End definition for \prop_get:NnN and others. These functions are documented on page ??.)

```

`\prop_pop:NnN` Popping a value also starts by doing the split. If the key is present, save the value in
`\prop_pop:NoN` the token list and update the property list as when deleting. If the key is missing, save
`\prop_pop:cnN` `\q_no_value` in the token list.
`\prop_pop:coN` 6208 `\cs_new_protected:Npn \prop_pop:NnN #1#2#3`
`\prop_gpop:NnN` 6209 `{`
`\prop_gpop:NoN` 6210 `\prop_split:NnTF #1 {#2}`
`\prop_gpop:cnN` 6211 `{ \prop_pop_aux:NNNnnn \tl_set:Nn #1 #3 }`
`\prop_gpop:coN` 6212 `{ \tl_set:Nn #3 { \q_no_value } }`
`\prop_pop_aux:NNNnnn` 6213 `}`
6214 `\cs_new_protected:Npn \prop_gpop:NnN #1#2#3`
6215 `{`
6216 `\prop_split:NnTF #1 {#2}`
6217 `{ \prop_pop_aux:NNNnnn \tl_gset:Nn #1 #3 }`
6218 `{ \tl_set:Nn #3 { \q_no_value } }`
6219 `}`
6220 `\cs_new_protected:Npn \prop_pop_aux:NNNnnn #1#2#3#4#5#6`
6221 `{`
6222 `\tl_set:Nn #3 {#5}`
6223 `#1 #2 { #4 #6 }`
6224 `}`
6225 `\cs_generate_variant:Nn \prop_pop:NnN { No }`
6226 `\cs_generate_variant:Nn \prop_pop:NnN { c , co }`
6227 `\cs_generate_variant:Nn \prop_gpop:NnN { No }`
6228 `\cs_generate_variant:Nn \prop_gpop:NnN { c , co }`
(End definition for \prop_pop:NnN and others. These functions are documented on page ??.)

`\prop_put:Nnn` Putting a key–value pair in a property list starts by splitting to remove any existing
`\prop_put:NnV` value. The property list is then reconstructed with the two remaining parts #5 and #7
`\prop_put:Nno` first, followed by the new or updated entry.
`\prop_put:Nnx` 6229 `\cs_new_protected:Npn \prop_put:Nnn { \prop_put_aux:NNnn \tl_set:Nx }`
`\prop_put:NVn` 6230 `\cs_new_protected:Npn \prop_gput:Nnn { \prop_put_aux:NNnn \tl_gset:Nx }`
`\prop_put:NVV` 6231 `\cs_new_protected:Npn \prop_put_aux:NNnn #1#2#3#4`
`\prop_put:Non` 6232 `{`
`\prop_put:Noo` 6233 `\prop_split:Nnn #2 {#3} { \prop_put_aux:NNnnnnn #1 #2 {#3} {#4} }`
`\prop_put:cnn` 6234 `}`
`\prop_put:cnV` 6235 `\cs_new_protected:Npn \prop_put_aux:NNnnnnn #1#2#3#4#5#6#7`
`\prop_put:cno` 6236 `{`
`\prop_put:cnx` 6237 `#1 #2`
`\prop_put:cVn` 6238 `{`
`\prop_put:cVV` 6239 `\exp_not:n { #5 #7 }`
`\prop_put:con` 6240 `\tl_to_str:n {#3} \exp_not:n { \q_prop {#4} \q_prop }`
`\prop_put:coo` 6241 `}`
6242 `}`
6243 `\cs_generate_variant:Nn \prop_put:Nnn`
6244 `{ NnV , Nno , Nnx , NV , NVV , No , Noo }`
6245 `\cs_generate_variant:Nn \prop_put:Nnn`
6246 `{ c , cnV , cno , cnx , cV , cVV , co , coo }`
6247 `\cs_generate_variant:Nn \prop_gput:Nnn`
6248 `{ NnV , Nno , Nnx , NV , NVV , No , Noo }`

```

6249 \cs_generate_variant:Nn \prop_gput:Nnn
6250 { c , cnV , cno , cnx , cV , cVV , co , coo }

```

(End definition for \prop_put:Nnn and others. These functions are documented on page ??.)

\prop_put_if_new:Nnn Adding conditionally also splits. If the key is already present, the three brace groups
\prop_put_if_new:cnn given by \prop_split:NnTF are removed. If the key is new, then the value is added,
\prop_gput_if_new:Nnn being careful to convert the key to a string using \tl_to_str:n.
\prop_gput_if_new:cnn

```

6251 \cs_new_protected_nopar:Npn \prop_put_if_new:Nnn
6252 { \prop_put_if_new_aux:NNnn \tl_put_right:Nx }
6253 \cs_new_protected_nopar:Npn \prop_gput_if_new:Nnn
6254 { \prop_put_if_new_aux:NNnn \tl_gput_right:Nx }
6255 \cs_new_protected:Npn \prop_put_if_new_aux:NNnn #1#2#3#4
6256 {
6257   \prop_split:NnTF #2 {#3}
6258   { \use_none:nnn }
6259   {
6260     #1 #2
6261     { \tl_to_str:n {#3} \exp_not:n { \q_prop {#4} \q_prop } }
6262   }
6263 }
6264 \cs_generate_variant:Nn \prop_put_if_new:Nnn { c }
6265 \cs_generate_variant:Nn \prop_gput_if_new:Nnn { c }

```

(End definition for \prop_put_if_new:Nnn and \prop_gput_if_new:Nnn. These functions are documented on page ??.)

195.3 Property list conditionals

\prop_if_empty:N The test here uses \c_empty_prop as it is not really empty!

```

\prop_if_empty:c
6266 \prg_new_conditional:Npnn \prop_if_empty:N #1 { p , T , F , TF }
6267 {
6268   \if_meaning:w #1 \c_empty_prop
6269   \prg_return_true:
6270   \else:
6271   \prg_return_false:
6272   \fi:
6273 }
6274 \cs_generate_variant:Nn \prop_if_empty_p:N {c}
6275 \cs_generate_variant:Nn \prop_if_empty:NnTF {c}
6276 \cs_generate_variant:Nn \prop_if_empty:NT {c}
6277 \cs_generate_variant:Nn \prop_if_empty:NF {c}

```

(End definition for \prop_if_empty:N and \prop_if_empty:c. These functions are documented on page ??.)

\prop_if_in:Nn Testing expandably if a key is in a property list requires to go through the key-value
\prop_if_in:NV pairs one by one. This is rather slow, and a faster test would be

```

\prop_if_in:No
\prop_if_in:cn
\prop_if_in:cV
\prop_if_in:co
\prop_if_in_aux:nwn
\prop_if_in_aux:N

```

```

\prg_new_protected_conditional:Npnn \prop_if_in:Nn #1 #2
{
  \prop_split:NnTF #1 {#2}

```

```

    {
      \prg_return_true:
      \use_none:nnn
    }
    { \prg_return_false: }
  }

```

but `\prop_split:NnTF` is non-expandable.

Instead, the key is compared to each key in turn using `\str_if_eq:xx`, which is expandable. To terminate the mapping, we add the key that is search for at the end of the property list. This second `\tl_to_str:n` is not expanded at the start, but only when included in the `\str_if_eq:xx`. It cannot make the breaking mechanism choke, because the arbitrary token list material is enclosed in braces. When ending, we test the next token: it is either `\q_prop` or `\q_recursion_tail` in the case of a missing key. Here, `\prop_map_function:NN` is not sufficient for the mapping, since it can only map a single token, and cannot carry the key that is searched for.

```

6278 \prg_new_conditional:Npnn \prop_if_in:Nn #1#2 { p , T , F , TF }
6279 {
6280   \exp_last_unbraced:Noo \prop_if_in_aux:nwn
6281   { \tl_to_str:n {#2} } #1
6282   \tl_to_str:n {#2} \q_prop { }
6283   \q_recursion_tail
6284   \prg_break_point:n { }
6285 }
6286 \cs_new:Npn \prop_if_in_aux:nwn #1 \q_prop #2 \q_prop #3
6287 {
6288   \str_if_eq:xxTF {#1} {#2}
6289   { \prop_if_in_aux:N }
6290   { \prop_if_in_aux:nwn {#1} }
6291 }
6292 \cs_new:Npn \prop_if_in_aux:N #1
6293 {
6294   \if_meaning:w \q_prop #1
6295   \prg_return_true:
6296   \else:
6297   \prg_return_false:
6298   \fi:
6299   \prop_map_break:
6300 }
6301 \cs_generate_variant:Nn \prop_if_in_p:Nn { NV , No }
6302 \cs_generate_variant:Nn \prop_if_in_p:Nn { c , cV , co }
6303 \cs_generate_variant:Nn \prop_if_in:NnT { NV , No }
6304 \cs_generate_variant:Nn \prop_if_in:NnT { c , cV , co }
6305 \cs_generate_variant:Nn \prop_if_in:NnF { NV , No }
6306 \cs_generate_variant:Nn \prop_if_in:NnF { c , cV , co }
6307 \cs_generate_variant:Nn \prop_if_in:NnTF { NV , No }
6308 \cs_generate_variant:Nn \prop_if_in:NnTF { c , cV , co }

```

(End definition for `\prop_if_in:Nn` and others. These functions are documented on page ??.)

195.4 Recovering values from property lists with branching

`\prop_get:NnN` Getting the value corresponding to a key, keeping track of whether the key was present
`\prop_get:NVN` or not, is implemented as a conditional (with side effects). If the key was absent, the
`\prop_get:NoN` token list is not altered.
`\prop_get:cnN` 6309 `\prg_new_protected_conditional:Npnn \prop_get:NnN #1#2#3 { T , F , TF }`
`\prop_get:cVN` 6310 `{`
`\prop_get:coN` 6311 `\prop_split:NnTF #1 {#2}`
`\prop_get_aux_true:Nnnn` 6312 `{ \prop_get_aux_true:Nnnn #3 }`
6313 `{ \prg_return_false: }`
6314 `}`
6315 `\cs_new_protected:Npn \prop_get_aux_true:Nnnn #1#2#3#4`
6316 `{`
6317 `\tl_set:Nn #1 {#3}`
6318 `\prg_return_true:`
6319 `}`
6320 `\cs_generate_variant:Nn \prop_get:NnNT { NV , No }`
6321 `\cs_generate_variant:Nn \prop_get:NnNF { NV , No }`
6322 `\cs_generate_variant:Nn \prop_get:NnNTF { NV , No }`
6323 `\cs_generate_variant:Nn \prop_get:NnNT { c , cV , co }`
6324 `\cs_generate_variant:Nn \prop_get:NnNF { c , cV , co }`
6325 `\cs_generate_variant:Nn \prop_get:NnNTF { c , cV , co }`
(End definition for \prop_get:NnN and others. These functions are documented on page ??.)

195.5 Mapping to property lists

`\prop_map_function:NN` The fastest way to do a recursion here is to use an `\if_meaning:w` test: the keys are
`\prop_map_function:Nc` strings, and thus cannot match the marker `\q_recursion_tail`.
`\prop_map_function:cN` 6326 `\cs_new:Npn \prop_map_function:NN #1#2`
`\prop_map_function:cc` 6327 `{`
`\prop_map_function_aux:Nwn` 6328 `\exp_last_unbraced:NNo \prop_map_function_aux:Nwn #2`
6329 `#1 \q_recursion_tail \q_prop { }`
6330 `\prg_break_point:n { }`
6331 `}`
6332 `\cs_new:Npn \prop_map_function_aux:Nwn #1 \q_prop #2 \q_prop #3`
6333 `{`
6334 `\if_meaning:w \q_recursion_tail #2`
6335 `\exp_after:wN \prop_map_break:`
6336 `\fi:`
6337 `#1 {#2} {#3}`
6338 `\prop_map_function_aux:Nwn #1`
6339 `}`
6340 `\cs_generate_variant:Nn \prop_map_function:NN { Nc }`
6341 `\cs_generate_variant:Nn \prop_map_function:NN { c , cc }`
(End definition for \prop_map_function:NN and others. These functions are documented on page ??.)

`\prop_map_inline:Nn` Mapping in line requires a nesting level counter.

`\prop_map_inline:cn` 6342 `\cs_new_protected:Npn \prop_map_inline:Nn #1#2`

```

6343 {
6344   \int_gincr:N \g_prg_map_int
6345   \cs_gset:cpn { prop_map_inline_ \int_use:N \g_prg_map_int :nn }
6346   ##1##2 {#2}
6347   \exp_last_unbraced:Nco \prop_map_function_aux:Nwn
6348   { prop_map_inline_ \int_use:N \g_prg_map_int :nn }
6349   #1
6350   \q_recursion_tail \q_prop { }
6351   \prg_break_point:n { \int_gdecr:N \g_prg_map_int }
6352 }
6353 \cs_generate_variant:Nn \prop_map_inline:Nn { c }

```

(End definition for `\prop_map_inline:Nn` and `\prop_map_inline:cn`. These functions are documented on page ??.)

`\prop_map_break:` The break statements are simply copies.

```

\prop_map_break:n
6354 \cs_new_eq:NN \prop_map_break: \prg_map_break:
6355 \cs_new_eq:NN \prop_map_break:n \prg_map_break:n

```

(End definition for `\prop_map_break:`. This function is documented on page 116.)

195.6 Viewing property lists

`\prop_show:N` Apply the general `\msg_aux_show:Nnx`. Contrarily to sequences and comma lists, we use
`\prop_show:c` `\msg_aux_show:nn` to format both the key and the value for each pair.

```

6356 \cs_new_protected:Npn \prop_show:N #1
6357 {
6358   \msg_aux_show:Nnx
6359   #1
6360   { prop }
6361   { \prop_map_function:NN #1 \msg_aux_show:nn }
6362 }
6363 \cs_generate_variant:Nn \prop_show:N { c }

```

(End definition for `\prop_show:N` and `\prop_show:c`. These functions are documented on page ??.)

195.7 Experimental functions

`\prop_pop:NnN` Popping an item from a property list, keeping track of whether the key was present or
`\prop_pop:cnN` not, is implemented as a conditional. If the key was missing, neither the property list, nor
`\prop_gpop:cnN` the token list are altered. Otherwise, `\prg_return_true:` is used after the assignments.

```

\prop_gpop:cnN
\prop_pop_aux_true:NNNnnn
6364 \prg_new_protected_conditional:Npnn \prop_pop:NnN #1#2#3 { T , F , TF }
6365 {
6366   \prop_split:NnTF #1 {#2}
6367   { \prop_pop_aux_true:NNNnnn \tl_set:Nn #1 #3 }
6368   { \prg_return_false: }
6369 }
6370 \prg_new_protected_conditional:Npnn \prop_gpop:NnN #1#2#3 { T , F , TF }
6371 {
6372   \prop_split:NnTF #1 {#2}

```

```

6373     { \prop_pop_aux_true:NNNnnn \tl_gset:Nn #1 #3 }
6374     { \prg_return_false: }
6375   }
6376 \cs_new_protected:Npn \prop_pop_aux_true:NNNnnn #1#2#3#4#5#6
6377 {
6378   \tl_set:Nn #3 {#5}
6379   #1 #2 { #4 #6 }
6380   \prg_return_true:
6381 }
6382 \cs_generate_variant:Nn \prop_pop:NnNT { c }
6383 \cs_generate_variant:Nn \prop_pop:NnNF { c }
6384 \cs_generate_variant:Nn \prop_pop:NnNTF { c }
6385 \cs_generate_variant:Nn \prop_gpop:NnNT { c }
6386 \cs_generate_variant:Nn \prop_gpop:NnNF { c }
6387 \cs_generate_variant:Nn \prop_gpop:NnNTF { c }

```

(End definition for \prop_pop:NnN and others. These functions are documented on page ??.)

\prop_map_tokens:Nn The mapping grabs one key–value pair at a time, and stops when reaching the marker
 \prop_map_tokens:cn key \q_recursion_tail, which cannot appear in normal keys since those are strings.
 \prop_map_tokens_aux:nwn The odd construction \use:n {#1} allows #1 to contain any token.

```

6388 \cs_new:Npn \prop_map_tokens:Nn #1#2
6389 {
6390   \exp_last_unbraced:Nno \prop_map_tokens_aux:nwn {#2} #1
6391   \q_recursion_tail \q_prop { }
6392   \prg_break_point:n { }
6393 }
6394 \cs_new:Npn \prop_map_tokens_aux:nwn #1 \q_prop #2 \q_prop #3
6395 {
6396   \if_meaning:w \q_recursion_tail #2
6397     \exp_after:wN \prop_map_break:
6398   \fi:
6399   \use:n {#1} {#2} {#3}
6400   \prop_map_tokens_aux:nwn {#1}
6401 }
6402 \cs_generate_variant:Nn \prop_map_tokens:Nn { c }

```

(End definition for \prop_map_tokens:Nn and \prop_map_tokens:cn. These functions are documented on page ??.)

\prop_get:Nn Getting the value corresponding to a key in a property list in an expandable fashion is a
 \prop_get:cn simple instance of mapping some tokens. Map the function \prop_get_aux:nnn which
 \prop_get_Nn_aux:nwn takes as its three arguments the <key> that we are looking for, the current <key> and the
 current <value>. If the <keys> match, the <value> is returned. If none of the keys match,
 this expands to nothing.

```

6403 \cs_new:Npn \prop_get:Nn #1#2
6404 {
6405   \exp_last_unbraced:Noo \prop_get_Nn_aux:nwn
6406     { \tl_to_str:n {#2} } #1
6407     \tl_to_str:n {#2} \q_prop { }
6408   \q_recursion_stop

```

```

6409 }
6410 \cs_new:Npn \prop_get:Nn_aux:nwn #1 \q_prop #2 \q_prop #3
6411 {
6412   \str_if_eq:xxTF {#1} {#2}
6413     { \use_i_delimit_by_q_recursion_stop:nw {#3} }
6414     { \prop_get:Nn_aux:nwn {#1} }
6415 }
6416 \cs_generate_variant:Nn \prop_get:Nn { c }

```

(End definition for `\prop_get:Nn` and `\prop_get:cn`. These functions are documented on page ??.)

195.8 Deprecated interfaces

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\prop_display:N` An older name for `\prop_show:N`.

```

\prop_display:c
6417 \*deprecated
6418 \cs_new_eq:NN \prop_display:N \prop_show:N
6419 \cs_new_eq:NN \prop_display:c \prop_show:c
6420 \*deprecated

```

(End definition for `\prop_display:N` and `\prop_display:c`. These functions are documented on page ??.)

`\prop_gget:NnN` Getting globally is no longer supported: this is a conceptual change, so the necessary code for the transition is provided directly.

```

\prop_gget:NnN
\prop_gget:NVN
\prop_gget:cnN
\prop_gget:cVN
\prop_gget_aux:Nnnn
6421 \*deprecated
6422 \cs_new_protected:Npn \prop_gget:NnN #1#2#3
6423 { \prop_split:Nnn #1 {#2} { \prop_gget_aux:Nnnn #3 } }
6424 \cs_new_protected:Npn \prop_gget_aux:Nnnn #1#2#3#4
6425 { \tl_gset:Nn #1 {#3} }
6426 \cs_generate_variant:Nn \prop_gget:NnN { NV }
6427 \cs_generate_variant:Nn \prop_gget:NnN { c , cV }
6428 \*deprecated

```

(End definition for `\prop_gget:NnN` and others. These functions are documented on page ??.)

`\prop_get_gdel:NnN` This name seems very odd.

```

6429 \*deprecated
6430 \cs_new_eq:NN \prop_get_gdel:NnN \prop_gpop:NnN
6431 \*deprecated

```

(End definition for `\prop_get_gdel:NnN`. This function is documented on page ??.)

`\prop_if_in:cc` A hang-over from an ancient implementation

```

6432 \*deprecated
6433 \cs_generate_variant:Nn \prop_if_in:NnT { cc }
6434 \cs_generate_variant:Nn \prop_if_in:NnF { cc }
6435 \cs_generate_variant:Nn \prop_if_in:NnTF { cc }
6436 \*deprecated

```

(End definition for `\prop_if_in:cc`. This function is documented on page ??.)

`\prop_gput:ccx` Another one.

```
6437 <*deprecated>
6438 \cs_generate_variant:Nn \prop_gput:Nnn { ccx }
6439 </deprecated>
(End definition for \prop_gput:ccx. This function is documented on page ??.)
```

`\prop_if_eq:NN` These ones do no even make sense!

```
\prop_if_eq:Nc 6440 <*deprecated>
\prop_if_eq:cN 6441 \prg_new_eq_conditional:NNn \prop_if_eq:NN \tl_if_eq:NN { p , T , F , TF }
\prop_if_eq:cc 6442 \prg_new_eq_conditional:NNn \prop_if_eq:cN \tl_if_eq:cN { p , T , F , TF }
6443 \prg_new_eq_conditional:NNn \prop_if_eq:Nc \tl_if_eq:Nc { p , T , F , TF }
6444 \prg_new_eq_conditional:NNn \prop_if_eq:cc \tl_if_eq:cc { p , T , F , TF }
6445 </deprecated>
(End definition for \prop_if_eq:NN and others. These functions are documented on page ??.)
6446 </initex | package>
```

196 l3box implementation

```
6447 <*initex | package>
6448 <*package>
6449 \ProvidesExplPackage
6450 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
6451 \package_check_loaded_expl:
6452 </package>
```

The code in this module is very straight forward so I'm not going to comment it very extensively.

196.1 Creating and initialising boxes

The following test files are used for this code: `m3box001.lvt`.

`\box_new:N` Defining a new `<box>` register: remember that box 255 is not generally available.

```
\box_new:c 6453 <*package>
6454 \cs_new_protected:Npn \box_new:N #1
6455 {
6456   \chk_if_free_cs:N #1
6457   \newbox #1
6458 }
6459 </package>
6460 \cs_generate_variant:Nn \box_new:N { c }
```

`\box_clear:N` Clear a `<box>` register.

```
\box_clear:c 6461 \cs_new_protected:Npn \box_clear:N #1
\box_gclear:N 6462 { \box_set_eq:NN #1 \c_empty_box }
\box_gclear:c 6463 \cs_new_protected:Npn \box_gclear:N #1
6464 { \box_gset_eq:NN #1 \c_empty_box }
6465 \cs_generate_variant:Nn \box_clear:N { c }
6466 \cs_generate_variant:Nn \box_gclear:N { c }
```

`\box_clear_new:N` Clear or new.

```

\box_clear_new:c 6467 \cs_new_protected:Npn \box_clear_new:N #1
\box_gclear_new:N 6468 {
\box_gclear_new:c 6469   \cs_if_exist:NTF #1
6470     { \box_set_eq:NN #1 \c_empty_box }
6471     { \box_new:N #1 }
6472   }
6473 \cs_new_protected:Npn \box_gclear_new:N #1
6474 {
6475   \cs_if_exist:NTF #1
6476     { \box_gset_eq:NN #1 \c_empty_box }
6477     { \box_new:N #1 }
6478   }
6479 \cs_generate_variant:Nn \box_clear_new:N { c }
6480 \cs_generate_variant:Nn \box_gclear_new:N { c }

```

`\box_set_eq:NN` Assigning the contents of a box to be another box.

```

\box_set_eq:cN 6481 \cs_new_protected:Npn \box_set_eq:NN #1#2
\box_set_eq:Nc 6482 { \tex_setbox:D #1 \tex_copy:D #2 }
\box_set_eq:cc 6483 \cs_new_protected:Npn \box_gset_eq:NN
\box_gset_eq:NN 6484 { \tex_global:D \box_set_eq:NN }
\box_gset_eq:cN 6485 \cs_generate_variant:Nn \box_set_eq:NN { cN , Nc , cc }
\box_gset_eq:Nc 6486 \cs_generate_variant:Nn \box_gset_eq:NN { cN , Nc , cc }

```

`\box_gset_eq:cc` Assigning the contents of a box to be another box. This clears the second box globally (that's how \TeX does it).

```

\box_set_eq_clear:NN 6487 \cs_new_protected:Npn \box_set_eq_clear:NN #1#2
\box_set_eq_clear:cN 6488 { \tex_setbox:D #1 \tex_box:D #2 }
\box_set_eq_clear:Nc 6489 \cs_new_protected:Npn \box_gset_eq_clear:NN
\box_gset_eq_clear:NN 6490 { \tex_global:D \box_set_eq_clear:NN }
\box_gset_eq_clear:cN 6491 \cs_generate_variant:Nn \box_set_eq_clear:NN { cN , Nc , cc }
\box_gset_eq_clear:Nc 6492 \cs_generate_variant:Nn \box_gset_eq_clear:NN { cN , Nc , cc }
\box_gset_eq_clear:cc

```

196.2 Measuring and setting box dimensions

`\box_ht:N` Accessing the height, depth, and width of a $\langle box \rangle$ register.

```

\box_ht:c 6493 \cs_new_eq:NN \box_ht:N \tex_ht:D
\box_dp:N 6494 \cs_new_eq:NN \box_dp:N \tex_dp:D
\box_dp:c 6495 \cs_new_eq:NN \box_wd:N \tex_wd:D
\box_wd:N 6496 \cs_generate_variant:Nn \box_ht:N { c }
\box_wd:c 6497 \cs_generate_variant:Nn \box_dp:N { c }
6498 \cs_generate_variant:Nn \box_wd:N { c }

```

`\box_set_ht:Nn` Measuring is easy: all primitive work. These primitives are not expandable, so the derived functions are not either.

```

\box_set_dp:Nn 6499 \cs_new_protected:Npn \box_set_dp:Nn #1#2
\box_set_dp:cn 6500 { \box_dp:N #1 \dim_eval:w #2 \dim_eval_end: }
\box_set_wd:Nn 6501 \cs_new_protected:Npn \box_set_ht:Nn #1#2
\box_set_wd:cn

```

```

6502 { \box_ht:N #1 \dim_eval:w #2 \dim_eval_end: }
6503 \cs_new_protected:Npn \box_set_wd:Nn #1#2
6504 { \box_wd:N #1 \dim_eval:w #2 \dim_eval_end: }
6505 \cs_generate_variant:Nn \box_set_ht:Nn { c }
6506 \cs_generate_variant:Nn \box_set_dp:Nn { c }
6507 \cs_generate_variant:Nn \box_set_wd:Nn { c }

```

196.3 Using boxes

`\box_use_clear:N` Using a $\langle box \rangle$. These are just TeX primitives with meaningful names.

```

\box_use_clear:c 6508 \cs_new_eq:NN \box_use_clear:N \tex_box:D
\box_use:N       6509 \cs_new_eq:NN \box_use:N \tex_copy:D
\box_use:c       6510 \cs_generate_variant:Nn \box_use_clear:N { c }
                 6511 \cs_generate_variant:Nn \box_use:N { c }

```

`\box_move_left:nn` Move box material in different directions.

```

\box_move_right:nn 6512 \cs_new_protected:Npn \box_move_left:nn #1#2
\box_move_up:nn    6513 { \tex_moveleft:D \dim_eval:w #1 \dim_eval_end: #2 }
\box_move_down:nn  6514 \cs_new_protected:Npn \box_move_right:nn #1#2
                   6515 { \tex_moveright:D \dim_eval:w #1 \dim_eval_end: #2 }
                   6516 \cs_new_protected:Npn \box_move_up:nn #1#2
                   6517 { \tex_raise:D \dim_eval:w #1 \dim_eval_end: #2 }
                   6518 \cs_new_protected:Npn \box_move_down:nn #1#2
                   6519 { \tex_lower:D \dim_eval:w #1 \dim_eval_end: #2 }

```

196.4 Box conditionals

`\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 6520 \cs_new_eq:NN \if_hbox:N \tex_ifhbox:D
\if_box_empty:N 6521 \cs_new_eq:NN \if_vbox:N \tex_ifvbox:D
               6522 \cs_new_eq:NN \if_box_empty:N \tex_ifvoid:D

```

```

\box_if_horizontal:N 6523 \prg_new_conditional:Npnn \box_if_horizontal:N #1 { p , T , F , TF }
\box_if_horizontal:c 6524 { \if_hbox:N #1 \prg_return_true: \else: \prg_return_false: \fi: }
\box_if_vertical:N   6525 \prg_new_conditional:Npnn \box_if_vertical:N #1 { p , T , F , TF }
\box_if_vertical:c   6526 { \if_vbox:N #1 \prg_return_true: \else: \prg_return_false: \fi: }
                   6527 \cs_generate_variant:Nn \box_if_horizontal_p:N { c }
                   6528 \cs_generate_variant:Nn \box_if_horizontal:NT { c }
                   6529 \cs_generate_variant:Nn \box_if_horizontal:NF { c }
                   6530 \cs_generate_variant:Nn \box_if_horizontal:NTF { c }
                   6531 \cs_generate_variant:Nn \box_if_vertical_p:N { c }
                   6532 \cs_generate_variant:Nn \box_if_vertical:NT { c }
                   6533 \cs_generate_variant:Nn \box_if_vertical:NF { c }
                   6534 \cs_generate_variant:Nn \box_if_vertical:NTF { c }

```

```

\box_if_empty:N Testing if a  $\langle box \rangle$  is empty/void.
\box_if_empty:c
6535 \prg_new_conditional:Npnn \box_if_empty:N #1 { p , T , F , TF }
6536 { \if_box_empty:N #1 \prg_return_true: \else: \prg_return_false: \fi: }
6537 \cs_generate_variant:Nn \box_if_empty_p:N { c }
6538 \cs_generate_variant:Nn \box_if_empty_NT { c }
6539 \cs_generate_variant:Nn \box_if_empty_NF { c }
6540 \cs_generate_variant:Nn \box_if_empty_NTF { c }
        (End definition for \box_new:N and \box_new:c. These functions are documented on page ??.)

```

196.5 The last box inserted

```

\box_set_to_last:N Set a box to the previous box.
\box_set_to_last:c
\box_gset_to_last:N
\box_gset_to_last:c
6541 \cs_new_protected:Npn \box_set_to_last:N #1
6542 { \tex_setbox:D #1 \tex_lastbox:D }
6543 \cs_new_protected:Npn \box_gset_to_last:N
6544 { \tex_global:D \box_set_to_last:N }
6545 \cs_generate_variant:Nn \box_set_to_last:N { c }
6546 \cs_generate_variant:Nn \box_gset_to_last:N { c }
        (End definition for \box_set_to_last:N and \box_set_to_last:c. These functions are documented on page ??.)

```

196.6 Constant boxes

```

\c_empty_box
6547 <*package>
6548 \cs_new_eq:NN \c_empty_box \voidb@x
6549 </package>
6550 <*initex>
6551 \box_new:N \c_empty_box
6552 </initex>
        (End definition for \c_empty_box. This function is documented on page 123.)

```

196.7 Scratch boxes

```

\l_tmpa_box
\l_tmpb_box
6553 <*package>
6554 \cs_new_eq:NN \l_tmpa_box \@tempboxa
6555 </package>
6556 <*initex>
6557 \box_new:N \l_tmpa_box
6558 </initex>
6559 \box_new:N \l_tmpb_box
        (End definition for \l_tmpa_box and \l_tmpb_box. These functions are documented on page 123.)

```

196.8 Viewing box contents

`\box_show:N` Check that the variable exists, then show the contents of the box and write it into the log file. The spurious `\use:n` gives a nicer output.

```

\box_show:c
6560 \cs_new_protected:Npn \box_show:N #1
6561 {
6562   \cs_if_exist:NTF #1
6563   { \tex_showbox:D \use:n {#1} }
6564   {
6565     \msg_kernel_error:nxx { kernel } { variable-not-defined }
6566     { \token_to_str:N #1 }
6567   }
6568 }
6569 \cs_generate_variant:Nn \box_show:N { c }

```

(End definition for \box_show:N and \box_show:c. These functions are documented on page ??.)

`\box_show:Nnn` Show the contents of a box and write it into the log file, after setting the parameters `\showboxbreadth` and `\showboxdepth` to the values provided by the user.

```

\box_show:cnn
\box_show_full:N
\box_show_full:c
6570 \cs_new_protected:Npn \box_show:Nnn #1#2#3
6571 {
6572   \group_begin:
6573   \int_set:Nn \tex_showboxbreadth:D {#2}
6574   \int_set:Nn \tex_showboxdepth:D {#3}
6575   \int_set_eq:NN \tex_tracingonline:D \c_one
6576   \box_show:N #1
6577   \group_end:
6578 }
6579 \cs_generate_variant:Nn \box_show:Nnn { c }
6580 \cs_new_protected:Npn \box_show_full:N #1
6581 { \box_show:Nnn #1 { \c_max_int } { \c_max_int } }
6582 \cs_generate_variant:Nn \box_show_full:N { c }

```

(End definition for \box_show:Nnn and \box_show:cnn. These functions are documented on page ??.)

196.9 Horizontal mode boxes

`\hbox:n` *(The test suite for this command, and others in this file, is `m3box002.lvt`.)*

Put a horizontal box directly into the input stream.

```

6583 \cs_new_protected:Npn \hbox:n { \tex_hbox:D \scan_stop: }

```

(End definition for \hbox:n. This function is documented on page 123.)

```

\hbox_set:Nn
\hbox_set:cn
6584 \cs_new_protected:Npn \hbox_set:Nn #1#2 { \tex_setbox:D #1 \tex_hbox:D {#2} }
\hbox_gset:Nn
6585 \cs_new_protected:Npn \hbox_gset:Nn { \tex_global:D \hbox_set:Nn }
\hbox_gset:cn
6586 \cs_generate_variant:Nn \hbox_set:Nn { c }
6587 \cs_generate_variant:Nn \hbox_gset:Nn { c }

```

(End definition for \hbox_set:Nn and \hbox_set:cn. These functions are documented on page ??.)

<code>\hbox_set_to_wd:Nnn</code>	Storing material in a horizontal box with a specified width.
<code>\hbox_set_to_wd:cnn</code>	6588 <code>\cs_new_protected:Npn \hbox_set_to_wd:Nnn #1#2#3</code>
<code>\hbox_gset_to_wd:Nnn</code>	6589 <code>{ \tex_setbox:D #1 \tex_hbox:D to \dim_eval:w #2 \dim_eval_end: {#3} }</code>
<code>\hbox_gset_to_wd:cnn</code>	6590 <code>\cs_new_protected:Npn \hbox_gset_to_wd:Nnn</code>
	6591 <code>{ \tex_global:D \hbox_set_to_wd:Nnn }</code>
	6592 <code>\cs_generate_variant:Nn \hbox_set_to_wd:Nnn { c }</code>
	6593 <code>\cs_generate_variant:Nn \hbox_gset_to_wd:Nnn { c }</code>
	<i>(End definition for \hbox_set_to_wd:Nnn and \hbox_set_to_wd:cnn. These functions are documented on page ??.)</i>
 <code>\hbox_set:Nw</code>	Storing material in a horizontal box. This type is useful in environment definitions.
<code>\hbox_set:cw</code>	6594 <code>\cs_new_protected:Npn \hbox_set:Nw #1</code>
<code>\hbox_gset:Nw</code>	6595 <code>{ \tex_setbox:D #1 \tex_hbox:D \c_group_begin_token }</code>
<code>\hbox_gset:cw</code>	6596 <code>\cs_new_protected:Npn \hbox_gset:Nw</code>
<code>\hbox_set_end:</code>	6597 <code>{ \tex_global:D \hbox_set:Nw }</code>
<code>\hbox_gset_end:</code>	6598 <code>\cs_generate_variant:Nn \hbox_set:Nw { c }</code>
	6599 <code>\cs_generate_variant:Nn \hbox_gset:Nw { c }</code>
	6600 <code>\cs_new_eq:NN \hbox_set_end: \c_group_end_token</code>
	6601 <code>\cs_new_eq:NN \hbox_gset_end: \c_group_end_token</code>
	<i>(End definition for \hbox_set:Nw and \hbox_set:cw. These functions are documented on page ??.)</i>
 <code>\hbox_set_inline_begin:N</code>	Renamed September 2011.
<code>\hbox_set_inline_begin:c</code>	6602 <code>\cs_new_eq:NN \hbox_set_inline_begin:N \hbox_set:Nw</code>
<code>\hbox_gset_inline_begin:N</code>	6603 <code>\cs_new_eq:NN \hbox_set_inline_begin:c \hbox_set:cw</code>
<code>\hbox_gset_inline_begin:c</code>	6604 <code>\cs_new_eq:NN \hbox_set_inline_end: \hbox_set_end:</code>
<code>\hbox_set_inline_end:</code>	6605 <code>\cs_new_eq:NN \hbox_gset_inline_begin:N \hbox_gset:Nw</code>
<code>\hbox_gset_inline_end:</code>	6606 <code>\cs_new_eq:NN \hbox_gset_inline_begin:c \hbox_gset:cw</code>
	6607 <code>\cs_new_eq:NN \hbox_gset_inline_end: \hbox_gset_end:</code>
	<i>(End definition for \hbox_set_inline_begin:N and \hbox_set_inline_begin:c. These functions are documented on page ??.)</i>
 <code>\hbox_to_wd:nn</code>	Put a horizontal box directly into the input stream.
<code>\hbox_to_zero:n</code>	6608 <code>\cs_new_protected:Npn \hbox_to_wd:nn #1#2</code>
	6609 <code>{ \tex_hbox:D to \dim_eval:w #1 \dim_eval_end: {#2} }</code>
	6610 <code>\cs_new_protected:Npn \hbox_to_zero:n #1 { \tex_hbox:D to \c_zero_skip {#1} }</code>
	<i>(End definition for \hbox_to_wd:nn. This function is documented on page 124.)</i>
 <code>\hbox_overlap_left:n</code>	Put a zero-sized box with the contents pushed against one side (which makes it stick out
<code>\hbox_overlap_right:n</code>	on the other) directly into the input stream.
	6611 <code>\cs_new_protected:Npn \hbox_overlap_left:n #1</code>
	6612 <code>{ \hbox_to_zero:n { \tex_hss:D #1 } }</code>
	6613 <code>\cs_new_protected:Npn \hbox_overlap_right:n #1</code>
	6614 <code>{ \hbox_to_zero:n { #1 \tex_hss:D } }</code>
	<i>(End definition for \hbox_overlap_left:n. This function is documented on page 124.)</i>

`\hbox_unpack:N` Unpacking a box and if requested also clear it.

`\hbox_unpack:c` 6615 `\cs_new_eq:NN \hbox_unpack:N \tex_unhcopy:D`

`\hbox_unpack_clear:N` 6616 `\cs_new_eq:NN \hbox_unpack_clear:N \tex_unhbox:D`

`\hbox_unpack_clear:c` 6617 `\cs_generate_variant:Nn \hbox_unpack:N { c }`

6618 `\cs_generate_variant:Nn \hbox_unpack_clear:N { c }`

(End definition for `\hbox_unpack:N` and `\hbox_unpack:c`. These functions are documented on page ??.)

196.10 Vertical mode boxes

TeX ends these boxes directly with the internal `end_graf` routine. This means that there is no `\par` at the end of vertical boxes unless we insert one.

`\vbox:n` The following test files are used for this code: `m3box003.lvt`.

`\vbox_top:n` The following test files are used for this code: `m3box003.lvt`.

Put a vertical box directly into the input stream.

6619 `\cs_new_protected:Npn \vbox:n #1 { \tex_vbox:D { #1 \par } }`

6620 `\cs_new_protected:Npn \vbox_top:n #1 { \tex_vtop:D { #1 \par } }`

(End definition for `\vbox:n`. This function is documented on page 125.)

`\vbox_to_ht:nn` Put a vertical box directly into the input stream.

`\vbox_to_zero:n` 6621 `\cs_new_protected:Npn \vbox_to_ht:nn #1#2`

`\vbox_to_ht:nn` 6622 `{ \tex_vbox:D to \dim_eval:w #1 \dim_eval_end: { #2 \par } }`

`\vbox_to_zero:n` 6623 `\cs_new_protected:Npn \vbox_to_zero:n #1`

6624 `{ \tex_vbox:D to \c_zero_dim { #1 \par } }`

(End definition for `\vbox_to_ht:nn` and `\vbox_to_zero:n`. These functions are documented on page 125.)

`\vbox_set:Nn` Storing material in a vertical box with a natural height.

`\vbox_set:cn` 6625 `\cs_new_protected:Npn \vbox_set:Nn #1#2`

`\vbox_gset:Nn` 6626 `{ \tex_setbox:D #1 \tex_vbox:D { #2 \par } }`

`\vbox_gset:cn` 6627 `\cs_new_protected:Npn \vbox_gset:Nn { \tex_global:D \vbox_set:Nn }`

6628 `\cs_generate_variant:Nn \vbox_set:Nn { c }`

6629 `\cs_generate_variant:Nn \vbox_gset:Nn { c }`

(End definition for `\vbox_set:Nn` and `\vbox_set:cn`. These functions are documented on page ??.)

`\vbox_set_top:Nn` Storing material in a vertical box with a natural height and reference point at the baseline

`\vbox_set_top:cn` of the first object in the box.

`\vbox_gset_top:Nn` 6630 `\cs_new_protected:Npn \vbox_set_top:Nn #1#2`

`\vbox_gset_top:cn` 6631 `{ \tex_setbox:D #1 \tex_vtop:D { #2 \par } }`

6632 `\cs_new_protected:Npn \vbox_gset_top:Nn`

6633 `{ \tex_global:D \vbox_set_top:Nn }`

6634 `\cs_generate_variant:Nn \vbox_set_top:Nn { c }`

6635 `\cs_generate_variant:Nn \vbox_gset_top:Nn { c }`

(End definition for `\vbox_set_top:Nn` and `\vbox_set_top:cn`. These functions are documented on page ??.)

`\vbox_set_to_ht:Nnn` Storing material in a vertical box with a specified height.

```

\vbox_set_to_ht:cnn 6636 \cs_new_protected:Npn \vbox_set_to_ht:Nnn #1#2#3
\vbox_gset_to_ht:Nnn 6637 { \tex_setbox:D #1 \tex_vbox:D to \dim_eval:w #2 \dim_eval_end: { #3 \par } }
\vbox_gset_to_ht:cnn 6638 \cs_new_protected:Npn \vbox_gset_to_ht:Nnn
6639 { \tex_global:D \vbox_set_to_ht:Nnn }
6640 \cs_generate_variant:Nn \vbox_set_to_ht:Nnn { c }
6641 \cs_generate_variant:Nn \vbox_gset_to_ht:Nnn { c }

```

(End definition for \vbox_set_to_ht:Nnn and \vbox_set_to_ht:cnn. These functions are documented on page ??.)

`\vbox_set:Nw` Storing material in a vertical box. This type is useful in environment definitions.

```

\vbox_set:cw 6642 \cs_new_protected:Npn \vbox_set:Nw #1
\vbox_gset:Nw 6643 { \tex_setbox:D #1 \tex_vbox:D \c_group_begin_token }
\vbox_gset:cw 6644 \cs_new_protected:Npn \vbox_gset:Nw
\vbox_set_end: 6645 { \tex_global:D \vbox_set:Nw }
\vbox_gset_end: 6646 \cs_generate_variant:Nn \vbox_set:Nw { c }
6647 \cs_generate_variant:Nn \vbox_gset:Nw { c }
6648 \cs_new_protected:Npn \vbox_set_end:
6649 {
6650 \par
6651 \c_group_end_token
6652 }
6653 \cs_new_eq:NN \vbox_gset_end: \vbox_set_end:

```

(End definition for \vbox_set:Nw and \vbox_set:cw. These functions are documented on page ??.)

`\vbox_set_inline_begin:N` Renamed September 2011.

```

\vbox_set_inline_begin:c 6654 \cs_new_eq:NN \vbox_set_inline_begin:N \vbox_set:Nw
\vbox_gset_inline_begin:N 6655 \cs_new_eq:NN \vbox_set_inline_begin:c \vbox_set:cw
\vbox_gset_inline_begin:c 6656 \cs_new_eq:NN \vbox_set_inline_end: \vbox_set_end:
\vbox_set_inline_end: 6657 \cs_new_eq:NN \vbox_gset_inline_begin:N \vbox_gset:Nw
\vbox_gset_inline_end: 6658 \cs_new_eq:NN \vbox_gset_inline_begin:c \vbox_gset:cw
6659 \cs_new_eq:NN \vbox_gset_inline_end: \vbox_gset_end:

```

(End definition for \vbox_set_inline_begin:N and \vbox_set_inline_begin:c. These functions are documented on page ??.)

`\vbox_unpack:N` Unpacking a box and if requested also clear it.

```

\vbox_unpack:c 6660 \cs_new_eq:NN \vbox_unpack:N \tex_unvcopy:D
\vbox_unpack_clear:N 6661 \cs_new_eq:NN \vbox_unpack_clear:N \tex_unvbox:D
\vbox_unpack_clear:c 6662 \cs_generate_variant:Nn \vbox_unpack:N { c }
6663 \cs_generate_variant:Nn \vbox_unpack_clear:N { c }

```

(End definition for \vbox_unpack:N and \vbox_unpack:c. These functions are documented on page ??.)

`\vbox_set_split_to_ht:NNn` Splitting a vertical box in two.

```

6664 \cs_new_protected:Npn \vbox_set_split_to_ht:NNn #1#2#3
6665 { \tex_setbox:D #1 \tex_vsplit:D #2 to \dim_eval:w #3 \dim_eval_end: }

```

(End definition for \vbox_set_split_to_ht:NNn. This function is documented on page 126.)

196.11 Affine transformations

`\l_box_angle_fp` When rotating boxes, the angle itself may be needed by the engine-dependent code. This is done using the `fp` module so that the value is tidied up properly.

```
6666 \fp_new:N \l_box_angle_fp
      (End definition for \l_box_angle_fp. This function is documented on page ??.)
```

`\l_box_cos_fp` These are used to hold the calculated sine and cosine values while carrying out a rotation.

```
\l_box_sin_fp 6667 \fp_new:N \l_box_cos_fp
6668 \fp_new:N \l_box_sin_fp
      (End definition for \l_box_cos_fp and \l_box_sin_fp. These functions are documented on page ??.)
```

`\l_box_top_dim` These are the positions of the four edges of a box before manipulation.

```
\l_box_bottom_dim 6669 \dim_new:N \l_box_top_dim
\l_box_left_dim 6670 \dim_new:N \l_box_bottom_dim
\l_box_right_dim 6671 \dim_new:N \l_box_left_dim
6672 \dim_new:N \l_box_right_dim
      (End definition for \l_box_top_dim and others. These functions are documented on page ??.)
```

`\l_box_top_new_dim` These are the positions of the four edges of a box after manipulation.

```
\l_box_bottom_new_dim 6673 \dim_new:N \l_box_top_new_dim
\l_box_left_new_dim 6674 \dim_new:N \l_box_bottom_new_dim
\l_box_right_new_dim 6675 \dim_new:N \l_box_left_new_dim
6676 \dim_new:N \l_box_right_new_dim
      (End definition for \l_box_top_new_dim and others. These functions are documented on page ??.)
```

`\l_box_tmp_box` Scratch space.

```
\l_box_tmp_fp 6677 \box_new:N \l_box_tmp_box
6678 \fp_new:N \l_box_tmp_fp
      (End definition for \l_box_tmp_box and \l_box_tmp_fp. These functions are documented on page ??.)
```

`\l_box_x_fp` Used as the input and output values for a point when manipulation the location.

```
\l_box_y_fp 6679 \fp_new:N \l_box_x_fp
\l_box_x_new_fp 6680 \fp_new:N \l_box_y_fp
\l_box_y_new_fp 6681 \fp_new:N \l_box_x_new_fp
6682 \fp_new:N \l_box_y_new_fp
      (End definition for \l_box_x_fp and others. These functions are documented on page ??.)
```

`\box_rotate:Nn` Rotation of a box starts with working out the relevant sine and cosine. There is then a check to avoid doing any real work for the trivial rotation.

```
\box_rotate_aux:N 6683 \cs_new_protected:Npn \box_rotate:Nn #1#2
\box_rotate_set_sin_cos: 6684 {
\box_rotate_x:nnN 6685 \hbox_set:Nn #1
\box_rotate_y:nnN 6686 {
\box_rotate_quadrant_one: 6687 \group_begin:
\box_rotate_quadrant_two: 6688 \fp_set:Nn \l_box_angle_fp {#2}
\box_rotate_quadrant_three:
\box_rotate_quadrant_four:
```

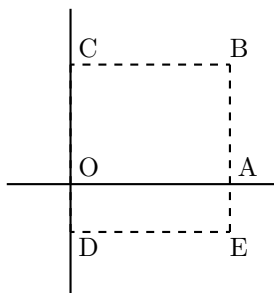


Figure 1: Co-ordinates of a box prior to rotation.

```

6689 \box_rotate_set_sin_cos:
6690 \fp_compare:NNTF \l_box_sin_fp = \c_zero_fp
6691 {
6692     \fp_compare:NNTF \l_box_cos_fp = \c_one_fp
6693     { \box_use:N #1 }
6694     { \box_rotate_aux:N #1 }
6695 }
6696 { \box_rotate_aux:N #1 }
6697 \group_end:
6698 }
6699 }

```

The edges of the box are then recorded: the left edge will always be at zero. Rotation of the four edges then takes place: this is most efficiently done on a quadrant by quadrant basis.

```

6700 \cs_new_protected:Npn \box_rotate_aux:N #1
6701 {
6702     \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6703     \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6704     \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6705     \dim_zero:N \l_box_left_dim

```

The next step is to work out the x and y coordinates of vertices of the rotated box in relation to its original coordinates. The box can be visualized with vertices B , C , D and E is illustrated (Figure 1). The vertex O is the reference point on the baseline, and in this implementation is also the centre of rotation. The formulae are, for a point P and angle α :

$$\begin{aligned}
 P'_x &= P_x - O_x \\
 P'_y &= P_y - O_y \\
 P''_x &= (P'_x \cos(\alpha)) - (P'_y \sin(\alpha)) \\
 P''_y &= (P'_x \sin(\alpha)) + (P'_y \cos(\alpha)) \\
 P'''_x &= P''_x + O_x + L_x \\
 P'''_y &= P''_y + O_y
 \end{aligned}$$

The “extra” horizontal translation L_x at the end is calculated so that the leftmost point of the resulting box has x -coordinate 0. This is desirable as \TeX boxes must have the

reference point at the left edge of the box. (As O is always $(0,0)$, this part of the calculation is omitted here.)

```

6706 \fp_compare:NNTF \l_box_sin_fp > \c_zero_fp
6707 {
6708   \fp_compare:NNTF \l_box_cos_fp > \c_zero_fp
6709   { \box_rotate_quadrant_one: }
6710   { \box_rotate_quadrant_two: }
6711 }
6712 {
6713   \fp_compare:NNTF \l_box_cos_fp < \c_zero_fp
6714   { \box_rotate_quadrant_three: }
6715   { \box_rotate_quadrant_four: }
6716 }

```

The position of the box edges are now known, but the box at this stage be misplaced relative to the current \TeX reference point. So the content of the box is moved such that the reference point of the rotated box will be in the same place as the original.

```

6717 \hbox_set:Nn \l_box_tmp_box { \box_use:N #1 }
6718 \hbox_set:Nn \l_box_tmp_box
6719 {
6720   \tex_kern:D -\l_box_left_new_dim
6721   \hbox:n
6722   {
6723     \driver_box_rotate_begin:
6724     \box_use:N \l_box_tmp_box
6725     \driver_box_rotate_end:
6726   }
6727 }

```

Tidy up the size of the box so that the material is actually inside the bounding box. The result can then be used to reset the original box.

```

6728 \box_set_ht:Nn \l_box_tmp_box { \l_box_top_new_dim }
6729 \box_set_dp:Nn \l_box_tmp_box { -\l_box_bottom_new_dim }
6730 \box_set_wd:Nn \l_box_tmp_box
6731 { \l_box_right_new_dim - \l_box_left_new_dim }
6732 \box_use:N \l_box_tmp_box
6733 }

```

A simple conversion from degrees to radians followed by calculation of the sine and cosine.

```

6734 \cs_new_protected:Npn \box_rotate_set_sin_cos:
6735 {
6736   \fp_set_eq:NN \l_box_tmp_fp \l_box_angle_fp
6737   \fp_div:Nn \l_box_tmp_fp { 180 }
6738   \fp_mul:Nn \l_box_tmp_fp { \c_pi_fp }
6739   \fp_sin:Nn \l_box_sin_fp { \l_box_tmp_fp }
6740   \fp_cos:Nn \l_box_cos_fp { \l_box_tmp_fp }
6741 }

```

These functions take a general point $(\#1,\#2)$ and rotate its location about the origin, using the previously-set sine and cosine values. Each function gives only one component of the location of the updated point. This is because for rotation of a box each step needs

only one value, and so performance is gained by avoiding working out both x' and y' at the same time. Contrast this with the equivalent function in the `l3coffins` module, where both parts are needed.

```

6742 \cs_new_protected:Npn \box_rotate_x:nnN #1#2#3
6743 {
6744   \fp_set_from_dim:Nn \l_box_x_fp {#1}
6745   \fp_set_from_dim:Nn \l_box_y_fp {#2}
6746   \fp_set_eq:NN \l_box_x_new_fp \l_box_x_fp
6747   \fp_set_eq:NN \l_box_tmp_fp \l_box_y_fp
6748   \fp_mul:Nn \l_box_x_new_fp { \l_box_cos_fp }
6749   \fp_mul:Nn \l_box_tmp_fp { \l_box_sin_fp }
6750   \fp_sub:Nn \l_box_x_new_fp { \l_box_tmp_fp }
6751   \dim_set:Nn #3 { \fp_to_dim:N \l_box_x_new_fp }
6752 }
6753 \cs_new_protected:Npn \box_rotate_y:nnN #1#2#3
6754 {
6755   \fp_set_from_dim:Nn \l_box_x_fp {#1}
6756   \fp_set_from_dim:Nn \l_box_y_fp {#2}
6757   \fp_set_eq:NN \l_box_y_new_fp \l_box_y_fp
6758   \fp_set_eq:NN \l_box_tmp_fp \l_box_x_fp
6759   \fp_mul:Nn \l_box_y_new_fp { \l_box_cos_fp }
6760   \fp_mul:Nn \l_box_tmp_fp { \l_box_sin_fp }
6761   \fp_add:Nn \l_box_y_new_fp { \l_box_tmp_fp }
6762   \dim_set:Nn #3 { \fp_to_dim:N \l_box_y_new_fp }
6763 }

```

Rotation of the edges is done using a different formula for each quadrant. In every case, the top and bottom edges only need the resulting y -values, whereas the left and right edges need the x -values. Each case is a question of picking out which corner ends up at with the maximum top, bottom, left and right value. Doing this by hand means a lot less calculating and avoids lots of comparisons.

```

6764 \cs_new_protected:Npn \box_rotate_quadrant_one:
6765 {
6766   \box_rotate_y:nnN \l_box_right_dim \l_box_top_dim
6767   \l_box_top_new_dim
6768   \box_rotate_y:nnN \l_box_left_dim \l_box_bottom_dim
6769   \l_box_bottom_new_dim
6770   \box_rotate_x:nnN \l_box_left_dim \l_box_top_dim
6771   \l_box_left_new_dim
6772   \box_rotate_x:nnN \l_box_right_dim \l_box_bottom_dim
6773   \l_box_right_new_dim
6774 }
6775 \cs_new_protected:Npn \box_rotate_quadrant_two:
6776 {
6777   \box_rotate_y:nnN \l_box_right_dim \l_box_bottom_dim
6778   \l_box_top_new_dim
6779   \box_rotate_y:nnN \l_box_left_dim \l_box_top_dim
6780   \l_box_bottom_new_dim
6781   \box_rotate_x:nnN \l_box_right_dim \l_box_top_dim

```

```

6782     \l_box_left_new_dim
6783     \box_rotate_x:nnN \l_box_left_dim \l_box_bottom_dim
6784     \l_box_right_new_dim
6785 }
6786 \cs_new_protected:Npn \box_rotate_quadrant_three:
6787 {
6788     \box_rotate_y:nnN \l_box_left_dim \l_box_bottom_dim
6789     \l_box_top_new_dim
6790     \box_rotate_y:nnN \l_box_right_dim \l_box_top_dim
6791     \l_box_bottom_new_dim
6792     \box_rotate_x:nnN \l_box_right_dim \l_box_bottom_dim
6793     \l_box_left_new_dim
6794     \box_rotate_x:nnN \l_box_left_dim \l_box_top_dim
6795     \l_box_right_new_dim
6796 }
6797 \cs_new_protected:Npn \box_rotate_quadrant_four:
6798 {
6799     \box_rotate_y:nnN \l_box_left_dim \l_box_top_dim
6800     \l_box_top_new_dim
6801     \box_rotate_y:nnN \l_box_right_dim \l_box_bottom_dim
6802     \l_box_bottom_new_dim
6803     \box_rotate_x:nnN \l_box_left_dim \l_box_bottom_dim
6804     \l_box_left_new_dim
6805     \box_rotate_x:nnN \l_box_right_dim \l_box_top_dim
6806     \l_box_right_new_dim
6807 }

```

(End definition for \box_rotate:Nn. This function is documented on page ??.)

`\l_box_scale_x_fp` Scaling is potentially-different in the two axes.

```

\l_box_scale_y_fp 6808 \fp_new:N \l_box_scale_x_fp
6809 \fp_new:N \l_box_scale_y_fp

```

(End definition for \l_box_scale_x_fp and \l_box_scale_y_fp. These functions are documented on page ??.)

`\box_resize:Nnn` Resizing a box starts by working out the various dimensions of the existing box.

```

\box_resize:cnn 6810 \cs_new_protected:Npn \box_resize:Nnn #1#2#3
\box_resize_aux:Nnn 6811 {
6812     \hbox_set:Nn #1
6813     {
6814         \group_begin:
6815         \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6816         \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6817         \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6818         \dim_zero:N \l_box_left_dim

```

The x -scaling and resulting box size is easy enough to work out: the dimension is that given as #2, and the scale is simply the new width divided by the old one.

```

6819         \fp_set_from_dim:Nn \l_box_scale_x_fp {#2}
6820         \fp_set_from_dim:Nn \l_box_tmp_fp { \l_box_right_dim }
6821         \fp_div:Nn \l_box_scale_x_fp { \l_box_tmp_fp }

```

The y -scaling needs both the height and the depth of the current box.

```

6822         \fp_set_from_dim:Nn \l_box_scale_y_fp {#3}
6823         \fp_set_from_dim:Nn \l_box_tmp_fp
6824         { \l_box_top_dim - \l_box_bottom_dim }
6825         \fp_div:Nn \l_box_scale_y_fp { \l_box_tmp_fp }

```

At this stage, check for trivial scaling. If both scalings are unity, then the code does nothing. Otherwise, pass on to the auxiliary function to find the new dimensions.

```

6826         \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6827         {
6828             \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp
6829             { \box_use:N #1 }
6830             { \box_resize_aux:Nnn #1 {#2} {#3} }
6831         }
6832         { \box_resize_aux:Nnn #1 {#2} {#3} }
6833     \group_end:
6834 }
6835 }
6836 \cs_generate_variant:Nn \box_resize:Nnn { c }

```

With at least one real scaling to do, the next phase is to find the new edge co-ordinates. In the x direction this is relatively easy: just scale the right edge. This is done using the absolute value of the scale so that the new edge is in the correct place. In the y direction, both dimensions have to be scaled, and this again needs the absolute scale value. Once that is all done, the common resize/rescale code can be employed.

```

6837 \cs_new_protected:Npn \box_resize_aux:Nnn #1#2#3
6838 {
6839     \dim_compare:nNnTF {#2} > \c_zero_dim
6840     { \dim_set:Nn \l_box_right_new_dim {#2} }
6841     { \dim_set:Nn \l_box_right_new_dim { \c_zero_dim - ( #2 ) } }
6842     \dim_compare:nNnTF {#3} > \c_zero_dim
6843     {
6844         \dim_set:Nn \l_box_top_new_dim
6845         { \fp_use:N \l_box_scale_y_fp \l_box_top_dim }
6846         \dim_set:Nn \l_box_bottom_new_dim
6847         { \fp_use:N \l_box_scale_y_fp \l_box_bottom_dim }
6848     }
6849     {
6850         \dim_set:Nn \l_box_top_new_dim
6851         { - \fp_use:N \l_box_scale_y_fp \l_box_top_dim }
6852         \dim_set:Nn \l_box_bottom_new_dim
6853         { - \fp_use:N \l_box_scale_y_fp \l_box_bottom_dim }
6854     }
6855     \box_resize_common:N #1
6856 }

```

(End definition for \box_resize:Nnn and \box_resize:cnn. These functions are documented on page ??.)

```

\box_resize_to_ht_plus_dp:Nn
\box_resize_to_ht_plus_dp:cn
\box_resize_to_wd:Nn
\box_resize_to_wd:cn

```

Scaling to a total height or to a width is a simplified version of the main resizing operation, with the scale simply copied between the two parts. The internal auxiliary is called using

the scaling value twice, as the sign for both parts is needed (as this allows the same internal code to be used as for the general case).

```

6857 \cs_new_protected:Npn \box_resize_to_ht_plus_dp:Nn #1#2
6858 {
6859   \hbox_set:Nn #1
6860   {
6861     \group_begin:
6862     \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6863     \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6864     \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6865     \dim_zero:N \l_box_left_dim
6866     \fp_set_from_dim:Nn \l_box_scale_y_fp {#2}
6867     \fp_set_from_dim:Nn \l_box_tmp_fp
6868     { \l_box_top_dim - \l_box_bottom_dim }
6869     \fp_div:Nn \l_box_scale_y_fp { \l_box_tmp_fp }
6870     \fp_set_eq:NN \l_box_scale_x_fp \l_box_scale_y_fp
6871     \fp_compare:NNNTF \l_box_scale_y_fp = \c_one_fp
6872     { \box_use:N #1 }
6873     { \box_resize_aux:Nnn #1 {#2} {#2} }
6874   \group_end:
6875 }
6876 }
6877 \cs_generate_variant:Nn \box_resize_to_ht_plus_dp:Nn { c }
6878 \cs_new_protected:Npn \box_resize_to_wd:Nn #1#2
6879 {
6880   \hbox_set:Nn #1
6881   {
6882     \group_begin:
6883     \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6884     \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6885     \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6886     \dim_zero:N \l_box_left_dim
6887     \fp_set_from_dim:Nn \l_box_scale_x_fp {#2}
6888     \fp_set_from_dim:Nn \l_box_tmp_fp { \l_box_right_dim }
6889     \fp_div:Nn \l_box_scale_x_fp { \l_box_tmp_fp }
6890     \fp_set_eq:NN \l_box_scale_y_fp \l_box_scale_x_fp
6891     \fp_compare:NNNTF \l_box_scale_x_fp = \c_one_fp
6892     { \box_use:N #1 }
6893     { \box_resize_aux:Nnn #1 {#2} {#2} }
6894   \group_end:
6895 }
6896 }
6897 \cs_generate_variant:Nn \box_resize_to_wd:Nn { c }

```

(End definition for \box_resize_to_ht_plus_dp:Nn and \box_resize_to_ht_plus_dp:cn. These functions are documented on page ??.)

\box_scale:Nnn	When scaling a box, setting the scaling itself is easy enough. The new dimensions are
\box_scale:cnn	also relatively easy to find, allowing only for the need to keep them positive in all cases.
\box_scale_aux:Nnn	Once that is done then after a check for the trivial scaling a hand-off can be made

to the common code. The dimension scaling operations are carried out using the \TeX mechanism as it avoids needing to use fp operations.

```

6898 \cs_new_protected:Npn \box_scale:Nnn #1#2#3
6899 {
6900   \hbox_set:Nn #1
6901   {
6902     \group_begin:
6903       \fp_set:Nn \l_box_scale_x_fp {#2}
6904       \fp_set:Nn \l_box_scale_y_fp {#3}
6905       \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6906       \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6907       \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6908       \dim_zero:N \l_box_left_dim
6909       \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6910       {
6911         \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp
6912         { \box_use:N #1 }
6913         { \box_scale_aux:Nnn #1 {#2} {#3} }
6914       }
6915       { \box_scale_aux:Nnn #1 {#2} {#3} }
6916     \group_end:
6917   }
6918 }
6919 \cs_generate_variant:Nn \box_scale:Nnn { c }
6920 \cs_new_protected:Npn \box_scale_aux:Nnn #1#2#3
6921 {
6922   \fp_compare:NNTF \l_box_scale_y_fp > \c_zero_fp
6923   {
6924     \dim_set:Nn \l_box_top_new_dim { #3 \l_box_top_dim }
6925     \dim_set:Nn \l_box_bottom_new_dim { #3 \l_box_bottom_dim }
6926   }
6927   {
6928     \dim_set:Nn \l_box_top_new_dim { -#3 \l_box_bottom_dim }
6929     \dim_set:Nn \l_box_bottom_new_dim { -#3 \l_box_top_dim }
6930   }
6931   \fp_compare:NNTF \l_box_scale_x_fp > \c_zero_fp
6932   { \l_box_right_new_dim #2 \l_box_right_dim }
6933   { \l_box_right_new_dim -#2 \l_box_right_dim }
6934   \box_resize_common:N #1
6935 }

```

(End definition for \box_scale:Nnn and \box_scale:cnn. These functions are documented on page ??.)

\backslash box_resize_common:N The main resize function places in input into a box which will start of with zero width, and includes the handles for engine rescaling.

```

6936 \cs_new_protected:Npn \box_resize_common:N #1
6937 {
6938   \hbox_set:Nn \l_box_tmp_box
6939   {

```



```

6940     \driver_box_scale_begin:
6941     \hbox_overlap_right:n { \box_use:N #1 }
6942     \driver_box_scale_end:
6943 }

```

The new height and depth can be applied directly.

```

6944     \box_set_ht:Nn \l_box_tmp_box { \l_box_top_new_dim }
6945     \box_set_dp:Nn \l_box_tmp_box { \l_box_bottom_new_dim }

```

Things are not quite as obvious for the width, as the reference point needs to remain unchanged. For positive scaling factors resizing the box is all that is needed. However, for case of a negative scaling the material must be shifted such that the reference point ends up in the right place.

```

6946     \fp_compare:NNNTF \l_box_scale_x_fp < \c_zero_fp
6947     {
6948         \hbox_to_wd:nn { \l_box_right_new_dim }
6949         {
6950             \tex_kern:D \l_box_right_new_dim
6951             \box_use:N \l_box_tmp_box
6952             \tex_hss:D
6953         }
6954     }
6955     {
6956         \box_set_wd:Nn \l_box_tmp_box { \l_box_right_new_dim }
6957         \box_use:N \l_box_tmp_box
6958     }
6959 }

```

(End definition for \box_resize_common:N. This function is documented on page ??.)

196.12 Viewing part of a box

`\box_clip:N` A wrapper around the driver-dependent code.

```

\box_clip:c
6960 \cs_new_protected:Npn \box_clip:N #1
6961 { \hbox_set:Nn #1 { \driver_box_use_clip:N #1 } }
6962 \cs_generate_variant:Nn \box_clip:N { c }

```

(End definition for \box_clip:N and \box_clip:c. These functions are documented on page ??.)

`\box_trim:Nnnnn` Trimming from the left- and right-hand edges of the box is easy. The total width is set to remove from the right, and a skip will shift the material to remove from the left.

```

\box_trim:cnnnn
6963 \cs_new_protected:Npn \box_trim:Nnnnn #1#2#3#4#5
6964 {
6965     \box_set_wd:Nn #1 { \box_wd:N #1 - \dim_eval:n {#4} - \dim_eval:n {#2} }
6966     \hbox_set:Nn #1
6967     {
6968         \skip_horizontal:n { - \dim_eval:n {#2} }
6969         \box_use:N #1
6970     }

```

For the height and depth, there is a need to watch the baseline is respected. Material always has to stay on the correct side, so trimming has to check that there is enough material to trim.

```

6971 \dim_compare:nNnTF { \box_dp:N #1 } > {#3}
6972 { \box_set_dp:Nn #1 { \box_dp:N #1 - \dim_eval:n {#3} } }
6973 {
6974   \hbox_set:Nn #1
6975   {
6976     \box_move_down:nn { \dim_eval:n {#3} - \box_dp:N #1 }
6977     { \box_use:N #1 }
6978   }
6979   \box_set_dp:Nn #1 \c_zero_dim
6980 }
6981 \dim_compare:nNnTF { \box_ht:N #1 } > {#5}
6982 { \box_set_ht:Nn #1 { \box_ht:N #1 - \dim_eval:n {#5} } }
6983 {
6984   \hbox_set:Nn #1
6985   {
6986     \box_move_up:nn { \dim_eval:n {#5} - \box_ht:N #1 }
6987     { \box_use:N #1 }
6988   }
6989   \box_set_ht:Nn #1 \c_zero_dim
6990 }
6991 }
6992 \cs_generate_variant:Nn \box_trim:Nnnnn { c }

```

(End definition for \box_trim:Nnnnn and \box_trim:cnnnn. These functions are documented on page ??.)

`\box_viewport:Nnnnn` The same general logic as for clipping, but with absolute dimensions. Thus again width is easy and height is harder.

```

6993 \cs_new_protected:Npn \box_viewport:Nnnnn #1#2#3#4#5
6994 {
6995   \box_set_wd:Nn #1 { \dim_eval:n {#4} - \dim_eval:n {#2} }
6996   \hbox_set:Nn #1
6997   {
6998     \skip_horizontal:n { - \dim_eval:n {#2} }
6999     \box_use:N #1
7000   }
7001   \dim_compare:nNnTF {#3} > \c_zero_dim
7002   {
7003     \hbox_set:Nn #1 { \box_move_down:nn {#3} { \box_use:N #1 } }
7004     \box_set_dp:Nn #1 \c_zero_dim
7005   }
7006   { \box_set_dp:Nn #1 { - \dim_eval:n {#3} } }
7007   \dim_compare:nNnTF {#5} > \c_zero_dim
7008   { \box_set_ht:Nn #1 {#5} }
7009   {
7010     \hbox_set:Nn #1
7011     { \box_move_up:nn { - \dim_eval:n {#5} } { \box_use:N #1 } }

```

```

7012         \box_set_ht:Nn #1 \c_zero_dim
7013     }
7014 }
7015 \cs_generate_variant:Nn \box_viewport:Nnnnn { c }
    (End definition for \box_viewport:Nnnnn and \box_viewport:cnnnn. These functions are docu-
    mented on page ??.)

```

196.13 Deprecated functions

`\l_last_box` Deprecated 2011-11-13, for removal by 2012-02-28.

```

7016 \cs_new_eq:NN \l_last_box \tex_lastbox:D
    (End definition for \l_last_box. This function is documented on page ??.)
7017 </initex | package>

```

197 l3coffins Implementation

```

7018 <*initex | package>
7019 <*package>
7020 \ProvidesExplPackage
7021   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
7022 \package_check_loaded_expl:
7023 </package>

```

197.1 Coffins: data structures and general variables

`\l_coffin_tmp_box` Scratch variables.

```

\l_coffin_tmp_dim 7024 \box_new:N \l_coffin_tmp_box
\l_coffin_tmp_fp   7025 \dim_new:N \l_coffin_tmp_dim
\l_coffin_tmp_tl   7026 \fp_new:N \l_coffin_tmp_fp
                   7027 \tl_new:N \l_coffin_tmp_tl
    (End definition for \l_coffin_tmp_box. This function is documented on page ??.)

```

`\c_coffin_corners_prop` The “corners”; of a coffin define the real content, as opposed to the T_EX bounding box. They all start off in the same place, of course.

```

7028 \prop_new:N \c_coffin_corners_prop
7029 \prop_put:Nnn \c_coffin_corners_prop { tl } { { 0 pt } { 0 pt } }
7030 \prop_put:Nnn \c_coffin_corners_prop { tr } { { 0 pt } { 0 pt } }
7031 \prop_put:Nnn \c_coffin_corners_prop { bl } { { 0 pt } { 0 pt } }
7032 \prop_put:Nnn \c_coffin_corners_prop { br } { { 0 pt } { 0 pt } }
    (End definition for \c_coffin_corners_prop. This function is documented on page ??.)

```

`\c_coffin_poles_prop` Pole positions are given for horizontal, vertical and reference-point based values.

```

7033 \prop_new:N \c_coffin_poles_prop
7034 \tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } { 0 pt } { 1000 pt } }
7035 \prop_put:Nno \c_coffin_poles_prop { l } { \l_coffin_tmp_tl }
7036 \prop_put:Nno \c_coffin_poles_prop { hc } { \l_coffin_tmp_tl }
7037 \prop_put:Nno \c_coffin_poles_prop { r } { \l_coffin_tmp_tl }

```

```

7038 \tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } { 1000 pt } { 0 pt } }
7039 \prop_put:Nno \c_coffin_poles_prop { b } { \l_coffin_tmp_tl }
7040 \prop_put:Nno \c_coffin_poles_prop { vc } { \l_coffin_tmp_tl }
7041 \prop_put:Nno \c_coffin_poles_prop { t } { \l_coffin_tmp_tl }
7042 \prop_put:Nno \c_coffin_poles_prop { B } { \l_coffin_tmp_tl }
7043 \prop_put:Nno \c_coffin_poles_prop { H } { \l_coffin_tmp_tl }
7044 \prop_put:Nno \c_coffin_poles_prop { T } { \l_coffin_tmp_tl }
      (End definition for \c_coffin_poles_prop. This function is documented on page ??.)

\l_coffin_calc_a_fp Used for calculations of intersections and in other internal places.
\l_coffin_calc_b_fp 7045 \fp_new:N \l_coffin_calc_a_fp
\l_coffin_calc_c_fp 7046 \fp_new:N \l_coffin_calc_b_fp
\l_coffin_calc_d_fp 7047 \fp_new:N \l_coffin_calc_c_fp
\l_coffin_calc_result_fp 7048 \fp_new:N \l_coffin_calc_d_fp
7049 \fp_new:N \l_coffin_calc_result_fp
      (End definition for \l_coffin_calc_a_fp. This function is documented on page ??.)

\l_coffin_error_bool For propagating errors so that parts of the code can work around them.
7050 \bool_new:N \l_coffin_error_bool
      (End definition for \l_coffin_error_bool. This function is documented on page ??.)

\l_coffin_offset_x_dim The offset between two sets of coffin handles when typesetting. These values are corrected
\l_coffin_offset_y_dim from those requested in an alignment for the positions of the handles.
7051 \dim_new:N \l_coffin_offset_x_dim
7052 \dim_new:N \l_coffin_offset_y_dim
      (End definition for \l_coffin_offset_x_dim. This function is documented on page ??.)

\l_coffin_pole_a_tl Needed for finding the intersection of two poles.
\l_coffin_pole_b_tl 7053 \tl_new:N \l_coffin_pole_a_tl
7054 \tl_new:N \l_coffin_pole_b_tl
      (End definition for \l_coffin_pole_a_tl. This function is documented on page ??.)

\l_coffin_sin_fp Used for rotations to get the sine and cosine values.
\l_coffin_cos_fp 7055 \fp_new:N \l_coffin_sin_fp
7056 \fp_new:N \l_coffin_cos_fp
      (End definition for \l_coffin_sin_fp. This function is documented on page ??.)

\l_coffin_x_dim For calculating intersections and so forth.
\l_coffin_y_dim 7057 \dim_new:N \l_coffin_x_dim
\l_coffin_x_prime_dim 7058 \dim_new:N \l_coffin_y_dim
\l_coffin_y_prime_dim 7059 \dim_new:N \l_coffin_x_prime_dim
7060 \dim_new:N \l_coffin_y_prime_dim
      (End definition for \l_coffin_x_dim. This function is documented on page ??.)

```

<pre> \l_coffin_x_fp \l_coffin_y_fp \l_coffin_x_prime_fp \l_coffin_y_prime_fp </pre>	<p>Used for calculations where there are clear x- and y-components, for example during vector rotation.</p> <pre> 7061 \fp_new:N \l_coffin_x_fp 7062 \fp_new:N \l_coffin_y_fp 7063 \fp_new:N \l_coffin_x_prime_fp 7064 \fp_new:N \l_coffin_y_prime_fp </pre> <p>(End definition for <code>\l_coffin_x_fp</code>. This function is documented on page ??.)</p>
<pre> \l_coffin_Depth_dim \l_coffin_Height_dim \l_coffin_TotalHeight_dim \l_coffin_Width_dim </pre>	<p>Dimensions for the various parts of a coffin.</p> <pre> 7065 \dim_new:N \l_coffin_Depth_dim 7066 \dim_new:N \l_coffin_Height_dim 7067 \dim_new:N \l_coffin_TotalHeight_dim 7068 \dim_new:N \l_coffin_Width_dim </pre> <p>(End definition for <code>\l_coffin_Depth_dim</code>. This function is documented on page ??.)</p>
<pre> \coffin_saved_Depth: \coffin_saved_Height: \coffin_saved_TotalHeight: \coffin_saved_Width: </pre>	<p>Used to save the meaning of <code>\Depth</code>, <code>\Height</code>, <code>\TotalHeight</code> and <code>\Width</code>.</p> <pre> 7069 \cs_new_nopar:Npn \coffin_saved_Depth: { } 7070 \cs_new_nopar:Npn \coffin_saved_Height: { } 7071 \cs_new_nopar:Npn \coffin_saved_TotalHeight: { } 7072 \cs_new_nopar:Npn \coffin_saved_Width: { } </pre> <p>(End definition for <code>\coffin_saved_Depth:</code>. This function is documented on page ??.)</p>

197.2 Basic coffin functions

There are a number of basic functions needed for creating coffins and placing material in them. This all relies on the following data structures.

`\coffin_if_exist:NT` Several of the higher-level coffin functions will give multiple errors if the coffin does not exist. A cleaner way to handle this is provided here: both the box and the coffin structure are checked.

```

7073 \cs_new_protected:Npn \coffin_if_exist:NT #1#2
7074 {
7075   \cs_if_exist:NTF #1
7076   {
7077     \cs_if_exist:cTF { l_coffin_poles_ \int_value:w #1 _prop }
7078     { #2 }
7079     {
7080       \msg_kernel_error:nnx { coffins } { unknown-coffin }
7081       { \token_to_str:N #1 }
7082     }
7083   }
7084   {
7085     \msg_kernel_error:nnx { coffins } { unknown-coffin }
7086     { \token_to_str:N #1 }
7087   }
7088 }

```

(End definition for `\coffin_if_exist:NT`. This function is documented on page ??.)

`\coffin_clear:N` Clearing coffins means emptying the box and resetting all of the structures.

```
\coffin_clear:c
7089 \cs_new_protected:Npn \coffin_clear:N #1
7090 {
7091   \coffin_if_exist:NT #1
7092   {
7093     \box_clear:N #1
7094     \coffin_reset_structure:N #1
7095   }
7096 }
7097 \cs_generate_variant:Nn \coffin_clear:N { c }
```

(End definition for \coffin_clear:N and \coffin_clear:c. These functions are documented on page ??.)

`\coffin_new:N` Creating a new coffin means making the underlying box and adding the data structures.
`\coffin_new:c` These are created globally, as there is a need to avoid any strange effects if the coffin is created inside a group. This means that the usual rule about `\l_...` variables has to be broken.

```
7098 \cs_new_protected:Npn \coffin_new:N #1
7099 {
7100   \box_new:N #1
7101   \prop_clear_new:c { l_coffin_corners_ \int_value:w #1 _prop }
7102   \prop_clear_new:c { l_coffin_poles_ \int_value:w #1 _prop }
7103   \prop_gset_eq:cN { l_coffin_corners_ \int_value:w #1 _prop }
7104     \c_coffin_corners_prop
7105   \prop_gset_eq:cN { l_coffin_poles_ \int_value:w #1 _prop }
7106     \c_coffin_poles_prop
7107 }
7108 \cs_generate_variant:Nn \coffin_new:N { c }
```

(End definition for \coffin_new:N and \coffin_new:c. These functions are documented on page ??.)

`\hcoffin_set:Nn` Horizontal coffins are relatively easy: set the appropriate box, reset the structures then
`\hcoffin_set:cn` update the handle positions.

```
7109 \cs_new_protected:Npn \hcoffin_set:Nn #1#2
7110 {
7111   \coffin_if_exist:NT #1
7112   {
7113     \hbox_set:Nn #1
7114     {
7115       \color_group_begin:
7116       \color_ensure_current:
7117       #2
7118       \color_group_end:
7119     }
7120     \coffin_reset_structure:N #1
7121     \coffin_update_poles:N #1
7122     \coffin_update_corners:N #1
7123   }
7124 }
```

7125 `\cs_generate_variant:Nn \hcoffin_set:Nn { c }`

(End definition for `\hcoffin_set:Nn` and `\hcoffin_set:cn`. These functions are documented on page ??.)

`\vcoffin_set:Nnn` Setting vertical coffins is more complex. First, the material is typeset with a given width.
`\vcoffin_set:cnn` The default handles and poles are set as for a horizontal coffin, before finding the top baseline using a temporary box.

```

7126 \cs_new_protected:Npn \vcoffin_set:Nnn #1#2#3
7127 {
7128   \coffin_if_exist:NT #1
7129   {
7130     \vbox_set:Nn #1
7131     {
7132       \dim_set:Nn \tex_hsize:D {#2}
7133       \color_group_begin:
7134       \color_ensure_current:
7135       #3
7136       \color_group_end:
7137     }
7138     \coffin_reset_structure:N #1
7139     \coffin_update_poles:N #1
7140     \coffin_update_corners:N #1
7141     \vbox_set_top:Nn \l_coffin_tmp_box { \vbox_unpack:N #1 }
7142     \coffin_set_pole:Nnx #1 { T }
7143     {
7144       { 0 pt }
7145       { \dim_eval:n { \box_ht:N #1 - \box_ht:N \l_coffin_tmp_box } }
7146       { 1000 pt }
7147       { 0 pt }
7148     }
7149     \box_clear:N \l_coffin_tmp_box
7150   }
7151 }
7152 \cs_generate_variant:Nn \vcoffin_set:Nnn { c }

```

(End definition for `\vcoffin_set:Nnn` and `\vcoffin_set:cnn`. These functions are documented on page ??.)

`\hcoffin_set:Nw` These are the “begin”/“end” versions of the above: watch the grouping!
`\hcoffin_set:cw`
`\hcoffin_set_end:`

```

7153 \cs_new_protected:Npn \hcoffin_set:Nw #1
7154 {
7155   \coffin_if_exist:NT #1
7156   {
7157     \hbox_set:Nw #1 \color_group_begin: \color_ensure_current:
7158     \cs_set_protected_nopar:Npn \hcoffin_set_end:
7159     {
7160       \color_group_end:
7161       \hbox_set_end:
7162       \coffin_reset_structure:N #1
7163       \coffin_update_poles:N #1

```

```

7164         \coffin_update_corners:N #1
7165     }
7166 }
7167 }
7168 \cs_new_protected_nopar:Npn \hcoffin_set_end: { }
7169 \cs_generate_variant:Nn \hcoffin_set:Nw { c }

```

(End definition for \hcoffin_set:Nw and \hcoffin_set:cw. These functions are documented on page ??.)

\vcoffin_set:Nnw The same for vertical coffins.

```

\vcoffin_set:cnw 7170 \cs_new_protected:Npn \vcoffin_set:Nnw #1#2
\vcoffin_set_end: 7171 {
7172     \coffin_if_exist:NT #1
7173     {
7174         \vbox_set:Nw #1
7175         \dim_set:Nn \tex_hsize:D {#2}
7176         \color_group_begin: \color_ensure_current:
7177         \cs_set_protected:Npn \vcoffin_set_end:
7178         {
7179             \color_group_end:
7180             \vbox_set_end:
7181             \coffin_reset_structure:N #1
7182             \coffin_update_poles:N #1
7183             \coffin_update_corners:N #1
7184             \vbox_set_top:Nn \l_coffin_tmp_box { \vbox_unpack:N #1 }
7185             \coffin_set_pole:Nnx #1 { T }
7186             {
7187                 { 0 pt }
7188                 {
7189                     \dim_eval:n { \box_ht:N #1 - \box_ht:N \l_coffin_tmp_box }
7190                 }
7191                 { 1000 pt }
7192                 { 0 pt }
7193             }
7194             \box_clear:N \l_coffin_tmp_box
7195         }
7196     }
7197 }
7198 \cs_new_protected_nopar:Npn \vcoffin_set_end: { }
7199 \cs_generate_variant:Nn \vcoffin_set:Nnw { c }

```

(End definition for \vcoffin_set:Nnw and \vcoffin_set:cw. These functions are documented on page ??.)

\coffin_set_eq:NN Setting two coffins equal is just a wrapper around other functions.

```

\coffin_set_eq:Nc 7200 \cs_new_protected:Npn \coffin_set_eq:NN #1#2
\coffin_set_eq:cN 7201 {
\coffin_set_eq:cc 7202     \coffin_if_exist:NT #1
7203     {
7204         \box_set_eq:NN #1 #2

```



```

7205         \coffin_set_eq_structure:NN #1 #2
7206     }
7207 }
7208 \cs_generate_variant:Nn \coffin_set_eq:NN { c , Nc , cc }

```

(End definition for \coffin_set_eq:NN and others. These functions are documented on page ??.)

\c_empty_coffin Special coffins: these cannot be set up earlier as they need \coffin_new:N. The empty coffin is set as a box as the full coffin-setting system needs some material which is not yet available.

\l_coffin_aligned_coffin

\l_coffin_aligned_internal_coffin

```

7209 \coffin_new:N \c_empty_coffin
7210 \hbox_set:Nn \c_empty_coffin { }
7211 \coffin_new:N \l_coffin_aligned_coffin
7212 \coffin_new:N \l_coffin_aligned_internal_coffin

```

(End definition for \c_empty_coffin. This function is documented on page ??.)

197.3 Measuring coffins

\coffin_dp:N Coffins are just boxes when it comes to measurement. However, semantically a separate set of functions are required.

\coffin_dp:c

```

\coffin_ht:N      7213 \cs_new_eq:NN \coffin_dp:N \box_dp:N
\coffin_ht:c      7214 \cs_new_eq:NN \coffin_dp:c \box_dp:c
\coffin_wd:N      7215 \cs_new_eq:NN \coffin_ht:N \box_ht:N
\coffin_wd:c      7216 \cs_new_eq:NN \coffin_ht:c \box_ht:c
                  7217 \cs_new_eq:NN \coffin_wd:N \box_wd:N
                  7218 \cs_new_eq:NN \coffin_wd:c \box_wd:c

```

(End definition for \coffin_dp:N and others. These functions are documented on page ??.)

197.4 Coffins: handle and pole management

\coffin_get_pole:NnN A simple wrapper around the recovery of a coffin pole, with some error checking and recovery built-in.

```

7219 \cs_new_protected:Npn \coffin_get_pole:NnN #1#2#3
7220 {
7221     \prop_get:cnNF
7222     { l_coffin_poles_ \int_value:w #1 _prop } {#2} #3
7223     {
7224         \msg_kernel_error:nxxx { coffins } { unknown-coffin-pole }
7225         {#2} { \token_to_str:N #1 }
7226         \tl_set:Nn #3 { { 0 pt } { 0 pt } { 0 pt } { 0 pt } }
7227     }
7228 }

```

(End definition for \coffin_get_pole:NnN. This function is documented on page ??.)

\coffin_reset_structure:N Resetting the structure is a simple copy job.

```

7229 \cs_new_protected:Npn \coffin_reset_structure:N #1
7230 {
7231     \prop_set_eq:cN { l_coffin_corners_ \int_value:w #1 _prop }
7232     \c_coffin_corners_prop

```

```

7233     \prop_set_eq:cN { l_coffin_poles_ \int_value:w #1 _prop }
7234     \c_coffin_poles_prop
7235 }

```

(End definition for \coffin_reset_structure:N. This function is documented on page ??.)

\coffin_set_eq_structure:NN Setting coffin structures equal simply means copying the property list.

```

\coffin_gset_eq_structure:NN
7236 \cs_new_protected:Npn \coffin_set_eq_structure:NN #1#2
7237 {
7238   \prop_set_eq:cc { l_coffin_corners_ \int_value:w #1 _prop }
7239   { l_coffin_corners_ \int_value:w #2 _prop }
7240   \prop_set_eq:cc { l_coffin_poles_ \int_value:w #1 _prop }
7241   { l_coffin_poles_ \int_value:w #2 _prop }
7242 }
7243 \cs_new_protected:Npn \coffin_gset_eq_structure:NN #1#2
7244 {
7245   \prop_gset_eq:cc { l_coffin_corners_ \int_value:w #1 _prop }
7246   { l_coffin_corners_ \int_value:w #2 _prop }
7247   \prop_gset_eq:cc { l_coffin_poles_ \int_value:w #1 _prop }
7248   { l_coffin_poles_ \int_value:w #2 _prop }
7249 }

```

(End definition for \coffin_set_eq_structure:NN and \coffin_gset_eq_structure:NN. These functions are documented on page ??.)

\coffin_set_user_dimensions:N These make design-level names for the dimensions of a coffin easy to get at.

```

\coffin_end_user_dimensions:
  \Depth
  \Height
  \TotalHeight
  \Width
7250 \cs_new_protected:Npn \coffin_set_user_dimensions:N #1
7251 {
7252   \cs_set_eq:NN \coffin_saved_Height: \Height
7253   \cs_set_eq:NN \coffin_saved_Depth: \Depth
7254   \cs_set_eq:NN \coffin_saved_TotalHeight: \TotalHeight
7255   \cs_set_eq:NN \coffin_saved_Width: \Width
7256   \cs_set_eq:NN \Height \l_coffin_Height_dim
7257   \cs_set_eq:NN \Depth \l_coffin_Depth_dim
7258   \cs_set_eq:NN \TotalHeight \l_coffin_TotalHeight_dim
7259   \cs_set_eq:NN \Width \l_coffin_Width_dim
7260   \dim_set:Nn \Height { \box_ht:N #1 }
7261   \dim_set:Nn \Depth { \box_dp:N #1 }
7262   \dim_set:Nn \TotalHeight { \box_ht:N #1 + \box_dp:N #1 }
7263   \dim_set:Nn \Width { \box_wd:N #1 }
7264 }
7265 \cs_new_protected_nopar:Npn \coffin_end_user_dimensions:
7266 {
7267   \cs_set_eq:NN \Height \coffin_saved_Height:
7268   \cs_set_eq:NN \Depth \coffin_saved_Depth:
7269   \cs_set_eq:NN \TotalHeight \coffin_saved_TotalHeight:
7270   \cs_set_eq:NN \Width \coffin_saved_Width:
7271 }

```

(End definition for \coffin_set_user_dimensions:N. This function is documented on page ??.)

`\coffin_set_horizontal_pole:Nnn`
`\coffin_set_horizontal_pole:cnn`
`\coffin_set_vertical_pole:Nnn`
`\coffin_set_vertical_pole:cnn`
`\coffin_set_pole:Nnn`
`\coffin_set_pole:Nnx`

Setting the pole of a coffin at the user/designer level requires a bit more care. The idea here is to provide a reasonable interface to the system, then to do the setting with full expansion. The three-argument version is used internally to do a direct setting.

```

7272 \cs_new_protected:Npn \coffin_set_horizontal_pole:Nnn #1#2#3
7273 {
7274   \coffin_if_exist:NT #1
7275   {
7276     \coffin_set_user_dimensions:N #1
7277     \coffin_set_pole:Nnx #1 {#2}
7278     {
7279       { 0 pt } { \dim_eval:n {#3} }
7280       { 1000 pt } { 0 pt }
7281     }
7282     \coffin_end_user_dimensions:
7283   }
7284 }
7285 \cs_new_protected:Npn \coffin_set_vertical_pole:Nnn #1#2#3
7286 {
7287   \coffin_if_exist:NT #1
7288   {
7289     \coffin_set_user_dimensions:N #1
7290     \coffin_set_pole:Nnx #1 {#2}
7291     {
7292       { \dim_eval:n {#3} } { 0 pt }
7293       { 0 pt } { 1000 pt }
7294     }
7295     \coffin_end_user_dimensions:
7296   }
7297 }
7298 \cs_new_protected:Npn \coffin_set_pole:Nnn #1#2#3
7299 { \prop_put:cnn { l_coffin_poles_ \int_value:w #1 _prop } {#2} {#3} }
7300 \cs_generate_variant:Nn \coffin_set_horizontal_pole:Nnn { c }
7301 \cs_generate_variant:Nn \coffin_set_vertical_pole:Nnn { c }
7302 \cs_generate_variant:Nn \coffin_set_pole:Nnn { Nnx }

```

(End definition for `\coffin_set_horizontal_pole:Nnn` and `\coffin_set_horizontal_pole:cnn`. These functions are documented on page ??.)

`\coffin_update_corners:N`

Updating the corners of a coffin is straight-forward as at this stage there can be no rotation. So the corners of the content are just those of the underlying \TeX box.

```

7303 \cs_new_protected:Npn \coffin_update_corners:N #1
7304 {
7305   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { tl }
7306   { { 0 pt } { \dim_use:N \box_ht:N #1 } }
7307   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { tr }
7308   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
7309   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { bl }
7310   { { 0 pt } { \dim_eval:n { - \box_dp:N #1 } } }
7311   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { br }
7312   { { \dim_use:N \box_wd:N #1 } { \dim_eval:n { - \box_dp:N #1 } } }

```

7313 }

(End definition for \coffin_update_corners:N. This function is documented on page ??.)

\coffin_update_poles:N This function is called when a coffin is set, and updates the poles to reflect the nature of size of the box. Thus this function only alters poles where the default position is dependent on the size of the box. It also does not set poles which are relevant only to vertical coffins.

```

7314 \cs_new_protected:Npn \coffin_update_poles:N #1
7315 {
7316   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { hc }
7317   {
7318     { \dim_eval:n { 0.5 \box_wd:N #1 } }
7319     { 0 pt } { 0 pt } { 1000 pt }
7320   }
7321   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { r }
7322   {
7323     { \dim_use:N \box_wd:N #1 }
7324     { 0 pt } { 0 pt } { 1000 pt }
7325   }
7326   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { vc }
7327   {
7328     { 0 pt }
7329     { \dim_eval:n { ( \box_ht:N #1 - \box_dp:N #1 ) / 2 } }
7330     { 1000 pt }
7331     { 0 pt }
7332   }
7333   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { t }
7334   {
7335     { 0 pt }
7336     { \dim_use:N \box_ht:N #1 }
7337     { 1000 pt }
7338     { 0 pt }
7339   }
7340   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { b }
7341   {
7342     { 0 pt }
7343     { \dim_eval:n { - \box_dp:N #1 } }
7344     { 1000 pt }
7345     { 0 pt }
7346   }
7347 }
```

(End definition for \coffin_update_poles:N. This function is documented on page ??.)

197.5 Coffins: calculation of pole intersections

\coffin_calculate_intersection:Nmn The lead off in finding intersections is to recover the two poles and then hand off to the auxiliary for the actual calculation. There may of course not be an intersection, for which an error trap is needed.

\coffin_calculate_intersection:nmmmmmm

\coffin_calculate_intersection_aux:nmmmmN

```

7348 \cs_new_protected:Npn \coffin_calculate_intersection:Nnn #1#2#3
7349 {
7350   \coffin_get_pole:NnN #1 {#2} \l_coffin_pole_a_tl
7351   \coffin_get_pole:NnN #1 {#3} \l_coffin_pole_b_tl
7352   \bool_set_false:N \l_coffin_error_bool
7353   \exp_last_two_unbraced:Noo
7354   \coffin_calculate_intersection:nnnnnnnn
7355   \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7356   \bool_if:NT \l_coffin_error_bool
7357   {
7358     \msg_kernel_error:nn { coffins } { no-pole-intersection }
7359     \dim_zero:N \l_coffin_x_dim
7360     \dim_zero:N \l_coffin_y_dim
7361   }
7362 }

```

The two poles passed here each have four values (as dimensions), (a, b, c, d) and (a', b', c', d') . These are arguments 1–4 and 5–8, respectively. In both cases a and b are the co-ordinates of a point on the pole and c and d define the direction of the pole. Finding the intersection depends on the directions of the poles, which are given by d/c and d'/c' . However, if one of the poles is either horizontal or vertical then one or more of c , d , c' and d' will be zero and a special case is needed.

```

7363 \cs_new_protected:Npn \coffin_calculate_intersection:nnnnnnnn
7364 #1#2#3#4#5#6#7#8
7365 {
7366   \dim_compare:nNnTF {#3} = { \c_zero_dim }

```

The case where the first pole is vertical. So the x -component of the interaction will be at a . There is then a test on the second pole: if it is also vertical then there is an error.

```

7367 {
7368   \dim_set:Nn \l_coffin_x_dim {#1}
7369   \dim_compare:nNnTF {#7} = { \c_zero_dim
7370     { \bool_set_true:N \l_coffin_error_bool }

```

The second pole may still be horizontal, in which case the y -component of the intersection will be b' . If not,

$$y = \frac{d'}{c'}(x - a') + b'$$

with the x -component already known to be $\#1$. This calculation is done as a generalised auxiliary.

```

7371 {
7372   \dim_compare:nNnTF {#8} = { \c_zero_dim
7373     { \dim_set:Nn \l_coffin_y_dim {#6} }
7374     {
7375       \coffin_calculate_intersection_aux:nnnnnnN
7376       {#1} {#5} {#6} {#7} {#8} \l_coffin_y_dim
7377     }
7378   }
7379 }

```

If the first pole is not vertical then it may be horizontal. If so, then the procedure is essentially the same as that already done but with the x - and y -components interchanged.

```

7380 {
7381   \dim_compare:nNnTF {#4} = \c_zero_dim
7382   {
7383     \dim_set:Nn \l_coffin_y_dim {#2}
7384     \dim_compare:nNnTF {#8} = { \c_zero_dim }
7385     { \bool_set_true:N \l_coffin_error_bool }
7386     {
7387       \dim_compare:nNnTF {#7} = \c_zero_dim
7388       { \dim_set:Nn \l_coffin_x_dim {#5} }

```

The formula for the case where the second pole is neither horizontal nor vertical is

$$x = \frac{c'}{d'}(y - b') + a'$$

which is again handled by the same auxiliary.

```

7389 {
7390   \coffin_calculate_intersection_aux:nnnnnN
7391   {#2} {#6} {#5} {#8} {#7} \l_coffin_x_dim
7392 }
7393 }
7394 }

```

The first pole is neither horizontal nor vertical. This still leaves the second pole, which may be a special case. For those possibilities, the calculations are the same as above with the first and second poles interchanged.

```

7395 {
7396   \dim_compare:nNnTF {#7} = \c_zero_dim
7397   {
7398     \dim_set:Nn \l_coffin_x_dim {#5}
7399     \coffin_calculate_intersection_aux:nnnnnN
7400     {#5} {#1} {#2} {#3} {#4} \l_coffin_y_dim
7401   }
7402   {
7403     \dim_compare:nNnTF {#8} = \c_zero_dim
7404     {
7405       \dim_set:Nn \l_coffin_x_dim {#6}
7406       \coffin_calculate_intersection_aux:nnnnnN
7407       {#6} {#2} {#1} {#4} {#3} \l_coffin_x_dim
7408     }

```

If none of the special cases apply then there is still a need to check that there is a unique intersection between the two pole. This is the case if they have different slopes.

```

7409 {
7410   \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#3}
7411   \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#4}
7412   \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#7}
7413   \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#8}
7414   \fp_div:Nn \l_coffin_calc_b_fp \l_coffin_calc_a_fp

```

```

7415         \fp_div:Nn \l_coffin_calc_d_fp \l_coffin_calc_c_fp
7416         \fp_compare:nNnTF
7417             \l_coffin_calc_b_fp = \l_coffin_calc_d_fp
7418             { \bool_set_true:N \l_coffin_error_bool }

```

All of the tests pass, so there is the full complexity of the calculation:

$$x = \frac{a(d/c) - a'(d'/c') - b + b'}{(d/c) - (d'/c')}$$

and noting that the two ratios are already worked out from the test just performed. There is quite a bit of shuffling from dimensions to floating points in order to do the work. The y -values is then worked out using the standard auxiliary starting from the x -position.

```

7419         {
7420             \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#6}
7421             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
7422             \fp_sub:Nn \l_coffin_calc_result_fp
7423                 { \l_coffin_calc_a_fp }
7424             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#1}
7425             \fp_mul:Nn \l_coffin_calc_a_fp
7426                 { \l_coffin_calc_b_fp }
7427             \fp_add:Nn \l_coffin_calc_result_fp
7428                 { \l_coffin_calc_a_fp }
7429             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#5}
7430             \fp_mul:Nn \l_coffin_calc_a_fp
7431                 { \l_coffin_calc_d_fp }
7432             \fp_sub:Nn \l_coffin_calc_result_fp
7433                 { \l_coffin_calc_a_fp }
7434             \fp_sub:Nn \l_coffin_calc_b_fp
7435                 { \l_coffin_calc_d_fp }
7436             \fp_div:Nn \l_coffin_calc_result_fp
7437                 { \l_coffin_calc_b_fp }
7438             \dim_set:Nn \l_coffin_x_dim
7439                 { \fp_to_dim:N \l_coffin_calc_result_fp }
7440             \coffin_calculate_intersection_aux:nnnnnN
7441                 { \l_coffin_x_dim }
7442                 {#5} {#6} {#8} {#7} \l_coffin_y_dim
7443         }
7444     }
7445 }
7446 }
7447 }
7448 }

```

The formula for finding the intersection point is in most cases the same. The formula here is

$$\#6 = \frac{\#5}{\#4} (\#1 - \#2) + \#3$$

Thus #4 and #5 should be the directions of the pole while #2 and #3 are co-ordinates.

```

7449 \cs_new_protected:Npn \coffin_calculate_intersection_aux:nnnnnN
7450   #1#2#3#4#5#6
7451   {
7452     \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#1}
7453     \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
7454     \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#3}
7455     \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#4}
7456     \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#5}
7457     \fp_sub:Nn \l_coffin_calc_result_fp { \l_coffin_calc_a_fp }
7458     \fp_div:Nn \l_coffin_calc_result_fp { \l_coffin_calc_d_fp }
7459     \fp_mul:Nn \l_coffin_calc_result_fp { \l_coffin_calc_c_fp }
7460     \fp_add:Nn \l_coffin_calc_result_fp { \l_coffin_calc_b_fp }
7461     \dim_set:Nn #6 { \fp_to_dim:N \l_coffin_calc_result_fp }
7462   }

```

(End definition for `\coffin_calculate_intersection:Nnn`. This function is documented on page ??.)

197.6 Aligning and typesetting of coffins

```

\coffin_join:NnnNnnnn
\coffin_join:cnnNnnnn
\coffin_join:Nnncnnnn
\coffin_join:cnncnnnn

```

This command joins two coffins, using a horizontal and vertical pole from each coffin and making an offset between the two. The result is stored as the as a third coffin, which will have all of its handles reset to standard values. First, the more basic alignment function is used to get things started.

```

7463 \cs_new_protected:Npn \coffin_join:NnnNnnnn #1#2#3#4#5#6#7#8
7464   {
7465     \coffin_align:NnnNnnnnN
7466     #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin

```

Correct the placement of the reference point. If the x -offset is negative then the reference point of the second box is to the left of that of the first, which is corrected using a kern. On the right side the first box might stick out, which will show up if it is wider than the sum of the x -offset and the width of the second box. So a second kern may be needed.

```

7467   \hbox_set:Nn \l_coffin_aligned_coffin
7468   {
7469     \dim_compare:nNnT { \l_coffin_offset_x_dim } < \c_zero_dim
7470     { \tex_kern:D -\l_coffin_offset_x_dim }
7471     \hbox_unpack:N \l_coffin_aligned_coffin
7472     \dim_set:Nn \l_coffin_tmp_dim
7473     { \l_coffin_offset_x_dim - \box_wd:N #1 + \box_wd:N #4 }
7474     \dim_compare:nNnT \l_coffin_tmp_dim < \c_zero_dim
7475     { \tex_kern:D -\l_coffin_tmp_dim }
7476   }

```

The coffin structure is reset, and the corners are cleared: only those from the two parent coffins are needed.

```

7477   \coffin_reset_structure:N \l_coffin_aligned_coffin
7478   \prop_clear:c
7479   { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _ prop }
7480   \coffin_update_poles:N \l_coffin_aligned_coffin

```


The structures of the parent coffins are now transferred to the new coffin, which requires that the appropriate offsets are applied. That will then depend on whether any shift was needed.

```

7481 \dim_compare:nNnTF \l_coffin_offset_x_dim < \c_zero_dim
7482 {
7483   \coffin_offset_poles:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
7484   \coffin_offset_poles:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
7485   \coffin_offset_corners:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
7486   \coffin_offset_corners:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
7487 }
7488 {
7489   \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
7490   \coffin_offset_poles:Nnn #4
7491     { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7492   \coffin_offset_corners:Nnn #1 { 0 pt } { 0 pt }
7493   \coffin_offset_corners:Nnn #4
7494     { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7495 }
7496 \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
7497 \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
7498 }
7499 \cs_generate_variant:Nn \coffin_join:NnnNnnnn { c , Nnnc , cncnc }

```

(End definition for \coffin_join:NnnNnnnn and others. These functions are documented on page ??.)

<pre> \coffin_attach:NnnNnnnn \coffin_attach:cnnNnnnn \coffin_attach:Nnncnnnn \coffin_attach:cncncnnn \coffin_attach_mark:NnnNnnnn </pre>	<p>A more simple version of the above, as it simply uses the size of the first coffin for the new one. This means that the work here is rather simplified compared to the above code.</p> <p>The function used when marking a position is hear also as it is similar but without the structure updates.</p>
---	---

```

7500 \cs_new_protected:Npn \coffin_attach:NnnNnnnn #1#2#3#4#5#6#7#8
7501 {
7502   \coffin_align:NnnNnnnnN
7503     #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
7504   \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
7505   \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
7506   \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
7507   \coffin_reset_structure:N \l_coffin_aligned_coffin
7508   \prop_set_eq:cc
7509     { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7510     { l_coffin_corners_ \int_value:w #1 _prop }
7511   \coffin_update_poles:N \l_coffin_aligned_coffin
7512   \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
7513   \coffin_offset_poles:Nnn #4
7514     { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7515   \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
7516   \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
7517 }
7518 \cs_new_protected:Npn \coffin_attach_mark:NnnNnnnn #1#2#3#4#5#6#7#8
7519 {

```

```

7520 \coffin_align:NnnNnnnnN
7521   #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
7522 \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
7523 \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
7524 \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
7525 \box_set_eq:NN #1 \l_coffin_aligned_coffin
7526 }
7527 \cs_generate_variant:Nn \coffin_attach:NnnNnnnn { c , Nnnc , cnnc }

```

(End definition for \coffin_attach:NnnNnnnn and others. These functions are documented on page ??.)

\coffin_align:NnnNnnnnN The internal function aligns the two coffins into a third one, but performs no corrections on the resulting coffin poles. The process begins by finding the points of intersection for the poles for each of the input coffins. Those for the first coffin are worked out after those for the second coffin, as this allows the ‘primed’ storage area to be used for the second coffin. The ‘real’ box offsets are then calculated, before using these to re-box the input coffins. The default poles are then set up, but the final result will depend on how the bounding box is being handled.

```

7528 \cs_new_protected:Npn \coffin_align:NnnNnnnnN #1#2#3#4#5#6#7#8#9
7529 {
7530   \coffin_calculate_intersection:Nnn #4 {#5} {#6}
7531   \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
7532   \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
7533   \coffin_calculate_intersection:Nnn #1 {#2} {#3}
7534   \dim_set:Nn \l_coffin_offset_x_dim
7535     { \l_coffin_x_dim - \l_coffin_x_prime_dim + #7 }
7536   \dim_set:Nn \l_coffin_offset_y_dim
7537     { \l_coffin_y_dim - \l_coffin_y_prime_dim + #8 }
7538   \hbox_set:Nn \l_coffin_aligned_internal_coffin
7539     {
7540     \box_use:N #1
7541     \tex_kern:D -\box_wd:N #1
7542     \tex_kern:D \l_coffin_offset_x_dim
7543     \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #4 }
7544     }
7545   \coffin_set_eq:NN #9 \l_coffin_aligned_internal_coffin
7546 }

```

(End definition for \coffin_align:NnnNnnnnN. This function is documented on page ??.)

\coffin_offset_poles:Nnn
\coffin_offset_pole:Nnnnnnn Transferring structures from one coffin to another requires that the positions are updated by the offset between the two coffins. This is done by mapping to the property list of the source coffins, moving as appropriate and saving to the new coffin data structures. The test for a - means that the structures from the parent coffins are uniquely labelled and do not depend on the order of alignment. The pay off for this is that - should not be used in coffin pole or handle names, and that multiple alignments do not result in a whole set of values.

```

7547 \cs_new_protected:Npn \coffin_offset_poles:Nnn #1#2#3
7548 {

```

```

7549     \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7550     { \coffin_offset_pole:Nnnnnnn #1 {##1} ##2 {#2} {#3} }
7551   }
7552   \cs_new_protected:Npn \coffin_offset_pole:Nnnnnnn #1#2#3#4#5#6#7#8
7553   {
7554     \dim_set:Nn \l_coffin_x_dim { #3 + #7 }
7555     \dim_set:Nn \l_coffin_y_dim { #4 + #8 }
7556     \tl_if_in:nnTF {#2} { - }
7557     { \tl_set:Nn \l_coffin_tmp_tl { {#2} } }
7558     { \tl_set:Nn \l_coffin_tmp_tl { { #1 - #2 } } }
7559     \exp_last_unbraced:NNo \coffin_set_pole:Nnx \l_coffin_aligned_coffin
7560     { \l_coffin_tmp_tl }
7561     {
7562       { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7563       {#5} {#6}
7564     }
7565   }

```

(End definition for \coffin_offset_poles:Nnn. This function is documented on page ??.)

`\coffin_offset_corners:Nnn` Saving the offset corners of a coffin is very similar, except that there is no need to worry
`\coffin_offset_corners:Nnnnn` about naming: every corner can be saved here as order is unimportant.

```

7566   \cs_new_protected:Npn \coffin_offset_corners:Nnn #1#2#3
7567   {
7568     \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7569     { \coffin_offset_corner:Nnnnn #1 {##1} ##2 {#2} {#3} }
7570   }
7571   \cs_new_protected:Npn \coffin_offset_corner:Nnnnn #1#2#3#4#5#6
7572   {
7573     \prop_put:cnx
7574     { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7575     { #1 - #2 }
7576     {
7577       { \dim_eval:n { #3 + #5 } }
7578       { \dim_eval:n { #4 + #6 } }
7579     }
7580   }

```

(End definition for \coffin_offset_corners:Nnn. This function is documented on page ??.)

`\coffin_update_vertical_poles:NNN` The T and B poles will need to be recalculated after alignment. These functions find the
`\coffin_update_T:nnnnnnnnN` larger absolute value for the poles, but this is of course only logical when the poles are
`\coffin_update_B:nnnnnnnnN` horizontal.

```

7581   \cs_new_protected:Npn \coffin_update_vertical_poles:NNN #1#2#3
7582   {
7583     \coffin_get_pole:NnN #3 { #1 -T } \l_coffin_pole_a_tl
7584     \coffin_get_pole:NnN #3 { #2 -T } \l_coffin_pole_b_tl
7585     \exp_last_two_unbraced:Noo \coffin_update_T:nnnnnnnnN
7586     \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
7587     \coffin_get_pole:NnN #3 { #1 -B } \l_coffin_pole_a_tl
7588     \coffin_get_pole:NnN #3 { #2 -B } \l_coffin_pole_b_tl

```

```

7589 \exp_last_two_unbraced:Noo \coffin_update_B:nnnnnnnnN
7590 \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
7591 }
7592 \cs_new_protected:Npn \coffin_update_T:nnnnnnnnN #1#2#3#4#5#6#7#8#9
7593 {
7594   \dim_compare:nNnTF {#2} < {#6}
7595   {
7596     \coffin_set_pole:Nnx #9 { T }
7597     { { 0 pt } {#6} { 1000 pt } { 0 pt } }
7598   }
7599   {
7600     \coffin_set_pole:Nnx #9 { T }
7601     { { 0 pt } {#2} { 1000 pt } { 0 pt } }
7602   }
7603 }
7604 \cs_new_protected:Npn \coffin_update_B:nnnnnnnnN #1#2#3#4#5#6#7#8#9
7605 {
7606   \dim_compare:nNnTF {#2} < {#6}
7607   {
7608     \coffin_set_pole:Nnx #9 { B }
7609     { { 0 pt } {#2} { 1000 pt } { 0 pt } }
7610   }
7611   {
7612     \coffin_set_pole:Nnx #9 { B }
7613     { { 0 pt } {#6} { 1000 pt } { 0 pt } }
7614   }
7615 }

```

(End definition for `\coffin_update_vertical_poles:NNN`. This function is documented on page ??.)

`\coffin_typeset:Nnnnn` Typesetting a coffin means aligning it with the current position, which is done using a coffin with no content at all. As well as aligning to the empty coffin, there is also a need to leave vertical mode, if necessary.

```

7616 \cs_new_protected:Npn \coffin_typeset:Nnnnn #1#2#3#4#5
7617 {
7618   \coffin_align:NnnNnnnnN \c_empty_coffin { H } { 1 }
7619   #1 {#2} {#3} {#4} {#5} \l_coffin_aligned_coffin
7620   \hbox_unpack:N \c_empty_box
7621   \box_use:N \l_coffin_aligned_coffin
7622 }
7623 \cs_generate_variant:Nn \coffin_typeset:Nnnnn { c }

```

(End definition for `\coffin_typeset:Nnnnn` and `\coffin_typeset:cnnnn`. These functions are documented on page ??.)

197.7 Rotating coffins

`\l_coffin_bounding_prop` A property list for the bounding box of a coffin. This is only needed during the rotation, so there is just the one.

```

7624 \prop_new:N \l_coffin_bounding_prop

```

(End definition for \l_coffin_bounding_prop. This function is documented on page ??.)

\l_coffin_bounding_shift_dim The shift of the bounding box of a coffin from the real content.

7625 \dim_new:N \l_coffin_bounding_shift_dim

(End definition for \l_coffin_bounding_shift_dim. This function is documented on page ??.)

\l_coffin_left_corner_dim These are used to hold maxima for the various corner values: these thus define the
 \l_coffin_right_corner_dim minimum size of the bounding box after rotation.

\l_coffin_bottom_corner_dim 7626 \dim_new:N \l_coffin_left_corner_dim

\l_coffin_top_corner_dim 7627 \dim_new:N \l_coffin_right_corner_dim

7628 \dim_new:N \l_coffin_bottom_corner_dim

7629 \dim_new:N \l_coffin_top_corner_dim

(End definition for \l_coffin_left_corner_dim. This function is documented on page ??.)

\coffin_rotate:Nn Rotating a coffin requires several steps which can be conveniently run together. The
 \coffin_rotate:cn first step is to convert the angle given in degrees to one in radians. This is then used
 to set \l_coffin_sin_fp and \l_coffin_cos_fp, which are carried through unchanged
 for the rest of the procedure.

7630 \cs_new_protected:Npn \coffin_rotate:Nn #1#2

7631 {

7632 \fp_set:Nn \l_coffin_tmp_fp {#2}

7633 \fp_div:Nn \l_coffin_tmp_fp { 180 }

7634 \fp_mul:Nn \l_coffin_tmp_fp { \c_pi_fp }

7635 \fp_sin:Nn \l_coffin_sin_fp { \l_coffin_tmp_fp }

7636 \fp_cos:Nn \l_coffin_cos_fp { \l_coffin_tmp_fp }

The corners and poles of the coffin can now be rotated around the origin. This is best
 achieved using mapping functions.

7637 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }

7638 { \coffin_rotate_corner:Nnnn #1 {##1} ##2 }

7639 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }

7640 { \coffin_rotate_pole:Nnnnnn #1 {##1} ##2 }

The bounding box of the coffin needs to be rotated, and to do this the corners have to be
 found first. They are then rotated in the same way as the corners of the coffin material
 itself.

7641 \coffin_set_bounding:N #1

7642 \prop_map_inline:Nn \l_coffin_bounding_prop

7643 { \coffin_rotate_bounding:nnn {##1} ##2 }

At this stage, there needs to be a calculation to find where the corners of the content
 and the box itself will end up.

7644 \coffin_find_corner_maxima:N #1

7645 \coffin_find_bounding_shift:

7646 \box_rotate:Nn #1 {#2}

The correction of the box position itself takes place here. The idea is that the bounding box for a coffin is tight up to the content, and has the reference point at the bottom-left. The x -direction is handled by moving the content by the difference in the positions of the bounding box and the content left edge. The y -direction is dealt with by moving the box down by any depth it has acquired.

```

7647 \hbox_set:Nn #1
7648 {
7649   \tex_kern:D \l_coffin_bounding_shift_dim
7650   \tex_kern:D -\l_coffin_left_corner_dim
7651   \box_move_down:nn { \l_coffin_bottom_corner_dim }
7652   { \box_use:N #1 }
7653 }

```

If there have been any previous rotations then the size of the bounding box will be bigger than the contents. This can be corrected easily by setting the size of the box to the height and width of the content.

```

7654 \box_set_ht:Nn #1
7655 { \l_coffin_top_corner_dim - \l_coffin_bottom_corner_dim }
7656 \box_set_dp:Nn #1 { 0 pt }
7657 \box_set_wd:Nn #1
7658 { \l_coffin_right_corner_dim - \l_coffin_left_corner_dim }

```

The final task is to move the poles and corners such that they are back in alignment with the box reference point.

```

7659 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7660 { \coffin_shift_corner:Nnnn #1 {##1} ##2 }
7661 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7662 { \coffin_shift_pole:Nnnnnn #1 {##1} ##2 }
7663 }
7664 \cs_generate_variant:Nn \coffin_rotate:Nn { c }

```

(End definition for \coffin_rotate:Nn and \coffin_rotate:cn. These functions are documented on page ??.)

\coffin_set_bounding:N The bounding box corners for a coffin are easy enough to find: this is the same code as for the corners of the material itself, but using a dedicated property list.

```

7665 \cs_new_protected:Npn \coffin_set_bounding:N #1
7666 {
7667   \prop_put:Nnx \l_coffin_bounding_prop { tl }
7668   { { 0 pt } { \dim_use:N \box_ht:N #1 } }
7669   \prop_put:Nnx \l_coffin_bounding_prop { tr }
7670   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
7671   \dim_set:Nn \l_coffin_tmp_dim { - \box_dp:N #1 }
7672   \prop_put:Nnx \l_coffin_bounding_prop { bl }
7673   { { 0 pt } { \dim_use:N \l_coffin_tmp_dim } }
7674   \prop_put:Nnx \l_coffin_bounding_prop { br }
7675   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \l_coffin_tmp_dim } }
7676 }

```

(End definition for \coffin_set_bounding:N. This function is documented on page ??.)

`\coffin_rotate_bounding:nnn` Rotating the position of the corner of the coffin is just a case of treating this as a vector
`\coffin_rotate_corner:Nnnn` from the reference point. The same treatment is used for the corners of the material itself
and the bounding box.

```

7677 \cs_new_protected:Npn \coffin_rotate_bounding:nnn #1#2#3
7678 {
7679   \coffin_rotate_vector:nnNN {#2} {#3} \l_coffin_x_dim \l_coffin_y_dim
7680   \prop_put:Nnx \l_coffin_bounding_prop {#1}
7681   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7682 }
7683 \cs_new_protected:Npn \coffin_rotate_corner:Nnnn #1#2#3#4
7684 {
7685   \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7686   \prop_put:cnx { \l_coffin_corners_ \int_value:w #1 _prop } {#2}
7687   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7688 }

```

(End definition for \coffin_rotate_bounding:nnn. This function is documented on page ??.)

`\coffin_rotate_pole:Nnnnnn` Rotating a single pole simply means shifting the co-ordinate of the pole and its direction.
The rotation here is about the bottom-left corner of the coffin.

```

7689 \cs_new_protected:Npn \coffin_rotate_pole:Nnnnnn #1#2#3#4#5#6
7690 {
7691   \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7692   \coffin_rotate_vector:nnNN {#5} {#6}
7693   \l_coffin_x_prime_dim \l_coffin_y_prime_dim
7694   \coffin_set_pole:Nnx #1 {#2}
7695   {
7696     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7697     { \dim_use:N \l_coffin_x_prime_dim }
7698     { \dim_use:N \l_coffin_y_prime_dim }
7699   }
7700 }

```

(End definition for \coffin_rotate_pole:Nnnnnn. This function is documented on page ??.)

`\coffin_rotate_vector:nnNN` A rotation function, which needs only an input vector (as dimensions) and an output
space. The values `\l_coffin_cos_fp` and `\l_coffin_sin_fp` should previously have
been set up correctly. Working this way means that the floating point work is kept to a
minimum: for any given rotation the sin and cosine values do no change, after all.

```

7701 \cs_new_protected:Npn \coffin_rotate_vector:nnNN #1#2#3#4
7702 {
7703   \fp_set_from_dim:Nn \l_coffin_x_fp {#1}
7704   \fp_set_from_dim:Nn \l_coffin_y_fp {#2}
7705   \fp_set_eq:NN \l_coffin_x_prime_fp \l_coffin_x_fp
7706   \fp_set_eq:NN \l_coffin_y_prime_fp \l_coffin_y_fp
7707   \fp_mul:Nn \l_coffin_x_prime_fp { \l_coffin_cos_fp }
7708   \fp_mul:Nn \l_coffin_y_prime_fp { \l_coffin_sin_fp }
7709   \fp_sub:Nn \l_coffin_x_prime_fp { \l_coffin_tmp_fp }
7710   \fp_set_eq:NN \l_coffin_y_prime_fp \l_coffin_y_fp
7711   \fp_set_eq:NN \l_coffin_tmp_fp \l_coffin_x_fp

```

```

7712     \fp_mul:Nn \l_coffin_y_prime_fp { \l_coffin_cos_fp }
7713     \fp_mul:Nn \l_coffin_tmp_fp      { \l_coffin_sin_fp }
7714     \fp_add:Nn \l_coffin_y_prime_fp { \l_coffin_tmp_fp }
7715     \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_x_prime_fp }
7716     \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_y_prime_fp }
7717 }

```

(End definition for \coffin_rotate_vector:nnNN. This function is documented on page ??.)

\coffin_find_corner_maxima:N The idea here is to find the extremities of the content of the coffin. This is done by
\coffin_find_corner_maxima_aux:nn looking for the smallest values for the bottom and left corners, and the largest values for the top and right corners. The values start at the maximum dimensions so that the case where all are positive or all are negative works out correctly.

```

7718 \cs_new_protected:Npn \coffin_find_corner_maxima:N #1
7719 {
7720     \dim_set:Nn \l_coffin_top_corner_dim { -\c_max_dim }
7721     \dim_set:Nn \l_coffin_right_corner_dim { -\c_max_dim }
7722     \dim_set:Nn \l_coffin_bottom_corner_dim { \c_max_dim }
7723     \dim_set:Nn \l_coffin_left_corner_dim { \c_max_dim }
7724     \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7725     { \coffin_find_corner_maxima_aux:nn ##2 }
7726 }
7727 \cs_new_protected:Npn \coffin_find_corner_maxima_aux:nn #1#2
7728 {
7729     \dim_set_min:Nn \l_coffin_left_corner_dim {#1}
7730     \dim_set_max:Nn \l_coffin_right_corner_dim {#1}
7731     \dim_set_min:Nn \l_coffin_bottom_corner_dim {#2}
7732     \dim_set_max:Nn \l_coffin_top_corner_dim {#2}
7733 }

```

(End definition for \coffin_find_corner_maxima:N. This function is documented on page ??.)

\coffin_find_bounding_shift: The approach to finding the shift for the bounding box is similar to that for the corners.
\coffin_find_bounding_shift_aux:nn However, there is only one value needed here and a fixed input property list, so things are a bit clearer.

```

7734 \cs_new_protected_nopar:Npn \coffin_find_bounding_shift:
7735 {
7736     \dim_set:Nn \l_coffin_bounding_shift_dim { \c_max_dim }
7737     \prop_map_inline:Nn \l_coffin_bounding_prop
7738     { \coffin_find_bounding_shift_aux:nn ##2 }
7739 }
7740 \cs_new_protected:Npn \coffin_find_bounding_shift_aux:nn #1#2
7741 { \dim_set_min:Nn \l_coffin_bounding_shift_dim {#1} }

```

(End definition for \coffin_find_bounding_shift:. This function is documented on page ??.)

\coffin_shift_corner:Nnnn Shifting the corners and poles of a coffin means subtracting the appropriate values from
\coffin_shift_pole:Nnnnnn the x - and y -components. For the poles, this means that the direction vector is unchanged.

```

7742 \cs_new_protected:Npn \coffin_shift_corner:Nnnn #1#2#3#4
7743 {

```



```

7744 \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _ prop } {#2}
7745 {
7746   { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
7747   { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
7748 }
7749 }
7750 \cs_new_protected:Npn \coffin_shift_pole:Nnnnnn #1#2#3#4#5#6
7751 {
7752   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _ prop } {#2}
7753   {
7754     { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
7755     { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
7756     {#5} {#6}
7757   }
7758 }

```

(End definition for \coffin_shift_corner:Nnnn. This function is documented on page ??.)

197.8 Resizing coffins

`\l_coffin_scale_x_fp` Storage for the scaling factors in x and y , respectively.

```

\l_coffin_scale_y_fp 7759 \fp_new:N \l_coffin_scale_x_fp
7760 \fp_new:N \l_coffin_scale_y_fp

```

(End definition for \l_coffin_scale_x_fp. This function is documented on page ??.)

`\l_coffin_scaled_total_height_dim` When scaling, the values given have to be turned into absolute values.

```

\l_coffin_scaled_width_dim 7761 \dim_new:N \l_coffin_scaled_total_height_dim
7762 \dim_new:N \l_coffin_scaled_width_dim

```

(End definition for \l_coffin_scaled_total_height_dim. This function is documented on page ??.)

`\coffin_resize:Nnn` Resizing a coffin begins by setting up the user-friendly names for the dimensions of the coffin box. The new sizes are then turned into scale factor. This is the same operation as takes place for the underlying box, but that operation is grouped and so the same calculation is done here.

`\coffin_resize:cnx`

```

7763 \cs_new_protected:Npn \coffin_resize:Nnn #1#2#3
7764 {
7765   \coffin_set_user_dimensions:N #1
7766   \box_resize:Nnn #1 {#2} {#3}
7767   \fp_set_from_dim:Nn \l_coffin_scale_x_fp {#2}
7768   \fp_set_from_dim:Nn \l_coffin_tmp_fp { \Width }
7769   \fp_div:Nn \l_coffin_scale_x_fp { \l_coffin_tmp_fp }
7770   \fp_set_from_dim:Nn \l_coffin_scale_y_fp {#3}
7771   \fp_set_from_dim:Nn \l_coffin_tmp_fp { \TotalHeight }
7772   \fp_div:Nn \l_coffin_scale_y_fp { \l_coffin_tmp_fp }
7773   \coffin_resize_common:Nnn #1 {#2} {#3}
7774 }
7775 \cs_generate_variant:Nn \coffin_resize:Nnn { c }

```

(End definition for \coffin_resize:Nnn and \coffin_resize:cnx. These functions are documented on page ??.)

`\coffin_resize_common:Nnn` The poles and corners of the coffin are scaled to the appropriate places before actually resizing the underlying box.

```

7776 \cs_new_protected:Npn \coffin_resize_common:Nnn #1#2#3
7777 {
7778   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7779   { \coffin_scale_corner:Nnnn #1 {##1} ##2 }
7780   \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7781   { \coffin_scale_pole:Nnnnnn #1 {##1} ##2 }

```

Negative x -scaling values will place the poles in the wrong location: this is corrected here.

```

7782   \fp_compare:NNNT \l_coffin_scale_x_fp < \c_zero_fp
7783   {
7784     \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7785     { \coffin_x_shift_corner:Nnnn #1 {##1} ##2 }
7786     \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7787     { \coffin_x_shift_pole:Nnnnnn #1 {##1} ##2 }
7788   }
7789   \coffin_end_user_dimensions:
7790 }

```

(End definition for \coffin_resize_common:Nnn. This function is documented on page ??.)

`\coffin_scale:Nnn` For scaling, the opposite calculation is done to find the new dimensions for the coffin.
`\coffin_scale:cnn` Only the total height is needed, as this is the shift required for corners and poles. The scaling is done the T_EX way as this works properly with floating point values without needing to use the fp module.

```

7791 \cs_new_protected:Npn \coffin_scale:Nnn #1#2#3
7792 {
7793   \box_scale:Nnn #1 {#2} {#3}
7794   \coffin_set_user_dimensions:N #1
7795   \fp_set:Nn \l_coffin_scale_x_fp {#2}
7796   \fp_set:Nn \l_coffin_scale_y_fp {#3}
7797   \fp_compare:NNNTF \l_coffin_scale_y_fp > \c_zero_fp
7798   { \l_coffin_scaled_total_height_dim #3 \TotalHeight }
7799   { \l_coffin_scaled_total_height_dim -#3 \TotalHeight }
7800   \fp_compare:NNNTF \l_coffin_scale_x_fp > \c_zero_fp
7801   { \l_coffin_scaled_width_dim -#2 \Width }
7802   { \l_coffin_scaled_width_dim #2 \Width }
7803   \coffin_resize_common:Nnn #1
7804   { \l_coffin_scaled_width_dim } { \l_coffin_scaled_total_height_dim }
7805 }
7806 \cs_generate_variant:Nn \coffin_scale:Nnn { c }

```

(End definition for \coffin_scale:Nnn and \coffin_scale:cnn. These functions are documented on page ??.)

`\coffin_scale_vector:nnNN` This functions scales a vector from the origin using the pre-set scale factors in x and y . This is a much less complex operation than rotation, and as a result the code is a lot clearer.

```

7807 \cs_new_protected:Npn \coffin_scale_vector:nnNN #1#2#3#4

```

```

7808 {
7809   \fp_set_from_dim:Nn \l_coffin_tmp_fp {#1}
7810   \fp_mul:Nn \l_coffin_tmp_fp { \l_coffin_scale_x_fp }
7811   \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_tmp_fp }
7812   \fp_set_from_dim:Nn \l_coffin_tmp_fp {#2}
7813   \fp_mul:Nn \l_coffin_tmp_fp { \l_coffin_scale_y_fp }
7814   \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_tmp_fp }
7815 }

```

(End definition for \coffin_scale_vector:nnNN. This function is documented on page ??.)

\coffin_scale_corner:Nnnn Scaling both corners and poles is a simple calculation using the preceding vector scaling.

```

\coffin_scale_pole:Nnnnnn
7816 \cs_new_protected:Npn \coffin_scale_corner:Nnnn #1#2#3#4
7817 {
7818   \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7819   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7820   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7821 }
7822 \cs_new_protected:Npn \coffin_scale_pole:Nnnnnn #1#2#3#4#5#6
7823 {
7824   \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7825   \coffin_set_pole:Nnx #1 {#2}
7826   {
7827     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7828     {#5} {#6}
7829   }
7830 }

```

(End definition for \coffin_scale_corner:Nnnn. This function is documented on page ??.)

\coffin_x_shift_corner:Nnnn These functions correct for the x displacement that takes place with a negative horizontal
\coffin_x_shift_pole:Nnnnnn scaling.

```

7831 \cs_new_protected:Npn \coffin_x_shift_corner:Nnnn #1#2#3#4
7832 {
7833   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7834   {
7835     { \dim_eval:n { #3 + \box_wd:N #1 } } {#4}
7836   }
7837 }
7838 \cs_new_protected:Npn \coffin_x_shift_pole:Nnnnnn #1#2#3#4#5#6
7839 {
7840   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } {#2}
7841   {
7842     { \dim_eval:n { #3 + \box_wd:N #1 } } {#4}
7843     {#5} {#6}
7844   }
7845 }

```

(End definition for \coffin_x_shift_corner:Nnnn. This function is documented on page ??.)

197.9 Coffin diagnostics

`\l_coffin_display_coffin` Used for printing coffins with data structures attached.

```

\l_coffin_display_coord_coffin 7846 \coffin_new:N \l_coffin_display_coffin
\l_coffin_display_pole_coffin 7847 \coffin_new:N \l_coffin_display_coord_coffin
7848 \coffin_new:N \l_coffin_display_pole_coffin
(End definition for \l_coffin_display_coffin. This function is documented on page ??.)

```

`\l_coffin_display_handles_prop` This property list is used to print coffin handles at suitable positions. The offsets are expressed as multiples of the basic offset value, which therefore acts as a scale-factor.

```

7849 \prop_new:N \l_coffin_display_handles_prop
7850 \prop_put:Nnn \l_coffin_display_handles_prop { tl }
7851 { { b } { r } { -1 } { 1 } }
7852 \prop_put:Nnn \l_coffin_display_handles_prop { thc }
7853 { { b } { hc } { 0 } { 1 } }
7854 \prop_put:Nnn \l_coffin_display_handles_prop { tr }
7855 { { b } { l } { 1 } { 1 } }
7856 \prop_put:Nnn \l_coffin_display_handles_prop { vc1 }
7857 { { vc } { r } { -1 } { 0 } }
7858 \prop_put:Nnn \l_coffin_display_handles_prop { vhc }
7859 { { vc } { hc } { 0 } { 0 } }
7860 \prop_put:Nnn \l_coffin_display_handles_prop { vcr }
7861 { { vc } { l } { 1 } { 0 } }
7862 \prop_put:Nnn \l_coffin_display_handles_prop { bl }
7863 { { t } { r } { -1 } { -1 } }
7864 \prop_put:Nnn \l_coffin_display_handles_prop { bhc }
7865 { { t } { hc } { 0 } { -1 } }
7866 \prop_put:Nnn \l_coffin_display_handles_prop { br }
7867 { { t } { l } { 1 } { -1 } }
7868 \prop_put:Nnn \l_coffin_display_handles_prop { Tl }
7869 { { t } { r } { -1 } { -1 } }
7870 \prop_put:Nnn \l_coffin_display_handles_prop { Thc }
7871 { { t } { hc } { 0 } { -1 } }
7872 \prop_put:Nnn \l_coffin_display_handles_prop { Tr }
7873 { { t } { l } { 1 } { -1 } }
7874 \prop_put:Nnn \l_coffin_display_handles_prop { Hl }
7875 { { vc } { r } { -1 } { 1 } }
7876 \prop_put:Nnn \l_coffin_display_handles_prop { Hhc }
7877 { { vc } { hc } { 0 } { 1 } }
7878 \prop_put:Nnn \l_coffin_display_handles_prop { Hr }
7879 { { vc } { l } { 1 } { 1 } }
7880 \prop_put:Nnn \l_coffin_display_handles_prop { Bl }
7881 { { b } { r } { -1 } { -1 } }
7882 \prop_put:Nnn \l_coffin_display_handles_prop { Bhc }
7883 { { b } { hc } { 0 } { -1 } }
7884 \prop_put:Nnn \l_coffin_display_handles_prop { Br }
7885 { { b } { l } { 1 } { -1 } }

```

(End definition for `\l_coffin_display_handles_prop`. This function is documented on page ??.)

<code>\l_coffin_display_offset_dim</code>	<p>The standard offset for the label from the handle position when displaying handles.</p> <pre> 7886 \dim_new:N \l_coffin_display_offset_dim 7887 \dim_set:Nn \l_coffin_display_offset_dim { 2 pt } (End definition for \l_coffin_display_offset_dim. This function is documented on page ??.) </pre>
<code>\l_coffin_display_x_dim</code> <code>\l_coffin_display_y_dim</code>	<p>As the intersections of poles have to be calculated to find which ones to print, there is a need to avoid repetition. This is done by saving the intersection into two dedicated values.</p> <pre> 7888 \dim_new:N \l_coffin_display_x_dim 7889 \dim_new:N \l_coffin_display_y_dim (End definition for \l_coffin_display_x_dim. This function is documented on page ??.) </pre>
<code>\l_coffin_display_poles_prop</code>	<p>A property list for printing poles: various things need to be deleted from this to get a “nice” output.</p> <pre> 7890 \prop_new:N \l_coffin_display_poles_prop (End definition for \l_coffin_display_poles_prop. This function is documented on page ??.) </pre>
<code>\l_coffin_display_font_tl</code>	<p>Stores the settings used to print coffin data: this keeps things flexible.</p> <pre> 7891 \tl_new:N \l_coffin_display_font_tl 7892 <*initex> 7893 \tl_set:Nn \l_coffin_display_font_tl { } % TODO 7894 </initex> 7895 <*package> 7896 \tl_set:Nn \l_coffin_display_font_tl { \sffamily \tiny } 7897 </package> (End definition for \l_coffin_display_font_tl. This function is documented on page ??.) </pre>
<code>\l_coffin_handles_tmp_prop</code>	<p>Used for displaying coffins, as the handles need to be stored in this case, at least temporarily.</p> <pre> 7898 \prop_new:N \l_coffin_handles_tmp_prop (End definition for \l_coffin_handles_tmp_prop. This function is documented on page ??.) </pre>
<code>\coffin_mark_handle:Nnnn</code> <code>\coffin_mark_handle:cnnn</code> <code>\coffin_mark_handle_aux:nnnnNnn</code>	<p>Marking a single handle is relatively easy. The standard attachment function is used, meaning that there are two calculations for the location. However, this is likely to be okay given the load expected. Contrast with the more optimised version for showing all handles which comes next.</p> <pre> 7899 \cs_new_protected:Npn \coffin_mark_handle:Nnnn #1#2#3#4 7900 { 7901 \hcoffin_set:Nn \l_coffin_display_pole_coffin 7902 { 7903 <*initex> 7904 \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TODO 7905 </initex> 7906 <*package> 7907 \color {#4} 7908 \rule { 1 pt } { 1 pt } 7909 </package> </pre>

```

7910     }
7911     \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
7912     \l_coffin_display_pole_coffin { hc } { vc } { 0 pt } { 0 pt }
7913     \hcoffin_set:Nn \l_coffin_display_coord_coffin
7914     {
7915     <*initex>
7916         % TODO
7917     </initex>
7918     <*package>
7919         \color {#4}
7920     </package>
7921     \l_coffin_display_font_tl
7922     ( \tl_to_str:n { #2 , #3 } )
7923     }
7924     \prop_get:NnN \l_coffin_display_handles_prop
7925     { #2 #3 } \l_coffin_tmp_tl
7926     \quark_if_no_value:NTF \l_coffin_tmp_tl
7927     {
7928         \prop_get:NnN \l_coffin_display_handles_prop
7929         { #3 #2 } \l_coffin_tmp_tl
7930         \quark_if_no_value:NTF \l_coffin_tmp_tl
7931         {
7932             \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
7933             \l_coffin_display_coord_coffin { l } { vc }
7934             { 1 pt } { 0 pt }
7935         }
7936         {
7937             \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
7938             \l_coffin_tmp_tl #1 {#2} {#3}
7939         }
7940     }
7941     {
7942         \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
7943         \l_coffin_tmp_tl #1 {#2} {#3}
7944     }
7945     }
7946     \cs_new_protected:Npn \coffin_mark_handle_aux:nnnnNnn #1#2#3#4#5#6#7
7947     {
7948         \coffin_attach_mark:NnnNnnnn #5 {#6} {#7}
7949         \l_coffin_display_coord_coffin {#1} {#2}
7950         { #3 \l_coffin_display_offset_dim }
7951         { #4 \l_coffin_display_offset_dim }
7952     }
7953     \cs_generate_variant:Nn \coffin_mark_handle:Nnnn { c }

```

(End definition for \coffin_mark_handle:Nnnn and \coffin_mark_handle:cnnn. These functions are documented on page ??.)

\coffin_display_handles:Nn Printing the poles starts by removing any duplicates, for which the H poles is used as
\coffin_display_handles:cn the definitive version for the baseline and bottom. Two loops are then used to find the
\coffin_display_handles_aux:nnnnnn
\coffin_display_handles_aux:nnnn
\coffin_display_attach:Nnnnnn

combinations of handles for all of these poles. This is done such that poles are removed during the loops to avoid duplication.

```

7954 \cs_new_protected:Npn \coffin_display_handles:Nn #1#2
7955 {
7956   \hcoffin_set:Nn \l_coffin_display_pole_coffin
7957   {
7958     \*initex
7959     \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TODO
7960   }
7961   \*package
7962   \color {#2}
7963   \rule { 1 pt } { 1 pt }
7964   \*package
7965 }
7966 \prop_set_eq:Nc \l_coffin_display_poles_prop
7967 { \l_coffin_poles_ \int_value:w #1 _prop }
7968 \coffin_get_pole:NnN #1 { H } \l_coffin_pole_a_tl
7969 \coffin_get_pole:NnN #1 { T } \l_coffin_pole_b_tl
7970 \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7971 { \prop_del:Nn \l_coffin_display_poles_prop { T } }
7972 \coffin_get_pole:NnN #1 { B } \l_coffin_pole_b_tl
7973 \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7974 { \prop_del:Nn \l_coffin_display_poles_prop { B } }
7975 \coffin_set_eq:NN \l_coffin_display_coffin #1
7976 \prop_clear:N \l_coffin_handles_tmp_prop
7977 \prop_map_inline:Nn \l_coffin_display_poles_prop
7978 {
7979   \prop_del:Nn \l_coffin_display_poles_prop {##1}
7980   \coffin_display_handles_aux:nnnnnn {##1} ##2 {#2}
7981 }
7982 \box_use:N \l_coffin_display_coffin
7983 }

```

For each pole there is a check for an intersection, which here does not give an error if none is found. The successful values are stored and used to align the pole coffin with the main coffin for output. The positions are recovered from the preset list if available.

```

7984 \cs_new_protected:Npn \coffin_display_handles_aux:nnnnnn #1#2#3#4#5#6
7985 {
7986   \prop_map_inline:Nn \l_coffin_display_poles_prop
7987   {
7988     \bool_set_false:N \l_coffin_error_bool
7989     \coffin_calculate_intersection:nnnnnnnn {#2} {#3} {#4} {#5} ##2
7990     \bool_if:NF \l_coffin_error_bool
7991     {
7992       \dim_set:Nn \l_coffin_display_x_dim { \l_coffin_x_dim }
7993       \dim_set:Nn \l_coffin_display_y_dim { \l_coffin_y_dim }
7994       \coffin_display_attach:Nnnnn
7995       \l_coffin_display_pole_coffin { hc } { vc }
7996       { 0 pt } { 0 pt }
7997       \hcoffin_set:Nn \l_coffin_display_coord_coffin

```

```

7998         {
7999     <*initex>
8000         % TODO
8001     </initex>
8002     <*package>
8003         \color {#6}
8004     </package>
8005         \l_coffin_display_font_tl
8006         ( \tl_to_str:n { #1 , ##1 } )
8007     }
8008     \prop_get:NnN \l_coffin_display_handles_prop
8009     { #1 ##1 } \l_coffin_tmp_tl
8010     \quark_if_no_value:NTF \l_coffin_tmp_tl
8011     {
8012         \prop_get:NnN \l_coffin_display_handles_prop
8013         { ##1 #1 } \l_coffin_tmp_tl
8014         \quark_if_no_value:NTF \l_coffin_tmp_tl
8015         {
8016             \coffin_display_attach:Nnnnn
8017             \l_coffin_display_coord_coffin { 1 } { vc }
8018             { 1 pt } { 0 pt }
8019         }
8020         {
8021             \exp_last_unbraced:No
8022             \coffin_display_handles_aux:nnnn
8023             \l_coffin_tmp_tl
8024         }
8025     }
8026     {
8027         \exp_last_unbraced:No \coffin_display_handles_aux:nnnn
8028         \l_coffin_tmp_tl
8029     }
8030 }
8031 }
8032 }
8033 \cs_new_protected:Npn \coffin_display_handles_aux:nnnn #1#2#3#4
8034 {
8035     \coffin_display_attach:Nnnnn
8036     \l_coffin_display_coord_coffin {#1} {#2}
8037     { #3 \l_coffin_display_offset_dim }
8038     { #4 \l_coffin_display_offset_dim }
8039 }
8040 \cs_generate_variant:Nn \coffin_display_handles:Nn { c }

```

This is a dedicated version of `\coffin_attach:NnnNnnnn` with a hard-wired first coffin. As the intersection is already known and stored for the display coffin the code simply uses it directly, with no calculation.

```

8041 \cs_new_protected:Npn \coffin_display_attach:Nnnnn #1#2#3#4#5
8042 {
8043     \coffin_calculate_intersection:Nnn #1 {#2} {#3}

```



```

8044 \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
8045 \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
8046 \dim_set:Nn \l_coffin_offset_x_dim
8047 { \l_coffin_display_x_dim - \l_coffin_x_prime_dim + #4 }
8048 \dim_set:Nn \l_coffin_offset_y_dim
8049 { \l_coffin_display_y_dim - \l_coffin_y_prime_dim + #5 }
8050 \hbox_set:Nn \l_coffin_aligned_coffin
8051 {
8052   \box_use:N \l_coffin_display_coffin
8053   \tex_kern:D -\box_wd:N \l_coffin_display_coffin
8054   \tex_kern:D \l_coffin_offset_x_dim
8055   \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #1 }
8056 }
8057 \box_set_ht:Nn \l_coffin_aligned_coffin
8058 { \box_ht:N \l_coffin_display_coffin }
8059 \box_set_dp:Nn \l_coffin_aligned_coffin
8060 { \box_dp:N \l_coffin_display_coffin }
8061 \box_set_wd:Nn \l_coffin_aligned_coffin
8062 { \box_wd:N \l_coffin_display_coffin }
8063 \box_set_eq:NN \l_coffin_display_coffin \l_coffin_aligned_coffin
8064 }

```

(End definition for `\coffin_display_handles:Nn` and `\coffin_display_handles:cn`. These functions are documented on page ??.)

`\coffin_show_structure:N` For showing the various internal structures attached to a coffin in a way that keeps things relatively readable. If there is no apparent structure then the code complains.

```

\coffin_show_structure:c
  \coffin_show_aux:n
  \coffin_show_aux:w
8065 \cs_new_protected:Npn \coffin_show_structure:N #1
8066 {
8067   \cs_if_exist:cTF { l_coffin_poles_ \int_value:w #1 _prop }
8068   {
8069     \iow_term:x
8070     {
8071       \iow_newline:
8072       Size-of~coffin~\token_to_str:N #1 : \iow_newline:
8073       > ~ ht~~\dim_use:N \box_ht:N #1 \iow_newline:
8074       > ~ dp~~\dim_use:N \box_dp:N #1 \iow_newline:
8075       > ~ wd~~\dim_use:N \box_wd:N #1 \iow_newline:
8076     }
8077     \iow_term:x { Poles-of~coffin~\token_to_str:N #1 : }
8078     \tl_set:Nx \l_coffin_tmp_tl
8079     {
8080       \prop_map_function:cn
8081       { l_coffin_poles_ \int_value:w #1 _prop }
8082       \coffin_show_aux:nn
8083     }
8084     \etex_showtokens:D \exp_after:wN \exp_after:wN \exp_after:wN
8085     { \exp_after:wN \coffin_show_aux:w \l_coffin_tmp_tl }
8086   }
8087   {
8088     \iow_term:x { ---No~poles~found--- }

```

```

8089         \tl_show:n { Is~this~really~a~coffin? }
8090     }
8091 }
8092 \cs_new:Npn \coffin_show_aux:nn #1#2
8093 {
8094     \iow_newline: > \c_space_tl \c_space_tl
8095     #1 \c_space_tl \c_space_tl => \c_space_tl \c_space_tl \exp_not:n {#2}
8096 }
8097 \cs_new:Npn \coffin_show_aux:w #1 > ~ { }
8098 \cs_generate_variant:Nn \coffin_show_structure:N { c }

```

(End definition for `\coffin_show_structure:N` and `\coffin_show_structure:c`. These functions are documented on page ??.)

197.10 Messages

```

8099 \msg_kernel_new:nnnn { coffins } { no-pole-intersection }
8100 { No~intersection~between~coffin~poles. }
8101 {
8102     \c_msg_coding_error_text_tl
8103     LaTeX~was~asked~to~find~the~intersection~between~two~poles,~
8104     but~they~do~not~have~a~unique~meeting~point:~
8105     the~value~(0~pt,~0~pt)~will~be~used.
8106 }
8107 \msg_kernel_new:nnnn { coffins } { unknown-coffin }
8108 { Unknown~coffin~'#1'. }
8109 { The~coffin~'#1'~was~never~defined. }
8110 \msg_kernel_new:nnnn { coffins } { unknown-coffin-pole }
8111 { Pole~'#1'~unknown~for~coffin~'#2'. }
8112 {
8113     \c_msg_coding_error_text_tl
8114     LaTeX~was~asked~to~find~a~typesetting~pole~for~a~coffin,~
8115     but~either~the~coffin~does~not~exist~or~the~pole~name~is~wrong.
8116 }
8117 </initex | package>

```

198 l3color Implementation

```

8118 <*initex | package>
8119 <*package>
8120 \ProvidesExplPackage
8121 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
8122 \package_check_loaded_expl:
8123 </package>

```

`\color_group_begin:` Grouping for colour is almost the same as using the basic `\group_begin:` and `\group_end:` functions. However, in vertical mode the end-of-group needs a `\par`, which in horizontal mode does nothing.

```

8124 \cs_new_eq:NN \color_group_begin: \group_begin:
8125 \cs_new_protected_nopar:Npn \color_group_end:

```

```

8126 {
8127     \tex_par:D
8128     \group_end:
8129 }

```

(End definition for \color_group_begin: and \color_group_end:. These functions are documented on page ??.)

\color_ensure_current: A driver-independent wrapper for setting the foreground colour to the current colour “now”.

```

8130 <*initex>
8131 \cs_new_protected_nopar:Npn \color_ensure_current:
8132 { \driver_color_ensure_current: }
8133 </initex>
8134 <*package>
8135 \cs_new_protected_nopar:Npn \color_ensure_current: { \set@color }
8136 </package>
8137 </initex | package>

```

(End definition for \color_ensure_current:. This function is documented on page ??.)

199 l3io implementation

```

8138 <*initex | package>
8139 <*package>
8140 \ProvidesExplPackage
8141 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
8142 \package_check_loaded_expl:
8143 </package>

```

199.1 Primitives

\if_eof:w The primitive conditional

```

8144 \cs_new_eq:NN \if_eof:w \tex_ifeof:D

```

(End definition for \if_eof:w. This function is documented on page 138.)

199.2 Variables and constants

\c_term_ior Reading from the terminal (with a prompt) is done using a positive but non-existent stream number. Unlike writing, there is no concept of reading from the log.

```

8145 \cs_new_eq:NN \c_term_ior \c_sixteen

```

(End definition for \c_term_ior. This function is documented on page ??.)

\c_log_iow Here we allocate two output streams for writing to the transcript file only (**\c_log_iow**) and to both the terminal and transcript file (**\c_term_iow**).

```

8146 \cs_new_eq:NN \c_log_iow \c_minus_one
8147 \cs_new_eq:NN \c_term_iow \c_sixteen

```

(End definition for \c_log_iow and \c_term_iow. These functions are documented on page ??.)

`\c_iow_streams_tl` The list of streams available, by number.

`\c_ior_streams_tl`

```

8148 \tl_const:Nn \c_iow_streams_tl
8149 {
8150   \c_zero
8151   \c_one
8152   \c_two
8153   \c_three
8154   \c_four
8155   \c_five
8156   \c_six
8157   \c_seven
8158   \c_eight
8159   \c_nine
8160   \c_ten
8161   \c_eleven
8162   \c_twelve
8163   \c_thirteen
8164   \c_fourteen
8165   \c_fifteen
8166 }
8167 \cs_new_eq:NN \c_ior_streams_tl \c_iow_streams_tl
      (End definition for \c_iow_streams_tl and \c_ior_streams_tl. These functions are documented
      on page ??.)

```

`\g_iow_streams_prop` The allocations for streams are stored in property lists, which are set up to have a “full”

`\g_ior_streams_prop` set of allocations from the start. In package mode, a few slots are always taken, so these are blocked off from use.

```

8168 \prop_new:N \g_iow_streams_prop
8169 \prop_new:N \g_ior_streams_prop
8170 <*package>
8171 \prop_put:Nnn \g_iow_streams_prop { 0 } { LaTeX2e~reserved }
8172 \prop_put:Nnn \g_iow_streams_prop { 1 } { LaTeX2e~reserved }
8173 \prop_put:Nnn \g_iow_streams_prop { 2 } { LaTeX2e~reserved }
8174 \prop_put:Nnn \g_ior_streams_prop { 0 } { LaTeX2e~reserved }
8175 </package>
      (End definition for \g_iow_streams_prop and \g_ior_streams_prop. These functions are docu-
      mented on page ??.)

```

`\l_iow_stream_int` Used to track the number allocated to the stream being created: this is taken from the

`\l_ior_stream_int` property list but does alter.

```

8176 \int_new:N \l_iow_stream_int
8177 \cs_new_eq:NN \l_ior_stream_int \l_iow_stream_int
      (End definition for \l_iow_stream_int and \l_ior_stream_int. These functions are documented
      on page ??.)

```

199.3 Stream management

`\ior_raw_new:N` The lowest level for stream management is actually creating raw T_EX streams. As these
`\ior_raw_new:c` are very limited (even with ε -T_EX), this should not be addressed directly.
`\iow_raw_new:N`
`\iow_raw_new:c`

```

8178 <*initex>
8179 \alloc_setup_type:nnn { ior } \c_zero \c_sixteen
8180 \cs_new_protected:Npn \ior_raw_new:N #1
8181 { \alloc_reg:nnn { ior } \tex_chardef:D #1 }
8182 \alloc_setup_type:nnn { iow } \c_zero \c_sixteen
8183 \cs_new_protected:Npn \iow_raw_new:N #1
8184 { \alloc_reg:nnn { iow } \tex_chardef:D #1 }
8185 </initex>
8186 <*package>
8187 \cs_set_eq:NN \iow_raw_new:N \newwrite
8188 \cs_set_eq:NN \ior_raw_new:N \newread
8189 </package>
8190 \cs_generate_variant:Nn \ior_raw_new:N { c }
8191 \cs_generate_variant:Nn \iow_raw_new:N { c }

```

(End definition for \ior_raw_new:N and \iow_raw_new:c. These functions are documented on page ??.)

`\ior_new:N` Reserving a new stream is done by defining the name as equal to using the terminal.

```

\ior_new:c      8192 \cs_new_protected:Npn \ior_new:N #1 { \cs_new_eq:NN #1 \c_term_ior }
\iow_new:N      8193 \cs_generate_variant:Nn \ior_new:N { c }
\iow_new:c      8194 \cs_new_protected:Npn \iow_new:N #1 { \cs_new_eq:NN #1 \c_term_iow }
                 8195 \cs_generate_variant:Nn \iow_new:N { c }

```

(End definition for \ior_new:N and others. These functions are documented on page ??.)

`\ior_open:Nn` In both cases, opening a stream starts with a call to the closing function: this is safest.
`\ior_open:cn` There is then a loop through the allocation number list to find the first free stream
`\iow_open:Nn` number. When one is found the allocation can take place, the information can be stored
`\iow_open:cn` and finally the file can actually be opened.

```

8196 \cs_new_protected:Npn \ior_open:Nn #1#2
8197 {
8198   \ior_close:N #1
8199   \int_set:Nn \l_ior_stream_int \c_sixteen
8200   \tl_map_function:NN \c_ior_streams_tl \ior_alloc_read:n
8201   \int_compare:nNnTF \l_ior_stream_int = \c_sixteen
8202   { \msg_kernel_fatal:nn { ior } { streams-exhausted } }
8203   {
8204     \ior_stream_alloc:N #1
8205     \prop_gput:NVn \g_ior_streams_prop \l_ior_stream_int {#2}
8206     \tex_openin:D #1#2 \scan_stop:
8207   }
8208 }
8209 \cs_new_protected:Npn \iow_open:Nn #1#2
8210 {
8211   \iow_close:N #1
8212   \int_set:Nn \l_iow_stream_int \c_sixteen

```

```

8213 \tl_map_function:NN \c_iow_streams_tl \iow_alloc_write:n
8214 \int_compare:nNnTF \l_iow_stream_int = \c_sixteen
8215 { \msg_kernel_fatal:nn { iow } { streams-exhausted } }
8216 {
8217   \iow_stream_alloc:N #1
8218   \prop_gput:NVn \g_iow_streams_prop \l_iow_stream_int {#2}
8219   \tex_immediate:D \tex_openout:D #1#2 \scan_stop:
8220 }
8221 }
8222 \cs_generate_variant:Nn \ior_open:Nn { c }
8223 \cs_generate_variant:Nn \iow_open:Nn { c }
      (End definition for \ior_open:Nn and \iow_open:Nn. These functions are documented on page
      ??.)

```

`\ior_alloc_read:n` These functions are used to see if a particular stream is available. The property list
`\iow_alloc_write:n` contains file names for streams in use, so any unused ones are for the taking.

```

8224 \cs_new_protected:Npn \iow_alloc_write:n #1
8225 {
8226   \prop_if_in:NnF \g_iow_streams_prop {#1}
8227   {
8228     \int_set:Nn \l_iow_stream_int {#1}
8229     \tl_map_break:
8230   }
8231 }
8232 \cs_new_protected:Npn \ior_alloc_read:n #1
8233 {
8234   \prop_if_in:NnF \g_iow_streams_prop {#1}
8235   {
8236     \int_set:Nn \l_iow_stream_int {#1}
8237     \tl_map_break:
8238   }
8239 }
      (End definition for \ior_alloc_read:n. This function is documented on page ??.)

```

`\iow_stream_alloc:N` Allocating a raw stream is much easier in `IniTEX` mode than for the package. For the
`\ior_stream_alloc:N` format, all streams will be allocated by `l3io` and so there is a simple check to see if a
`\iow_stream_alloc_aux:` raw stream is actually available. On the other hand, for the package there will be non-
`\ior_stream_alloc_aux:` managed streams. So if the managed one is not open, a check is made to see if some
`\g_iow_tmp_stream` other managed stream is available before deciding to open a new one. If a new one is
`\g_ior_tmp_stream` needed, we get the number allocated by `LATEX 2ε` to get “back on track” with allocation.

```

8240 \cs_new_protected:Npn \iow_stream_alloc:N #1
8241 {
8242   \cs_if_exist:cTF { g_iow_ \int_use:N \l_iow_stream_int _iow }
8243   { \cs_gset_eq:Nc #1 { g_iow_ \int_use:N \l_iow_stream_int _iow } }
8244   {
8245     <*package>
8246     \iow_stream_alloc_aux:
8247     \int_compare:nNnTF \l_iow_stream_int = \c_sixteen
8248     {

```

```

8249         \iow_raw_new:N \g_iow_tmp_iow
8250         \int_set:Nn \l_iow_stream_int { \g_iow_tmp_iow }
8251         \cs_gset_eq:cN
8252         { \g_iow_ \int_use:N \l_iow_stream_int _iow } \g_iow_tmp_iow
8253     }
8254 </package>
8255 <*initex>
8256     \iow_raw_new:c { \g_iow_ \int_use:N \l_iow_stream_int _iow }
8257 </initex>
8258     \cs_gset_eq:Nc #1 { \g_iow_ \int_use:N \l_iow_stream_int _iow }
8259 }
8260 }
8261 <*package>
8262 \cs_new_protected_nopar:Npn \iow_stream_alloc_aux:
8263 {
8264     \int_incr:N \l_iow_stream_int
8265     \int_compare:nNnT \l_iow_stream_int < \c_sixteen
8266     {
8267         \cs_if_exist:cTF { \g_iow_ \int_use:N \l_iow_stream_int _iow }
8268         {
8269             \prop_if_in:NVT \g_iow_streams_prop \l_iow_stream_int
8270             { \iow_stream_alloc_aux: }
8271         }
8272         { \iow_stream_alloc_aux: }
8273     }
8274 }
8275 </package>
8276 \cs_new_protected:Npn \ior_stream_alloc:N #1
8277 {
8278     \cs_if_exist:cTF { \g_ior_ \int_use:N \l_ior_stream_int _ior }
8279     { \cs_gset_eq:Nc #1 { \g_ior_ \int_use:N \l_ior_stream_int _ior } }
8280     {
8281 <*package>
8282         \ior_stream_alloc_aux:
8283         \int_compare:nNnT \l_ior_stream_int = \c_sixteen
8284         {
8285             \ior_raw_new:N \g_ior_tmp_ior
8286             \int_set:Nn \l_ior_stream_int { \g_ior_tmp_ior }
8287             \cs_gset_eq:cN
8288             { \g_ior_ \int_use:N \l_ior_stream_int _ior } \g_ior_tmp_ior
8289         }
8290 </package>
8291 <*initex>
8292         \ior_raw_new:c { \g_ior_ \int_use:N \l_ior_stream_int _ior }
8293 </initex>
8294         \cs_gset_eq:Nc #1 { \g_ior_ \int_use:N \l_ior_stream_int _ior }
8295     }
8296 }
8297 <*package>
8298 \cs_new_protected_nopar:Npn \ior_stream_alloc_aux:

```

```

8299 {
8300   \int_incr:N \l_ior_stream_int
8301   \int_compare:nNnT \l_ior_stream_int < \c_sixteen
8302   {
8303     \cs_if_exist:cTF { g_ior_ \int_use:N \l_ior_stream_int _ior }
8304     {
8305       \prop_if_in:NVT \g_ior_streams_prop \l_ior_stream_int
8306       { \ior_stream_alloc_aux: }
8307     }
8308     { \ior_stream_alloc_aux: }
8309   }
8310 }
8311 \</package>

```

(End definition for \ior_stream_alloc:N. This function is documented on page ??.)

\ior_close:N Closing a stream is not quite the reverse of opening one. First, the close operation is easier than the open one, and second as the stream is actually a number we can use it directly to show that the slot has been freed up.

```

\ior_close:c
\ior_close:N
\ior_close:c
8312 \cs_new_protected:Npn \ior_close:N #1
8313 {
8314   \cs_if_exist:NT #1
8315   {
8316     \int_compare:nNnF #1 = \c_minus_one
8317     {
8318       \int_compare:nNnF #1 = \c_sixteen
8319       { \tex_closein:D #1 }
8320       \prop_gdel:NV \g_ior_streams_prop #1
8321       \cs_gset_eq:NN #1 \c_term_ior
8322     }
8323   }
8324 }
8325 \cs_new_protected:Npn \ior_close:c #1
8326 {
8327   \cs_if_exist:NT #1
8328   {
8329     \int_compare:nNnF #1 = \c_minus_one
8330     {
8331       \int_compare:nNnF #1 = \c_sixteen
8332       { \tex_closein:D #1 }
8333       \prop_gdel:NV \g_ior_streams_prop #1
8334       \cs_gset_eq:NN #1 \c_term_ior
8335     }
8336   }
8337 }
8338 \cs_generate_variant:Nn \ior_close:N { c }
8339 \cs_generate_variant:Nn \ior_close:c { c }

```

(End definition for \ior_close:N and \ior_close:c. These functions are documented on page ??.)

`\ior_list_streams:` Show the property lists, but with some “pretty printing”. See the `l3msg` module. If there are no open read streams, issue the message `show-no-stream`, and show an empty token list. If there are open read streams, format them with `\msg_aux_show_unbraced:nn`, and with the message `show-open-streams`.

```

8340 \cs_new_protected_nopar:Npn \ior_list_streams:
8341   { \ior_list_streams_aux:Nn \g_ior_streams_prop { ior } }
8342 \cs_new_protected_nopar:Npn \iow_list_streams:
8343   { \ior_list_streams_aux:Nn \g_iow_streams_prop { iow } }
8344 \cs_new_protected:Npn \ior_list_streams_aux:Nn #1#2
8345   {
8346     \msg_aux_use:nn { LaTeX / #2 }
8347     { \prop_if_empty:NTF #1 { show-no-stream } { show-open-streams } }
8348     \msg_aux_show:x
8349     { \prop_map_function:NN #1 \msg_aux_show_unbraced:nn }
8350   }

```

(End definition for \ior_list_streams:. This function is documented on page ??.)

Text for the error messages.

```

8351 \msg_kernel_new:nnnn { iow } { streams-exhausted }
8352   { Output~streams-exhausted }
8353   {
8354     TeX~can~only~open~up~to~16~output~streams~at~one~time.\\
8355     All~16 are currently~in~use,~and~something~wanted~to~open
8356     another~one.
8357   }
8358 \msg_kernel_new:nnnn { ior } { streams-exhausted }
8359   { Input~streams-exhausted }
8360   {
8361     TeX~can~only~open~up~to~16~input~streams~at~one~time.\\
8362     All~16 are currently~in~use,~and~something~wanted~to~open
8363     another~one.
8364   }

```

199.4 Deferred writing

`\iow_shipout_x:Nn` First the easy part, this is the primitive.

`\iow_shipout_x:Nx`

```

8365 \cs_new_eq:NN \iow_shipout_x:Nn \tex_write:D
8366 \cs_generate_variant:Nn \iow_shipout_x:Nn { Nx }

```

(End definition for \iow_shipout_x:Nn and \iow_shipout_x:Nx. These functions are documented on page ??.)

`\iow_shipout:Nn` With ε -TeX available deferred writing is easy.

`\iow_shipout:Nx`

```

8367 \cs_new_protected:Npn \iow_shipout:Nn #1#2
8368   { \iow_shipout_x:Nn #1 { \exp_not:n {#2} } }
8369 \cs_generate_variant:Nn \iow_shipout:Nn { Nx }

```

(End definition for \iow_shipout:Nn and \iow_shipout:Nx. These functions are documented on page ??.)

199.5 Immediate writing

`\iow_now:Nx` An abbreviation for an often used operation, which immediately writes its second argument expanded to the output stream.

```
8370 \cs_new_protected_nopar:Npn \iow_now:Nx { \tex_immediate:D \iow_shipout_x:Nn }
```

(End definition for \iow_now:Nx. This function is documented on page ??.)

`\iow_now:Nn` This routine writes the second argument onto the output stream without expansion. If this stream isn't open, the output goes to the terminal instead. If the first argument is no output stream at all, we get an internal error.

```
8371 \cs_new_protected:Npn \iow_now:Nn #1#2
8372 { \iow_now:Nx #1 { \exp_not:n {#2} } }
```

(End definition for \iow_now:Nn. This function is documented on page 134.)

`\iow_log:n` Writing to the log and the terminal directly are relatively easy.

```
\iow_log:x 8373 \cs_set_protected_nopar:Npn \iow_log:x { \iow_now:Nx \c_log_iow }
\iow_term:n 8374 \cs_new_protected_nopar:Npn \iow_log:n { \iow_now:Nn \c_log_iow }
\iow_term:x 8375 \cs_set_protected_nopar:Npn \iow_term:x { \iow_now:Nx \c_term_iow }
8376 \cs_new_protected_nopar:Npn \iow_term:n { \iow_now:Nn \c_term_iow }
```

(End definition for \iow_log:n and \iow_log:x. These functions are documented on page ??.)

`\iow_now_when_avail:Nn` For writing only if the stream requested is open at all.

```
\iow_now_when_avail:Nx 8377 \cs_new_protected:Npn \iow_now_when_avail:Nn #1
8378 { \cs_if_free:NTF #1 { \use_none:n } { \iow_now:Nn #1 } }
8379 \cs_new_protected:Npn \iow_now_when_avail:Nx #1
8380 { \cs_if_free:NTF #1 { \use_none:n } { \iow_now:Nx #1 } }
```

(End definition for \iow_now_when_avail:Nn and \iow_now_when_avail:Nx. These functions are documented on page ??.)

199.6 Special characters for writing

`\iow_newline:` Global variable holding the character that forces a new line when something is written to an output stream

```
8381 \cs_new_nopar:Npn \iow_newline: { ^^J }
```

(End definition for \iow_newline:. This function is documented on page ??.)

`\iow_char:N` Function to write any escaped char to an output stream.

```
8382 \cs_new_eq:NN \iow_char:N \cs_to_str:N
```

(End definition for \iow_char:N. This function is documented on page 135.)

199.7 Hard-wrapping lines based on length

The code here implements a generic hard-wrapping function. This is used by the messaging system, but is designed such that it is available for other uses.

<code>\l_iow_line_length_int</code>	<p>This is the “raw” length of a line which can be written to a file. The standard value is the line length typically used by \TeXLive and \MikTeX.</p> <pre> 8383 \int_new:N \l_iow_line_length_int 8384 \int_set:Nn \l_iow_line_length_int { 78 } (End definition for \l_iow_line_length_int. This function is documented on page 136.) </pre>
<code>\l_iow_target_length_int</code>	<p>This stores the target line length: the full length minus any part for a leader at the start of each line.</p> <pre> 8385 \int_new:N \l_iow_target_length_int (End definition for \l_iow_target_length_int. This function is documented on page ??.) </pre>
<code>\l_iow_current_line_int</code> <code>\l_iow_current_word_int</code> <code>\l_iow_current_indentation_int</code>	<p>These store the number of characters in the line and word currently being constructed, and the current indentation, respectively.</p> <pre> 8386 \int_new:N \l_iow_current_line_int 8387 \int_new:N \l_iow_current_word_int 8388 \int_new:N \l_iow_current_indentation_int (End definition for \l_iow_current_line_int, \l_iow_current_word_int, and \l_iow_current_indentation_int. These functions are documented on page ??.) </pre>
<code>\l_iow_current_line_tl</code> <code>\l_iow_current_word_tl</code> <code>\l_iow_current_indentation_tl</code>	<p>These hold the current line of text and current word, and a number of spaces for indentation, respectively.</p> <pre> 8389 \tl_new:N \l_iow_current_line_tl 8390 \tl_new:N \l_iow_current_word_tl 8391 \tl_new:N \l_iow_current_indentation_tl (End definition for \l_iow_current_line_tl, \l_iow_current_word_tl, and \l_iow_current_indentation_tl. These functions are documented on page ??.) </pre>
<code>\l_iow_wrap_tl</code>	<p>Used for the expansion step before detokenizing.</p> <pre> 8392 \tl_new:N \l_iow_wrap_tl (End definition for \l_iow_wrap_tl. This function is documented on page ??.) </pre>
<code>\l_iow_wrapped_tl</code>	<p>The output from wrapping text: fully expanded and with lines which are not overly long.</p> <pre> 8393 \tl_new:N \l_iow_wrapped_tl (End definition for \l_iow_wrapped_tl. This function is documented on page ??.) </pre>
<code>\l_iow_line_start_bool</code>	<p>Boolean to avoid adding a space at the beginning of forced newlines.</p> <pre> 8394 \bool_new:N \l_iow_line_start_bool (End definition for \l_iow_line_start_bool. This function is documented on page ??.) </pre>

`\c_catcode_other_space_tl` Lowercase a character with category code 12 to produce an “other” space. We can do everything within the group, because `\tl_const:Nn` defines its argument globally.

```
8395 \group_begin:
8396   \char_set_catcode_other:N \*
8397   \char_set_lccode:n { '\* } { '\ }
8398   \tl_to_lowercase:n { \tl_const:Nn \c_catcode_other_space_tl { * } }
8399 \group_end:
      (End definition for \c_catcode_other_space_tl. This function is documented on page 136.)
```

`\c_iow_wrap_marker_tl` Every special action of the wrapping code is preceeded by the same recognizable string, `\c_iow_wrap_end_marker_tl` `\c_iow_wrap_marker_tl`. Upon seeing that “word”, the wrapping code reads one space-delimited argument to know what operation to perform. The setting of `\escapechar` here is not very important, but makes `\c_iow_wrap_marker_tl` look nicer. Note that `\iow_wrap_new_marker:n` does not survive the group, but all constants are defined globally.

```
8400 \group_begin:
8401   \int_set_eq:NN \tex_escapechar:D \c_minus_one
8402   \tl_const:Nx \c_iow_wrap_marker_tl
8403     { \tl_to_str:n { \^^I \^^O \^^W \^^_ \^^W \^^R \^^A \^^P } }
8404   \cs_set:Npn \iow_wrap_new_marker:n #1
8405     {
8406       \tl_const:cx { c_iow_wrap_ #1 _marker_tl }
8407       {
8408         \c_catcode_other_space_tl
8409         \c_iow_wrap_marker_tl
8410         \c_catcode_other_space_tl
8411         #1
8412         \c_catcode_other_space_tl
8413       }
8414     }
8415   \iow_wrap_new_marker:n { end }
8416   \iow_wrap_new_marker:n { newline }
8417   \iow_wrap_new_marker:n { indent }
8418   \iow_wrap_new_marker:n { unindent }
8419 \group_end:
      (End definition for \c_iow_wrap_marker_tl. This function is documented on page ??.)
```

`\iow_indent:n` We give a dummy (protected) definition to `\iow_indent:n` when outside messages. `\iow_indent_expandable:n` Within wrapped message, it places the instruction for increasing the indentation before its argument, and the instruction for unindenting afterwards. Note that there will be no forced line-break, so the indentation only changes when the next line is started.

```
8420 \cs_new_protected:Npn \iow_indent:n #1 { }
8421 \cs_new:Npx \iow_indent_expandable:n #1
8422   {
8423     \c_iow_wrap_indent_marker_tl
8424     #1
8425     \c_iow_wrap_unindent_marker_tl
8426   }
      (End definition for \iow_indent:n. This function is documented on page ??.)
```

`\iow_wrap:xnnnN` The main wrapping function works as follows. The target number of characters in a line is calculated, before fully-expanding the input such that `\\` and `_` are converted into the appropriate values. There is then a loop over each word in the input, which will do the actual wrapping. After the loop, the resulting text is passed on to the function which has been given as a post-processor. The argument `#4` is available for additional set up steps for the output. The definition of `\\` and `_` use an “other” space rather than a normal space, because the latter might be absorbed by TeX to end a number or other f-type expansions. The `\tl_to_str:N` step converts the “other” space back to a normal space.

```

8427 \cs_new_protected:Npn \iow_wrap:xnnnN #1#2#3#4#5
8428 {
8429   \group_begin:
8430     \int_set:Nn \l_iow_target_length_int { \l_iow_line_length_int - ( #3 ) }
8431     \int_zero:N \l_iow_current_indentation_int
8432     \tl_clear:N \l_iow_current_indentation_tl
8433     \int_zero:N \l_iow_current_line_int
8434     \tl_clear:N \l_iow_current_line_tl
8435     \tl_clear:N \l_iow_wrap_tl
8436     \bool_set_true:N \l_iow_line_start_bool
8437     \int_set_eq:NN \tex_escapechar:D \c_minus_one
8438     \cs_set_nopar:Npx \{ { \token_to_str:N \{ }
8439     \cs_set_nopar:Npx \# { \token_to_str:N \# }
8440     \cs_set_nopar:Npx \} { \token_to_str:N \} }
8441     \cs_set_nopar:Npx \% { \token_to_str:N \% }
8442     \cs_set_nopar:Npx \~ { \token_to_str:N \~ }
8443     \int_set:Nn \tex_escapechar:D { 92 }
8444     \cs_set_eq:NN \\ \c_iow_wrap_newline_marker_tl
8445     \cs_set_eq:NN \_ \c_catcode_other_space_tl
8446     \cs_set_eq:NN \iow_indent:n \iow_indent_expandable:n
8447     #4
8448     <*initex>
8449     \tl_set:Nx \l_iow_wrap_tl {#1}
8450     </initex>
8451     <*package>
8452     \protected@edef \l_iow_wrap_tl {#1}
8453     </package>
8454     \cs_set:Npn \\ { \iow_newline: #2 }
8455     \use:x
8456     {
8457       \iow_wrap_loop:w
8458       \tl_to_str:N \l_iow_wrap_tl
8459       \tl_to_str:N \c_iow_wrap_end_marker_tl
8460       \c_space_tl \c_space_tl
8461       \exp_not:N \q_stop
8462     }
8463     \exp_args:NNo \group_end:
8464     #5 \l_iow_wrapped_tl
8465   }

```

(End definition for `\iow_wrap:xnnnN`. This function is documented on page 136.)

`\iow_wrap_loop:w` The loop grabs one word in the input, and checks whether it is the special marker, or a normal word.

```

8466 \cs_new_protected:Npn \iow_wrap_loop:w #1 ~ %
8467 {
8468   \tl_set:Nn \l_iow_current_word_tl {#1}
8469   \tl_if_eq:NNTF \l_iow_current_word_tl \c_iow_wrap_marker_tl
8470     { \iow_wrap_special:w }
8471     { \iow_wrap_word: }
8472 }

```

(End definition for \iow_wrap_loop:w. This function is documented on page ??.)

`\iow_wrap_word:` For a normal word, update the line length, then test if the current word would fit in the current line, and call the appropriate function. If the word fits in the current line, `\iow_wrap_word_fits:` add it to the line, preceded by a space unless it is the first word of the line. Otherwise, `\iow_wrap_word_newline:` the current line is added to the result, with the run-on text. The current word (and its length) are then put in the new line.

```

8473 \cs_new_protected_nopar:Npn \iow_wrap_word:
8474 {
8475   \int_set:Nn \l_iow_current_word_int
8476     { \str_length_skip_spaces:N \l_iow_current_word_tl }
8477   \int_add:Nn \l_iow_current_line_int { \l_iow_current_word_int }
8478   \int_compare:nNnTF \l_iow_current_line_int < \l_iow_target_length_int
8479     { \iow_wrap_word_fits: }
8480     { \iow_wrap_word_newline: }
8481   \iow_wrap_loop:w
8482 }
8483 \cs_new_protected_nopar:Npn \iow_wrap_word_fits:
8484 {
8485   \bool_if:NNTF \l_iow_line_start_bool
8486     {
8487       \bool_set_false:N \l_iow_line_start_bool
8488       \tl_put_right:Nx \l_iow_current_line_tl
8489         { \l_iow_current_indentation_tl \l_iow_current_word_tl }
8490       \int_add:Nn \l_iow_current_line_int
8491         { \l_iow_current_indentation_int }
8492     }
8493     {
8494       \tl_put_right:Nx \l_iow_current_line_tl
8495         { ~ \l_iow_current_word_tl }
8496       \int_incr:N \l_iow_current_line_int
8497     }
8498 }
8499 \cs_new_protected_nopar:Npn \iow_wrap_word_newline:
8500 {
8501   \tl_put_right:Nx \l_iow_wrapped_tl
8502     { \l_iow_current_line_tl \\ }
8503   \int_set:Nn \l_iow_current_line_int
8504     {
8505     \l_iow_current_word_int

```

```

8506         + \l_iow_current_indentation_int
8507     }
8508     \tl_set:Nx \l_iow_current_line_tl
8509     { \l_iow_current_indentation_tl \l_iow_current_word_tl }
8510 }

```

(End definition for `\iow_wrap_word:`. This function is documented on page ??.)

`\iow_wrap_special:w` When the “special” marker is encountered, read what operation to perform, as a space-delimited argument, perform it, and remember to loop. In fact, to avoid spurious spaces when two special actions follow each other, we look ahead for another copy of the marker. `\iow_wrap_newline:w` Forced newlines are almost identical to those caused by overflow, except that here the word is empty. To indent more, add four spaces to the start of the indentation token list. `\iow_wrap_indent:w` To reduce indentation, rebuild the indentation token list using `\prg_replicate:nn`. `\iow_wrap_unindent:w` At the end, we simply save the last line (without the run-on text), and prevent the loop. `\iow_wrap_end:w`

```

8511 \cs_new_protected:Npn \iow_wrap_special:w #1 ~ #2 ~ #3 ~ %
8512 {
8513     \use:c { iow_wrap_#1: }
8514     \str_if_eq:xxTF { #2~#3 } { ~ \c_iow_wrap_marker_tl }
8515     { \iow_wrap_special:w }
8516     { \iow_wrap_loop:w #2 ~ #3 ~ }
8517 }
8518 \cs_new_protected_nopar:Npn \iow_wrap_newline:
8519 {
8520     \tl_put_right:Nx \l_iow_wrapped_tl
8521     { \l_iow_current_line_tl \ }
8522     \int_zero:N \l_iow_current_line_int
8523     \tl_clear:N \l_iow_current_line_tl
8524     \bool_set_true:N \l_iow_line_start_bool
8525 }
8526 \cs_new_protected_nopar:Npx \iow_wrap_indent:
8527 {
8528     \int_add:Nn \l_iow_current_indentation_int \c_four
8529     \tl_put_right:Nx \exp_not:N \l_iow_current_indentation_tl
8530     { \c_space_tl \c_space_tl \c_space_tl \c_space_tl }
8531 }
8532 \cs_new_protected_nopar:Npn \iow_wrap_unindent:
8533 {
8534     \int_sub:Nn \l_iow_current_indentation_int \c_four
8535     \tl_set:Nx \l_iow_current_indentation_tl
8536     { \prg_replicate:nn \l_iow_current_indentation_int { ~ } }
8537 }
8538 \cs_new_protected_nopar:Npn \iow_wrap_end:
8539 {
8540     \tl_put_right:Nx \l_iow_wrapped_tl
8541     { \l_iow_current_line_tl }
8542     \use_none_delimit_by_q_stop:w
8543 }

```

(End definition for `\iow_wrap_special:w`. This function is documented on page ??.)

`\str_length_skip_spaces:N`
`\str_length_skip_spaces:n`
`\str_length_loop:NNNNNNNNN`

The wrapping code requires to measure the number of character in each word. This could be done with `\tl_length:n`, but it is ten times faster (literally) to use the code below.

```

8544 \cs_new_nopar:Npn \str_length_skip_spaces:N
8545 { \exp_args:No \str_length_skip_spaces:n }
8546 \cs_new:Npn \str_length_skip_spaces:n #1
8547 {
8548   \int_value:w \int_eval:w
8549     \exp_after:wN \str_length_loop:NNNNNNNNN \tl_to_str:n {#1}
8550     {X8}{X7}{X6}{X5}{X4}{X3}{X2}{X1}{X0} \q_stop
8551   \int_eval_end:
8552 }
8553 \cs_new:Npn \str_length_loop:NNNNNNNNN #1#2#3#4#5#6#7#8#9
8554 {
8555   \if_catcode:w X #9
8556     \exp_after:wN \use_none_delimit_by_q_stop:w
8557   \else:
8558     9 +
8559     \exp_after:wN \str_length_loop:NNNNNNNNN
8560   \fi:
8561 }

```

(End definition for `\str_length_skip_spaces:N`. This function is documented on page ??.)

199.8 Reading input

`\ior_if_eof_p:N` To test if some particular input stream is exhausted the following conditional is provided. As the pool model means that closed streams are undefined control sequences, the test has two parts.

```

8562 \prg_new_conditional:Nnn \ior_if_eof:N { p , T , F , TF }
8563 {
8564   \cs_if_exist:NTF #1
8565   {
8566     \if_int_compare:w #1 = \c_sixteen
8567       \prg_return_true:
8568     \else:
8569       \if_eof:w #1
8570         \prg_return_true:
8571       \else:
8572         \prg_return_false:
8573       \fi:
8574     \fi:
8575   }
8576   { \prg_return_true: }
8577 }

```

(End definition for `\ior_if_eof_p:N`. This function is documented on page 138.)

`\ior_to:NN`
`\ior_gto:NN`

And here we read from files.

```

8578 \cs_new_protected:Npn \ior_to:NN #1#2
8579 { \tex_read:D #1 to #2 }

```



```

8580 \cs_new_protected:Npn \ior_gto:NN #1#2
8581 { \tex_global:D \tex_read:D #1 to #2 }
      (End definition for \ior_to:NN. This function is documented on page 137.)

```

\ior_str_to:NN Reading as strings is also a primitive wrapper.

```

\ior_str_gto:NN
8582 \cs_new_protected:Npn \ior_str_to:NN #1#2
8583 { \etex_readline:D #1 to #2 }
8584 \cs_new_protected:Npn \ior_str_gto:NN #1#2
8585 { \tex_global:D \etex_readline:D #1 to #2 }
      (End definition for \ior_str_to:NN. This function is documented on page 137.)

```

199.9 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

\iow_now_buffer_safe:Nn This is much more easily done using the wrapping system: there is an expansion there,
 \iow_now_buffer_safe:Nx so a bit of a hack is needed.

```

8586 < *deprecated >
8587 \cs_new_protected:Npn \iow_now_buffer_safe:Nn #1#2
8588 { \iow_wrap:xnnnN { \exp_not:n {#2} } { } \c_zero { } \iow_now:Nn #1 }
8589 \cs_new_protected:Npn \iow_now_buffer_safe:Nx #1#2
8590 { \iow_wrap:xnnnN {#2} { } \c_zero { } \iow_now:Nn #1 }
8591 < /deprecated >
      (End definition for \iow_now_buffer_safe:Nn and \iow_now_buffer_safe:Nx. These functions are
      documented on page ??.)

```

\ior_open_streams: Slightly misleading names.

```

\ior_open_streams:
8592 < *deprecated >
8593 \cs_new_eq:NN \ior_open_streams: \ior_list_streams:
8594 \cs_new_eq:NN \iow_open_streams: \iow_list_streams:
8595 < /deprecated >
      (End definition for \ior_open_streams:. This function is documented on page ??.)
8596 < /initex | package >

```

200 l3msg implementation

```

8597 < *initex | package >
8598 < *package >
8599 \ProvidesExplPackage
8600 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
8601 \package_check_loaded_expl:
8602 < /package >

```

\l_msg_tmp_tl A general scratch for the module.

```

8603 \tl_new:N \l_msg_tmp_tl
      (End definition for \l_msg_tmp_tl. This function is documented on page ??.)

```

200.1 Creating messages

Messages are created and used separately, so there two parts to the code here. First, a mechanism for creating message text. This is pretty simple, as there is not actually a lot to do.

```
\c_msg_text_prefix_tl Locations for the text of messages.
\c_msg_more_text_prefix_tl
8604 \tl_const:Nn \c_msg_text_prefix_tl { msg~text~>~ }
8605 \tl_const:Nn \c_msg_more_text_prefix_tl { msg~extra~text~>~ }
      (End definition for \c_msg_text_prefix_tl and \c_msg_more_text_prefix_tl. These functions
      are documented on page ??.)

\msg_new:nnnn Setting a message simply means saving the appropriate text into two functions. A sanity
\msg_new:nnn check first.
\msg_gset:nnnn
\msg_gset:nnn
\msg_set:nnnn
\msg_set:nnn
8606 \cs_new_protected:Npn \msg_new:nnnn #1#2
8607 {
8608   \cs_if_exist:cT { \c_msg_text_prefix_tl #1 / #2 }
8609   {
8610     \msg_kernel_error:nnxx { msg } { message-already-defined }
8611     {#1} {#2}
8612   }
8613   \msg_gset:nnnn {#1} {#2}
8614 }
8615 \cs_new_protected:Npn \msg_new:nnn #1#2#3
8616 { \msg_new:nnnn {#1} {#2} {#3} { } }
8617 \cs_new_protected:Npn \msg_set:nnnn #1#2#3#4
8618 {
8619   \cs_set:cpn { \c_msg_text_prefix_tl #1 / #2 }
8620   ##1##2##3##4 {#3}
8621   \cs_set:cpn { \c_msg_more_text_prefix_tl #1 / #2 }
8622   ##1##2##3##4 {#4}
8623 }
8624 \cs_new_protected:Npn \msg_set:nnn #1#2#3
8625 { \msg_set:nnnn {#1} {#2} {#3} { } }
8626 \cs_new_protected:Npn \msg_gset:nnnn #1#2#3#4
8627 {
8628   \cs_gset:cpn { \c_msg_text_prefix_tl #1 / #2 }
8629   ##1##2##3##4 {#3}
8630   \cs_gset:cpn { \c_msg_more_text_prefix_tl #1 / #2 }
8631   ##1##2##3##4 {#4}
8632 }
8633 \cs_new_protected:Npn \msg_gset:nnn #1#2#3
8634 { \msg_gset:nnnn {#1} {#2} {#3} { } }
      (End definition for \msg_new:nnnn and \msg_new:nnn. These functions are documented on page
      ??.)
```

200.2 Messages: support functions and text

```

\c_msg_coding_error_text_tl Simple pieces of text for messages.
\c_msg_continue_text_tl      8635 \tl_const:Nn \c_msg_coding_error_text_tl
\c_msg_critical_text_tl      8636 {
\c_msg_fatal_text_tl         8637   This~is~a~coding~error.
\c_msg_help_text_tl          8638   \\ \\
\c_msg_no_info_text_tl       8639 }
\c_msg_on_line_text_tl        8640 \tl_const:Nn \c_msg_continue_text_tl
\c_msg_return_text_tl         8641 { Type~<return>~to~continue }
\c_msg_trouble_text_tl        8642 \tl_const:Nn \c_msg_critical_text_tl
                                8643 { Reading~the~current~file~will~stop }
                                8644 \tl_const:Nn \c_msg_fatal_text_tl
                                8645 { This~is~a~fatal~error:~LaTeX~will~abort }
                                8646 \tl_const:Nn \c_msg_help_text_tl
                                8647 { For~immediate~help~type~H~<return> }
                                8648 \tl_const:Nn \c_msg_no_info_text_tl
                                8649 {
                                8650   LaTeX~does~not~know~anything~more~about~this~error,~sorry.
                                8651   \c_msg_return_text_tl
                                8652 }
                                8653 \tl_const:Nn \c_msg_on_line_text_tl { on~line }
                                8654 \tl_const:Nn \c_msg_return_text_tl
                                8655 {
                                8656   \\ \\
                                8657   Try~typing~<return>~to~proceed.
                                8658   \\
                                8659   If~that~doesn't~work,~type~X~<return>~to~quit.
                                8660 }
                                8661 \tl_const:Nn \c_msg_trouble_text_tl
                                8662 {
                                8663   \\ \\
                                8664   More~errors~will~almost~certainly~follow: \\
                                8665   the~LaTeX~run~should~be~aborted.
                                8666 }
                                (End definition for \c_msg_coding_error_text_tl and others. These functions are documented
                                on page 140.)

\msg_newline: New lines are printed in the same way as for low-level file writing.
\msg_two_newlines: 8667 \cs_new_nopar:Npn \msg_newline: { ^^J }
                    8668 \cs_new_nopar:Npn \msg_two_newlines: { ^^J ^^J }
                    (End definition for \msg_newline: and \msg_two_newlines:. These functions are documented on
                    page ??.)

\msg_line_number: For writing the line number nicely.
\msg_line_context: 8669 \cs_new_nopar:Npn \msg_line_number: { \int_use:N \tex_inputlineno:D }
                    8670 \cs_set_nopar:Npn \msg_line_context:
                    8671 {
                    8672   \c_msg_on_line_text_tl
                    8673   \c_space_tl

```

```

8674     \msg_line_number:
8675 }
      (End definition for \msg_line_number:. This function is documented on page ??.)

```

200.3 Showing messages: low level mechanism

`\c_msg_hide_tl` An empty variable with a number of (category code 11) periods at the end of its name. `\c_msg_hide_tl<dots>` This is used to push the T_EX part of an error message “off the screen”. Using two variables here means that later life is a little easier.

```

8676 \char_set_catcode_letter:N \.
8677 \tl_new:N
8678   \c_msg_hide_tl.....
8679 \tl_const:Nn \c_msg_hide_tl
8680 { \c_msg_hide_tl..... }
8681 \char_set_catcode_other:N \.
      (End definition for \c_msg_hide_tl. This function is documented on page ??.)

```

`\l_msg_text_tl` For wrapping message text.

```

8682 \tl_new:N \l_msg_text_tl
      (End definition for \l_msg_text_tl. This function is documented on page ??.)

```

`\msg_interrupt:xxx` The low-level interruption macro is rather opaque, unfortunately. The idea here is to create a message which hides all of T_EX’s own information by filling the output up with dots. To achieve this, dots have to be letters. The odd `\c_msg_hide_tl<dots>` actually does the hiding: it is the large run of dots in the name that is important here. The meaning of `\` is altered so that the explanation text is a simple run whilst the initial error has line-continuation shown.

```

8683 \cs_new_protected:Npn \msg_interrupt:xxx #1#2#3
8684 {
8685   \group_begin:
8686   \tl_if_empty:nTF {#3}
8687     { \msg_interrupt_no_details:xx {#1} {#2} }
8688     { \msg_interrupt_details:xxx {#1} {#2} {#3} }
8689   \msg_interrupt_aux:
8690   \group_end:
8691 }

```

Depending on the availability of more information there is a choice of how to set up the further help. The extra help text has to be set before the message itself can be issued. Everything is done using x-type expansion as the new line markers are different for the two type of text and need to be correctly set up.

```

8692 \cs_new_protected:Npn \msg_interrupt_no_details:xx #1#2
8693 {
8694   \iow_wrap:xnnnN
8695   { \ \c_msg_no_info_text_tl }
8696   { |~ } { 2 } { } \msg_interrupt_more_text:n
8697   \iow_wrap:xnnnN { #1 \ \ #2 \ \ \c_msg_continue_text_tl }
8698   { ! ~ } { 2 } { } \msg_interrupt_text:n

```

```

8699 }
8700 \cs_new_protected:Npn \msg_interrupt_details:xxx #1#2#3
8701 {
8702   \iow_wrap:xnnnN
8703     { \ \ #3 }
8704     { |~ } { 2 } { } \msg_interrupt_more_text:n
8705   \iow_wrap:xnnnN { #1 \ \ \ #2 \ \ \ \c_msg_help_text_tl }
8706     { ! ~ } { 2 } { } \msg_interrupt_text:n
8707 }
8708 \cs_new_protected:Npn \msg_interrupt_text:n #1
8709 { \tl_set:Nn \l_msg_text_tl {#1} }
8710 \cs_new_protected:Npn \msg_interrupt_more_text:n #1
8711 {
8712   <*initex>
8713     \tl_set:Nx \l_msg_tmp_tl
8714   </initex>
8715   <*package>
8716     \protected@edef \l_msg_tmp_tl
8717   </package>
8718   {
8719     |,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
8720     #1
8721     \msg_newline:
8722     |.....
8723   }
8724   \tex_errhelp:D \exp_after:wN { \l_msg_tmp_tl }
8725 }

```

The business end of the process starts by producing some visual separation of the message from the main part of the log. It then adds the hiding text to the message to print. The error message needs to be printed with everything made “invisible”: this is where the strange business with & comes in: this is made into another !. There is also a closing brace that will show up in the output, which is turned into a blank space.

```

8726 \group_begin: % {
8727   \char_set_lccode:nn {'\} } {'\ }
8728   \char_set_lccode:nn {'&} {'\!}
8729   \char_set_catcode_active:N \&
8730   \tl_to_lowercase:n
8731   {
8732     \group_end:
8733     \cs_new_protected:Npn \msg_interrupt_aux:
8734     {
8735       \iow_term:x
8736       {
8737         \iow_newline:
8738         !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
8739         \iow_newline:
8740         !
8741       }
8742       \tl_put_right:No \l_msg_text_tl { \c_msg_hide_tl }

```

```

8743     \cs_set_protected_nopar:Npx &
8744     { \tex_errmessage:D { \exp_not:o { \l_msg_text_tl } } }
8745     &
8746     }
8747 }

```

(End definition for `\msg_interrupt:xxx`. This function is documented on page ??.)

`\msg_log:x` Printing to the log or terminal without a stop is rather easier. A bit of simple visual
`\msg_term:x` work sets things off nicely.

```

8748 \cs_new_protected:Npn \msg_log:x #1
8749 {
8750   \iow_log:x { ..... }
8751   \iow_wrap:xnnnN { . ~ #1 } { . ~ } { 2 } { }
8752   \iow_log:x
8753   \iow_log:x { ..... }
8754 }
8755 \cs_new_protected:Npn \msg_term:x #1
8756 {
8757   \iow_term:x { ***** }
8758   \iow_wrap:xnnnN { * ~ #1 } { * ~ } { 2 } { }
8759   \iow_term:x
8760   \iow_term:x { ***** }
8761 }

```

(End definition for `\msg_log:x`. This function is documented on page 144.)

200.4 Displaying messages

L^AT_EX is handling error messages and so the T_EX ones are disabled.

```

8762 \int_set:Nn \tex_errorcontextlines:D { -1 }

```

`\msg_fatal_text:n` A function for issuing messages: both the text and order could in principal vary.

```

\msg_critical_text:n 8763 \cs_new:Npn \msg_fatal_text:n #1 { Fatal~#1~error }
\msg_error_text:n    8764 \cs_new:Npn \msg_critical_text:n #1 { Critical~#1~error }
\msg_warning_text:n  8765 \cs_new:Npn \msg_error_text:n #1 { #1~error }
\msg_info_text:n     8766 \cs_new:Npn \msg_warning_text:n #1 { #1~warning }
                     8767 \cs_new:Npn \msg_info_text:n #1 { #1~info }

```

(End definition for `\msg_fatal_text:n` and others. These functions are documented on page 141.)

`\msg_see_documentation_text:n` Contextual footer information.

```

8768 \cs_new:Npn \msg_see_documentation_text:n #1
8769 { \ \ \ See~the~#1~documentation~for~further~information. }

```

(End definition for `\msg_see_documentation_text:n`. This function is documented on page ??.)

`\l_msg_redirect_classes_prop` For filtering messages, a list of all messages and of those which have to be modified is
`\l_msg_redirect_names_prop` required.

```

8770 \prop_new:N \l_msg_redirect_classes_prop
8771 \prop_new:N \l_msg_redirect_names_prop

```

(End definition for `\l_msg_redirect_classes_prop` and `\l_msg_redirect_names_prop`. These functions are documented on page ??.)

`\msg_class_set:nn` Setting up a message class does two tasks. Any existing redirection is cleared, and the various message functions are created to simply use the code stored for the message.

```

8772 \cs_new_protected:Npn \msg_class_set:nn #1#2
8773 {
8774   \prop_clear_new:c { l_msg_redirect_ #1 _prop }
8775   \cs_set_protected:cpn { msg_ #1 :nnxxxx } ##1##2##3##4##5##6
8776   { \msg_use:nnnnxxxx {#1} {#2} {##1} {##2} {##3} {##4} {##5} {##6} }
8777   \cs_set_protected:cpx { msg_ #1 :nnxxx } ##1##2##3##4##5
8778   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} {##4} {##5} { } }
8779   \cs_set_protected:cpx { msg_ #1 :nnxx } ##1##2##3##4
8780   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} {##4} { } { } }
8781   \cs_set_protected:cpx { msg_ #1 :nnx } ##1##2##3
8782   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} { } { } { } }
8783   \cs_set_protected:cpx { msg_ #1 :nn } ##1##2
8784   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} { } { } { } { } }
8785 }

```

(End definition for `\msg_class_set:nn`. This function is documented on page 141.)

`\msg_if_more_text:N` A test to see if any more text is available, using a permanently-empty text function.

```

\msg_if_more_text:c 8786 \prg_set_conditional:Npnn \msg_if_more_text:N #1 { p , T , F , TF }
\msg_no_more_text:xxxx 8787 {
8788   \cs_if_eq:NNTF #1 \msg_no_more_text:xxxx
8789   { \prg_return_false: }
8790   { \prg_return_true: }
8791 }
8792 \cs_new:Npn \msg_no_more_text:xxxx #1#2#3#4 { }
8793 \cs_generate_variant:Nn \msg_if_more_text_p:N { c }
8794 \cs_generate_variant:Nn \msg_if_more_text_NT { c }
8795 \cs_generate_variant:Nn \msg_if_more_text_NF { c }
8796 \cs_generate_variant:Nn \msg_if_more_text_NTF { c }

```

(End definition for `\msg_if_more_text:N` and `\msg_if_more_text:c`. These functions are documented on page ??.)

`\msg_fatal:nnxxxx` For fatal errors, after the error message T_EX bails out.

```

\msg_fatal:nnxxxx 8797 \msg_class_set:nn { fatal }
\msg_fatal:nnxx 8798 {
\msg_fatal:nnx 8799   \msg_interrupt:xxx
\msg_fatal:nn 8800   { \msg_fatal_text:n {#1} : ~ "#2" }
8801   {
8802     \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8803     \msg_see_documentation_text:n {#1}
8804   }
8805   { \c_msg_fatal_text_tl }
8806   \tex_end:D
8807 }

```

(End definition for `\msg_fatal:nnxxxx` and others. These functions are documented on page ??.)

`\msg_critical:nnxxxx` Not quite so bad: just end the current file.

```

\msg_critical:nnxxx 8808 \msg_class_set:nn { critical }
\msg_critical:nnxx 8809 {
\msg_critical:nnx 8810 \msg_interrupt:xxx
\msg_critical:nn 8811 { \msg_critical_text:n {#1} : ~ "#2" }
8812 {
8813 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8814 \msg_see_documentation_text:n {#1}
8815 }
8816 { \c_msg_critical_text_tl }
8817 \tex_endinput:D
8818 }

```

(End definition for \msg_critical:nnxxxx and others. These functions are documented on page ??.)

`\msg_error:nnxxxx` For an error, the interrupt routine is called, then any recovery code is tried.

```

\msg_error:nnxxx 8819 \msg_class_set:nn { error }
\msg_error:nnxx 8820 {
\msg_error:nnx 8821 \msg_if_more_text:cTF { \c_msg_more_text_prefix_tl #1 / #2 }
\msg_error:nn 8822 {
8823 \msg_interrupt:xxx
8824 { \msg_error_text:n {#1} : ~ "#2" }
8825 {
8826 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8827 \msg_see_documentation_text:n {#1}
8828 }
8829 { \use:c { \c_msg_more_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6} }
8830 }
8831 {
8832 \msg_interrupt:xxx
8833 { \msg_error_text:n {#1} : ~ "#2" }
8834 {
8835 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8836 \msg_see_documentation_text:n {#1}
8837 }
8838 { }
8839 }
8840 }

```

(End definition for \msg_error:nnxxxx and others. These functions are documented on page ??.)

`\msg_warning:nnxxxx` Warnings are printed to the terminal.

```

\msg_warning:nnxxx 8841 \msg_class_set:nn { warning }
\msg_warning:nnxx 8842 {
\msg_warning:nnx 8843 \msg_term:x
\msg_warning:nn 8844 {
8845 \msg_warning_text:n {#1} : ~ "#2" \\ \\
8846 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8847 }
8848 }

```


(End definition for \msg_warning:nnxxxx and others. These functions are documented on page ??.)

```
\msg_info:nnxxxx Information only goes into the log.
\msg_info:nnxxxx 8849 \msg_class_set:nn { info }
\msg_info:nnxxx 8850 {
\msg_info:nnx 8851 \msg_log:x
\msg_info:nn 8852 {
8853 \msg_info_text:n {#1} : ~ "#2" \\ \\
8854 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8855 }
8856 }
```

(End definition for \msg_info:nnxxxx and others. These functions are documented on page ??.)

```
\msg_log:nnxxxx "Log" data is very similar to information, but with no extras added.
\msg_log:nnxxxx 8857 \msg_class_set:nn { log }
\msg_log:nnxxx 8858 {
\msg_log:nnx 8859 \msg_log:x
\msg_log:nn 8860 { \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6} }
8861 }
```

(End definition for \msg_log:nnxxxx and others. These functions are documented on page ??.)

\msg_none:nnxxxx The none message type is needed so that input can be gobbled.

```
\msg_none:nnxxxx 8862 \msg_class_set:nn { none } { }
```

(End definition for \msg_none:nnxxxx and others. These functions are documented on page ??.)

\l_msg_redirect_classes_seq Support variables needed for the redirection system.

```
\l_msg_class_tl 8863 \seq_new:N \l_msg_redirect_classes_seq
\l_msg_current_class_tl 8864 \tl_new:N \l_msg_class_tl
\l_msg_current_module_tl 8865 \tl_new:N \l_msg_current_class_tl
8866 \tl_new:N \l_msg_current_module_tl
```

(End definition for \l_msg_redirect_classes_seq and others. These functions are documented on page ??.)

\msg_use:nnnnxxxx The main message-using macro creates two auxiliary functions: one containing the code for the message, and the second a loop function. There is then a hand-off to the system for checking if redirection is needed.

```
\msg_use_loop_check:nn 8867 \cs_new_protected:Npn \msg_use:nnnnxxxx #1#2#3#4#5#6#7#8
\msg_use_code: 8868 {
\msg_use_loop:n 8869 \cs_set_protected_nopar:Npx \msg_use_code:
\msg_use_loop:o 8870 {
8871 \seq_clear:N \exp_not:N \l_msg_redirect_classes_seq
8872 \exp_not:n {#2}
8873 }
8874 \cs_set_protected:Npx \msg_use_loop:n ##1
8875 {
8876 \seq_if_in:NnTF \exp_not:n \l_msg_redirect_classes_seq {#1}
8877 { \msg_kernel_error:nn { msg } { message-loop } {#1} }
8878 }
```

```

8879         \seq_put_right:Nn \exp_not:N \l_msg_redirect_classes_seq {#1}
8880         \exp_not:N \cs_if_exist:cTF { msg_ ##1 :nnxxxx }
8881         {
8882             \exp_not:N \use:c { msg_ ##1 :nnxxxx }
8883             \exp_not:n { {#3} {#4} {#5} {#6} {#7} {#8} }
8884         }
8885         {
8886             \msg_kernel_error:nnx { msg } { message-class-unknown } {##1}
8887         }
8888     }
8889 }
8890 \cs_if_exist:cTF { \c_msg_text_prefix_tl #3 / #4 }
8891 { \msg_use_aux:nnn {#1} {#3} {#4} }
8892 { \msg_kernel_error:nnxx { msg } { message-unknown } {#3} {#4} }
8893 }

```

The first auxiliary macro looks for a match by name: the most restrictive check.

```

8894 \cs_new_protected:Npn \msg_use_aux:nnn #1#2#3
8895 {
8896     \tl_set:Nn \l_msg_current_class_tl {#1}
8897     \tl_set:Nn \l_msg_current_module_tl {#2}
8898     \prop_if_in:NnTF \l_msg_redirect_names_prop { // #2 / #3 / }
8899     { \msg_use_loop_check:nn { names } { // #2 / #3 / } }
8900     { \msg_use_aux:nn {#1} {#2} }
8901 }

```

The second function checks for general matches by module or for all modules.

```

8902 \cs_new_protected:Npn \msg_use_aux:nn #1#2
8903 {
8904     \prop_if_in:cnTF { l_msg_redirect_ #1 _prop } {#2}
8905     { \msg_use_loop_check:nn {#1} {#2} }
8906     {
8907         \prop_if_in:cnTF { l_msg_redirect_ #1 _prop } { * }
8908         { \msg_use_loop_check:nn {#1} { * } }
8909         { \msg_use_code: }
8910     }
8911 }

```

When checking whether to loop, the same code is needed in a few places.

```

8912 \cs_new_protected:Npn \msg_use_loop_check:nn #1#2
8913 {
8914     \prop_get:cnN { l_msg_redirect_ #1 _prop } {#2} \l_msg_class_tl
8915     \tl_if_eq:NNTF \l_msg_current_class_tl \l_msg_class_tl
8916     {
8917         { \msg_use_code: }
8918         { \msg_use_loop:o \l_msg_class_tl }
8919     }
8920 }
8921 \cs_new_protected_nopar:Npn \msg_use_code: { }
8922 \cs_new_protected:Npn \msg_use_loop:n #1 { }
8923 \cs_generate_variant:Nn \msg_use_loop:n { o }

```

(End definition for \msg_use:nnnnxxxx. This function is documented on page ??.)

\msg_redirect_class:nn Converts class one into class two.

```
8924 \cs_new_protected:Npn \msg_redirect_class:nn #1#2
8925 { \prop_put:cnn { l_msg_redirect_ #1 _prop } { * } {#2} }
(End definition for \msg_redirect_class:nn. This function is documented on page 143.)
```

\msg_redirect_module:nnn For when all messages of a class should be altered for a given module.

```
8926 \cs_new_protected:Npn \msg_redirect_module:nnn #1#2#3
8927 { \prop_put:cnn { l_msg_redirect_ #2 _prop } {#1} {#3} }
(End definition for \msg_redirect_module:nnn. This function is documented on page 143.)
```

\msg_redirect_name:nnn Named message will always use the given class.

```
8928 \cs_new_protected:Npn \msg_redirect_name:nnn #1#2#3
8929 { \prop_put:Nnn \l_msg_redirect_names_prop { // #1 / #2 / } {#3} }
(End definition for \msg_redirect_name:nnn. This function is documented on page 143.)
```

200.5 Kernel-specific functions

\msg_kernel_new:nnnn The kernel needs some messages of its own. These are created using pre-built functions.
 \msg_kernel_new:nnn Two functions are provided: one more general and one which only has the short text part.
 \msg_kernel_set:nnnn
 \msg_kernel_set:nnn

```
8930 \cs_new_protected:Npn \msg_kernel_new:nnnn #1#2
8931 { \msg_new:nnnn { LaTeX } { #1 / #2 } }
8932 \cs_new_protected:Npn \msg_kernel_new:nnn #1#2
8933 { \msg_new:nnn { LaTeX } { #1 / #2 } }
8934 \cs_new_protected:Npn \msg_kernel_set:nnnn #1#2
8935 { \msg_set:nnnn { LaTeX } { #1 / #2 } }
8936 \cs_new_protected:Npn \msg_kernel_set:nnn #1#2
8937 { \msg_set:nnn { LaTeX } { #1 / #2 } }
(End definition for \msg_kernel_new:nnnn. This function is documented on page ??.)
```

\msg_kernel_fatal:nnxxxx Fatal kernel errors cannot be re-defined.

```
\msg_kernel_fatal:nnxxx 8938 \cs_new_protected:Npn \msg_kernel_fatal:nnxxxx #1#2#3#4#5#6
\msg_kernel_fatal:nnxx 8939 {
\msg_kernel_fatal:nnx 8940 \msg_interrupt:xxx
\msg_kernel_fatal:nn 8941 { \msg_fatal_text:n { LaTeX } : ~ "#1 / #2" }
8942 {
8943 \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8944 {#3} {#4} {#5} {#6}
8945 \msg_see_documentation_text:n { LaTeX3 }
8946 }
8947 { \c_msg_fatal_text_tl }
8948 \tex_end:D
8949 }
8950 \cs_new_protected:Npn \msg_kernel_fatal:nnxxx #1#2#3#4#5
8951 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
8952 \cs_new_protected:Npn \msg_kernel_fatal:nnxx #1#2#3#4
```

```

8953 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} {#4} { } { } }
8954 \cs_new_protected:Npn \msg_kernel_fatal:nnx #1#2#3
8955 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} { } { } { } }
8956 \cs_new_protected:Npn \msg_kernel_fatal:nn #1#2
8957 { \msg_kernel_fatal:nnxxxx {#1} {#2} { } { } { } { } }
      (End definition for \msg_kernel_fatal:nnxxxx. This function is documented on page ??.)

```

\msg_kernel_error:nnxxxx Neither can kernel errors.

```

\msg_kernel_error:nnxxx 8958 \cs_new_protected:Npn \msg_kernel_error:nnxxxx #1#2#3#4#5#6
\msg_kernel_error:nnxxx 8959 {
\msg_kernel_error:nnx 8960   \msg_if_more_text:cTF { \c_msg_more_text_prefix_tl LaTeX / #1 / #2 }
\msg_kernel_error:nn 8961   {
8962     \msg_interrupt:xxx
8963     { \msg_error_text:n { LaTeX } : ~ " #1 / #2 " }
8964     {
8965       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8966       {#3} {#4} {#5} {#6}
8967       \msg_see_documentation_text:n { LaTeX3 }
8968     }
8969     {
8970       \use:c { \c_msg_more_text_prefix_tl LaTeX / #1 / #2 }
8971       {#3} {#4} {#5} {#6}
8972     }
8973   }
8974   {
8975     \msg_interrupt:xxx
8976     { \msg_error_text:n { LaTeX } : ~ " #1 / #2 " }
8977     {
8978       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
8979       {#3} {#4} {#5} {#6}
8980       \msg_see_documentation_text:n { LaTeX3 }
8981     }
8982     { }
8983   }
8984 }
8985 \cs_new_protected:Npn \msg_kernel_error:nnxxx #1#2#3#4#5
8986 { \msg_kernel_error:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
8987 \cs_set_protected:Npn \msg_kernel_error:nnxx #1#2#3#4
8988 { \msg_kernel_error:nnxxxx {#1} {#2} {#3} {#4} { } { } }
8989 \cs_set_protected:Npn \msg_kernel_error:nnx #1#2#3
8990 { \msg_kernel_error:nnxxxx {#1} {#2} {#3} { } { } { } }
8991 \cs_set_protected:Npn \msg_kernel_error:nn #1#2
8992 { \msg_kernel_error:nnxxxx {#1} {#2} { } { } { } { } }
      (End definition for \msg_kernel_error:nnxxxx. This function is documented on page ??.)

```

\msg_kernel_warning:nnxxxx Kernel messages which can be redirected.

```

\msg_kernel_warning:nnxxx 8993 \prop_new:N \l_msg_redirect_kernel_warning_prop
\msg_kernel_warning:nnxxx 8994 \cs_new_protected:Npn \msg_kernel_warning:nnxxxx #1#2#3#4#5#6
\msg_kernel_warning:nnx 8995 {
\msg_kernel_warning:nn
\msg_kernel_info:nnxxxx
\msg_kernel_info:nnxxx
\msg_kernel_info:nnxx
\msg_kernel_info:nnx
\msg_kernel_info:nn

```

```

8996 \msg_use:nnnnxxxx { warning }
8997 {
8998     \msg_term:x
8999     {
9000         \msg_warning_text:n { LaTeX } : ~ " #1 / #2 " \\ \\
9001         \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
9002         {#3} {#4} {#5} {#6}
9003     }
9004 }
9005 { LaTeX } { #1 / #2 } {#3} {#4} {#5} {#6}
9006 }
9007 \cs_new_protected:Npn \msg_kernel_warning:nnxxx #1#2#3#4#5
9008 { \msg_kernel_warning:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
9009 \cs_new_protected:Npn \msg_kernel_warning:nnxx #1#2#3#4
9010 { \msg_kernel_warning:nnxxxx {#1} {#2} {#3} {#4} { } { } }
9011 \cs_new_protected:Npn \msg_kernel_warning:nnx #1#2#3
9012 { \msg_kernel_warning:nnxxxx {#1} {#2} {#3} { } { } { } }
9013 \cs_new_protected:Npn \msg_kernel_warning:nn #1#2
9014 { \msg_kernel_warning:nnxxxx {#1} {#2} { } { } { } { } }
9015 \prop_new:N \l_msg_redirect_kernel_info_prop
9016 \cs_new_protected:Npn \msg_kernel_info:nnxxxx #1#2#3#4#5#6
9017 {
9018     \msg_use:nnnnxxxx { info }
9019     {
9020         \msg_log:x
9021         {
9022             \msg_info_text:n { LaTeX } : ~ " #1 / #2 " \\ \\
9023             \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
9024             {#3} {#4} {#5} {#6}
9025         }
9026     }
9027     { LaTeX } { #1 / #2 } {#3} {#4} {#5} {#6}
9028 }
9029 \cs_new_protected:Npn \msg_kernel_info:nnxxx #1#2#3#4#5
9030 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
9031 \cs_new_protected:Npn \msg_kernel_info:nnxx #1#2#3#4
9032 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} {#4} { } { } }
9033 \cs_new_protected:Npn \msg_kernel_info:nnx #1#2#3
9034 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} { } { } { } }
9035 \cs_new_protected:Npn \msg_kernel_info:nn #1#2
9036 { \msg_kernel_info:nnxxxx {#1} {#2} { } { } { } { } }

```

(End definition for \msg_kernel_warning:nnxxxx. This function is documented on page ??.)

Error messages needed to actually implement the message system itself.

```

9037 \msg_kernel_new:nnnn { msg } { message-already-defined }
9038 { Message~'#2'~for~module~'#1'~already-defined. }
9039 {
9040     \c_msg_coding_error_text_tl
9041     LaTeX~was~asked~to~define~a~new~message~called~'#2'\
9042     by~the~module~'#1':~this~message~already~exists.

```

```

9043     \c_msg_return_text_tl
9044   }
9045   \msg_kernel_new:nnnn { msg } { message-unknown }
9046   { Unknown~message~'#2'~for~module~'#1'. }
9047   {
9048     \c_msg_coding_error_text_tl
9049     LaTeX~was~asked~to~display~a~message~called~'#2'\\
9050     by~the~module~'#1'~module:~this~message~does~not~exist.
9051     \c_msg_return_text_tl
9052   }
9053   \msg_kernel_new:nnnn { msg } { message-class-unknown }
9054   { Unknown~message~class~'#1'. }
9055   {
9056     LaTeX~has~been~asked~to~redirect~messages~to~a~class~'#1':\\
9057     this~was~never~defined.
9058     \c_msg_return_text_tl
9059   }
9060   \msg_kernel_new:nnnn { msg } { redirect-loop }
9061   { Message~redirection~loop~for~message~class~'#1'. }
9062   {
9063     LaTeX~has~been~asked~to~redirect~messages~in~an~infinite~loop.\\
9064     The~original~message~here~has~been~lost.
9065     \c_msg_return_text_tl
9066   }

```

Messages for earlier kernel modules.

```

9067   \msg_kernel_new:nnnn { kernel } { bad-number-of-arguments }
9068   { Function~'#1'~cannot~be~defined~with~#2~arguments. }
9069   {
9070     \c_msg_coding_error_text_tl
9071     LaTeX~has~been~asked~to~define~a~function~'#1'~with~
9072     #2~arguments. \\
9073     TeX~allows~between~0~and~9~arguments~for~a~single~function.
9074   }
9075   \msg_kernel_new:nnnn { kernel } { command-already-defined }
9076   { Control~sequence~#1~already~defined. }
9077   {
9078     \c_msg_coding_error_text_tl
9079     LaTeX~has~been~asked~to~create~a~new~control~sequence~'#1'~
9080     but~this~name~has~already~been~used~elsewhere. \\ \\
9081     The~current~meaning~is:\\
9082     \ \ #2
9083   }
9084   \msg_kernel_new:nnnn { kernel } { command-not-defined }
9085   { Control~sequence~#1~undefined. }
9086   {
9087     \c_msg_coding_error_text_tl
9088     LaTeX~has~been~asked~to~use~a~command~#1,~but~this~has~not~
9089     been~defined~yet.
9090   }

```

```

9091 \msg_kernel_new:nnnn { kernel } { variable-not-defined }
9092 { Variable~#1~undefined. }
9093 {
9094   \c_msg_coding_error_text_tl
9095   LaTeX~has~been~asked~to~show~a~variable~#1,~but~this~has~not~
9096   been~defined~yet.
9097 }
9098 \msg_kernel_new:nnnn { seq } { empty-sequence }
9099 { Empty~sequence~#1. }
9100 {
9101   \c_msg_coding_error_text_tl
9102   LaTeX~has~been~asked~to~recover~an~entry~from~a~sequence~that~
9103   has~no~content:~that~cannot~happen!
9104 }
9105 \msg_kernel_new:nnnn { tl } { empty-search-pattern }
9106 { Empty~search~pattern. }
9107 {
9108   \c_msg_coding_error_text_tl
9109   LaTeX~has~been~asked~to~replace~an~empty~pattern~by~'~#1'~that~%
9110   would~lead~to~an~infinite~loop!
9111 }
9112 \msg_kernel_new:nnnn { scan } { already-defined }
9113 { Scan~mark~#1~already~defined. }
9114 {
9115   \c_msg_coding_error_text_tl
9116   LaTeX~has~been~asked~to~create~a~new~scan~mark~'~#1'~
9117   but~this~name~has~already~been~used~for~a~scan~mark.
9118 }

```

Some errors only appear in expandable settings, hence don't need a “more-text” argument.

```

9119 \msg_kernel_new:nnn { seq } { misused }
9120 { A~sequence~was~misused. }
9121 \msg_kernel_new:nnn { kernel } { bad-var }
9122 { Erroneous~variable~#1~used! }
9123 \msg_kernel_new:nnn { prg } { zero-step }
9124 { Zero~step~size~for~stepwise~function~#1. }
9125 \msg_kernel_new:nnn { prg } { replicate-neg }
9126 { Negative~argument~for~\prg_replicate:nn. }

```

Messages used by the “show” functions.

```

9127 \msg_kernel_new:nnn { seq } { show }
9128 {
9129   Sequence~\token_to_str:N #1~
9130   \seq_if_empty:NTF #1
9131     { is~empty }
9132     { contains~the~items~(without~outer~braces): }
9133 }
9134 \msg_kernel_new:nnn { prop } { show }
9135 {

```

```

9136 Property~list~\token_to_str:N #1~
9137 \prop_if_empty:NTF #1
9138 { is~empty }
9139 { contains~the~pairs~(without~outer~braces): }
9140 }
9141 \msg_kernel_new:nnn { clist } { show }
9142 {
9143   Comma~list~
9144   \str_if_eq:nnF {#1} { \l_clist_tmpa_clist } { \token_to_str:N #1~}
9145   \clist_if_empty:NTF #1
9146   { is~empty }
9147   { contains~the~items~(without~outer~braces): }
9148 }
9149 \msg_kernel_new:nnn { ior } { show-no-stream }
9150 { No~input~streams~are~open }
9151 \msg_kernel_new:nnn { ior } { show-open-streams }
9152 { The~following~input~streams~are~in~use: }
9153 \msg_kernel_new:nnn { iow } { show-no-stream }
9154 { No~output~streams~are~open }
9155 \msg_kernel_new:nnn { iow } { show-open-streams }
9156 { The~following~output~streams~are~in~use: }

```

200.6 Expandable errors

`\msg_expandable_error:n` In expansion only context, we cannot use the normal means of reporting errors. Instead, we feed \TeX an undefined control sequence, `\LaTeX3 error:.` It is thus interrupted, and shows the context, which thanks to the odd-looking `\use:n` is

```

<argument> \LaTeX3 error:
                The error message.

```

In other words, \TeX is processing the argument of `\use:n`, which is `\LaTeX3 error: <error message>`. Then `\msg_expandable_error_aux:w` cleans up. In fact, there is an extra subtlety: if the user inserts tokens for error recovery, they should be kept. Thus we also use an odd space character (with category code 7) and keep tokens until that space character, dropping everything else until `\q_stop`. The `\c_zero` prevents losing braces around the user-inserted text if any, and stops the expansion of `\romannumeral`.

```

9157 \group_begin:
9158 \char_set_catcode_math_superscript:N \^
9159 \char_set_lccode:nn {'^} {'\ }
9160 \char_set_lccode:nn {'L} {'L}
9161 \char_set_lccode:nn {'T} {'T}
9162 \char_set_lccode:nn {'X} {'X}
9163 \tl_to_lowercase:n
9164 {
9165   \cs_new:Npx \msg_expandable_error:n #1
9166   {
9167     \exp_not:n
9168     {

```



```

9169         \tex_romannumeral:D
9170         \exp_after:wN \exp_after:wN
9171         \exp_after:wN \msg_expandable_error_aux:w
9172         \exp_after:wN \exp_after:wN
9173         \exp_after:wN \c_zero
9174     }
9175     \exp_not:N \use:n { \exp_not:c { LaTeX3-error: } ^ #1 } ^
9176 }
9177 \cs_new:Npn \msg_expandable_error_aux:w #1 ^ #2 ^ { #1 }
9178 }
9179 \group_end:

```

(End definition for `\msg_expandable_error:n`. This function is documented on page 146.)

`\msg_expandable_kernel_error:nnnnnn` The command built from the csname `\c_msg_text_prefix_tl LaTeX / #1 / #2` takes four arguments and builds the error text, which is fed to `\msg_expandable_error:n`.

```

\msg_expandable_kernel_error:nnnnnn
\msg_expandable_kernel_error:nnnnn
\msg_expandable_kernel_error:nnnn
\msg_expandable_kernel_error:nnn
\msg_expandable_kernel_error:nn
9180 \cs_new:Npn \msg_expandable_kernel_error:nnnnnn #1#2#3#4#5#6
9181 {
9182     \exp_args:Nf \msg_expandable_error:n
9183     {
9184         \exp_args:Nnc \exp_after:wN \exp_stop_f:
9185         { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
9186         {#3} {#4} {#5} {#6}
9187     }
9188 }
9189 \cs_new:Npn \msg_expandable_kernel_error:nnnnn #1#2#3#4#5
9190 {
9191     \msg_expandable_kernel_error:nnnnnn
9192     {#1} {#2} {#3} {#4} {#5} { }
9193 }
9194 \cs_new:Npn \msg_expandable_kernel_error:nnnn #1#2#3#4
9195 {
9196     \msg_expandable_kernel_error:nnnnnn
9197     {#1} {#2} {#3} {#4} { } { }
9198 }
9199 \cs_new:Npn \msg_expandable_kernel_error:nnn #1#2#3
9200 {
9201     \msg_expandable_kernel_error:nnnnnn
9202     {#1} {#2} {#3} { } { } { }
9203 }
9204 \cs_new:Npn \msg_expandable_kernel_error:nn #1#2
9205 {
9206     \msg_expandable_kernel_error:nnnnnn
9207     {#1} {#2} { } { } { } { }
9208 }

```

(End definition for `\msg_expandable_kernel_error:nnnnnn` and others. These functions are documented on page ??.)

200.7 Showing variables

Functions defined in this section are used for diagnostic functions in `l3clist`, `l3io`, `l3prop`, `l3seq`, `xtemplate`

`\l_msg_show_tl` Used to store the material for the diagnostic functions of various modules.

```
9209 \tl_new:N \l_msg_show_tl
      (End definition for \l_msg_show_tl. This function is documented on page ??.)
```

`\msg_aux_use:nn` Print the text of a message to the terminal, without formatting.

```
\msg_aux_use:nnxxx
9210 \cs_new_protected:Npn \msg_aux_use:nn #1#2
9211 { \msg_aux_use:nnxxx {#1} {#2} { } { } { } { } }
9212 \cs_new_protected:Npn \msg_aux_use:nnxxx #1#2#3#4#5#6
9213 {
9214   \iow_term:x
9215   {
9216     \use:c { \c_msg_text_prefix_tl #1 / #2 }
9217     {#3} {#4} {#5} {#6}
9218   }
9219 }
      (End definition for \msg_aux_use:nn. This function is documented on page ??.)
```

`\msg_aux_show:Nnx` The arguments of `\msg_aux_show:Nnx` are

`\msg_aux_show:x`
`\msg_aux_show:w`

- The $\langle variable \rangle$ to be shown.
- The TF emptiness conditional for that type of variables.
- The type of the variable.
- A mapping of the form `\seq_map_function:NN $\langle variable \rangle$ \msg_aux_show:n`, which produces the formatted string.

We remove a new line and $\>_$ from the first item using a `w`-type auxiliary, and the fact that `f`-expansion removes a space. To avoid a low-level \TeX error if there is an empty argument, a simple test is used to keep the output “clean”. The odd `\exp_after:wN` and trailing `\prg_do_nothing:` improve the output slightly.

```
9220 \cs_new_protected:Npn \msg_aux_show:Nnx #1#2#3
9221 {
9222   \cs_if_exist:NTF #1
9223   {
9224     \msg_aux_use:nnxxx { LaTeX / #2 } { show } {#1} { } { } { }
9225     \msg_aux_show:x {#3}
9226   }
9227   {
9228     \msg_kernel_error:nnx { kernel } { variable-not-defined }
9229     { \token_to_str:N #1 }
9230   }
9231 }
9232 \cs_new_protected:Npn \msg_aux_show:x #1
```

```

9233 {
9234   \tl_set:Nx \l_msg_show_tl {#1}
9235   \tl_if_empty:NT \l_msg_show_tl
9236     { \tl_set:Nx \l_msg_show_tl { > } }
9237   \exp_args:Nf \etex_showtokens:D
9238     {
9239       \exp_after:wN \exp_after:wN
9240       \exp_after:wN \msg_aux_show:w
9241       \exp_after:wN \l_msg_show_tl
9242       \exp_after:wN
9243     }
9244   \prg_do_nothing:
9245 }
9246 \cs_new:Npn \msg_aux_show:w #1 > { }

```

(End definition for \msg_aux_show:Nnx. This function is documented on page ??.)

`\msg_aux_show:n` Each item in the variable is formatted using one of the following functions.

```

\msg_aux_show:nn 9247 \cs_new:Npn \msg_aux_show:n #1
\msg_aux_show_unbraced:nn 9248 {
9249   \iow_newline: > \c_space_tl \c_space_tl { \exp_not:n {#1} }
9250 }
9251 \cs_new:Npn \msg_aux_show:nn #1#2
9252 {
9253   \iow_newline: > \c_space_tl \c_space_tl { \exp_not:n {#1} }
9254   \c_space_tl \c_space_tl => \c_space_tl \c_space_tl { \exp_not:n {#2} }
9255 }
9256 \cs_new:Npn \msg_aux_show_unbraced:nn #1#2
9257 {
9258   \iow_newline: > \c_space_tl \c_space_tl \exp_not:n {#1}
9259   \c_space_tl \c_space_tl => \c_space_tl \c_space_tl \exp_not:n {#2}
9260 }

```

(End definition for \msg_aux_show:n. This function is documented on page ??.)

200.8 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\msg_class_new:nn` This is only ever used in a `set` fashion.

```

9261 <deprecated>
9262 \cs_new_eq:NN \msg_class_new:nn \msg_class_set:nn
9263 </deprecated>

```

(End definition for \msg_class_new:nn. This function is documented on page ??.)

`\msg_trace:nnxxxx` The performance here is never going to be good enough for tracing code, so let's be realistic.

```

\msg_trace:nnxx 9264 <deprecated>
\msg_trace:nnx 9265 \cs_new_eq:NN \msg_trace:nnxxxx \msg_log:nnxxxx
\msg_trace:nn 9266 \cs_new_eq:NN \msg_trace:nnxxx \msg_log:nnxxx
9267 \cs_new_eq:NN \msg_trace:nnxx \msg_log:nnxx

```

```

9268 \cs_new_eq:NN \msg_trace:nnx \msg_log:nnx
9269 \cs_new_eq:NN \msg_trace:nn \msg_log:nn
9270 </deprecated>
      (End definition for \msg_trace:nnxxxx and others. These functions are documented on page ??.)

```

```

\msg_generic_new:nnn
\msg_generic_new:nn
\msg_generic_set:nnn
\msg_generic_set:nn
\msg_direct_interrupt:xxxxx
\msg_direct_log:xx
\msg_direct_term:xx

```

These were all too low-level.

```

9271 <*deprecated>
9272 \cs_new_protected:Npn \msg_generic_new:nnn #1#2#3 { \deprecated }
9273 \cs_new_protected:Npn \msg_generic_new:nn #1#2 { \deprecated }
9274 \cs_new_protected:Npn \msg_generic_set:nnn #1#2#3 { \deprecated }
9275 \cs_new_protected:Npn \msg_generic_set:nn #1#2 { \deprecated }
9276 \cs_new_protected:Npn \msg_direct_interrupt:xxxxx #1#2#3#4#5 { \deprecated }
9277 \cs_new_protected:Npn \msg_direct_log:xx #1#2 { \deprecated }
9278 \cs_new_protected:Npn \msg_direct_term:xx #1#2 { \deprecated }
9279 </deprecated>
      (End definition for \msg_generic_new:nnn. This function is documented on page ??.)

```

```

\msg_kernel_bug:x
\c_msg_kernel_bug_text_tl
\c_msg_kernel_bug_more_text_tl

```

```

9280 <*deprecated>
9281 \cs_set_protected:Npn \msg_kernel_bug:x #1
9282 {
9283   \msg_interrupt:xxx { \c_msg_kernel_bug_text_tl }
9284   {
9285     #1
9286     \msg_see_documentation_text:n { LaTeX3 }
9287   }
9288   { \c_msg_kernel_bug_more_text_tl }
9289 }
9290 \tl_const:Nn \c_msg_kernel_bug_text_tl
9291 { This~is~a~LaTeX~bug:~check~coding! }
9292 \tl_const:Nn \c_msg_kernel_bug_more_text_tl
9293 {
9294   There~is~a~coding~bug~somewhere~around~here. \\
9295   This~probably~needs~examining~by~an~expert.
9296   \c_msg_return_text_tl
9297 }
9298 </deprecated>
      (End definition for \msg_kernel_bug:x. This function is documented on page ??.)
9299 </initex | package>

```

201 l3keys Implementation

```

9300 <*initex | package>
9301 <*package>
9302 \ProvidesExplPackage
9303   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
9304 \package_check_loaded_expl:

```

9305 `\package`

201.1 Low-level interface

For historical reasons this code uses the ‘keyval’ module prefix.

`\g_keyval_level_int` For nesting purposes an integer is needed for the current level.

9306 `\int_new:N \g_keyval_level_int`
(End definition for \g_keyval_level_int. This function is documented on page ??.)

`\l_keyval_key_tl` The current key name and value.

`\l_keyval_value_tl` 9307 `\tl_new:N \l_keyval_key_tl`
 9308 `\tl_new:N \l_keyval_value_tl`
(End definition for \l_keyval_key_tl and \l_keyval_value_tl. These functions are documented on page ??.)

`\l_keyval_sanitise_tl` Token list variables for dealing with awkward category codes in the input.

`\l_keyval_parse_tl` 9309 `\tl_new:N \l_keyval_sanitise_tl`
 9310 `\tl_new:N \l_keyval_parse_tl`
(End definition for \l_keyval_sanitise_tl. This function is documented on page ??.)

`\keyval_parse:n` The parsing function first deals with the category codes for = and ,, so that there are no odd events. The input is then handed off to the element by element system.

9311 `\group_begin:`
 9312 `\char_set_catcode_active:n { '\= }`
 9313 `\char_set_catcode_active:n { '\, }`
 9314 `\char_set_lccode:nn { '\8 } { '\= }`
 9315 `\char_set_lccode:nn { '\9 } { '\, }`
 9316 `\tl_to_lowercase:n`
 9317 `{`
 9318 `\group_end:`
 9319 `\cs_new_protected:Npn \keyval_parse:n #1`
 9320 `{`
 9321 `\group_begin:`
 9322 `\tl_clear:N \l_keyval_sanitise_tl`
 9323 `\tl_set:Nn \l_keyval_sanitise_tl {#1}`
 9324 `\tl_replace_all:Nnn \l_keyval_sanitise_tl { = } { 8 }`
 9325 `\tl_replace_all:Nnn \l_keyval_sanitise_tl { , } { 9 }`
 9326 `\tl_clear:N \l_keyval_parse_tl`
 9327 `\exp_after:wN \keyval_parse_elt:w \exp_after:wN`
 9328 `\q_no_value \l_keyval_sanitise_tl 9 \q_nil 9`
 9329 `\exp_after:wN \group_end:`
 9330 `\l_keyval_parse_tl`
 9331 `}`
 9332 `}`

(End definition for \keyval_parse:n. This function is documented on page ??.)

`\keyval_parse_elt:w` Each item to be parsed will have `\q_no_value` added to the front. Hence the blank test here can always be used to find a totally empty argument. If this is the case, the system loops round. If there is something to parse, there is a check for the `\q_nil` marker and if not a hand-off.

```

9333 \cs_new_protected:Npn \keyval_parse_elt:w #1 ,
9334 {
9335   \tl_if_blank:oTF { \use_none:n #1 }
9336   { \keyval_parse_elt:w \q_no_value }
9337   {
9338     \quark_if_nil:oF { \use_ii:nn #1 }
9339     {
9340       \keyval_split_key_value:w #1 = = \q_stop
9341       \keyval_parse_elt:w \q_no_value
9342     }
9343   }
9344 }

```

(End definition for `\keyval_parse_elt:w`. This function is documented on page ??.)

`\keyval_split_key_value:w` The key and value are handled separately. First the key is grabbed and saved as `\l_keyval_key_tl`. Then a check is need to see if there is a value at all: if not then the key name is simply added to the output. If there is a value then there is a check to ensure that there was only one `=` in the input (remembering some extra ones are around at the moment to prevent errors). All being well, there is an hand-off to find the value: the `\q_nil` is there to prevent loss of braces.

`\keyval_split_key_value_aux:wTF`

```

9345 \cs_new_protected:Npn \keyval_split_key_value:w #1 = #2 \q_stop
9346 {
9347   \keyval_split_key:w #1 \q_stop
9348   \str_if_eq:nnTF {#2} { = }
9349   {
9350     \tl_put_right:Nx \l_keyval_parse_tl
9351     {
9352       \exp_not:c
9353       { keyval_key_no_value_elt_ \int_use:N \g_keyval_level_int :n }
9354       { \exp_not:o \l_keyval_key_tl }
9355     }
9356   }
9357   {
9358     \keyval_split_key_value_aux:wTF #2 \q_no_value \q_stop
9359     { \keyval_split_value:w \q_nil #2 }
9360     { \msg_kernel_error:nn { keyval } { misplaced-equals-sign } }
9361   }
9362 }
9363 \cs_new:Npn \keyval_split_key_value_aux:wTF #1 = #2#3 \q_stop
9364 { \tl_if_head_eq_meaning:nNTF {#3} \q_no_value }

```

(End definition for `\keyval_split_key_value:w`. This function is documented on page ??.)

`\keyval_split_key:w` The aim here is to remove spaces and also exactly one set of braces. There is also a quark to remove, hence the `\use_none:n` appearing before application of `\tl_trim_spaces:n`.

```

9365 \cs_new_protected:Npn \keyval_split_key:w #1 \q_stop
9366 {
9367   \tl_set:Nx \l_keyval_key_tl
9368   { \exp_after:wN \tl_trim_spaces:n \exp_after:wN { \use_none:n #1 } }
9369 }

```

(End definition for \keyval_split_key:w. This function is documented on page ??.)

\keyval_split_value:w Here the value has to be separated from the equals signs and the leading \q_nil added in to keep the brace levels. First the processing function can be added to the output list. If there is no value, setting \l_keyval_value_tl with three groups removed will leave nothing at all, and so an empty group can be added to the parsed list. On the other hand, if the value is entirely contained within a set of braces then \l_keyval_value_tl will contain \q_nil only. In that case, strip off the leading quark using \use_ii:nnn, which also deals with any spaces.

```

9370 \cs_new_protected:Npn \keyval_split_value:w #1 = =
9371 {
9372   \tl_put_right:Nx \l_keyval_parse_tl
9373   {
9374     \exp_not:c
9375     { keyval_key_value_elt_ \int_use:N \g_keyval_level_int :nn }
9376     { \exp_not:o \l_keyval_key_tl }
9377   }
9378   \tl_set:Nx \l_keyval_value_tl
9379   { \exp_not:o { \use_none:nnn #1 \q_nil \q_nil } }
9380   \tl_if_empty:NTF \l_keyval_value_tl
9381   { \tl_put_right:Nn \l_keyval_parse_tl { { } } }
9382   {
9383     \quark_if_nil:NTF \l_keyval_value_tl
9384     {
9385       \tl_put_right:Nx \l_keyval_parse_tl
9386       { { \exp_not:o { \use_ii:nnn #1 \q_nil } } }
9387     }
9388     { \keyval_split_value_aux:w #1 \q_stop }
9389   }
9390 }

```

A similar idea to the key code: remove the spaces from each end and deal with one set of braces.

```

9391 \cs_new_protected:Npn \keyval_split_value_aux:w \q_nil #1 \q_stop
9392 {
9393   \tl_set:Nx \l_keyval_value_tl { \tl_trim_spaces:n {#1} }
9394   \tl_put_right:Nx \l_keyval_parse_tl
9395   { { \exp_not:o \l_keyval_value_tl } }
9396 }

```

(End definition for \keyval_split_value:w. This function is documented on page ??.)

\keyval_parse:NNn The outer parsing routine just sets up the processing functions and hands off.

```

9397 \cs_new_protected:Npn \keyval_parse:NNn #1#2#3
9398 {

```

```

9399 \int_gincr:N \g_keyval_level_int
9400 \cs_gset_eq:cN
9401 { keyval_key_no_value_elt_ \int_use:N \g_keyval_level_int :n } #1
9402 \cs_gset_eq:cN
9403 { keyval_key_value_elt_ \int_use:N \g_keyval_level_int :nn } #2
9404 \keyval_parse:n {#3}
9405 \int_gdecr:N \g_keyval_level_int
9406 }
(End definition for \keyval_parse:NNn. This function is documented on page 157.)
One message for the low level parsing system.
9407 \msg_kernel_new:nnnn { keyval } { misplaced-equals-sign }
9408 { Misplaced-equals-sign-in-key-value-input~\msg_line_number: }
9409 {
9410 LaTeX-is-attempting-to-parse-some-key-value-input-but-found~
9411 two-equals-signs-not-separated-by-a-comma.
9412 }

```

201.2 Constants and variables

`\c_keys_code_root_tl` The prefixes for the code and variables of the keys themselves.

```

\c_keys_vars_root_tl
9413 \tl_const:Nn \c_keys_code_root_tl { key~code~>~ }
9414 \tl_const:Nn \c_keys_vars_root_tl { key~var~>~ }

```

(End definition for `\c_keys_code_root_tl` and `\c_keys_vars_root_tl`. These functions are documented on page ??.)

`\c_keys_props_root_tl` The prefix for storing properties.

```

9415 \tl_const:Nn \c_keys_props_root_tl { key~prop~>~ }

```

(End definition for `\c_keys_props_root_tl`. This function is documented on page ??.)

`\c_keys_value_forbidden_tl` Two marker token lists.

```

\c_keys_value_required_tl
9416 \tl_const:Nn \c_keys_value_forbidden_tl { forbidden }
9417 \tl_const:Nn \c_keys_value_required_tl { required }

```

(End definition for `\c_keys_value_forbidden_tl` and `\c_keys_value_required_tl`. These functions are documented on page ??.)

`\l_keys_choice_int` Publicly accessible data on which choice is being used when several are generated as a set.

`\l_keys_choices_tl`

```

9418 \int_new:N \l_keys_choice_int
9419 \tl_new:N \l_keys_choices_tl

```

(End definition for `\l_keys_choice_int` and `\l_keys_choices_tl`. These functions are documented on page ??.)

`\l_keys_key_tl` The name of a key itself: needed when setting keys.

```

9420 \tl_new:N \l_keys_key_tl

```

(End definition for `\l_keys_key_tl`. This function is documented on page 155.)

`\l_keys_module_tl` The module for an entire set of keys.

```

9421 \tl_new:N \l_keys_module_tl

```


(End definition for \l_keys_module_tl. This function is documented on page ??.)

`\l_keys_no_value_bool` A marker is needed internally to show if only a key or a key plus a value was seen: this is recorded here.

9422 `\bool_new:N \l_keys_no_value_bool`

(End definition for \l_keys_no_value_bool. This function is documented on page ??.)

`\l_keys_path_tl` The “path” of the current key is stored here: this is available to the programmer and so is public.

9423 `\tl_new:N \l_keys_path_tl`

(End definition for \l_keys_path_tl. This function is documented on page 155.)

`\l_keys_property_tl` The “property” begin set for a key at definition time is stored here.

9424 `\tl_new:N \l_keys_property_tl`

(End definition for \l_keys_property_tl. This function is documented on page ??.)

`\l_keys_unknown_clist` Used when setting only known keys to store those left over.

9425 `\tl_new:N \l_keys_unknown_clist`

(End definition for \l_keys_unknown_clist. This function is documented on page ??.)

`\l_keys_value_tl` The value given for a key: may be empty if no value was given.

9426 `\tl_new:N \l_keys_value_tl`

(End definition for \l_keys_value_tl. This function is documented on page 155.)

201.3 The key defining mechanism

`\keys_define:nn` The public function for definitions is just a wrapper for the lower level mechanism, more or less. The outer function is designed to keep a track of the current module, to allow safe nesting. The module is set removing any leading / (which is not needed here).

`\keys_define_aux:nnn`
`\keys_define_aux:onn`

```
9427 \cs_new_protected:Npn \keys_define:nn
9428 { \keys_define_aux:onn \l_keys_module_tl }
9429 \cs_new_protected:Npn \keys_define_aux:nnn #1#2#3
9430 {
9431   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9432   \keyval_parse:NNn \keys_define_elt:n \keys_define_elt:nn {#3}
9433   \tl_set:Nn \l_keys_module_tl {#1}
9434 }
9435 \cs_generate_variant:Nn \keys_define_aux:nnn { o }
```

(End definition for \keys_define:nn. This function is documented on page ??.)

`\keys_define_elt:n` The outer functions here record whether a value was given and then converge on a common internal mechanism. There is first a search for a property in the current key name, then a check to make sure it is known before the code hands off to the next step.

`\keys_define_elt:nn`
`\keys_define_elt_aux:nn`

```
9436 \cs_new_protected:Npn \keys_define_elt:n #1
9437 {
9438   \bool_set_true:N \l_keys_no_value_bool
9439   \keys_define_elt_aux:nn {#1} { }
```

```

9440 }
9441 \cs_new_protected:Npn \keys_define_elt:nn #1#2
9442 {
9443   \bool_set_false:N \l_keys_no_value_bool
9444   \keys_define_elt_aux:nn {#1} {#2}
9445 }
9446 \cs_new_protected:Npn \keys_define_elt_aux:nn #1#2 {
9447   \keys_property_find:n {#1}
9448   \cs_if_exist:cTF { \c_keys_props_root_tl \l_keys_property_tl }
9449   { \keys_define_key:n {#2} }
9450   {
9451     \msg_kernel_error:nxx { keys } { property-unknown }
9452     { \l_keys_property_tl } { \l_keys_path_tl }
9453   }
9454 }

```

(End definition for \keys_define_elt:n. This function is documented on page ??.)

\keys_property_find:n Searching for a property means finding the last . in the input, and storing the text before
\keys_property_find_aux:w and after it. Everything is turned into strings, so there is no problem using an x-type expansion.

```

9455 \cs_new_protected:Npn \keys_property_find:n #1
9456 {
9457   \tl_set:Nx \l_keys_path_tl { \l_keys_module_tl / }
9458   \tl_if_in:nnTF {#1} { . }
9459   { \keys_property_find_aux:w #1 \q_stop }
9460   { \msg_kernel_error:nxx { keys } { key-no-property } {#1} }
9461 }
9462 \cs_new_protected:Npn \keys_property_find_aux:w #1 . #2 \q_stop
9463 {
9464   \tl_set:Nx \l_keys_path_tl { \l_keys_path_tl \tl_to_str:n {#1} }
9465   \tl_if_in:nnTF {#2} { . }
9466   {
9467     \tl_set:Nx \l_keys_path_tl { \l_keys_path_tl . }
9468     \keys_property_find_aux:w #2 \q_stop
9469   }
9470   { \tl_set:Nn \l_keys_property_tl { . #2 } }
9471 }

```

(End definition for \keys_property_find:n. This function is documented on page ??.)

\keys_define_key:n Two possible cases. If there is a value for the key, then just use the function. If not,
\keys_define_key_aux:w then a check to make sure there is no need for a value with the property. If there should be one then complain, otherwise execute it. There is no need to check for a : as if it is missing the earlier tests will have failed.

```

9472 \cs_new_protected:Npn \keys_define_key:n #1
9473 {
9474   \bool_if:NTF \l_keys_no_value_bool
9475   {
9476     \exp_after:wN \keys_define_key_aux:w
9477     \l_keys_property_tl \q_stop

```

```

9478         { \use:c { \c_keys_props_root_tl \l_keys_property_tl } }
9479         {
9480             \msg_kernel_error:nxxx { keys }
9481             { property-requires-value } { \l_keys_property_tl }
9482             { \l_keys_path_tl }
9483         }
9484     }
9485     { \use:c { \c_keys_props_root_tl \l_keys_property_tl } {#1} }
9486 }
9487 \cs_new_protected:Npn \keys_define_key_aux:w #1 : #2 \q_stop
9488 { \tl_if_empty:nTF {#2} }

```

(End definition for \keys_define_key:n. This function is documented on page ??.)

201.4 Turning properties into actions

\keys_bool_set:NN Boolean keys are really just choices, but all done by hand. The second argument here is the scope: either empty or g for global.

```

9489 \cs_new:Npn \keys_bool_set:NN #1#2
9490 {
9491     \cs_if_exist:NF #1 { \bool_new:N #1 }
9492     \keys_choice_make:
9493     \keys_cmd_set:nx { \l_keys_path_tl / true }
9494     { \exp_not:c { bool_ #2 set_true:N } \exp_not:N #1 }
9495     \keys_cmd_set:nx { \l_keys_path_tl / false }
9496     { \exp_not:c { bool_ #2 set_false:N } \exp_not:N #1 }
9497     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9498     {
9499         \msg_kernel_error:nnx { keys } { boolean-values-only }
9500         { \l_keys_key_tl }
9501     }
9502     \keys_default_set:n { true }
9503 }

```

(End definition for \keys_bool_set:NN. This function is documented on page ??.)

\keys_bool_set_inverse:NN Inverse boolean setting is much the same.

```

9504 \cs_new:Npn \keys_bool_set_inverse:NN #1#2
9505 {
9506     \cs_if_exist:NF #1 { \bool_new:N #1 }
9507     \keys_choice_make:
9508     \keys_cmd_set:nx { \l_keys_path_tl / true }
9509     { \exp_not:c { bool_ #2 set_false:N } \exp_not:N #1 }
9510     \keys_cmd_set:nx { \l_keys_path_tl / false }
9511     { \exp_not:c { bool_ #2 set_true:N } \exp_not:N #1 }
9512     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9513     {
9514         \msg_kernel_error:nnx { keys } { boolean-values-only }
9515         { \l_keys_key_tl }
9516     }
9517     \keys_default_set:n { true }

```

```

9518 }
      (End definition for \keys_bool_set_inverse:NN. This function is documented on page ??.)

```

\keys_choice_make: To make a choice from a key, two steps: set the code, and set the unknown key.

```

9519 \cs_new_protected_nopar:Npn \keys_choice_make:
9520 {
9521   \keys_cmd_set:nn { \l_keys_path_tl }
9522   { \keys_choice_find:n {##1} }
9523   \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9524   {
9525     \msg_kernel_error:nnxx { keys } { choice-unknown }
9526     { \l_keys_path_tl } {##1}
9527   }
9528 }
      (End definition for \keys_choice_make:. This function is documented on page ??.)

```

\keys_choices_make:nn Auto-generating choices means setting up the root key as a choice, then defining each choice in turn.

```

9529 \cs_new_protected:Npn \keys_choices_make:nn #1#2
9530 {
9531   \keys_choice_make:
9532   \int_zero:N \l_keys_choice_int
9533   \clist_map_inline:nn {#1}
9534   {
9535     \keys_cmd_set:nx { \l_keys_path_tl / ##1 }
9536     {
9537       \tl_set:Nn \exp_not:N \l_keys_choice_tl {##1}
9538       \int_set:Nn \exp_not:N \l_keys_choice_int
9539       { \int_use:N \l_keys_choice_int }
9540       \exp_not:n {#2}
9541     }
9542     \int_incr:N \l_keys_choice_int
9543   }
9544 }
      (End definition for \keys_choices_make:nn. This function is documented on page ??.)

```

\keys_choices_generate:n **\keys_choices_generate_aux:n** Creating multiple-choices means setting up the “indicator” code, then applying whatever the user wanted.

```

9545 \cs_new_protected:Npn \keys_choices_generate:n #1
9546 {
9547   \cs_if_exist:cTF
9548   { \c_keys_vars_root_tl \l_keys_path_tl .choice~code }
9549   {
9550     \keys_choice_make:
9551     \int_zero:N \l_keys_choice_int
9552     \clist_map_function:nn {#1} \keys_choices_generate_aux:n
9553   }
9554   {
9555     \msg_kernel_error:nnx { keys }

```

```

9556         { generate-choices-before-code } { \l_keys_path_tl }
9557     }
9558 }
9559 \cs_new_protected:Npn \keys_choices_generate_aux:n #1
9560 {
9561     \keys_cmd_set:nx { \l_keys_path_tl / #1 }
9562     {
9563         \tl_set:Nn \exp_not:N \l_keys_choice_tl {#1}
9564         \int_set:Nn \exp_not:N \l_keys_choice_int
9565         { \int_use:N \l_keys_choice_int }
9566         \exp_not:v
9567         { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9568     }
9569     \int_incr:N \l_keys_choice_int
9570 }

```

(End definition for \keys_choices_generate:n. This function is documented on page ??.)

\keys_choice_code_store:x The code for making multiple choices is stored in a token list.

```

9571 \cs_new_protected:Npn \keys_choice_code_store:x #1
9572 {
9573     \cs_if_exist:cF
9574     { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9575     {
9576         \tl_new:c
9577         { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9578     }
9579     \tl_set:cx { \c_keys_vars_root_tl \l_keys_path_tl .choice-code }
9580     {#1}
9581 }

```

(End definition for \keys_choice_code_store:x. This function is documented on page ??.)

\keys_cmd_set:nn Creating a new command means tidying up the properties and then making the internal
\keys_cmd_set:nx function which actually does the work.

```

\keys_cmd_set_aux:n 9582 \cs_new_protected:Npn \keys_cmd_set:nn #1#2
9583 {
9584     \keys_cmd_set_aux:n {#1}
9585     \cs_set:cpn { \c_keys_code_root_tl #1 } ##1 {#2}
9586 }
9587 \cs_new_protected:Npn \keys_cmd_set:nx #1#2
9588 {
9589     \keys_cmd_set_aux:n {#1}
9590     \cs_set:cpx { \c_keys_code_root_tl #1 } ##1 {#2}
9591 }
9592 \cs_new_protected:Npn \keys_cmd_set_aux:n #1
9593 {
9594     \tl_clear_new:c { \c_keys_vars_root_tl #1 .default }
9595     \tl_set:cn { \c_keys_vars_root_tl #1 .default } { \q_no_value }
9596     \tl_clear_new:c { \c_keys_vars_root_tl #1 .req }
9597 }

```

(End definition for `\keys_cmd_set:nn` and `\keys_cmd_set:nx`. These functions are documented on page ??.)

`\keys_default_set:n` Setting a default value is easy.

```
\keys_default_set:V
9598 \cs_new_protected:Npn \keys_default_set:n #1
9599 { \tl_set:cn { \c_keys_vars_root_tl \l_keys_path_tl .default } {#1} }
9600 \cs_generate_variant:Nn \keys_default_set:n { V }
```

(End definition for `\keys_default_set:n` and `\keys_default_set:V`. These functions are documented on page ??.)

`\keys_meta_make:n` To create a meta-key, simply set up to pass data through.

```
\keys_meta_make:x
9601 \cs_new_protected:Npn \keys_meta_make:n #1
9602 {
9603   \exp_args:NNo \keys_cmd_set:nn \l_keys_path_tl
9604   { \exp_after:wN \keys_set:nn \exp_after:wN { \l_keys_module_tl } {#1} }
9605 }
9606 \cs_new_protected:Npn \keys_meta_make:x #1
9607 {
9608   \keys_cmd_set:nx { \l_keys_path_tl }
9609   { \exp_not:N \keys_set:nn { \l_keys_module_tl } {#1} }
9610 }
```

(End definition for `\keys_meta_make:n` and `\keys_meta_make:x`. These functions are documented on page ??.)

`\keys_multichoice_find:n` Choices where several values can be selected are very similar to normal exclusive choices.

`\keys_multichoice_make:` There is just a slight change in implementation to map across a comma-separated list.

`\keys_multichoices_make:nn` This then requires that the appropriate set up takes place elsewhere.

```
9611 \cs_new:Npn \keys_multichoice_find:n #1
9612 { \clist_map_function:nN {#1} \keys_choice_find:n }
9613 \cs_new_protected_nopar:Npn \keys_multichoice_make:
9614 {
9615   \keys_cmd_set:nn { \l_keys_path_tl }
9616   { \keys_multichoice_find:n {##1} }
9617   \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9618   {
9619     \msg_kernel_error:nnxx { keys } { choice-unknown }
9620     { \l_keys_path_tl } {##1}
9621   }
9622 }
9623 \cs_new_protected:Npn \keys_multichoices_make:nn #1#2
9624 {
9625   \keys_multichoice_make:
9626   \int_zero:N \l_keys_choice_int
9627   \clist_map_inline:nn {#1}
9628   {
9629     \keys_cmd_set:nx { \l_keys_path_tl / ##1 }
9630     {
9631       \tl_set:Nn \exp_not:N \l_keys_choice_tl {##1}
9632       \int_set:Nn \exp_not:N \l_keys_choice_int
```

```

9633         { \int_use:N \l_keys_choice_int }
9634         \exp_not:n {#2}
9635     }
9636     \int_incr:N \l_keys_choice_int
9637 }
9638 }

```

(End definition for `\keys_multichoice_find:n`. This function is documented on page ??.)

`\keys_value_requirement:n` Values can be required or forbidden by having the appropriate marker set.

```

9639 \cs_new_protected:Npn \keys_value_requirement:n #1
9640 {
9641     \tl_set_eq:cc
9642     { \c_keys_vars_root_tl \l_keys_path_tl .req }
9643     { c_keys_value_ #1 _tl }
9644 }

```

(End definition for `\keys_value_requirement:n`. This function is documented on page ??.)

`\keys_variable_set:NnNN` Setting a variable takes the type and scope separately so that it is easy to make a new
`\keys_variable_set:cnNN` variable if needed. The three-argument version is set up so that the use of { } as an
`\keys_variable_set:NnN` N-type variable is only done once!
`\keys_variable_set:cnN`

```

9645 \cs_new_protected:Npn \keys_variable_set:NnNN #1#2#3#4
9646 {
9647     \cs_if_exist:NF #1 { \use:c { #2 _new:N } #1 }
9648     \keys_cmd_set:nx { \l_keys_path_tl }
9649     { \exp_not:c { #2 _ #3 set:N #4 } \exp_not:N #1 {##1} }
9650 }
9651 \cs_new_protected:Npn \keys_variable_set:NnN #1#2#3
9652 { \keys_variable_set:NnNN #1 {#2} { } #3 }
9653 \cs_generate_variant:Nn \keys_variable_set:NnNN { c }
9654 \cs_generate_variant:Nn \keys_variable_set:NnN { c }

```

(End definition for `\keys_variable_set:NnNN` and `\keys_variable_set:cnNN`. These functions are documented on page ??.)

201.5 Creating key properties

The key property functions are all wrappers for internal functions, meaning that things stay readable and can also be altered later on.

`.bool_set:N` One function for this.

```

9655 \cs_new_protected:cpn { \c_keys_props_root_tl .bool_set:N } #1
9656 { \keys_bool_set:NN #1 { } }
9657 \cs_new_protected:cpn { \c_keys_props_root_tl .bool_gset:N } #1
9658 { \keys_bool_set:NN #1 g }

```

(End definition for `.bool_set:N`. This function is documented on page 149.)

`.bool_set_inverse:N` One function for this.

`.bool_gset_inverse:N`

```

9659 \cs_new_protected:cpn { \c_keys_props_root_tl .bool_set_inverse:N } #1
9660 { \keys_bool_set_inverse:NN #1 { } }
9661 \cs_new_protected:cpn { \c_keys_props_root_tl .bool_gset_inverse:N } #1
9662 { \keys_bool_set_inverse:NN #1 g }

```

(End definition for .bool_set_inverse:N. This function is documented on page 149.)

`.choice:` Making a choice is handled internally, as it is also needed by `.generate_choices:n`.

```

9663 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .choice: }
9664 { \keys_choice_make: }

```

(End definition for .choice:. This function is documented on page ??.)

`.choices:nn` For auto-generation of a series of mutually-exclusive choices. Here, #1 will consist of two separate arguments, hence the slightly odd-looking implementation.

```

9665 \cs_new_protected:cpn { \c_keys_props_root_tl .choices:nn } #1
9666 { \keys_choices_make:nn #1 }

```

(End definition for .choices:nn. This function is documented on page 149.)

`.code:n` Creating code is simply a case of passing through to the underlying `set` function.

`.code:x`

```

9667 \cs_new_protected:cpn { \c_keys_props_root_tl .code:n } #1
9668 { \keys_cmd_set:nn { \l_keys_path_tl } {#1} }
9669 \cs_new_protected:cpn { \c_keys_props_root_tl .code:x } #1
9670 { \keys_cmd_set:nx { \l_keys_path_tl } {#1} }

```

(End definition for .code:n and .code:x. These functions are documented on page 150.)

`.choice_code:n` Storing the code for choices, using `\exp_not:n` to avoid needing two internal functions.

`.choice_code:x`

```

9671 \cs_new_protected:cpn { \c_keys_props_root_tl .choice_code:n } #1
9672 { \keys_choice_code_store:x { \exp_not:n {#1} } }
9673 \cs_new_protected:cpn { \c_keys_props_root_tl .choice_code:x } #1
9674 { \keys_choice_code_store:x {#1} }

```

(End definition for .choice_code:n and .choice_code:x. These functions are documented on page 149.)

`.clist_set:N`

`.clist_set:c`

`.clist_gset:N`

`.clist_gset:c`

```

9675 \cs_new_protected:cpn { \c_keys_props_root_tl .clist_set:N } #1
9676 { \keys_variable_set:NnN #1 { clist } n }
9677 \cs_new_protected:cpn { \c_keys_props_root_tl .clist_set:c } #1
9678 { \keys_variable_set:cnN {#1} { clist } n }
9679 \cs_new_protected:cpn { \c_keys_props_root_tl .clist_gset:N } #1
9680 { \keys_variable_set:NnNN #1 { clist } g n }
9681 \cs_new_protected:cpn { \c_keys_props_root_tl .clist_gset:c } #1
9682 { \keys_variable_set:cnNN {#1} { clist } g n }

```

(End definition for .clist_set:N and .clist_set:c. These functions are documented on page 149.)

`.default:n` Expansion is left to the internal functions.

```
.default:V 9683 \cs_new_protected:cpn { \c_keys_props_root_tl .default:n } #1  
          9684 { \keys_default_set:n {#1} }  
          9685 \cs_new_protected:cpn { \c_keys_props_root_tl .default:V } #1  
          9686 { \keys_default_set:V #1 }  
          (End definition for .default:n and .default:V. These functions are documented on page 150.)
```

`.dim_set:N` Setting a variable is very easy: just pass the data along.

```
.dim_set:c 9687 \cs_new_protected:cpn { \c_keys_props_root_tl .dim_set:N } #1  
.dim_gset:N 9688 { \keys_variable_set:NnN #1 { dim } n }  
.dim_gset:c 9689 \cs_new_protected:cpn { \c_keys_props_root_tl .dim_set:c } #1  
          9690 { \keys_variable_set:cnN {#1} { dim } n }  
          9691 \cs_new_protected:cpn { \c_keys_props_root_tl .dim_gset:N } #1  
          9692 { \keys_variable_set:NnNN #1 { dim } g n }  
          9693 \cs_new_protected:cpn { \c_keys_props_root_tl .dim_gset:c } #1  
          9694 { \keys_variable_set:cnNN {#1} { dim } g n }  
          (End definition for .dim_set:N and .dim_set:c. These functions are documented on page 150.)
```

`.fp_set:N` Setting a variable is very easy: just pass the data along.

```
.fp_set:c 9695 \cs_new_protected:cpn { \c_keys_props_root_tl .fp_set:N } #1  
.fp_gset:N 9696 { \keys_variable_set:NnN #1 { fp } n }  
.fp_gset:c 9697 \cs_new_protected:cpn { \c_keys_props_root_tl .fp_set:c } #1  
          9698 { \keys_variable_set:cnN {#1} { fp } n }  
          9699 \cs_new_protected:cpn { \c_keys_props_root_tl .fp_gset:N } #1  
          9700 { \keys_variable_set:NnNN #1 { fp } g n }  
          9701 \cs_new_protected:cpn { \c_keys_props_root_tl .fp_gset:c } #1  
          9702 { \keys_variable_set:cnNN {#1} { fp } g n }  
          (End definition for .fp_set:N and .fp_set:c. These functions are documented on page 150.)
```

`.generate_choices:n` Making choices is easy.

```
          9703 \cs_new_protected:cpn { \c_keys_props_root_tl .generate_choices:n } #1  
          9704 { \keys_choices_generate:n {#1} }  
          (End definition for .generate_choices:n. This function is documented on page 151.)
```

`.int_set:N` Setting a variable is very easy: just pass the data along.

```
.int_set:c 9705 \cs_new_protected:cpn { \c_keys_props_root_tl .int_set:N } #1  
.int_gset:N 9706 { \keys_variable_set:NnN #1 { int } n }  
.int_gset:c 9707 \cs_new_protected:cpn { \c_keys_props_root_tl .int_set:c } #1  
          9708 { \keys_variable_set:cnN {#1} { int } n }  
          9709 \cs_new_protected:cpn { \c_keys_props_root_tl .int_gset:N } #1  
          9710 { \keys_variable_set:NnNN #1 { int } g n }  
          9711 \cs_new_protected:cpn { \c_keys_props_root_tl .int_gset:c } #1  
          9712 { \keys_variable_set:cnNN {#1} { int } g n }  
          (End definition for .int_set:N and .int_set:c. These functions are documented on page 151.)
```

`.meta:n` Making a meta is handled internally.

`.meta:x`

```

9713 \cs_new_protected:cpn { \c_keys_props_root_tl .meta:n } #1
9714 { \keys_meta_make:n {#1} }
9715 \cs_new_protected:cpn { \c_keys_props_root_tl .meta:x } #1
9716 { \keys_meta_make:x {#1} }

```

(End definition for .meta:n and .meta:x. These functions are documented on page 151.)

`.multichoice:` The same idea as `.choice:` and `.choices:nn`, but where more than one choice is allowed.

`.multichoices:nn`

```

9717 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .multichoice: }
9718 { \keys_multichoice_make: }
9719 \cs_new_protected:cpn { \c_keys_props_root_tl .multichoices:nn } #1
9720 { \keys_multichoices_make:nn #1 }

```

(End definition for .multichoice:. This function is documented on page ??.)

`.skip_set:N` Setting a variable is very easy: just pass the data along.

`.skip_set:c`

`.skip_gset:N`

`.skip_gset:c`

```

9721 \cs_new_protected:cpn { \c_keys_props_root_tl .skip_set:N } #1
9722 { \keys_variable_set:NnN #1 { skip } n }
9723 \cs_new_protected:cpn { \c_keys_props_root_tl .skip_set:c } #1
9724 { \keys_variable_set:cnN {#1} { skip } n }
9725 \cs_new_protected:cpn { \c_keys_props_root_tl .skip_gset:N } #1
9726 { \keys_variable_set:NnNN #1 { skip } g n }
9727 \cs_new_protected:cpn { \c_keys_props_root_tl .skip_gset:c } #1
9728 { \keys_variable_set:cnNN {#1} { skip } g n }

```

(End definition for .skip_set:N and .skip_set:c. These functions are documented on page 151.)

`.tl_set:N` Setting a variable is very easy: just pass the data along.

`.tl_set:c`

`.tl_gset:N`

`.tl_gset:c`

`.tl_set_x:N`

`.tl_set_x:c`

`.tl_gset_x:N`

`.tl_gset_x:c`

```

9729 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_set:N } #1
9730 { \keys_variable_set:NnN #1 { tl } n }
9731 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_set:c } #1
9732 { \keys_variable_set:cnN {#1} { tl } n }
9733 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_set_x:N } #1
9734 { \keys_variable_set:NnN #1 { tl } x }
9735 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_set_x:c } #1
9736 { \keys_variable_set:cnN {#1} { tl } x }
9737 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_gset:N } #1
9738 { \keys_variable_set:NnNN #1 { tl } g n }
9739 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_gset:c } #1
9740 { \keys_variable_set:cnNN {#1} { tl } g n }
9741 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_gset_x:N } #1
9742 { \keys_variable_set:NnNN #1 { tl } g x }
9743 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_gset_x:c } #1
9744 { \keys_variable_set:cnNN {#1} { tl } g x }

```

(End definition for .tl_set:N and .tl_set:c. These functions are documented on page 152.)

`.value_forbidden:` These are very similar, so both call the same function.

`.value_required:`

```

9745 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .value_forbidden: }
9746 { \keys_value_requirement:n { forbidden } }
9747 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .value_required: }
9748 { \keys_value_requirement:n { required } }

```

(End definition for .value_forbidden:. This function is documented on page ??.)

201.6 Setting keys

`\keys_set:nn` A simple wrapper again.

```

\keys_set:nV 9749 \cs_new_protected:Npn \keys_set:nn
\keys_set:nv 9750 { \keys_set_aux:onn { \l_keys_module_tl } }
\keys_set:no 9751 \cs_new_protected:Npn \keys_set_aux:nnn #1#2#3
\keys_set_aux:nnn 9752 {
\keys_set_aux:onn 9753   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9754   \keyval_parse:NNn \keys_set_elt:n \keys_set_elt:nn {#3}
9755   \tl_set:Nn \l_keys_module_tl {#1}
9756 }
9757 \cs_generate_variant:Nn \keys_set:nn { nV , nv , no }
9758 \cs_generate_variant:Nn \keys_set_aux:nnn { o }

```

(End definition for \keys_set:nn and others. These functions are documented on page ??.)

```

\keys_set_known:nnN
\keys_set_known:nVN 9759 \cs_new_protected:Npn \keys_set_known:nnN
\keys_set_known:nvN 9760 { \keys_set_known_aux:onnN { \l_keys_module_tl } }
\keys_set_known:noN 9761 \cs_new_protected:Npn \keys_set_known_aux:nnnN #1#2#3#4
\keys_set_known_aux:nnnN 9762 {
\keys_set_known_aux:onnN 9763   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9764   \clist_clear:N \l_keys_unknown_clist
9765   \cs_set_eq:NN \keys_execute_unknown: \keys_execute_unknown_alt:
9766   \keyval_parse:NNn \keys_set_elt:n \keys_set_elt:nn {#3}
9767   \cs_set_eq:NN \keys_execute_unknown: \keys_execute_unknown_std:
9768   \tl_set:Nn \l_keys_module_tl {#1}
9769   \clist_set_eq:NN #4 \l_keys_unknown_clist
9770 }
9771 \cs_generate_variant:Nn \keys_set_known:nnN { nV , nv , no }
9772 \cs_generate_variant:Nn \keys_set_known_aux:nnnN { o }

```

(End definition for \keys_set_known:nnN and others. These functions are documented on page ??.)

`\keys_set_elt:n` A shared system once again. First, set the current path and add a default if needed.
`\keys_set_elt:nn` There are then checks to see if the a value is required or forbidden. If everything passes,
`\keys_set_elt_aux:nn` move on to execute the code.

```

9773 \cs_new_protected:Npn \keys_set_elt:n #1
9774 {
9775   \bool_set_true:N \l_keys_no_value_bool
9776   \keys_set_elt_aux:nn {#1} { }
9777 }
9778 \cs_new_protected:Npn \keys_set_elt:nn #1#2
9779 {
9780   \bool_set_false:N \l_keys_no_value_bool
9781   \keys_set_elt_aux:nn {#1} {#2}
9782 }
9783 \cs_new_protected:Npn \keys_set_elt_aux:nn #1#2
9784 {
9785   \tl_set:Nx \l_keys_key_tl { \tl_to_str:n {#1} }

```

```

9786 \tl_set:Nx \l_keys_path_tl { \l_keys_module_tl / \l_keys_key_tl }
9787 \keys_value_or_default:n {#2}
9788 \bool_if:nTF
9789 {
9790   \keys_if_value_p:n { required } &&
9791   \l_keys_no_value_bool
9792 }
9793 {
9794   \msg_kernel_error:nnx { keys } { value-required }
9795   { \l_keys_path_tl }
9796 }
9797 {
9798   \bool_if:nTF
9799   {
9800     \keys_if_value_p:n { forbidden } &&
9801     ! \l_keys_no_value_bool
9802   }
9803   {
9804     \msg_kernel_error:nnxx { keys } { value-forbidden }
9805     { \l_keys_path_tl } { \l_keys_value_tl }
9806   }
9807   { \keys_execute: }
9808 }
9809 }

```

(End definition for \keys_set_elt:n and \keys_set_elt:nn. These functions are documented on page ??.)

\keys_value_or_default:n If a value is given, return it as #1, otherwise send a default if available.

```

9810 \cs_new_protected:Npn \keys_value_or_default:n #1
9811 {
9812   \tl_set:Nn \l_keys_value_tl {#1}
9813   \bool_if:NT \l_keys_no_value_bool
9814   {
9815     \quark_if_no_value:cF { \c_keys_vars_root_tl \l_keys_path_tl .default }
9816     {
9817       \cs_if_exist:cT { \c_keys_vars_root_tl \l_keys_path_tl .default }
9818       {
9819         \tl_set_eq:Nc \l_keys_value_tl
9820         { \c_keys_vars_root_tl \l_keys_path_tl .default }
9821       }
9822     }
9823   }
9824 }

```

(End definition for \keys_value_or_default:n. This function is documented on page ??.)

\keys_if_value_p:n To test if a value is required or forbidden. A simple check for the existence of the appropriate marker.

```

9825 \prg_new_conditional:Npnn \keys_if_value:n #1 { p }
9826 {

```

```

9827     \tl_if_eq:ccTF { c_keys_value_ #1 _tl }
9828     { \c_keys_vars_root_tl \l_keys_path_tl .req }
9829     { \prg_return_true: }
9830     { \prg_return_false: }
9831 }

```

(End definition for \keys_if_value_p:n. This function is documented on page ??.)

\keys_execute: Actually executing a key is done in two parts. First, look for the key itself, then look for the unknown key with the same path. If both of these fail, complain.

```

\keys_execute_unknown:
\keys_execute_unknown_std:
\keys_execute_unknown_alt:
\keys_execute:nn
9832 \cs_new_nopar:Npn \keys_execute:
9833 { \keys_execute:nn { \l_keys_path_tl } { \keys_execute_unknown: } }
9834 \cs_new_nopar:Npn \keys_execute_unknown:
9835 {
9836     \keys_execute:nn { \l_keys_module_tl / unknown }
9837     {
9838         \msg_kernel_error:nnxx { keys } { key-unknown }
9839         { \l_keys_path_tl } { \l_keys_module_tl }
9840     }
9841 }
9842 \cs_new_eq:NN \keys_execute_unknown_std: \keys_execute_unknown:
9843 \cs_new_nopar:Npn \keys_execute_unknown_alt:
9844 {
9845     \clist_put_right:Nx \l_keys_unknown_clist
9846     {
9847         \exp_not:o \l_keys_key_tl
9848         \bool_if:NF \l_keys_no_value_bool
9849         { = { \exp_not:o \l_keys_value_tl } }
9850     }
9851 }
9852 \cs_new:Npn \keys_execute:nn #1#2
9853 {
9854     \cs_if_exist:cTF { \c_keys_code_root_tl #1 }
9855     {
9856         \exp_args:Nno \use:c { \c_keys_code_root_tl #1 }
9857         \l_keys_value_tl
9858     }
9859     {#2}
9860 }

```

(End definition for \keys_execute:. This function is documented on page ??.)

\keys_choice_find:n Executing a choice has two parts. First, try the choice given, then if that fails call the unknown key. That will exist, as it is created when a choice is first made. So there is no need for any escape code.

```

9861 \cs_new:Npn \keys_choice_find:n #1
9862 {
9863     \keys_execute:nn { \l_keys_path_tl / \tl_to_str:n {#1} }
9864     { \keys_execute:nn { \l_keys_path_tl / unknown } { } }
9865 }

```

(End definition for \keys_choice_find:n. This function is documented on page ??.)

201.7 Utilities

`\keys_if_exist:nn` A utility for others to see if a key exists.

```

9866 \prg_new_conditional:Npnn \keys_if_exist:nn #1#2 { p , T , F , TF }
9867 {
9868   \cs_if_exist:cTF { \c_keys_code_root_tl #1 / #2 }
9869   { \prg_return_true: }
9870   { \prg_return_false: }
9871 }

```

(End definition for \keys_if_exist:nn. This function is documented on page 156.)

`\keys_if_choice_exist:nnn` Just an alternative view on `\keys_if_exist:nn(TF)`.

```

9872 \prg_new_conditional:Npnn \keys_if_choice_exist:nnn #1#2#3 { p , T , F , TF }
9873 {
9874   \cs_if_exist:cTF { \c_keys_code_root_tl #1 / #2 / #3 }
9875   { \prg_return_true: }
9876   { \prg_return_false: }
9877 }

```

(End definition for \keys_if_choice_exist:nnn. This function is documented on page ??.)

`\keys_show:nn` Showing a key is just a question of using the correct name.

```

9878 \cs_new:Npn \keys_show:nn #1#2
9879 { \cs_show:c { \c_keys_code_root_tl #1 / \tl_to_str:n {#2} } }

```

(End definition for \keys_show:nn. This function is documented on page 156.)

201.8 Messages

For when there is a need to complain.

```

9880 \msg_kernel_new:nnnn { keys } { boolean-values-only }
9881 { Key~'#1'~accepts~boolean-values-only. }
9882 { The~key~'#1'~only~accepts~the~values~'true'~and~'false'. }
9883 \msg_kernel_new:nnnn { keys } { choice-unknown }
9884 { Choice~'#2'~unknown~for~key~'#1'. }
9885 {
9886   The~key~'#1'~takes~a~limited~number~of~values.\\
9887   The~input~given,~'#2',~is~not~on~the~list~accepted.
9888 }
9889 \msg_kernel_new:nnnn { keys } { generate-choices-before-code }
9890 { No~code~available~to~generate~choices~for~key~'#1'. }
9891 {
9892   \c_msg_coding_error_text_tl
9893   Before~using~.generate_choices:n~the~code~should~be~defined~
9894   with~'.choice_code:n'~or~'.choice_code:x'.
9895 }
9896 \msg_kernel_new:nnnn { keys } { key-no-property }
9897 { No~property~given~in~definition~of~key~'#1'. }
9898 {
9899   \c_msg_coding_error_text_tl
9900   Inside~\keys_define:nn~each~key~name

```

```

9901     needs-a~property:  \\
9902     ~ ~ #1 .<property>  \\
9903     LaTeX~did~not~find~a~'. '~to~indicate~the~start~of~a~property.
9904 }
9905 \msg_kernel_new:nnnn { keys } { key-unknown }
9906 { The~key~'#1'~is~unknown~and~is~being~ignored. }
9907 {
9908     The~module~'#2'~does~not~have~a~key~called~'#1'.\\
9909     Check~that~you~have~spelled~the~key~name~correctly.
9910 }
9911 \msg_kernel_new:nnnn { keys } { option-unknown }
9912 { Unknown~option~'#1'~for~package~#2. }
9913 {
9914     LaTeX~has~been~asked~to~set~an~option~called~'#1'~
9915     but~the~#2~package~has~not~created~an~option~with~this~name.
9916 }
9917 \msg_kernel_new:nnnn { keys } { property-requires-value }
9918 { The~property~'#1'~requires~a~value. }
9919 {
9920     \c_msg_coding_error_text_tl
9921     LaTeX~was~asked~to~set~property~'#2'~for~key~'#1'.\\
9922     No~value~was~given~for~the~property,~and~one~is~required.
9923 }
9924 \msg_kernel_new:nnnn { keys } { property-unknown }
9925 { The~key~property~'#1'~is~unknown. }
9926 {
9927     \c_msg_coding_error_text_tl
9928     LaTeX~has~been~asked~to~set~the~property~'#1'~for~key~'#2':~
9929     this~property~is~not~defined.
9930 }
9931 \msg_kernel_new:nnnn { keys } { value-forbidden }
9932 { The~key~'#1'~does~not~taken~a~value. }
9933 {
9934     The~key~'#1'~should~be~given~without~a~value.\\
9935     LaTeX~will~ignore~the~given~value~'#2'.
9936 }
9937 \msg_kernel_new:nnnn { keys } { value-required }
9938 { The~key~'#1'~requires~a~value. }
9939 {
9940     The~key~'#1'~must~have~a~value.\\
9941     No~value~was~present:~the~key~will~be~ignored.
9942 }

```

201.9 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

There is just one function for this now.

```

9943 <*deprecated>

```

```

\KV_process_space_removal_sanitiz:NNn
\KV_process_space_removal_no_sanitiz:NNn
\KV_process_no_space_removal_no_sanitiz:NNn

```

```

9944 \cs_new_eq:NN \KV_process_space_removal_sanitiz:NNn \keyval_parse:NNn
9945 \cs_new_eq:NN \KV_process_space_removal_no_sanitiz:NNn \keyval_parse:NNn
9946 \cs_new_eq:NN \KV_process_no_space_removal_no_sanitiz:NNn \keyval_parse:NNn
9947 </deprecated>
      (End definition for \KV_process_space_removal_sanitiz:NNn. This function is documented on
page ??.)
9948 </initex | package>

```

202 l3file implementation

The following test files are used for this code: *m3file001*.

```

9949 <*initex | package>
9950 <*package>
9951 \ProvidesExplPackage
9952   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
9953   \package_check_loaded_expl:
9954 </package>

```

`\g_file_current_name_tl` The name of the current file should be available at all times.

```

9955 \tl_new:N \g_file_current_name_tl

```

For the format the file name needs to be picked up at the start of the file. In package mode the current file name is collected from L^AT_EX 2_ε.

```

9956 <*initex>
9957 \tex_everyjob:D \exp_after:wN
9958   {
9959     \tex_the:D \tex_everyjob:D
9960     \tl_gset:Nx \g_file_current_name_tl { \tex_jobname:D }
9961   }
9962 </initex>
9963 <*package>
9964 \tl_gset_eq:NN \g_file_current_name_tl \@currname
9965 </package>

```

(End definition for `\g_file_current_name_tl`. This function is documented on page 158.)

`\g_file_stack_seq` The input list of files is stored as a sequence stack.

```

9966 \seq_new:N \g_file_stack_seq

```

(End definition for `\g_file_stack_seq`. This function is documented on page 159.)

`\g_file_record_seq` The total list of files used is recorded separately from the current file stack, as nothing is ever popped from this list.

```

9967 \seq_new:N \g_file_record_seq

```


The current file name should be included in the file list!

```

9968 <*initex>
9969 \tex_everyjob:D \exp_after:wN
9970 {
9971   \tex_the:D \tex_everyjob:D
9972   \seq_gput_right:NV \g_file_record_seq \g_file_current_name_tl
9973 }
9974 </initex>

```

(End definition for \g_file_record_seq. This function is documented on page 159.)

\l_file_name_tl Used to return the fully-qualified name of a file.

```

9975 \tl_new:N \l_file_name_tl

```

(End definition for \l_file_name_tl. This function is documented on page 159.)

\l_file_search_path_seq The current search path.

```

9976 \seq_new:N \l_file_search_path_seq

```

(End definition for \l_file_search_path_seq. This function is documented on page 159.)

\l_file_search_path_saved_seq The current search path has to be saved for package use.

```

9977 <*package>
9978 \seq_new:N \l_file_search_path_saved_seq
9979 </package>

```

(End definition for \l_file_search_path_saved_seq. This function is documented on page 159.)

\l_file_tmpa_seq Scratch space for comma list conversion in package mode.

```

9980 <*package>
9981 \seq_new:N \l_file_tmpa_seq
9982 </package>

```

(End definition for \l_file_tmpa_seq. This function is documented on page 159.)

\file_add_path:nN The way to test if a file exists is to try to open it: if it does not exist then T_EX will report end-of-file. For files which are in the current directory, this is straight-forward.
 \g_file_test_ior
 \file_add_path_search:nN For other locations, a search has to be made looking at each potential path in turn. The first location is of course treated as the correct one. If nothing is found, #2 is returned empty.

```

9983 \cs_new_protected:Npn \file_add_path:nN #1#2
9984 {
9985   \ior_open:Nn \g_file_test_ior {#1}
9986   \ior_if_eof:NTF \g_file_test_ior
9987     { \file_add_path_search:nN {#1} #2 }
9988   {
9989     \ior_close:N \g_file_test_ior
9990     \tl_set:Nx #2 {#1}
9991   }
9992 }
9993 \cs_new_protected:Npn \file_add_path_search:nN #1#2
9994 {
9995   \tl_clear:N #2

```

```

9996 <*package>
9997   \cs_if_exist:NT \input@path
9998   {
9999     \seq_set_eq:NN \l_file_search_path_saved_seq \l_file_search_path_seq
10000     \seq_set_from_clist:NN \l_file_tmpa_seq \input@path
10001     \seq_concat:NNN \l_file_search_path_seq
10002       \l_file_search_path_seq \l_file_tmpa_seq
10003   }
10004 </package>
10005   \seq_map_inline:Nn \l_file_search_path_seq
10006   {
10007     \ior_open:Nn \g_file_test_ior { ##1 #1 }
10008     \ior_if_eof:NF \g_file_test_ior
10009     {
10010       \tl_set:Nx #2 { ##1 #1 }
10011       \seq_map_break:
10012     }
10013   }
10014 <*package>
10015   \cs_if_exist:NT \input@path
10016   { \seq_set_eq:NN \l_file_search_path_seq \l_file_search_path_saved_seq }
10017 </package>
10018   \ior_close:N \g_file_test_ior
10019 }

```

(End definition for \file_add_path:nN. This function is documented on page ??.)

\file_if_exist:n The test for the existence of a file is a wrapper around the function to add a path to a file. If the file was found, the path will contain something, whereas if the file was not located then the return value will be empty.

```

10020 \prg_new_protected_conditional:Nnn \file_if_exist:n { T , F , TF }
10021 {
10022   \file_add_path:nN {#1} \l_file_name_tl
10023   \tl_if_empty:NTF \l_file_name_tl
10024   { \prg_return_false: }
10025   { \prg_return_true: }
10026 }

```

(End definition for \file_if_exist:n. This function is documented on page 158.)

\file_input:n Loading a file is done in a safe way, checking first that the file exists and loading only if it does.

```

10027 \cs_new_protected:Npn \file_input:n #1
10028 {
10029   \file_add_path:nN {#1} \l_file_name_tl
10030   \tl_if_empty:NF \l_file_name_tl
10031   {
10032     <*initex>
10033     \seq_gput_right:Nx \g_file_record_seq {#1}
10034     </initex>
10035   }
<*package>

```

```

10036         \@addtofilelist {#1}
10037     \end{package}
10038     \seq_gpush:NV \g_file_stack_seq \g_file_current_name_tl
10039     \tl_gset:Nn \g_file_current_name_tl {#1}
10040     \exp_after:wN \tex_input:D \l_file_name_tl \c_space_tl
10041     \seq_gpop:NN \g_file_stack_seq \g_file_current_name_tl
10042 }
10043 }

```

(End definition for \file_input:n. This function is documented on page 159.)

\file_path_include:n Wrapper functions to manage the search path.

```

\file_path_remove:n
10044 \cs_new_protected:Npn \file_path_include:n #1
10045 {
10046     \seq_if_in:NnF \l_file_search_path_seq {#1}
10047     { \seq_put_right:Nn \l_file_search_path_seq {#1} }
10048 }
10049 \cs_new_protected:Npn \file_path_remove:n #1
10050 { \seq_remove_all:Nn \l_file_search_path_seq {#1} }

```

(End definition for \file_path_include:n. This function is documented on page 159.)

\file_list: A function to list all files used to the log.

```

10051 \cs_new_protected_nopar:Npn \file_list:
10052 {
10053     \seq_remove_duplicates:N \g_file_record_seq
10054     \iow_log:n { *~File~List~* }
10055     \seq_map_inline:Nn \g_file_record_seq { \iow_log:n {##1} }
10056     \iow_log:n { ***** }
10057 }

```

(End definition for \file_list:. This function is documented on page ??.)

When used as a package, there is a need to hold onto the standard file list as well as the new one here.

```

10058 \begin{package}
10059 \AtBeginDocument
10060 {
10061     \seq_set_from_clist:NN \l_file_tmpa_seq \@filelist
10062     \seq_gconcat:NNN \g_file_record_seq \g_file_record_seq \l_file_tmpa_seq
10063 }
10064 \end{package}
10065 \end{initex | package}

```

203 l3fp Implementation

The following test files are used for this code: m3fp003.lvt.

```

10066 \begin{initex | package}

```

```

10067 <*package>
10068 \ProvidesExplPackage
10069   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
10070 \package_check_loaded_expl:
10071 </package>

```

203.1 Constants

`\c_forty_four` There is some speed to gain by moving numbers into fixed positions.

```

\c_one_million 10072 \int_const:Nn \c_forty_four { 44 }
\c_one_hundred_million 10073 \int_const:Nn \c_one_million { 1 000 000 }
\c_five_hundred_million 10074 \int_const:Nn \c_one_hundred_million { 100 000 000 }
\c_one_thousand_million 10075 \int_const:Nn \c_five_hundred_million { 500 000 000 }
10076 \int_const:Nn \c_one_thousand_million { 1 000 000 000 }
(End definition for \c_forty_four. This function is documented on page ??.)

```

`\c_fp_pi_by_four_decimal_int` Parts of π for trigonometric range reduction, implemented as `int` variables for speed.

```

\c_fp_pi_by_four_extended_int 10077 \int_new:N \c_fp_pi_by_four_decimal_int
\c_fp_pi_decimal_int 10078 \int_set:Nn \c_fp_pi_by_four_decimal_int { 785 398 158 }
\c_fp_pi_extended_int 10079 \int_new:N \c_fp_pi_by_four_extended_int
\c_fp_two_pi_decimal_int 10080 \int_set:Nn \c_fp_pi_by_four_extended_int { 897 448 310 }
\c_fp_two_pi_extended_int 10081 \int_new:N \c_fp_pi_decimal_int
10082 \int_set:Nn \c_fp_pi_decimal_int { 141 592 653 }
10083 \int_new:N \c_fp_pi_extended_int
10084 \int_set:Nn \c_fp_pi_extended_int { 589 793 238 }
10085 \int_new:N \c_fp_two_pi_decimal_int
10086 \int_set:Nn \c_fp_two_pi_decimal_int { 283 185 307 }
10087 \int_new:N \c_fp_two_pi_extended_int
10088 \int_set:Nn \c_fp_two_pi_extended_int { 179 586 477 }
(End definition for \c_fp_pi_by_four_decimal_int. This function is documented on page ??.)

```

`\c_e_fp` The value e as a “machine number”.

```

10089 \tl_const:Nn \c_e_fp { + 2.718281828 e 0 }
(End definition for \c_e_fp. This function is documented on page 165.)

```

`\c_one_fp` The constant value 1: used for fast comparisons.

```

10090 \tl_const:Nn \c_one_fp { + 1.000000000 e 0 }
(End definition for \c_one_fp. This function is documented on page 165.)

```

`\c_pi_fp` The value π as a “machine number”.

```

10091 \tl_const:Nn \c_pi_fp { + 3.141592654 e 0 }
(End definition for \c_pi_fp. This function is documented on page 165.)

```

`\c_undefined_fp` A marker for undefined values.

```

10092 \tl_const:Nn \c_undefined_fp { X 0.000000000 e 0 }
(End definition for \c_undefined_fp. This function is documented on page 165.)

```

`\c_zero_fp` The constant zero value.

```

10093 \tl_const:Nn \c_zero_fp { + 0.000000000 e 0 }
(End definition for \c_zero_fp. This function is documented on page 165.)

```

203.2 Variables

`\l_fp_arg_tl` A token list to store the formalised representation of the input for transcendental functions.

10094 `\tl_new:N \l_fp_arg_tl`

(End definition for `\l_fp_arg_tl`. This function is documented on page ??.)

`\l_fp_count_int` A counter for things like the number of divisions possible.

10095 `\int_new:N \l_fp_count_int`

(End definition for `\l_fp_count_int`. This function is documented on page ??.)

`\l_fp_div_offset_int` When carrying out division, an offset is used for the results to get the decimal part correct.

10096 `\int_new:N \l_fp_div_offset_int`

(End definition for `\l_fp_div_offset_int`. This function is documented on page ??.)

`\l_fp_exp_integer_int` Used for the calculation of exponent values.

`\l_fp_exp_decimal_int` 10097 `\int_new:N \l_fp_exp_integer_int`

`\l_fp_exp_extended_int` 10098 `\int_new:N \l_fp_exp_decimal_int`

`\l_fp_exp_exponent_int` 10099 `\int_new:N \l_fp_exp_extended_int`

10100 `\int_new:N \l_fp_exp_exponent_int`

(End definition for `\l_fp_exp_integer_int`. This function is documented on page ??.)

`\l_fp_input_a_sign_int` Storage for the input: two storage areas as there are at most two inputs.

`\l_fp_input_a_integer_int` 10101 `\int_new:N \l_fp_input_a_sign_int`

`\l_fp_input_a_decimal_int` 10102 `\int_new:N \l_fp_input_a_integer_int`

`\l_fp_input_a_exponent_int` 10103 `\int_new:N \l_fp_input_a_decimal_int`

`\l_fp_input_b_sign_int` 10104 `\int_new:N \l_fp_input_a_exponent_int`

`\l_fp_input_b_integer_int` 10105 `\int_new:N \l_fp_input_b_sign_int`

`\l_fp_input_b_decimal_int` 10106 `\int_new:N \l_fp_input_b_integer_int`

`\l_fp_input_b_exponent_int` 10107 `\int_new:N \l_fp_input_b_decimal_int`

10108 `\int_new:N \l_fp_input_b_exponent_int`

(End definition for `\l_fp_input_a_sign_int`. This function is documented on page ??.)

`\l_fp_input_a_extended_int` For internal use, “extended” floating point numbers are needed.

`\l_fp_input_b_extended_int` 10109 `\int_new:N \l_fp_input_a_extended_int`

10110 `\int_new:N \l_fp_input_b_extended_int`

(End definition for `\l_fp_input_a_extended_int`. This function is documented on page ??.)

`\l_fp_mul_a_i_int` Multiplication requires that the decimal part is split into parts so that there are no overflows.

`\l_fp_mul_a_ii_int` 10111 `\int_new:N \l_fp_mul_a_i_int`

`\l_fp_mul_a_iii_int` 10112 `\int_new:N \l_fp_mul_a_ii_int`

`\l_fp_mul_a_iv_int` 10113 `\int_new:N \l_fp_mul_a_iii_int`

`\l_fp_mul_a_v_int` 10114 `\int_new:N \l_fp_mul_a_iv_int`

`\l_fp_mul_a_vi_int` 10115 `\int_new:N \l_fp_mul_a_v_int`

`\l_fp_mul_b_i_int` 10116 `\int_new:N \l_fp_mul_a_vi_int`

`\l_fp_mul_b_ii_int` 10117 `\int_new:N \l_fp_mul_b_i_int`

`\l_fp_mul_b_iii_int`

`\l_fp_mul_b_iv_int`

`\l_fp_mul_b_v_int`

`\l_fp_mul_b_vi_int`

	10118	<code>\int_new:N \l_fp_mul_b_ii_int</code>	
	10119	<code>\int_new:N \l_fp_mul_b_iii_int</code>	
	10120	<code>\int_new:N \l_fp_mul_b_iv_int</code>	
	10121	<code>\int_new:N \l_fp_mul_b_v_int</code>	
	10122	<code>\int_new:N \l_fp_mul_b_vi_int</code>	
			(End definition for <code>\l_fp_mul_a_i_int</code> . This function is documented on page ??.)
<code>\l_fp_mul_output_int</code>			Space for multiplication results.
<code>\l_fp_mul_output_tl</code>	10123	<code>\int_new:N \l_fp_mul_output_int</code>	
	10124	<code>\tl_new:N \l_fp_mul_output_tl</code>	
			(End definition for <code>\l_fp_mul_output_int</code> . This function is documented on page ??.)
<code>\l_fp_output_sign_int</code>			Output is stored in the same way as input.
<code>\l_fp_output_integer_int</code>	10125	<code>\int_new:N \l_fp_output_sign_int</code>	
<code>\l_fp_output_decimal_int</code>	10126	<code>\int_new:N \l_fp_output_integer_int</code>	
<code>\l_fp_output_exponent_int</code>	10127	<code>\int_new:N \l_fp_output_decimal_int</code>	
	10128	<code>\int_new:N \l_fp_output_exponent_int</code>	
			(End definition for <code>\l_fp_output_sign_int</code> . This function is documented on page ??.)
<code>\l_fp_output_extended_int</code>			Again, for calculations an extended part.
	10129	<code>\int_new:N \l_fp_output_extended_int</code>	
			(End definition for <code>\l_fp_output_extended_int</code> . This function is documented on page ??.)
<code>\l_fp_round_carry_bool</code>			To indicate that a digit needs to be carried forward.
	10130	<code>\bool_new:N \l_fp_round_carry_bool</code>	
			(End definition for <code>\l_fp_round_carry_bool</code> . This function is documented on page ??.)
<code>\l_fp_round_decimal_tl</code>			A temporary store when rounding, to build up the decimal part without needing to do any maths.
	10131	<code>\tl_new:N \l_fp_round_decimal_tl</code>	
			(End definition for <code>\l_fp_round_decimal_tl</code> . This function is documented on page ??.)
<code>\l_fp_round_position_int</code>			Used to check the position for rounding.
<code>\l_fp_round_target_int</code>	10132	<code>\int_new:N \l_fp_round_position_int</code>	
	10133	<code>\int_new:N \l_fp_round_target_int</code>	
			(End definition for <code>\l_fp_round_position_int</code> . This function is documented on page ??.)
<code>\l_fp_sign_tl</code>			There are places where the sign needs to be set up “early”, so that the registers can be re-used.
	10134	<code>\tl_new:N \l_fp_sign_tl</code>	
			(End definition for <code>\l_fp_sign_tl</code> . This function is documented on page ??.)
<code>\l_fp_split_sign_int</code>			When splitting the input it is fastest to use a fixed name for the sign part, and to transfer it after the split is complete.
	10135	<code>\int_new:N \l_fp_split_sign_int</code>	
			(End definition for <code>\l_fp_split_sign_int</code> . This function is documented on page ??.)

`\l_fp_tmp_int` A scratch int: used only where the value is not carried forward.

```

10136 \int_new:N \l_fp_tmp_int
      (End definition for \l_fp_tmp_int. This function is documented on page ??.)

```

`\l_fp_tmp_tl` A scratch token list variable for expanding material.

```

10137 \tl_new:N \l_fp_tmp_tl
      (End definition for \l_fp_tmp_tl. This function is documented on page ??.)

```

`\l_fp_trig_octant_int` To track which octant the trigonometric input is in.

```

10138 \int_new:N \l_fp_trig_octant_int
      (End definition for \l_fp_trig_octant_int. This function is documented on page ??.)

```

`\l_fp_trig_sign_int` Used for the calculation of trigonometric values.

```

10139 \int_new:N \l_fp_trig_sign_int
10140 \int_new:N \l_fp_trig_decimal_int
10141 \int_new:N \l_fp_trig_extended_int
      (End definition for \l_fp_trig_sign_int. This function is documented on page ??.)

```

203.3 Parsing numbers

`\fp_read:N` Reading a stored value is made easier as the format is designed to match the delimited
`\fp_read_aux:w` function. This is always used to read the first value (register a).

```

10142 \cs_new_protected:Npn \fp_read:N #1
10143 { \exp_after:wN \fp_read_aux:w #1 \q_stop }
10144 \cs_new_protected:Npn \fp_read_aux:w #1#2 . #3 e #4 \q_stop
10145 {
10146   \if:w #1 -
10147     \l_fp_input_a_sign_int \c_minus_one
10148   \else:
10149     \l_fp_input_a_sign_int \c_one
10150   \fi:
10151   \l_fp_input_a_integer_int #2 \scan_stop:
10152   \l_fp_input_a_decimal_int #3 \scan_stop:
10153   \l_fp_input_a_exponent_int #4 \scan_stop:
10154 }
      (End definition for \fp_read:N. This function is documented on page ??.)

```

`\fp_split:Nn` The aim here is to use as much of TeX's mechanism as possible to pick up the numerical
`\fp_split_sign:` input without any mistakes. In particular, negative numbers have to be filtered out first
`\fp_split_exponent:` in case the integer part is 0 (in which case TeX would drop the - sign). That process
`\fp_split_aux_i:w` has to be done in a loop for cases where the sign is repeated. Finding an exponent is
`\fp_split_aux_ii:w` relatively easy, after which the next phase is to find the integer part, which will terminate
`\fp_split_aux_iii:w` with a ., and trigger the decimal-finding code. The later will allow the decimal to be too
`\fp_split_decimal:w` long, truncating the result.

```

10155 \cs_new_protected:Npn \fp_split:Nn #1#2
10156 {
10157   \tl_set:Nx \l_fp_tmp_tl {#2}

```

```

10158 \tl_set_rescan:Nno \l_fp_tmp_tl { \char_set_catcode_ignore:n { 32 } }
10159 { \l_fp_tmp_tl }
10160 \l_fp_split_sign_int \c_one
10161 \fp_split_sign:
10162 \use:c { l_fp_input_ #1 _sign_int } \l_fp_split_sign_int
10163 \exp_after:wN \fp_split_exponent:w \l_fp_tmp_tl e e \q_stop #1
10164 }
10165 \cs_new_protected_nopar:Npn \fp_split_sign:
10166 {
10167 \if_int_compare:w \pdfTeX_strcmp:D
10168 { \exp_after:wN \tl_head:w \l_fp_tmp_tl ? \q_stop } { - }
10169 = \c_zero
10170 \tl_set:Nx \l_fp_tmp_tl
10171 {
10172 \exp_after:wN
10173 \tl_tail:w \l_fp_tmp_tl \prg_do_nothing: \q_stop
10174 }
10175 \l_fp_split_sign_int -\l_fp_split_sign_int
10176 \exp_after:wN \fp_split_sign:
10177 \else:
10178 \if_int_compare:w \pdfTeX_strcmp:D
10179 { \exp_after:wN \tl_head:w \l_fp_tmp_tl ? \q_stop } { + }
10180 = \c_zero
10181 \tl_set:Nx \l_fp_tmp_tl
10182 {
10183 \exp_after:wN
10184 \tl_tail:w \l_fp_tmp_tl \prg_do_nothing: \q_stop
10185 }
10186 \exp_after:wN \exp_after:wN \exp_after:wN \fp_split_sign:
10187 \fi:
10188 \fi:
10189 }
10190 \cs_new_protected:Npn \fp_split_exponent:w #1 e #2 e #3 \q_stop #4
10191 {
10192 \use:c { l_fp_input_ #4 _exponent_int }
10193 \int_eval:w 0 #2 \scan_stop:
10194 \tex_afterassignment:D \fp_split_aux_i:w
10195 \use:c { l_fp_input_ #4 _integer_int }
10196 \int_eval:w 0 #1 . . \q_stop #4
10197 }
10198 \cs_new_protected:Npn \fp_split_aux_i:w #1 . #2 . #3 \q_stop
10199 { \fp_split_aux_ii:w #2 000000000 \q_stop }
10200 \cs_new_protected:Npn \fp_split_aux_ii:w #1#2#3#4#5#6#7#8#9
10201 { \fp_split_aux_iii:w {#1#2#3#4#5#6#7#8#9} }
10202 \cs_new_protected:Npn \fp_split_aux_iii:w #1#2 \q_stop
10203 {
10204 \l_fp_tmp_int 1 #1 \scan_stop:
10205 \exp_after:wN \fp_split_decimal:w
10206 \int_use:N \l_fp_tmp_int 000000000 \q_stop
10207 }

```



```

10208 \cs_new_protected:Npn \fp_split_decimal:w #1#2#3#4#5#6#7#8#9
10209 { \fp_split_decimal_aux:w {#2#3#4#5#6#7#8#9} }
10210 \cs_new_protected:Npn \fp_split_decimal_aux:w #1#2#3 \q_stop #4
10211 {
10212   \use:c { l_fp_input_ #4 _decimal_int } #1#2 \scan_stop:
10213   \if_int_compare:w
10214     \int_eval:w
10215       \use:c { l_fp_input_ #4 _integer_int } +
10216       \use:c { l_fp_input_ #4 _decimal_int }
10217     \scan_stop:
10218     = \c_zero
10219     \use:c { l_fp_input_ #4 _sign_int } \c_one
10220   \fi:
10221   \if_int_compare:w
10222     \use:c { l_fp_input_ #4 _integer_int } < \c_one_thousand_million
10223   \else:
10224     \exp_after:wN \fp_overflow_msg:
10225   \fi:
10226 }

```

(End definition for \fp_split:Nn. This function is documented on page ??.)

\fp_standardise:NNNN The idea here is to shift the input into a known exponent range. This is done using \TeX tokens where possible, as this is faster than arithmetic.

```

\fp_standardise_aux:NNNN
\fp_standardise_aux:
\fp_standardise_aux:w
10227 \cs_new_protected:Npn \fp_standardise:NNNN #1#2#3#4
10228 {
10229   \if_int_compare:w
10230     \int_eval:w #2 + #3 = \c_zero
10231     #1 \c_one
10232     #4 \c_zero
10233     \exp_after:wN \use_none:nnnn
10234   \else:
10235     \exp_after:wN \fp_standardise_aux:NNNN
10236   \fi:
10237   #1#2#3#4
10238 }
10239 \cs_new_protected:Npn \fp_standardise_aux:NNNN #1#2#3#4
10240 {
10241   \cs_set_protected_nopar:Npn \fp_standardise_aux:
10242   {
10243     \if_int_compare:w #2 = \c_zero
10244       \tex_advance:D #3 \c_one_thousand_million
10245       \exp_after:wN \fp_standardise_aux:w
10246       \int_use:N #3 \q_stop
10247       \exp_after:wN \fp_standardise_aux:
10248     \fi:
10249   }
10250   \cs_set_protected:Npn
10251     \fp_standardise_aux:w ##1##2##3##4##5##6##7##8##9 \q_stop
10252   {

```

```

10253      #2 ##2 \scan_stop:
10254      #3 ##3##4##5##6##7##8##9 0 \scan_stop:
10255      \tex_advance:D #4 \c_minus_one
10256    }
10257  \fp_standardise_aux:
10258  \cs_set_protected_nopar:Npn \fp_standardise_aux:
10259  {
10260    \if_int_compare:w #2 > \c_nine
10261      \tex_advance:D #2 \c_one_thousand_million
10262      \exp_after:wN \use_i:nn \exp_after:wN
10263      \fp_standardise_aux:w \int_use:N #2
10264      \exp_after:wN \fp_standardise_aux:
10265    \fi:
10266  }
10267  \cs_set_protected:Npn
10268  \fp_standardise_aux:w ##1##2##3##4##5##6##7##8##9
10269  {
10270    #2 ##1##2##3##4##5##6##7##8 \scan_stop:
10271    \tex_advance:D #3 \c_one_thousand_million
10272    \tex_divide:D #3 \c_ten
10273    \tl_set:Nx \l_fp_tmp_tl
10274    {
10275      ##9
10276      \exp_after:wN \use_none:n \int_use:N #3
10277    }
10278    #3 \l_fp_tmp_tl \scan_stop:
10279    \tex_advance:D #4 \c_one
10280  }
10281  \fp_standardise_aux:
10282  \if_int_compare:w #4 < \c_one_hundred
10283    \if_int_compare:w #4 > -\c_one_hundred
10284    \else:
10285      #1 \c_one
10286      #2 \c_zero
10287      #3 \c_zero
10288      #4 \c_zero
10289    \fi:
10290  \else:
10291    \exp_after:wN \fp_overflow_msg:
10292  \fi:
10293  }
10294  \cs_new_protected_nopar:Npn \fp_standardise_aux: { }
10295  \cs_new_protected_nopar:Npn \fp_standardise_aux:w { }

```

(End definition for \fp_standardise:NNNN. This function is documented on page ??.)

203.4 Internal utilities

`\fp_level_input_exponents:` The routines here are similar to those used to standardise the exponent. However, the
`\fp_level_input_exponents_a:` aim here is different: the two exponents need to end up the same.
`\fp_level_input_exponents_a:NNNNNNNN`
`\fp_level_input_exponents_b:`
`\fp_level_input_exponents_b:NNNNNNNN`

```

10296 \cs_new_protected_nopar:Npn \fp_level_input_exponents:
10297 {
10298   \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
10299     \exp_after:wN \fp_level_input_exponents_a:
10300   \else:
10301     \exp_after:wN \fp_level_input_exponents_b:
10302   \fi:
10303 }
10304 \cs_new_protected_nopar:Npn \fp_level_input_exponents_a:
10305 {
10306   \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
10307     \tex_advance:D \l_fp_input_b_integer_int \c_one_thousand_million
10308     \exp_after:wN \use_i:nn \exp_after:wN
10309       \fp_level_input_exponents_a:NNNNNNNNN
10310       \int_use:N \l_fp_input_b_integer_int
10311     \exp_after:wN \fp_level_input_exponents_a:
10312   \fi:
10313 }
10314 \cs_new_protected:Npn \fp_level_input_exponents_a:NNNNNNNNN
10315   #1#2#3#4#5#6#7#8#9
10316 {
10317   \l_fp_input_b_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
10318   \tex_advance:D \l_fp_input_b_decimal_int \c_one_thousand_million
10319   \tex_divide:D \l_fp_input_b_decimal_int \c_ten
10320   \tl_set:Nx \l_fp_tmp_tl
10321     {
10322       #9
10323       \exp_after:wN \use_none:n
10324       \int_use:N \l_fp_input_b_decimal_int
10325     }
10326   \l_fp_input_b_decimal_int \l_fp_tmp_tl \scan_stop:
10327   \tex_advance:D \l_fp_input_b_exponent_int \c_one
10328 }
10329 \cs_new_protected_nopar:Npn \fp_level_input_exponents_b:
10330 {
10331   \if_int_compare:w \l_fp_input_b_exponent_int > \l_fp_input_a_exponent_int
10332     \tex_advance:D \l_fp_input_a_integer_int \c_one_thousand_million
10333     \exp_after:wN \use_i:nn \exp_after:wN
10334       \fp_level_input_exponents_b:NNNNNNNNN
10335       \int_use:N \l_fp_input_a_integer_int
10336     \exp_after:wN \fp_level_input_exponents_b:
10337   \fi:
10338 }
10339 \cs_new_protected:Npn \fp_level_input_exponents_b:NNNNNNNNN
10340   #1#2#3#4#5#6#7#8#9
10341 {
10342   \l_fp_input_a_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
10343   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10344   \tex_divide:D \l_fp_input_a_decimal_int \c_ten
10345   \tl_set:Nx \l_fp_tmp_tl

```

```

10346     {
10347     #9
10348     \exp_after:wN \use_none:n
10349     \int_use:N \l_fp_input_a_decimal_int
10350     }
10351     \l_fp_input_a_decimal_int \l_fp_tmp_tl \scan_stop:
10352     \tex_advance:D \l_fp_input_a_exponent_int \c_one
10353   }
  
```

(End definition for `\fp_level_input_exponents:`. This function is documented on page ??.)

`\fp_tmp:w` Used for output of results, cutting down on `\exp_after:wN`. This is just a place holder definition.

```

10354 \cs_new_protected:Npn \fp_tmp:w #1#2 { }
  
```

(End definition for `\fp_tmp:w`. This function is documented on page ??.)

203.5 Operations for fp variables

The format of `fp` variables is tightly defined, so that they can be read quickly by the internal code. The format is a single sign token, a single number, the decimal point, nine decimal numbers, an `e` and finally the exponent. This final part may vary in length. When stored, floating points will always be stored with a value in the integer position unless the number is zero.

`\fp_new:N` Fixed-points always have a value, and of course this has to be initialised globally.

```

\fp_new:c 10355 \cs_new_protected:Npn \fp_new:N #1
10356 {
10357   \tl_new:N #1
10358   \tl_gset_eq:NN #1 \c_zero_fp
10359 }
10360 \cs_generate_variant:Nn \fp_new:N { c }
  
```

(End definition for `\fp_new:N` and `\fp_new:c`. These functions are documented on page ??.)

`\fp_const:Nn` A simple wrapper.

```

\fp_const:cn 10361 \cs_new_protected:Npn \fp_const:Nn #1#2
10362 {
10363   \fp_new:N #1
10364   \fp_gset:Nn #1 {#2}
10365 }
10366 \cs_generate_variant:Nn \fp_const:Nn { c }
  
```

(End definition for `\fp_const:Nn` and `\fp_const:cn`. These functions are documented on page ??.)

`\fp_zero:N` Zeroing fixed-points is pretty obvious.

```

\fp_zero:c 10367 \cs_new_protected:Npn \fp_zero:N #1
\fp_gzero:N 10368 { \tl_set_eq:NN #1 \c_zero_fp }
\fp_gzero:c 10369 \cs_new_protected:Npn \fp_gzero:N #1
10370 { \tl_gset_eq:NN #1 \c_zero_fp }
10371 \cs_generate_variant:Nn \fp_zero:N { c }
10372 \cs_generate_variant:Nn \fp_gzero:N { c }
  
```

(End definition for \fp_zero:N and \fp_zero:c. These functions are documented on page ??.)

\fp_set:Nn To trap any input errors, a very simple version of the parser is run here. This will pick
 \fp_set:cn up any invalid characters at this stage, saving issues later. The splitting approach is the
 \fp_gset:Nn same as the more advanced function later.
 \fp_gset:cn
 \fp_set_aux:NNn

```

10373 \cs_new_protected_nopar:Npn \fp_set:Nn { \fp_set_aux:NNn \tl_set:Nn }
10374 \cs_new_protected_nopar:Npn \fp_gset:Nn { \fp_set_aux:NNn \tl_gset:Nn }
10375 \cs_new_protected:Npn \fp_set_aux:NNn #1#2#3
10376 {
10377   \group_begin:
10378   \fp_split:Nn a {#3}
10379   \fp_standardise:NNNN
10380   \l_fp_input_a_sign_int
10381   \l_fp_input_a_integer_int
10382   \l_fp_input_a_decimal_int
10383   \l_fp_input_a_exponent_int
10384   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10385   \cs_set_protected_nopar:Npx \fp_tmp:w
10386   {
10387     \group_end:
10388     #1 \exp_not:N #2
10389     {
10390       \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10391       -
10392       \else:
10393       +
10394       \fi:
10395       \int_use:N \l_fp_input_a_integer_int
10396       .
10397       \exp_after:wN \use_none:n
10398       \int_use:N \l_fp_input_a_decimal_int
10399       e
10400       \int_use:N \l_fp_input_a_exponent_int
10401     }
10402   }
10403   \fp_tmp:w
10404 }
10405 \cs_generate_variant:Nn \fp_set:Nn { c }
10406 \cs_generate_variant:Nn \fp_gset:Nn { c }

```

(End definition for \fp_set:Nn and \fp_set:cn. These functions are documented on page ??.)

\fp_set_from_dim:Nn Here, dimensions are converted to fixed-points *via* a temporary variable. This ensures
 \fp_set_from_dim:cn that they always convert as points. The code is then essentially the same as for \fp_
 \fp_gset_from_dim:Nn set:Nn, but with the dimension passed so that it will be striped of the pt on the way
 \fp_gset_from_dim:cn through. The passage through a skip is used to remove any rubber part.
 \fp_set_from_dim_aux:NNn
 \fp_set_from_dim_aux:w
 \l_fp_tmp_dim
 \l_fp_tmp_skip

```

10407 \cs_new_protected_nopar:Npn \fp_set_from_dim:Nn
10408 { \fp_set_from_dim_aux:NNn \tl_set:Nx }
10409 \cs_new_protected_nopar:Npn \fp_gset_from_dim:Nn
10410 { \fp_set_from_dim_aux:NNn \tl_gset:Nx }

```

```

10411 \cs_new_protected:Npn \fp_set_from_dim_aux:NNn #1#2#3
10412 {
10413   \group_begin:
10414     \l_fp_tmp_skip \etex_glueexpr:D #3 \scan_stop:
10415     \l_fp_tmp_dim \l_fp_tmp_skip
10416     \fp_split:Nn a
10417     {
10418       \exp_after:wN \fp_set_from_dim_aux:w
10419       \dim_use:N \l_fp_tmp_dim
10420     }
10421     \fp_standardise:NNNN
10422     \l_fp_input_a_sign_int
10423     \l_fp_input_a_integer_int
10424     \l_fp_input_a_decimal_int
10425     \l_fp_input_a_exponent_int
10426     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10427     \cs_set_protected_nopar:Npx \fp_tmp:w
10428     {
10429       \group_end:
10430       #1 \exp_not:N #2
10431       {
10432         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10433         -
10434         \else:
10435         +
10436         \fi:
10437         \int_use:N \l_fp_input_a_integer_int
10438         .
10439         \exp_after:wN \use_none:n
10440         \int_use:N \l_fp_input_a_decimal_int
10441         e
10442         \int_use:N \l_fp_input_a_exponent_int
10443       }
10444     }
10445     \fp_tmp:w
10446   }
10447   \cs_set_protected_nopar:Npx \fp_set_from_dim_aux:w
10448   {
10449     \cs_set:Npn \exp_not:N \fp_set_from_dim_aux:w
10450     ##1 \tl_to_str:n { pt } {##1}
10451   }
10452   \fp_set_from_dim_aux:w
10453   \cs_generate_variant:Nn \fp_set_from_dim:Nn { c }
10454   \cs_generate_variant:Nn \fp_gset_from_dim:Nn { c }
10455   \dim_new:N \l_fp_tmp_dim
10456   \skip_new:N \l_fp_tmp_skip

```

(End definition for \fp_set_from_dim:Nn and \fp_set_from_dim:cn. These functions are documented on page ??.)

\fp_set_eq:NN Pretty simple, really.
\fp_set_eq:cN
\fp_set_eq:Nc
\fp_set_eq:cc
\fp_gset_eq:NN
\fp_gset_eq:cN
\fp_gset_eq:Nc
\fp_gset_eq:cc

```

10457 \cs_new_eq:NN \fp_set_eq:NN \tl_set_eq:NN
10458 \cs_new_eq:NN \fp_set_eq:cN \tl_set_eq:cN
10459 \cs_new_eq:NN \fp_set_eq:Nc \tl_set_eq:Nc
10460 \cs_new_eq:NN \fp_set_eq:cc \tl_set_eq:cc
10461 \cs_new_eq:NN \fp_gset_eq:NN \tl_gset_eq:NN
10462 \cs_new_eq:NN \fp_gset_eq:cN \tl_gset_eq:cN
10463 \cs_new_eq:NN \fp_gset_eq:Nc \tl_gset_eq:Nc
10464 \cs_new_eq:NN \fp_gset_eq:cc \tl_gset_eq:cc

```

(End definition for \fp_set_eq:NN and others. These functions are documented on page ??.)

\fp_show:N Simple showing of the underlying variable.

```

\fp_show:c 10465 \cs_new_eq:NN \fp_show:N \tl_show:N
10466 \cs_new_eq:NN \fp_show:c \tl_show:c

```

(End definition for \fp_show:N and \fp_show:c. These functions are documented on page ??.)

\fp_use:N The idea of the \fp_use:N function to convert the stored value into something suitable for TeX to use as a number in an expandable manner. The first step is to deal with the sign, then work out how big the input is.

```

\fp_use:c
\fp_use_aux:w
\fp_use_none:w 10467 \cs_new:Npn \fp_use:N #1
\fp_use_small:w 10468 { \exp_after:wN \fp_use_aux:w #1 \q_stop }
\fp_use_large:w 10469 \cs_generate_variant:Nn \fp_use:N { c }
\fp_use_large_aux_i:w 10470 \cs_new:Npn \fp_use_aux:w #1#2 e #3 \q_stop
\fp_use_large_aux_1:w 10471 {
\fp_use_large_aux_2:w 10472 \if:w #1 -
\fp_use_large_aux_3:w 10473 -
\fp_use_large_aux_4:w 10474 \fi:
\fp_use_large_aux_5:w 10475 \if_int_compare:w #3 > \c_zero
\fp_use_large_aux_6:w 10476 \exp_after:wN \fp_use_large:w
\fp_use_large_aux_7:w 10477 \else:
\fp_use_large_aux_8:w 10478 \if_int_compare:w #3 < \c_zero
\fp_use_large_aux_i:w 10479 \exp_after:wN \exp_after:wN \exp_after:wN
\fp_use_large_aux_ii:w 10480 \fp_use_small:w
10481 \else:
10482 \exp_after:wN \exp_after:wN \exp_after:wN \fp_use_none:w
10483 \fi:
10484 \fi:
10485 #2 e #3 \q_stop
10486 }

```

When the exponent is zero, the input is simply returned as output.

```

10487 \cs_new:Npn \fp_use_none:w #1 e #2 \q_stop {#1}

```

For small numbers (less than 1) the correct number of zeros have to be inserted, but the decimal point is easy.

```

10488 \cs_new:Npn \fp_use_small:w #1 . #2 e #3 \q_stop
10489 {
10490 0 .
10491 \prg_replicate:nn { -#3 - 1 } { 0 }
10492 #1#2
10493 }

```

Life is more complex for large numbers. The decimal point needs to be shuffled, with potentially some zero-filling for very large values.

```

10494 \cs_new:Npn \fp_use_large:w #1 . #2 e #3 \q_stop
10495 {
10496   \if_int_compare:w #3 < \c_ten
10497     \exp_after:wN \fp_use_large_aux_i:w
10498   \else:
10499     \exp_after:wN \fp_use_large_aux_ii:w
10500   \fi:
10501   #1#2 e #3 \q_stop
10502 }
10503 \cs_new:Npn \fp_use_large_aux_i:w #1#2 e #3 \q_stop
10504 {
10505   #1
10506   \use:c { fp_use_large_aux_ #3 :w } #2 \q_stop
10507 }
10508 \cs_new:cpn { fp_use_large_aux_1:w } #1#2 \q_stop { #1 . #2 }
10509 \cs_new:cpn { fp_use_large_aux_2:w } #1#2#3 \q_stop
10510 { #1#2 . #3 }
10511 \cs_new:cpn { fp_use_large_aux_3:w } #1#2#3#4 \q_stop
10512 { #1#2#3 . #4 }
10513 \cs_new:cpn { fp_use_large_aux_4:w } #1#2#3#4#5 \q_stop
10514 { #1#2#3#4 . #5 }
10515 \cs_new:cpn { fp_use_large_aux_5:w } #1#2#3#4#5#6 \q_stop
10516 { #1#2#3#4#5 . #6 }
10517 \cs_new:cpn { fp_use_large_aux_6:w } #1#2#3#4#5#6#7 \q_stop
10518 { #1#2#3#4#5#6 . #7 }
10519 \cs_new:cpn { fp_use_large_aux_7:w } #1#2#3#4#5#6#7#8 \q_stop
10520 { #1#2#3#4#6#7 . #8 }
10521 \cs_new:cpn { fp_use_large_aux_8:w } #1#2#3#4#5#6#7#8#9 \q_stop
10522 { #1#2#3#4#5#6#7#8 . #9 }
10523 \cs_new:cpn { fp_use_large_aux_9:w } #1 \q_stop { #1 . }
10524 \cs_new:Npn \fp_use_large_aux_ii:w #1 e #2 \q_stop
10525 {
10526   #1
10527   \prg_replicate:nn { #2 - 9 } { 0 }
10528   .
10529 }

```

(End definition for \fp_use:N and \fp_use:c. These functions are documented on page ??.)

203.6 Transferring to other types

The `\fp_use:N` function converts a floating point variable to a form that can be used by \TeX . Here, the functions are slightly different, as some information may be discarded.

`\fp_to_dim:N` A very simple wrapper.

```

\fp_to_dim:c 10530 \cs_new:Npn \fp_to_dim:N #1 { \fp_use:N #1 pt }
10531 \cs_generate_variant:Nn \fp_to_dim:N { c }

```


(End definition for `\fp_to_dim:N` and `\fp_to_dim:c`. These functions are documented on page ??.)

`\fp_to_int:N` Converting to integers in an expandable manner is very similar to simply using floating
`\fp_to_int:c` point variables, particularly in the lead-off.
`\fp_to_int_aux:w` 10532 `\cs_new:Npn \fp_to_int:N #1`
`\fp_to_int_none:w` 10533 `{ \exp_after:wN \fp_to_int_aux:w #1 \q_stop }`
`\fp_to_int_small:w` 10534 `\cs_generate_variant:Nn \fp_to_int:N { c }`
`\fp_to_int_large:w` 10535 `\cs_new:Npn \fp_to_int_aux:w #1#2 e #3 \q_stop`
`\fp_to_int_large_aux_i:w` 10536 `{`
`\fp_to_int_large_aux_1:w` 10537 `\if:w #1 -`
`\fp_to_int_large_aux_2:w` 10538 `-`
`\fp_to_int_large_aux_3:w` 10539 `\fi:`
`\fp_to_int_large_aux_4:w` 10540 `\if_int_compare:w #3 < \c_zero`
`\fp_to_int_large_aux_5:w` 10541 `\exp_after:wN \fp_to_int_small:w`
`\fp_to_int_large_aux_6:w` 10542 `\else:`
`\fp_to_int_large_aux_7:w` 10543 `\exp_after:wN \fp_to_int_large:w`
`\fp_to_int_large_aux_8:w` 10544 `\fi:`
`\fp_to_int_large_aux_i:w` 10545 `#2 e #3 \q_stop`
`\fp_to_int_large_aux:nnn` 10546 `}`
`\fp_to_int_large_aux_ii:w` For small numbers, if the decimal part is greater than a half then there is rounding up
to do.

```

10547 \cs_new:Npn \fp_to_int_small:w #1 . #2 e #3 \q_stop
10548 {
10549   \if_int_compare:w #3 > \c_one
10550   \else:
10551     \if_int_compare:w #1 < \c_five
10552     0
10553   \else:
10554     1
10555   \fi:
10556 \fi:
10557 }

```

For large numbers, the idea is to split off the part for rounding, do the rounding and fill if needed.

```

10558 \cs_new:Npn \fp_to_int_large:w #1 . #2 e #3 \q_stop
10559 {
10560   \if_int_compare:w #3 < \c_ten
10561   \exp_after:wN \fp_to_int_large_aux_i:w
10562   \else:
10563     \exp_after:wN \fp_to_int_large_aux_ii:w
10564   \fi:
10565   #1#2 e #3 \q_stop
10566 }
10567 \cs_new:Npn \fp_to_int_large_aux_i:w #1#2 e #3 \q_stop
10568 { \use:c { fp_to_int_large_aux_ #3 :w } #2 \q_stop {#1} }
10569 \cs_new:cpn { fp_to_int_large_aux_1:w } #1#2 \q_stop
10570 { \fp_to_int_large_aux:nnn { #2 0 } {#1} }

```

```

10571 \cs_new:cpn { fp_to_int_large_aux_2:w } #1#2#3 \q_stop
10572 { \fp_to_int_large_aux:nnn { #3 00 } {#1#2} }
10573 \cs_new:cpn { fp_to_int_large_aux_3:w } #1#2#3#4 \q_stop
10574 { \fp_to_int_large_aux:nnn { #4 000 } {#1#2#3} }
10575 \cs_new:cpn { fp_to_int_large_aux_4:w } #1#2#3#4#5 \q_stop
10576 { \fp_to_int_large_aux:nnn { #5 0000 } {#1#2#3#4} }
10577 \cs_new:cpn { fp_to_int_large_aux_5:w } #1#2#3#4#5#6 \q_stop
10578 { \fp_to_int_large_aux:nnn { #6 00000 } {#1#2#3#4#5} }
10579 \cs_new:cpn { fp_to_int_large_aux_6:w } #1#2#3#4#5#6#7 \q_stop
10580 { \fp_to_int_large_aux:nnn { #7 000000 } {#1#2#3#4#5#6} }
10581 \cs_new:cpn { fp_to_int_large_aux_7:w } #1#2#3#4#5#6#7#8 \q_stop
10582 { \fp_to_int_large_aux:nnn { #8 0000000 } {#1#2#3#4#5#6#7} }
10583 \cs_new:cpn { fp_to_int_large_aux_8:w } #1#2#3#4#5#6#7#8#9 \q_stop
10584 { \fp_to_int_large_aux:nnn { #9 00000000 } {#1#2#3#4#5#6#7#8} }
10585 \cs_new:cpn { fp_to_int_large_aux_9:w } #1 \q_stop {#1}
10586 \cs_new:Npn \fp_to_int_large_aux:nnn #1#2#3
10587 {
10588   \if_int_compare:w #1 < \c_five_hundred_million
10589     #3#2
10590   \else:
10591     \int_value:w \int_eval:w #3#2 + 1 \int_eval_end:
10592   \fi:
10593 }
10594 \cs_new:Npn \fp_to_int_large_aux_ii:w #1 e #2 \q_stop
10595 {
10596   #1
10597   \prg_replicate:nn { #2 - 9 } { 0 }
10598 }

```

(End definition for \fp_to_int:N and \fp_to_int:c. These functions are documented on page ??.)

<pre> \fp_to_tl:N \fp_to_tl:c \fp_to_tl_aux:w \fp_to_tl_large:w \fp_to_tl_large_aux_i:w \fp_to_tl_large_aux_ii:w \fp_to_tl_large_0:w \fp_to_tl_large_1:w \fp_to_tl_large_2:w \fp_to_tl_large_3:w \fp_to_tl_large_4:w \fp_to_tl_large_5:w \fp_to_tl_large_6:w \fp_to_tl_large_7:w \fp_to_tl_large_8:w \fp_to_tl_large_8_aux:w \fp_to_tl_large_9:w \fp_to_tl_small:w \fp_to_tl_small_one:w \fp_to_tl_small_two:w \fp_to_tl_small_aux:w \fp_to_tl_large_zeros:NNNNNNNNN \fp_to_tl_small_zeros:NNNNNNNNN \fp_use_iix_ix:NNNNNNNNN \fp_use_ix:NNNNNNNNN \fp_use_i_to_vii:NNNNNNNNN \fp_use_i_to_iix:NNNNNNNNN </pre>	<pre> 10599 \cs_new:Npn \fp_to_tl:N #1 10600 { \exp_after:wN \fp_to_tl_aux:w #1 \q_stop } 10601 \cs_generate_variant:Nn \fp_to_tl:N { c } 10602 \cs_new:Npn \fp_to_tl_aux:w #1#2 e #3 \q_stop 10603 { 10604 \if:w #1 - 10605 - 10606 \fi: 10607 \if_int_compare:w #3 < \c_zero 10608 \exp_after:wN \fp_to_tl_small:w 10609 \else: 10610 \exp_after:wN \fp_to_tl_large:w 10611 \fi: 10612 #2 e #3 \q_stop 10613 } </pre>
---	--

For “large” numbers (exponent ≥ 0) there are two cases. For very large exponents (≥ 10) life is easy: apart from dropping extra zeros there is no work to do. On the other hand, for intermediate exponent values the decimal needs to be moved, then zeros can be dropped.

```

10614 \cs_new:Npn \fp_to_tl_large:w #1 e #2 \q_stop
10615 {
10616   \if_int_compare:w #2 < \c_ten
10617     \exp_after:wN \fp_to_tl_large_aux_i:w
10618   \else:
10619     \exp_after:wN \fp_to_tl_large_aux_ii:w
10620   \fi:
10621   #1 e #2 \q_stop
10622 }
10623 \cs_new:Npn \fp_to_tl_large_aux_i:w #1 e #2 \q_stop
10624 { \use:c { fp_to_tl_large_#2 :w } #1 \q_stop }
10625 \cs_new:Npn \fp_to_tl_large_aux_ii:w #1 . #2 e #3 \q_stop
10626 {
10627   #1
10628   \fp_to_tl_large_zeros:NNNNNNNN #2
10629   e #3
10630 }
10631 \cs_new:cpn { fp_to_tl_large_0:w } #1 . #2 \q_stop
10632 {
10633   #1
10634   \fp_to_tl_large_zeros:NNNNNNNN #2
10635 }
10636 \cs_new:cpn { fp_to_tl_large_1:w } #1 . #2#3 \q_stop
10637 {
10638   #1#2
10639   \fp_to_tl_large_zeros:NNNNNNNN #3 0
10640 }
10641 \cs_new:cpn { fp_to_tl_large_2:w } #1 . #2#3#4 \q_stop
10642 {
10643   #1#2#3
10644   \fp_to_tl_large_zeros:NNNNNNNN #4 00
10645 }
10646 \cs_new:cpn { fp_to_tl_large_3:w } #1 . #2#3#4#5 \q_stop
10647 {
10648   #1#2#3#4
10649   \fp_to_tl_large_zeros:NNNNNNNN #5 000
10650 }
10651 \cs_new:cpn { fp_to_tl_large_4:w } #1 . #2#3#4#5#6 \q_stop
10652 {
10653   #1#2#3#4#5
10654   \fp_to_tl_large_zeros:NNNNNNNN #6 0000
10655 }
10656 \cs_new:cpn { fp_to_tl_large_5:w } #1 . #2#3#4#5#6#7 \q_stop
10657 {
10658   #1#2#3#4#5#6
10659   \fp_to_tl_large_zeros:NNNNNNNN #7 00000

```

```

10660 }
10661 \cs_new:cpn { fp_to_tl_large_6:w } #1 . #2#3#4#5#6#7#8 \q_stop
10662 {
10663   #1#2#3#4#5#6#7
10664   \fp_to_tl_large_zeros:NNNNNNNN #8 000000
10665 }
10666 \cs_new:cpn { fp_to_tl_large_7:w } #1 . #2#3#4#5#6#7#8#9 \q_stop
10667 {
10668   #1#2#3#4#5#6#7#8
10669   \fp_to_tl_large_zeros:NNNNNNNN #9 0000000
10670 }
10671 \cs_new:cpn { fp_to_tl_large_8:w } #1 .
10672 {
10673   #1
10674   \use:c { fp_to_tl_large_8_aux:w }
10675 }
10676 \cs_new:cpn { fp_to_tl_large_8_aux:w } #1#2#3#4#5#6#7#8#9 \q_stop
10677 {
10678   #1#2#3#4#5#6#7#8
10679   \fp_to_tl_large_zeros:NNNNNNNN #9 00000000
10680 }
10681 \cs_new:cpn { fp_to_tl_large_9:w } #1 . #2 \q_stop {#1#2}

```

Dealing with small numbers is a bit more complex as there has to be rounding. This makes life rather awkward, as there need to be a series of tests and calculations, as things cannot be stored in an expandable system.

```

10682 \cs_new:Npn \fp_to_tl_small:w #1 e #2 \q_stop
10683 {
10684   \if_int_compare:w #2 = \c_minus_one
10685     \exp_after:wN \fp_to_tl_small_one:w
10686   \else:
10687     \if_int_compare:w #2 = -\c_two
10688       \exp_after:wN \exp_after:wN \exp_after:wN \fp_to_tl_small_two:w
10689     \else:
10690       \exp_after:wN \exp_after:wN \exp_after:wN \fp_to_tl_small_aux:w
10691     \fi:
10692   \fi:
10693   #1 e #2 \q_stop
10694 }
10695 \cs_new:Npn \fp_to_tl_small_one:w #1 . #2 e #3 \q_stop
10696 {
10697   \if_int_compare:w \fp_use_ix:NNNNNNNN #2 > \c_four
10698     \if_int_compare:w
10699       \int_eval:w #1 \fp_use_i_to_iix:NNNNNNNN #2 + 1
10700       < \c_one_thousand_million
10701     0.
10702     \exp_after:wN \fp_to_tl_small_zeros:NNNNNNNN
10703     \int_value:w \int_eval:w
10704       #1 \fp_use_i_to_iix:NNNNNNNN #2 + 1
10705     \int_eval_end:

```

```

10706         \else:
10707             1
10708         \fi:
10709     \else:
10710         0. #1
10711         \fp_to_tl_small_zeros:NNNNNNNN #2
10712     \fi:
10713 }
10714 \cs_new:Npn \fp_to_tl_small_two:w #1 . #2 e #3 \q_stop
10715 {
10716     \if_int_compare:w \fp_use_iix_ix:NNNNNNNN #2 > \c_forty_four
10717     \if_int_compare:w
10718         \int_eval:w #1 \fp_use_i_to_vii:NNNNNNNN #2 0 + \c_ten
10719         < \c_one_thousand_million
10720         0.0
10721         \exp_after:wN \fp_to_tl_small_zeros:NNNNNNNN
10722         \int_value:w \int_eval:w
10723         #1 \fp_use_i_to_vii:NNNNNNNN #2 0 + \c_ten
10724         \int_eval_end:
10725     \else:
10726         0.1
10727     \fi:
10728 \else:
10729     0.0
10730     #1
10731     \fp_to_tl_small_zeros:NNNNNNNN #2
10732 \fi:
10733 }
10734 \cs_new:Npn \fp_to_tl_small_aux:w #1 . #2 e #3 \q_stop
10735 {
10736     #1
10737     \fp_to_tl_large_zeros:NNNNNNNN #2
10738     e #3
10739 }

```

Rather than a complex recursion, the tests for finding trailing zeros are written out long-hand. The difference between the two is only the need for a decimal marker.

```

10740 \cs_new:Npn \fp_to_tl_large_zeros:NNNNNNNN #1#2#3#4#5#6#7#8#9
10741 {
10742     \if_int_compare:w #9 = \c_zero
10743     \if_int_compare:w #8 = \c_zero
10744     \if_int_compare:w #7 = \c_zero
10745     \if_int_compare:w #6 = \c_zero
10746     \if_int_compare:w #5 = \c_zero
10747     \if_int_compare:w #4 = \c_zero
10748     \if_int_compare:w #3 = \c_zero
10749     \if_int_compare:w #2 = \c_zero
10750     \if_int_compare:w #1 = \c_zero
10751     \else:
10752         . #1

```

```

10753         \fi:
10754         \else:
10755             . #1#2
10756         \fi:
10757         \else:
10758             . #1#2#3
10759         \fi:
10760         \else:
10761             . #1#2#3#4
10762         \fi:
10763         \else:
10764             . #1#2#3#4#5
10765         \fi:
10766         \else:
10767             . #1#2#3#4#5#6
10768         \fi:
10769         \else:
10770             . #1#2#3#4#5#6#7
10771         \fi:
10772         \else:
10773             . #1#2#3#4#5#6#7#8
10774         \fi:
10775         \else:
10776             . #1#2#3#4#5#6#7#8#9
10777         \fi:
10778     }
10779 \cs_new:Npn \fp_to_tl_small_zeros:NNNNNNNNN #1#2#3#4#5#6#7#8#9
10780 {
10781     \if_int_compare:w #9 = \c_zero
10782     \if_int_compare:w #8 = \c_zero
10783     \if_int_compare:w #7 = \c_zero
10784     \if_int_compare:w #6 = \c_zero
10785     \if_int_compare:w #5 = \c_zero
10786     \if_int_compare:w #4 = \c_zero
10787     \if_int_compare:w #3 = \c_zero
10788     \if_int_compare:w #2 = \c_zero
10789     \if_int_compare:w #1 = \c_zero
10790     \else:
10791         #1
10792     \fi:
10793     \else:
10794         #1#2
10795     \fi:
10796     \else:
10797         #1#2#3
10798     \fi:
10799     \else:
10800         #1#2#3#4
10801     \fi:
10802     \else:

```

```

10803         #1#2#3#4#5
10804         \fi:
10805         \else:
10806             #1#2#3#4#5#6
10807         \fi:
10808         \else:
10809             #1#2#3#4#5#6#7
10810         \fi:
10811         \else:
10812             #1#2#3#4#5#6#7#8
10813         \fi:
10814         \else:
10815             #1#2#3#4#5#6#7#8#9
10816         \fi:
10817     }

```

Some quick “return a few” functions.

```

10818 \cs_new:Npn \fp_use_iix_ix:NNNNNNNN #1#2#3#4#5#6#7#8#9 {#8#9}
10819 \cs_new:Npn \fp_use_ix:NNNNNNNN #1#2#3#4#5#6#7#8#9 {#9}
10820 \cs_new:Npn \fp_use_i_to_vii:NNNNNNNN #1#2#3#4#5#6#7#8#9
10821     {#1#2#3#4#5#6#7}
10822 \cs_new:Npn \fp_use_i_to_iix:NNNNNNNN #1#2#3#4#5#6#7#8#9
10823     {#1#2#3#4#5#6#7#8}

```

(End definition for \fp_to_tl:N and \fp_to_tl:c. These functions are documented on page ??.)

203.7 Rounding numbers

The results may well need to be rounded. A couple of related functions to do this for a stored value.

<pre> \fp_round_figures:Nn \fp_round_figures:cn \fp_ground_figures:Nn \fp_ground_figures:cn \fp_round_figures_aux:NNn </pre>	<p>Rounding to figures needs only an adjustment to the target by one (as the target is in decimal places).</p> <pre> 10824 \cs_new_protected_nopar:Npn \fp_round_figures:Nn 10825 { \fp_round_figures_aux:NNn \tl_set:Nn } 10826 \cs_generate_variant:Nn \fp_round_figures:Nn { c } 10827 \cs_new_protected_nopar:Npn \fp_ground_figures:Nn 10828 { \fp_round_figures_aux:NNn \tl_gset:Nn } 10829 \cs_generate_variant:Nn \fp_ground_figures:Nn { c } 10830 \cs_new_protected:Npn \fp_round_figures_aux:NNn #1#2#3 10831 { 10832 \group_begin: 10833 \fp_read:N #2 10834 \int_set:Nn \l_fp_round_target_int { #3 - 1 } 10835 \if_int_compare:w \l_fp_round_target_int < \c_ten 10836 \exp_after:wN \fp_round: 10837 \fi: 10838 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million 10839 \cs_set_protected_nopar:Npx \fp_tmp:w 10840 { </pre>
--	---

```

10841         \group_end:
10842         #1 \exp_not:N #2
10843         {
10844             \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10845             -
10846             \else:
10847             +
10848             \fi:
10849             \int_use:N \l_fp_input_a_integer_int
10850             .
10851             \exp_after:wN \use_none:n
10852             \int_use:N \l_fp_input_a_decimal_int
10853             e
10854             \int_use:N \l_fp_input_a_exponent_int
10855         }
10856     }
10857     \fp_tmp:w
10858 }

```

(End definition for `\fp_round_figures:Nn` and `\fp_round_figures:cn`. These functions are documented on page ??.)

`\fp_round_places:Nn` Rounding to places needs an adjustment for the exponent value, which will mean that everything should be correct.

```

\fp_round_places:cn
\fp_ground_places:Nn
\fp_ground_places:cn
\fp_round_places_aux:NNn
10859 \cs_new_protected_nopar:Npn \fp_round_places:Nn
10860 { \fp_round_places_aux:NNn \tl_set:Nn }
10861 \cs_generate_variant:Nn \fp_round_places:Nn { c }
10862 \cs_new_protected_nopar:Npn \fp_ground_places:Nn
10863 { \fp_round_places_aux:NNn \tl_gset:Nn }
10864 \cs_generate_variant:Nn \fp_ground_places:Nn { c }
10865 \cs_new_protected:Npn \fp_round_places_aux:NNn #1#2#3
10866 {
10867     \group_begin:
10868     \fp_read:N #2
10869     \int_set:Nn \l_fp_round_target_int
10870     { #3 + \l_fp_input_a_exponent_int }
10871     \if_int_compare:w \l_fp_round_target_int < \c_ten
10872     \exp_after:wN \fp_round:
10873     \fi:
10874     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10875     \cs_set_protected_nopar:Npx \fp_tmp:w
10876     {
10877         \group_end:
10878         #1 \exp_not:N #2
10879         {
10880             \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10881             -
10882             \else:
10883             +
10884             \fi:
10885             \int_use:N \l_fp_input_a_integer_int

```



```

10886      .
10887      \exp_after:wN \use_none:n
10888      \int_use:N \l_fp_input_a_decimal_int
10889      e
10890      \int_use:N \l_fp_input_a_exponent_int
10891    }
10892  }
10893  \fp_tmp:w
10894 }

```

(End definition for `\fp_round_places:Nn` and `\fp_round_places:cn`. These functions are documented on page ??.)

`\fp_round:` The rounding approach is the same for decimal places and significant figures. There are always nine decimal digits to round, so the code can be written to account for this. The basic logic is simply to find the rounding, track any carry digit and move along. At the end of the loop there is a possible shuffle if the integer part has become 10.

```

10895 \cs_new_protected_nopar:Npn \fp_round:
10896 {
10897   \bool_set_false:N \l_fp_round_carry_bool
10898   \l_fp_round_position_int \c_eight
10899   \tl_clear:N \l_fp_round_decimal_tl
10900   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10901   \exp_after:wN \use_i:nn \exp_after:wN
10902   \fp_round_aux:NNNNNNNNN \int_use:N \l_fp_input_a_decimal_int
10903 }
10904 \cs_new_protected:Npn \fp_round_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
10905 {
10906   \fp_round_loop:N #9#8#7#6#5#4#3#2#1
10907   \bool_if:NT \l_fp_round_carry_bool
10908   { \tex_advance:D \l_fp_input_a_integer_int \c_one }
10909   \l_fp_input_a_decimal_int \l_fp_round_decimal_tl \scan_stop:
10910   \if_int_compare:w \l_fp_input_a_integer_int < \c_ten
10911   \else:
10912     \l_fp_input_a_integer_int \c_one
10913     \tex_divide:D \l_fp_input_a_decimal_int \c_ten
10914     \tex_advance:D \l_fp_input_a_exponent_int \c_one
10915   \fi:
10916 }
10917 \cs_new_protected:Npn \fp_round_loop:N #1
10918 {
10919   \if_int_compare:w \l_fp_round_position_int < \l_fp_round_target_int
10920   \bool_if:NTF \l_fp_round_carry_bool
10921   { \l_fp_tmp_int \int_eval:w #1 + \c_one \scan_stop: }
10922   { \l_fp_tmp_int \int_eval:w #1 \scan_stop: }
10923   \if_int_compare:w \l_fp_tmp_int = \c_ten
10924   \l_fp_tmp_int \c_zero
10925   \else:
10926     \bool_set_false:N \l_fp_round_carry_bool
10927   \fi:

```

```

10928     \tl_set:Nx \l_fp_round_decimal_tl
10929     { \int_use:N \l_fp_tmp_int \l_fp_round_decimal_tl }
10930   \else:
10931     \tl_set:Nx \l_fp_round_decimal_tl { 0 \l_fp_round_decimal_tl }
10932     \if_int_compare:w \l_fp_round_position_int = \l_fp_round_target_int
10933       \if_int_compare:w #1 > \c_four
10934         \bool_set_true:N \l_fp_round_carry_bool
10935       \fi:
10936     \fi:
10937   \fi:
10938   \tex_advance:D \l_fp_round_position_int \c_minus_one
10939   \if_int_compare:w \l_fp_round_position_int > \c_minus_one
10940     \exp_after:wN \fp_round_loop:N
10941   \fi:
10942 }

```

(End definition for \fp_round:. This function is documented on page ??.)

203.8 Unary functions

\fp_abs:N Setting the absolute value is easy: read the value, ignore the sign, return the result.

```

\fp_abs:c 10943 \cs_new_protected_nopar:Npn \fp_abs:N { \fp_abs_aux:NN \tl_set:Nn }
\fp_gabs:N 10944 \cs_new_protected_nopar:Npn \fp_gabs:N { \fp_abs_aux:NN \tl_gset:Nn }
\fp_gabs:c 10945 \cs_generate_variant:Nn \fp_abs:N { c }
\fp_abs_aux:NN 10946 \cs_generate_variant:Nn \fp_gabs:N { c }
10947 \cs_new_protected:Npn \fp_abs_aux:NN #1#2
10948 {
10949   \group_begin:
10950   \fp_read:N #2
10951   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10952   \cs_set_protected_nopar:Npx \fp_tmp:w
10953   {
10954     \group_end:
10955     #1 \exp_not:N #2
10956     {
10957       +
10958       \int_use:N \l_fp_input_a_integer_int
10959       .
10960       \exp_after:wN \use_none:n
10961       \int_use:N \l_fp_input_a_decimal_int
10962       e
10963       \int_use:N \l_fp_input_a_exponent_int
10964     }
10965   }
10966   \fp_tmp:w
10967 }

```

(End definition for \fp_abs:N and \fp_abs:c. These functions are documented on page ??.)

\fp_neg:N Just a bit more complex: read the input, reverse the sign and output the result.

```

\fp_neg:c 10968 \cs_new_protected_nopar:Npn \fp_neg:N { \fp_neg_aux:NN \tl_set:Nn }
\fp_gneg:N
\fp_gneg:c
\fp_neg:NN

```

```

10969 \cs_new_protected_nopar:Npn \fp_gneg:N { \fp_neg_aux:NN \tl_gset:Nn }
10970 \cs_generate_variant:Nn \fp_neg:N { c }
10971 \cs_generate_variant:Nn \fp_gneg:N { c }
10972 \cs_new_protected:Npn \fp_neg_aux:NN #1#2
10973 {
10974   \group_begin:
10975   \fp_read:N #2
10976   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10977   \tl_set:Nx \l_fp_tmp_tl
10978   {
10979     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10980     +
10981     \else:
10982     -
10983     \fi:
10984     \int_use:N \l_fp_input_a_integer_int
10985     .
10986     \exp_after:wN \use_none:n
10987     \int_use:N \l_fp_input_a_decimal_int
10988     e
10989     \int_use:N \l_fp_input_a_exponent_int
10990   }
10991   \exp_after:wN \group_end: \exp_after:wN
10992   #1 \exp_after:wN #2 \exp_after:wN { \l_fp_tmp_tl }
10993 }

```

(End definition for `\fp_neg:N` and `\fp_neg:c`. These functions are documented on page ??.)

203.9 Basic arithmetic

`\fp_add:Nn` The various addition functions are simply different ways to call the single master function
`\fp_add:cn` below. This pattern is repeated for the other arithmetic functions.
`\fp_gadd:Nn` 10994 `\cs_new_protected_nopar:Npn \fp_add:Nn { \fp_add_aux:NNn \tl_set:Nn }`
`\fp_gadd:cn` 10995 `\cs_new_protected_nopar:Npn \fp_gadd:Nn { \fp_add_aux:NNn \tl_gset:Nn }`
`\fp_add_aux:NNn` 10996 `\cs_generate_variant:Nn \fp_add:Nn { c }`
`\fp_add_core:` 10997 `\cs_generate_variant:Nn \fp_gadd:Nn { c }`
`\fp_add_sum:` Addition takes place using one of two paths. If the signs of the two parts are the same,
`\fp_add_difference:` they are simply combined. On the other hand, if the signs are different the calculation
finds this difference.

```

10998 \cs_new_protected:Npn \fp_add_aux:NNn #1#2#3
10999 {
11000   \group_begin:
11001   \fp_read:N #2
11002   \fp_split:Nn b {#3}
11003   \fp_standardise:NNNN
11004   \l_fp_input_b_sign_int
11005   \l_fp_input_b_integer_int
11006   \l_fp_input_b_decimal_int
11007   \l_fp_input_b_exponent_int

```

```

11008     \fp_add_core:
11009     \fp_tmp:w #1#2
11010 }
11011 \cs_new_protected_nopar:Npn \fp_add_core:
11012 {
11013     \fp_level_input_exponents:
11014     \if_int_compare:w
11015         \int_eval:w
11016             \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
11017             > \c_zero
11018     \exp_after:wN \fp_add_sum:
11019 \else:
11020     \exp_after:wN \fp_add_difference:
11021 \fi:
11022 \l_fp_output_exponent_int \l_fp_input_a_exponent_int
11023 \fp_standardise:NNNN
11024     \l_fp_output_sign_int
11025     \l_fp_output_integer_int
11026     \l_fp_output_decimal_int
11027     \l_fp_output_exponent_int
11028 \cs_set_protected:Npx \fp_tmp:w ##1##2
11029 {
11030     \group_end:
11031     ##1 ##2
11032     {
11033         \if_int_compare:w \l_fp_output_sign_int < \c_zero
11034             -
11035         \else:
11036             +
11037         \fi:
11038         \int_use:N \l_fp_output_integer_int
11039         .
11040         \exp_after:wN \use_none:n
11041         \int_value:w \int_eval:w
11042             \l_fp_output_decimal_int + \c_one_thousand_million
11043         e
11044         \int_use:N \l_fp_output_exponent_int
11045     }
11046 }
11047 }

```

Finding the sum of two numbers is trivially easy.

```

11048 \cs_new_protected_nopar:Npn \fp_add_sum:
11049 {
11050     \l_fp_output_sign_int \l_fp_input_a_sign_int
11051     \l_fp_output_integer_int
11052     \int_eval:w
11053         \l_fp_input_a_integer_int + \l_fp_input_b_integer_int
11054     \scan_stop:
11055     \l_fp_output_decimal_int

```

```

11056     \int_eval:w
11057     \l_fp_input_a_decimal_int + \l_fp_input_b_decimal_int
11058     \scan_stop:
11059     \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
11060     \else:
11061     \tex_advance:D \l_fp_output_integer_int \c_one
11062     \tex_advance:D \l_fp_output_decimal_int -\c_one_thousand_million
11063     \fi:
11064 }

```

When the signs of the two parts of the input are different, the absolute difference is worked out first. There is then a calculation to see which way around everything has worked out, so that the final sign is correct. The difference might also give a zero result with a negative sign, which is reversed as zero is regarded as positive.

```

11065 \cs_new_protected_nopar:Npn \fp_add_difference:
11066 {
11067     \l_fp_output_integer_int
11068     \int_eval:w
11069     \l_fp_input_a_integer_int - \l_fp_input_b_integer_int
11070     \scan_stop:
11071     \l_fp_output_decimal_int
11072     \int_eval:w
11073     \l_fp_input_a_decimal_int - \l_fp_input_b_decimal_int
11074     \scan_stop:
11075     \if_int_compare:w \l_fp_output_decimal_int < \c_zero
11076     \tex_advance:D \l_fp_output_integer_int \c_minus_one
11077     \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
11078     \fi:
11079     \if_int_compare:w \l_fp_output_integer_int < \c_zero
11080     \l_fp_output_sign_int \l_fp_input_b_sign_int
11081     \if_int_compare:w \l_fp_output_decimal_int = \c_zero
11082     \l_fp_output_integer_int -\l_fp_output_integer_int
11083     \else:
11084     \l_fp_output_decimal_int
11085     \int_eval:w
11086     \c_one_thousand_million - \l_fp_output_decimal_int
11087     \scan_stop:
11088     \l_fp_output_integer_int
11089     \int_eval:w
11090     - \l_fp_output_integer_int - \c_one
11091     \scan_stop:
11092     \fi:
11093     \else:
11094     \l_fp_output_sign_int \l_fp_input_a_sign_int
11095     \fi:
11096 }

```

(End definition for \fp_add:Nn and \fp_add:cn. These functions are documented on page ??.)

`\fp_sub:Nn` Subtraction is essentially the same as addition, but with the sign of the second component
`\fp_sub:cn` reversed. Thus the core of the two function groups is the same, with just a little set up
`\fp_gsub:Nn`
`\fp_gsub:cn`
`\fp_sub_aux:NNn`

here.

```

11097 \cs_new_protected_nopar:Npn \fp_sub:Nn { \fp_sub_aux:NNn \tl_set:Nn }
11098 \cs_new_protected_nopar:Npn \fp_gsub:Nn { \fp_sub_aux:NNn \tl_gset:Nn }
11099 \cs_generate_variant:Nn \fp_sub:Nn { c }
11100 \cs_generate_variant:Nn \fp_gsub:Nn { c }
11101 \cs_new_protected:Npn \fp_sub_aux:NNn #1#2#3
11102 {
11103   \group_begin:
11104     \fp_read:N #2
11105     \fp_split:Nn b {#3}
11106     \fp_standardise:NNNN
11107     \l_fp_input_b_sign_int
11108     \l_fp_input_b_integer_int
11109     \l_fp_input_b_decimal_int
11110     \l_fp_input_b_exponent_int
11111     \tex_multiply:D \l_fp_input_b_sign_int \c_minus_one
11112     \fp_add_core:
11113     \fp_tmp:w #1#2
11114   }

```

(End definition for \fp_sub:Nn and \fp_sub:cn. These functions are documented on page ??.)

`\fp_mul:Nn` The pattern is much the same for multiplication.

```

\fp_mul:cn 11115 \cs_new_protected_nopar:Npn \fp_mul:Nn { \fp_mul_aux:NNn \tl_set:Nn }
\fp_gmul:Nn 11116 \cs_new_protected_nopar:Npn \fp_gmul:Nn { \fp_mul_aux:NNn \tl_gset:Nn }
\fp_gmul:cn 11117 \cs_generate_variant:Nn \fp_mul:Nn { c }
\fp_mul_aux:NNn 11118 \cs_generate_variant:Nn \fp_gmul:Nn { c }

```

`\fp_mul_internal:` The approach to multiplication is as follows. First, the two numbers are split into blocks
`\fp_mul_split:NNNN` of three digits. These are then multiplied together to find products for each group of three
`\fp_mul_split:w` output digits. This is all written out in full for speed reasons. Between each block of three
`\fp_mul_end_level:` digits in the output, there is a carry step. The very lowest digits are not calculated, while

```

\fp_mul_end_level:NNNNNNNN 11119 \cs_new_protected:Npn \fp_mul_aux:NNn #1#2#3
11120 {
11121   \group_begin:
11122     \fp_read:N #2
11123     \fp_split:Nn b {#3}
11124     \fp_standardise:NNNN
11125     \l_fp_input_b_sign_int
11126     \l_fp_input_b_integer_int
11127     \l_fp_input_b_decimal_int
11128     \l_fp_input_b_exponent_int
11129     \fp_mul_internal:
11130     \l_fp_output_exponent_int
11131     \int_eval:w
11132       \l_fp_input_a_exponent_int + \l_fp_input_b_exponent_int
11133     \scan_stop:
11134     \fp_standardise:NNNN
11135     \l_fp_output_sign_int
11136     \l_fp_output_integer_int

```

```

11137     \l_fp_output_decimal_int
11138     \l_fp_output_exponent_int
11139 \cs_set_protected_nopar:Npx \fp_tmp:w
11140 {
11141     \group_end:
11142     #1 \exp_not:N #2
11143     {
11144         \if_int_compare:w
11145             \int_eval:w
11146             \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
11147             < \c_zero
11148         \if_int_compare:w
11149             \int_eval:w
11150             \l_fp_output_integer_int + \l_fp_output_decimal_int
11151             = \c_zero
11152         +
11153         \else:
11154             -
11155         \fi:
11156         \else:
11157             +
11158         \fi:
11159         \int_use:N \l_fp_output_integer_int
11160         .
11161         \exp_after:wN \use_none:n
11162         \int_value:w \int_eval:w
11163         \l_fp_output_decimal_int + \c_one_thousand_million
11164         e
11165         \int_use:N \l_fp_output_exponent_int
11166     }
11167 }
11168 \fp_tmp:w
11169 }

```

Done separately so that the internal use is a bit easier.

```

11170 \cs_new_protected_nopar:Npn \fp_mul_internal:
11171 {
11172     \fp_mul_split:NNNN \l_fp_input_a_decimal_int
11173     \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11174     \fp_mul_split:NNNN \l_fp_input_b_decimal_int
11175     \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11176     \l_fp_mul_output_int \c_zero
11177     \tl_clear:N \l_fp_mul_output_tl
11178     \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11179     \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11180     \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11181     \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11182     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_iii_int
11183     \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11184     \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int

```

```

11185 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_input_b_integer_int
11186 \fp_mul_end_level:
11187 \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_ii_int
11188 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11189 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_input_b_integer_int
11190 \fp_mul_end_level:
11191 \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_i_int
11192 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_input_b_integer_int
11193 \fp_mul_end_level:
11194 \l_fp_output_decimal_int 0 \l_fp_mul_output_tl \scan_stop:
11195 \tl_clear:N \l_fp_mul_output_tl
11196 \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_input_b_integer_int
11197 \fp_mul_end_level:
11198 \l_fp_output_integer_int 0 \l_fp_mul_output_tl \scan_stop:
11199 }

```

The split works by making a 10 digit number, from which the first digit can then be dropped using a delimited argument. The groups of three digits are then assigned to the various parts of the input: notice that ##9 contains the last two digits of the smallest part of the input.

```

11200 \cs_new_protected:Npn \fp_mul_split:NNNN #1#2#3#4
11201 {
11202   \tex_advance:D #1 \c_one_thousand_million
11203   \cs_set_protected:Npn \fp_mul_split_aux:w
11204     ##1##2##3##4##5##6##7##8##9 \q_stop {
11205     #2 ##2##3##4 \scan_stop:
11206     #3 ##5##6##7 \scan_stop:
11207     #4 ##8##9 \scan_stop:
11208   }
11209   \exp_after:wN \fp_mul_split_aux:w \int_use:N #1 \q_stop
11210   \tex_advance:D #1 -\c_one_thousand_million
11211 }
11212 \cs_new_protected:Npn \fp_mul_product:NN #1#2
11213 {
11214   \l_fp_mul_output_int
11215   \int_eval:w \l_fp_mul_output_int + #1 * #2 \scan_stop:
11216 }

```

At the end of each output group of three, there is a transfer of information so that there is no danger of an overflow. This is done by expansion to keep the number of calculations down.

```

11217 \cs_new_protected_nopar:Npn \fp_mul_end_level:
11218 {
11219   \tex_advance:D \l_fp_mul_output_int \c_one_thousand_million
11220   \exp_after:wN \use_i:nn \exp_after:wN
11221   \fp_mul_end_level:NNNNNNNN \int_use:N \l_fp_mul_output_int
11222 }
11223 \cs_new_protected:Npn \fp_mul_end_level:NNNNNNNN #1#2#3#4#5#6#7#8#9
11224 {
11225   \tl_set:Nx \l_fp_mul_output_tl { #7#8#9 \l_fp_mul_output_tl }

```



```

11226     \l_fp_mul_output_int #1#2#3#4#5#6 \scan_stop:
11227 }

```

(End definition for \fp_mul:Nn and \fp_mul:cn. These functions are documented on page ??.)

```

\fp_div:Nn The pattern is much the same for multiplication.
\fp_div:cn
\fp_gdiv:Nn 11228 \cs_new_protected_nopar:Npn \fp_div:Nn { \fp_div_aux:NNn \tl_set:Nn }
\fp_gdiv:cn 11229 \cs_new_protected_nopar:Npn \fp_gdiv:Nn { \fp_div_aux:NNn \tl_gset:Nn }
\fp_gdiv:cn 11230 \cs_generate_variant:Nn \fp_div:Nn { c }
\fp_div_aux:NNn 11231 \cs_generate_variant:Nn \fp_gdiv:Nn { c }
\fp_div_internal: Division proper starts with a couple of tests. If the denominator is zero then a error is
\fp_div_loop: issued. On the other hand, if the numerator is zero then the result must be 0.0 and can
\fp_div_divide: be given with no further work.
\fp_div_divide_aux:
\fp_div_store: 11232 \cs_new_protected:Npn \fp_div_aux:NNn #1#2#3
\fp_div_store_integer: 11233 {
\fp_div_store_decimal: 11234 \group_begin:
11235 \fp_read:N #2
11236 \fp_split:Nn b {#3}
11237 \fp_standardise:NNNN
11238 \l_fp_input_b_sign_int
11239 \l_fp_input_b_integer_int
11240 \l_fp_input_b_decimal_int
11241 \l_fp_input_b_exponent_int
11242 \if_int_compare:w
11243 \int_eval:w
11244 \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
11245 = \c_zero
11246 \cs_set_protected:Npx \fp_tmp:w ##1##2
11247 {
11248 \group_end:
11249 #1 \exp_not:N #2 { \c_undefined_fp }
11250 }
11251 \else:
11252 \if_int_compare:w
11253 \int_eval:w
11254 \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
11255 = \c_zero
11256 \cs_set_protected:Npx \fp_tmp:w ##1##2
11257 {
11258 \group_end:
11259 #1 \exp_not:N #2 { \c_zero_fp }
11260 }
11261 \else:
11262 \exp_after:wN \exp_after:wN \exp_after:wN \fp_div_internal:
11263 \fi:
11264 \fi:
11265 \fp_tmp:w #1#2
11266 }

```

The main division algorithm works by finding how many times **b** can be removed from **a**, storing the result and doing the subtraction. Input **a** is then multiplied by 10, and the process is repeated. The looping ends either when there is nothing left of **a** (*i.e.* an exact result) or when the code reaches the ninth decimal place. Most of the process takes place in the loop function below.

```

11267 \cs_new_protected_nopar:Npn \fp_div_internal: {
11268   \l_fp_output_integer_int \c_zero
11269   \l_fp_output_decimal_int \c_zero
11270   \cs_set_eq:NN \fp_div_store: \fp_div_store_integer:
11271   \l_fp_div_offset_int \c_one_hundred_million
11272   \fp_div_loop:
11273   \l_fp_output_exponent_int
11274   \int_eval:w
11275     \l_fp_input_a_exponent_int - \l_fp_input_b_exponent_int
11276   \scan_stop:
11277   \fp_standardise:NNNN
11278   \l_fp_output_sign_int
11279   \l_fp_output_integer_int
11280   \l_fp_output_decimal_int
11281   \l_fp_output_exponent_int
11282   \cs_set_protected:Npx \fp_tmp:w ##1##2
11283   {
11284     \group_end:
11285     ##1 ##2
11286     {
11287       \if_int_compare:w
11288         \int_eval:w
11289           \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
11290         < \c_zero
11291       \if_int_compare:w
11292         \int_eval:w
11293           \l_fp_output_integer_int + \l_fp_output_decimal_int
11294         = \c_zero
11295       +
11296       \else:
11297         -
11298       \fi:
11299       \else:
11300         +
11301       \fi:
11302       \int_use:N \l_fp_output_integer_int
11303       .
11304       \exp_after:wN \use_none:n
11305       \int_value:w \int_eval:w
11306         \l_fp_output_decimal_int + \c_one_thousand_million
11307       \int_eval_end:
11308     e
11309     \int_use:N \l_fp_output_exponent_int
11310   }

```

```

11311     }
11312 }

```

The main loop implements the approach described above. The storing function is done as a function so that the integer and decimal parts can be done separately but rapidly.

```

11313 \cs_new_protected_nopar:Npn \fp_div_loop:
11314 {
11315     \l_fp_count_int \c_zero
11316     \fp_div_divide:
11317     \fp_div_store:
11318     \tex_multiply:D \l_fp_input_a_integer_int \c_ten
11319     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11320     \exp_after:wN \fp_div_loop_step:w
11321     \int_use:N \l_fp_input_a_decimal_int \q_stop
11322     \if_int_compare:w
11323     \int_eval:w \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
11324     > \c_zero
11325     \if_int_compare:w \l_fp_div_offset_int > \c_zero
11326     \exp_after:wN \exp_after:wN \exp_after:wN
11327     \fp_div_loop:
11328     \fi:
11329 \fi:
11330 }

```

Checking to see if the numerator can be divided needs quite an involved check. Either the integer part has to be bigger for the numerator or, if it is not smaller then the decimal part of the numerator must not be smaller than that of the denominator. Once the test is right the rest is much as elsewhere.

```

11331 \cs_new_protected_nopar:Npn \fp_div_divide:
11332 {
11333     \if_int_compare:w \l_fp_input_a_integer_int > \l_fp_input_b_integer_int
11334     \exp_after:wN \fp_div_divide_aux:
11335     \else:
11336     \if_int_compare:w \l_fp_input_a_integer_int < \l_fp_input_b_integer_int
11337     \else:
11338     \if_int_compare:w
11339     \l_fp_input_a_decimal_int < \l_fp_input_b_decimal_int
11340     \else:
11341     \exp_after:wN \exp_after:wN \exp_after:wN
11342     \exp_after:wN \exp_after:wN \exp_after:wN
11343     \exp_after:wN \fp_div_divide_aux:
11344     \fi:
11345     \fi:
11346     \fi:
11347 }
11348 \cs_new_protected_nopar:Npn \fp_div_divide_aux:
11349 {
11350     \tex_advance:D \l_fp_count_int \c_one
11351     \tex_advance:D \l_fp_input_a_integer_int -\l_fp_input_b_integer_int
11352     \tex_advance:D \l_fp_input_a_decimal_int -\l_fp_input_b_decimal_int

```

```

11353 \if_int_compare:w \l_fp_input_a_decimal_int < \c_zero
11354 \tex_advance:D \l_fp_input_a_integer_int \c_minus_one
11355 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11356 \fi:
11357 \fp_div_divide:
11358 }

```

Storing the number of each division is done differently for the integer and decimal. The integer is easy and a one-off, while the decimal also needs to account for the position of the digit to store.

```

11359 \cs_new_protected_nopar:Npn \fp_div_store: { }
11360 \cs_new_protected_nopar:Npn \fp_div_store_integer:
11361 {
11362   \l_fp_output_integer_int \l_fp_count_int
11363   \cs_set_eq:NN \fp_div_store: \fp_div_store_decimal:
11364 }
11365 \cs_new_protected_nopar:Npn \fp_div_store_decimal:
11366 {
11367   \l_fp_output_decimal_int
11368   \int_eval:w
11369     \l_fp_output_decimal_int +
11370     \l_fp_count_int * \l_fp_div_offset_int
11371   \int_eval_end:
11372   \tex_divide:D \l_fp_div_offset_int \c_ten
11373 }
11374 \cs_new_protected:Npn \fp_div_loop_step:w #1#2#3#4#5#6#7#8#9 \q_stop
11375 {
11376   \l_fp_input_a_integer_int
11377   \int_eval:w #2 + \l_fp_input_a_integer_int \int_eval_end:
11378   \l_fp_input_a_decimal_int #3#4#5#6#7#8#9 0 \scan_stop:
11379 }

```

(End definition for \fp_div:Nn and \fp_div:cn. These functions are documented on page ??.)

203.10 Arithmetic for internal use

For the more complex functions, it is only possible to deliver reliable 10 digit accuracy if the internal calculations are carried out to a higher degree of precision. This is done using a second set of functions so that the ‘user’ versions are not slowed down. These versions are also focussed on the needs of internal calculations. No error checking, sign checking or exponent levelling is done. For addition and subtraction, the arguments are:

- Integer part of input a.
- Decimal part of input a.
- Additional decimal part of input a.
- Integer part of input b.
- Decimal part of input b.

- Additional decimal part of input b.
- Integer part of output.
- Decimal part of output.
- Additional decimal part of output.

The situation for multiplication and division is a little different as they only deal with the decimal part.

`\fp_add:NNNNNNNNN` The internal sum is always exactly that: it is always a sum and there is no sign check.

```

11380 \cs_new_protected:Npn \fp_add:NNNNNNNNN #1#2#3#4#5#6#7#8#9
11381 {
11382   #7 \int_eval:w #1 + #4 \int_eval_end:
11383   #8 \int_eval:w #2 + #5 \int_eval_end:
11384   #9 \int_eval:w #3 + #6 \int_eval_end:
11385   \if_int_compare:w #9 < \c_one_thousand_million
11386   \else:
11387     \tex_advance:D #8 \c_one
11388     \tex_advance:D #9 -\c_one_thousand_million
11389   \fi:
11390   \if_int_compare:w #8 < \c_one_thousand_million
11391   \else:
11392     \tex_advance:D #7 \c_one
11393     \tex_advance:D #8 -\c_one_thousand_million
11394   \fi:
11395 }

```

(End definition for \fp_add:NNNNNNNNN. This function is documented on page ??.)

`\fp_sub:NNNNNNNNN` Internal subtraction is needed only when the first number is bigger than the second, so there is no need to worry about the sign. This is a good job as there are no arguments left. The flipping flag is used in the rare case where a sign change is possible.

```

11396 \cs_new_protected:Npn \fp_sub:NNNNNNNNN #1#2#3#4#5#6#7#8#9
11397 {
11398   #7 \int_eval:w #1 - #4 \int_eval_end:
11399   #8 \int_eval:w #2 - #5 \int_eval_end:
11400   #9 \int_eval:w #3 - #6 \int_eval_end:
11401   \if_int_compare:w #9 < \c_zero
11402     \tex_advance:D #8 \c_minus_one
11403     \tex_advance:D #9 \c_one_thousand_million
11404   \fi:
11405   \if_int_compare:w #8 < \c_zero
11406     \tex_advance:D #7 \c_minus_one
11407     \tex_advance:D #8 \c_one_thousand_million
11408   \fi:
11409   \if_int_compare:w #7 < \c_zero
11410     \if_int_compare:w \int_eval:w #8 + #9 = \c_zero
11411       #7 -#7
11412   \else:

```

```

11413 \tex_advance:D #7 \c_one
11414 #8 \int_eval:w \c_one_thousand_million - #8 \int_eval_end:
11415 #9 \int_eval:w \c_one_thousand_million - #9 \int_eval_end:
11416 \fi:
11417 \fi:
11418 }

```

(End definition for \fp_sub:NNNNNNNN. This function is documented on page ??.)

\fp_mul:NNNNNN Decimal-part only multiplication but with higher accuracy than the user version.

```

11419 \cs_new_protected:Npn \fp_mul:NNNNNN #1#2#3#4#5#6
11420 {
11421   \fp_mul_split:NNNN #1
11422   \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11423   \fp_mul_split:NNNN #2
11424   \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_a_vi_int
11425   \fp_mul_split:NNNN #3
11426   \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11427   \fp_mul_split:NNNN #4
11428   \l_fp_mul_b_iv_int \l_fp_mul_b_v_int \l_fp_mul_b_vi_int
11429   \l_fp_mul_output_int \c_zero
11430   \tl_clear:N \l_fp_mul_output_tl
11431   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_vi_int
11432   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_v_int
11433   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iv_int
11434   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_iii_int
11435   \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_ii_int
11436   \fp_mul_product:NN \l_fp_mul_a_vi_int \l_fp_mul_b_i_int
11437   \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11438   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_v_int
11439   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iv_int
11440   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iii_int
11441   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_ii_int
11442   \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_i_int
11443   \fp_mul_end_level:
11444   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iv_int
11445   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iii_int
11446   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_ii_int
11447   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_i_int
11448   \fp_mul_end_level:
11449   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11450   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11451   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11452   \fp_mul_end_level:
11453   #6 0 \l_fp_mul_output_tl \scan_stop:
11454   \tl_clear:N \l_fp_mul_output_tl
11455   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11456   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int
11457   \fp_mul_end_level:
11458   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11459   \fp_mul_end_level:

```

```

11460 \fp_mul_end_level:
11461 #5 0 \l_fp_mul_output_tl \scan_stop:
11462 }
      (End definition for \fp_mul:NNNNNN. This function is documented on page ??.)

```

\fp_mul:NNNNNNNN For internal multiplication where the integer does need to be retained. This means of course that this code is quite slow, and so is only used when necessary.

```

11463 \cs_new_protected:Npn \fp_mul:NNNNNNNNN #1#2#3#4#5#6#7#8#9
11464 {
11465   \fp_mul_split:NNNN #2
11466   \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11467   \fp_mul_split:NNNN #3
11468   \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_a_vi_int
11469   \fp_mul_split:NNNN #5
11470   \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11471   \fp_mul_split:NNNN #6
11472   \l_fp_mul_b_iv_int \l_fp_mul_b_v_int \l_fp_mul_b_vi_int
11473   \l_fp_mul_output_int \c_zero
11474   \tl_clear:N \l_fp_mul_output_tl
11475   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_vi_int
11476   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_v_int
11477   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iv_int
11478   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_iii_int
11479   \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_ii_int
11480   \fp_mul_product:NN \l_fp_mul_a_vi_int \l_fp_mul_b_i_int
11481   \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11482   \fp_mul_product:NN #1 \l_fp_mul_b_vi_int
11483   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_v_int
11484   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iv_int
11485   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iii_int
11486   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_ii_int
11487   \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_i_int
11488   \fp_mul_product:NN \l_fp_mul_a_vi_int #4
11489   \fp_mul_end_level:
11490   \fp_mul_product:NN #1 \l_fp_mul_b_v_int
11491   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iv_int
11492   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iii_int
11493   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_ii_int
11494   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_i_int
11495   \fp_mul_product:NN \l_fp_mul_a_v_int #4
11496   \fp_mul_end_level:
11497   \fp_mul_product:NN #1 \l_fp_mul_b_iv_int
11498   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11499   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11500   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11501   \fp_mul_product:NN \l_fp_mul_a_iv_int #4
11502   \fp_mul_end_level:
11503   #9 0 \l_fp_mul_output_tl \scan_stop:
11504   \tl_clear:N \l_fp_mul_output_tl

```

```

11505 \fp_mul_product:NN #1 \l_fp_mul_b_iii_int
11506 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11507 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int
11508 \fp_mul_product:NN \l_fp_mul_a_iii_int #4
11509 \fp_mul_end_level:
11510 \fp_mul_product:NN #1 \l_fp_mul_b_ii_int
11511 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11512 \fp_mul_product:NN \l_fp_mul_a_ii_int #4
11513 \fp_mul_end_level:
11514 \fp_mul_product:NN #1 \l_fp_mul_b_i_int
11515 \fp_mul_product:NN \l_fp_mul_a_i_int #4
11516 \fp_mul_end_level:
11517 #8 0 \l_fp_mul_output_tl \scan_stop:
11518 \tl_clear:N \l_fp_mul_output_tl
11519 \fp_mul_product:NN #1 #4
11520 \fp_mul_end_level:
11521 #7 0 \l_fp_mul_output_tl \scan_stop:
11522 }

```

(End definition for \fp_mul:NNNNNNNN. This function is documented on page ??.)

\fp_div_integer:NNNNN Here, division is always by an integer, and so it is possible to use TeX's native calculations rather than doing it in macros. The idea here is to divide the decimal part, find any remainder, then do the real division of the two parts before adding in what is needed for the remainder.

```

11523 \cs_new_protected:Npn \fp_div_integer:NNNNN #1#2#3#4#5
11524 {
11525   \l_fp_tmp_int #1
11526   \tex_divide:D \l_fp_tmp_int #3
11527   \l_fp_tmp_int \int_eval:w #1 - \l_fp_tmp_int * #3 \int_eval_end:
11528   #4 #1
11529   \tex_divide:D #4 #3
11530   #5 #2
11531   \tex_divide:D #5 #3
11532   \tex_multiply:D \l_fp_tmp_int \c_one_thousand
11533   \tex_divide:D \l_fp_tmp_int #3
11534   #5 \int_eval:w #5 + \l_fp_tmp_int * \c_one_million \int_eval_end:
11535   \if_int_compare:w #5 > \c_one_thousand_million
11536     \tex_advance:D #4 \c_one
11537     \tex_advance:D #5 -\c_one_thousand_million
11538   \fi:
11539 }

```

(End definition for \fp_div_integer:NNNNN. This function is documented on page ??.)

\fp_extended_normalise: The “extended” integers for internal use are mainly used in fixed-point mode. This comes up in a few places, so a generalised utility is made available to carry out the change. This function simply calls the two loops to shift the input to the point of having a zero exponent.

```

11540 \cs_new_protected_nopar:Npn \fp_extended_normalise:
11541 {

```



```

11542 \fp_extended_normalise_aux_i:
11543 \fp_extended_normalise_aux_ii:
11544 }
11545 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_i:
11546 {
11547 \if_int_compare:w \l_fp_input_a_exponent_int > \c_zero
11548 \tex_multiply:D \l_fp_input_a_integer_int \c_ten
11549 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11550 \exp_after:wN \fp_extended_normalise_aux_i:w
11551 \int_use:N \l_fp_input_a_decimal_int \q_stop
11552 \exp_after:wN \fp_extended_normalise_aux_i:
11553 \fi:
11554 }
11555 \cs_new_protected:Npn \fp_extended_normalise_aux_i:w
11556 #1#2#3#4#5#6#7#8#9 \q_stop
11557 {
11558 \l_fp_input_a_integer_int
11559 \int_eval:w \l_fp_input_a_integer_int + #2 \scan_stop:
11560 \l_fp_input_a_decimal_int #3#4#5#6#7#8#9 0 \scan_stop:
11561 \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
11562 \exp_after:wN \fp_extended_normalise_aux_ii:w
11563 \int_use:N \l_fp_input_a_extended_int \q_stop
11564 }
11565 \cs_new_protected:Npn \fp_extended_normalise_aux_ii:w
11566 #1#2#3#4#5#6#7#8#9 \q_stop
11567 {
11568 \l_fp_input_a_decimal_int
11569 \int_eval:w \l_fp_input_a_decimal_int + #2 \scan_stop:
11570 \l_fp_input_a_extended_int #3#4#5#6#7#8#9 0 \scan_stop:
11571 \tex_advance:D \l_fp_input_a_exponent_int \c_minus_one
11572 }
11573 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_ii:
11574 {
11575 \if_int_compare:w \l_fp_input_a_exponent_int < \c_zero
11576 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11577 \exp_after:wN \use_i:nn \exp_after:wN
11578 \fp_extended_normalise_ii_aux:NNNNNNNNN
11579 \int_use:N \l_fp_input_a_decimal_int
11580 \exp_after:wN \fp_extended_normalise_aux_ii:
11581 \fi:
11582 }
11583 \cs_new_protected:Npn \fp_extended_normalise_ii_aux:NNNNNNNNN
11584 #1#2#3#4#5#6#7#8#9
11585 {
11586 \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
11587 \l_fp_input_a_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
11588 \else:
11589 \tl_set:Nx \l_fp_tmp_tl
11590 {
11591 \int_use:N \l_fp_input_a_integer_int

```

```

11592         #1#2#3#4#5#6#7#8
11593     }
11594     \l_fp_input_a_integer_int \c_zero
11595     \l_fp_input_a_decimal_int \l_fp_tmp_tl \scan_stop:
11596 \fi:
11597 \tex_divide:D \l_fp_input_a_extended_int \c_ten
11598 \tl_set:Nx \l_fp_tmp_tl
11599 {
11600     #9
11601     \int_use:N \l_fp_input_a_extended_int
11602 }
11603 \l_fp_input_a_extended_int \l_fp_tmp_tl \scan_stop:
11604 \tex_advance:D \l_fp_input_a_exponent_int \c_one
11605 }

```

(End definition for \fp_extended_normalise:. This function is documented on page ??.)

\fp_extended_normalise_output: At some stages in working out extended output, it is possible for the value to need shifting to keep the integer part in range. This only ever happens such that the integer needs to be made smaller.

```

\fp_extended_normalise_output_aux_i:NNNNNNNNN
\fp_extended_normalise_output_aux_ii:NNNNNNNNN
\fp_extended_normalise_output_aux:N
11606 \cs_new_protected_nopar:Npn \fp_extended_normalise_output:
11607 {
11608     \if_int_compare:w \l_fp_output_integer_int > \c_nine
11609     \tex_advance:D \l_fp_output_integer_int \c_one_thousand_million
11610     \exp_after:wN \use_i:nn \exp_after:wN
11611     \fp_extended_normalise_output_aux_i:NNNNNNNNN
11612     \int_use:N \l_fp_output_integer_int
11613     \exp_after:wN \fp_extended_normalise_output:
11614 \fi:
11615 }
11616 \cs_new_protected:Npn \fp_extended_normalise_output_aux_i:NNNNNNNNN
11617 #1#2#3#4#5#6#7#8#9
11618 {
11619     \l_fp_output_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
11620     \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
11621     \tl_set:Nx \l_fp_tmp_tl
11622     {
11623         #9
11624         \exp_after:wN \use_none:n
11625         \int_use:N \l_fp_output_decimal_int
11626     }
11627     \exp_after:wN \fp_extended_normalise_output_aux_ii:NNNNNNNNN
11628     \l_fp_tmp_tl
11629 }
11630 \cs_new_protected:Npn \fp_extended_normalise_output_aux_ii:NNNNNNNNN
11631 #1#2#3#4#5#6#7#8#9
11632 {
11633     \l_fp_output_decimal_int #1#2#3#4#5#6#7#8#9 \scan_stop:
11634     \fp_extended_normalise_output_aux:N
11635 }

```

```

11636 \cs_new_protected:Npn \fp_extended_normalise_output_aux:N #1
11637 {
11638   \tex_advance:D \l_fp_output_extended_int \c_one_thousand_million
11639   \tex_divide:D \l_fp_output_extended_int \c_ten
11640   \tl_set:Nx \l_fp_tmp_tl
11641   {
11642     #1
11643     \exp_after:wN \use_none:n
11644     \int_use:N \l_fp_output_extended_int
11645   }
11646   \l_fp_output_extended_int \l_fp_tmp_tl \scan_stop:
11647   \tex_advance:D \l_fp_output_exponent_int \c_one
11648 }

```

(End definition for \fp_extended_normalise_output:. This function is documented on page ??.)

203.11 Trigonometric functions

\fp_trig_normalise: For normalisation, the code essentially switches to fixed-point arithmetic. There is a shift of the exponent, then repeated subtractions. The end result is a number in the range $-\pi < x \leq \pi$.

```

11649 \cs_new_protected_nopar:Npn \fp_trig_normalise:
11650 {
11651   \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
11652     \l_fp_input_a_extended_int \c_zero
11653     \fp_extended_normalise:
11654     \fp_trig_normalise_aux:
11655     \if_int_compare:w \l_fp_input_a_integer_int < \c_zero
11656       \l_fp_input_a_sign_int -\l_fp_input_a_sign_int
11657       \l_fp_input_a_integer_int -\l_fp_input_a_integer_int
11658     \fi:
11659     \exp_after:wN \fp_trig_octant:
11660   \else:
11661     \l_fp_input_a_sign_int \c_one
11662     \l_fp_output_integer_int \c_zero
11663     \l_fp_output_decimal_int \c_zero
11664     \l_fp_output_exponent_int \c_zero
11665     \exp_after:wN \fp_trig_overflow_msg:
11666   \fi:
11667 }
11668 \cs_new_protected_nopar:Npn \fp_trig_normalise_aux:
11669 {
11670   \if_int_compare:w \l_fp_input_a_integer_int > \c_three
11671     \fp_trig_sub:NNN
11672     \c_six \c_fp_two_pi_decimal_int \c_fp_two_pi_extended_int
11673     \exp_after:wN \fp_trig_normalise_aux:
11674   \else:
11675     \if_int_compare:w \l_fp_input_a_integer_int > \c_two
11676       \if_int_compare:w \l_fp_input_a_decimal_int > \c_fp_pi_decimal_int
11677         \fp_trig_sub:NNN

```

```

11678         \c_six \c_fp_two_pi_decimal_int \c_fp_two_pi_extended_int
11679         \exp_after:wN \exp_after:wN \exp_after:wN
11680         \exp_after:wN \exp_after:wN \exp_after:wN
11681         \exp_after:wN \fp_trig_normalise_aux:
11682         \fi:
11683     \fi:
11684 \fi:
11685 }

```

Here, there may be a sign change but there will never be any variation in the input. So a dedicated function can be used.

```

11686 \cs_new_protected:Npn \fp_trig_sub:NNN #1#2#3
11687 {
11688     \l_fp_input_a_integer_int
11689     \int_eval:w \l_fp_input_a_integer_int - #1 \int_eval_end:
11690     \l_fp_input_a_decimal_int
11691     \int_eval:w \l_fp_input_a_decimal_int - #2 \int_eval_end:
11692     \l_fp_input_a_extended_int
11693     \int_eval:w \l_fp_input_a_extended_int - #3 \int_eval_end:
11694     \if_int_compare:w \l_fp_input_a_extended_int < \c_zero
11695         \tex_advance:D \l_fp_input_a_decimal_int \c_minus_one
11696         \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
11697     \fi:
11698     \if_int_compare:w \l_fp_input_a_decimal_int < \c_zero
11699         \tex_advance:D \l_fp_input_a_integer_int \c_minus_one
11700         \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11701     \fi:
11702     \if_int_compare:w \l_fp_input_a_integer_int < \c_zero
11703         \l_fp_input_a_sign_int -\l_fp_input_a_sign_int
11704         \if_int_compare:w
11705             \int_eval:w
11706             \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
11707             = \c_zero
11708             \l_fp_input_a_integer_int -\l_fp_input_a_integer_int
11709         \else:
11710             \l_fp_input_a_integer_int
11711             \int_eval:w
11712             - \l_fp_input_a_integer_int - \c_one
11713             \int_eval_end:
11714             \l_fp_input_a_decimal_int
11715             \int_eval:w
11716             \c_one_thousand_million - \l_fp_input_a_decimal_int
11717             \int_eval_end:
11718             \l_fp_input_a_extended_int
11719             \int_eval:w
11720             \c_one_thousand_million - \l_fp_input_a_extended_int
11721             \int_eval_end:
11722     \fi:
11723 \fi:
11724 }

```

(End definition for \fp_trig_normalise:. This function is documented on page ??.)

\fp_trig_octant: Here, the input is further reduced into the range $0 < x \leq \pi/4$. This is pretty simple:
\fp_trig_octant_aux_i: check if $\pi/4$ can be taken off and if it can do it and loop. The check at the end is to “mop
\fp_trig_octant_aux_ii: up” values which are so close to $\pi/4$ that they should be treated as such. The test for
an even octant is needed as the ‘remainder’ needed is from the nearest $\pi/2$. The check
for octant 4 is needed as an exact π input will otherwise end up in the wrong place!

```

11725 \cs_new_protected_nopar:Npn \fp_trig_octant:
11726 {
11727   \l_fp_trig_octant_int \c_one
11728   \fp_trig_octant_aux_i:
11729   \if_int_compare:w \l_fp_input_a_decimal_int < \c_ten
11730     \l_fp_input_a_decimal_int \c_zero
11731     \l_fp_input_a_extended_int \c_zero
11732   \fi:
11733   \if_int_odd:w \l_fp_trig_octant_int
11734   \else:
11735     \fp_sub:NNNNNNNNN
11736     \c_zero \c_fp_pi_by_four_decimal_int \c_fp_pi_by_four_extended_int
11737     \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11738     \l_fp_input_a_extended_int
11739     \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11740     \l_fp_input_a_extended_int
11741   \fi:
11742 }
11743 \cs_new_protected_nopar:Npn \fp_trig_octant_aux_i:
11744 {
11745   \if_int_compare:w \l_fp_trig_octant_int > \c_four
11746     \l_fp_trig_octant_int \c_four
11747     \l_fp_input_a_decimal_int \c_fp_pi_by_four_decimal_int
11748     \l_fp_input_a_extended_int \c_fp_pi_by_four_extended_int
11749   \else:
11750     \exp_after:wN \fp_trig_octant_aux_ii:
11751   \fi:
11752 }
11753 \cs_new_protected_nopar:Npn \fp_trig_octant_aux_ii:
11754 {
11755   \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
11756     \fp_sub:NNNNNNNNN
11757     \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11758     \l_fp_input_a_extended_int
11759     \c_zero \c_fp_pi_by_four_decimal_int \c_fp_pi_by_four_extended_int
11760     \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11761     \l_fp_input_a_extended_int
11762     \tex_advance:D \l_fp_trig_octant_int \c_one
11763     \exp_after:wN \fp_trig_octant_aux_i:
11764   \else:
11765     \if_int_compare:w
11766       \l_fp_input_a_decimal_int > \c_fp_pi_by_four_decimal_int

```

```

11767 \fp_sub:NNNNNNNNN
11768 \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11769 \l_fp_input_a_extended_int
11770 \c_zero \c_fp_pi_by_four_decimal_int
11771 \c_fp_pi_by_four_extended_int
11772 \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11773 \l_fp_input_a_extended_int
11774 \tex_advance:D \l_fp_trig_octant_int \c_one
11775 \exp_after:wN \exp_after:wN \exp_after:wN
11776 \fp_trig_octant_aux_i:
11777 \fi:
11778 \fi:
11779 }

```

(End definition for \fp_trig_octant:. This function is documented on page ??.)

\fp_sin:Nn Calculating the sine starts off in the usual way. There is a check to see if the value has
\fp_sin:cn already been worked out before proceeding further.

\fp_gsin:Nn 11780 \cs_new_protected_nopar:Npn \fp_sin:Nn { \fp_sin_aux:NNn \tl_set:Nn }
\fp_gsin:cn 11781 \cs_new_protected_nopar:Npn \fp_gsin:Nn { \fp_sin_aux:NNn \tl_gset:Nn }
\fp_sin_aux:NNn 11782 \cs_generate_variant:Nn \fp_sin:Nn { c }
\fp_sin_aux_i: 11783 \cs_generate_variant:Nn \fp_gsin:Nn { c }
\fp_sin_aux_ii: The internal routine for sines does a check to see if the value is already known. This
saves a lot of repetition when doing rotations. For very small values it is best to simply
return the input as the sine: the cut-off is 1×10^{-5} .

```

11784 \cs_new_protected:Npn \fp_sin_aux:NNn #1#2#3
11785 {
11786 \group_begin:
11787 \fp_split:Nn a {#3}
11788 \fp_standardise:NNNN
11789 \l_fp_input_a_sign_int
11790 \l_fp_input_a_integer_int
11791 \l_fp_input_a_decimal_int
11792 \l_fp_input_a_exponent_int
11793 \tl_set:Nx \l_fp_arg_tl
11794 {
11795 \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11796 -
11797 \else:
11798 +
11799 \fi:
11800 \int_use:N \l_fp_input_a_integer_int
11801 .
11802 \exp_after:wN \use_none:n
11803 \int_value:w \int_eval:w
11804 \l_fp_input_a_decimal_int + \c_one_thousand_million
11805 e
11806 \int_use:N \l_fp_input_a_exponent_int
11807 }
11808 \if_int_compare:w \l_fp_input_a_exponent_int < -\c_five

```

```

11809     \cs_set_protected_nopar:Npx \fp_tmp:w
11810     {
11811         \group_end:
11812         #1 \exp_not:N #2 { \l_fp_arg_tl }
11813     }
11814 \else:
11815     \if_cs_exist:w
11816         c_fp_sin ( \l_fp_arg_tl ) _fp
11817     \cs_end:
11818     \else:
11819         \exp_after:wN \exp_after:wN \exp_after:wN
11820         \fp_sin_aux_i:
11821     \fi:
11822     \cs_set_protected_nopar:Npx \fp_tmp:w
11823     {
11824         \group_end:
11825         #1 \exp_not:N #2
11826         { \use:c { c_fp_sin ( \l_fp_arg_tl ) _fp } }
11827     }
11828 \fi:
11829 \fp_tmp:w
11830 }

```

The internals for sine first normalise the input into an octant, then choose the correct set up for the Taylor series. The sign for the sine function is easy, so there is no worry about it. So the only thing to do is to get the output standardised.

```

11831 \cs_new_protected_nopar:Npn \fp_sin_aux_i:
11832 {
11833     \fp_trig_normalise:
11834     \fp_sin_aux_ii:
11835     \if_int_compare:w \l_fp_output_integer_int = \c_one
11836         \l_fp_output_exponent_int \c_zero
11837     \else:
11838         \l_fp_output_integer_int \l_fp_output_decimal_int
11839         \l_fp_output_decimal_int \l_fp_output_extended_int
11840         \l_fp_output_exponent_int -\c_nine
11841     \fi:
11842     \fp_standardise:NNNN
11843     \l_fp_input_a_sign_int
11844     \l_fp_output_integer_int
11845     \l_fp_output_decimal_int
11846     \l_fp_output_exponent_int
11847     \tl_new:c { c_fp_sin ( \l_fp_arg_tl ) _fp }
11848     \tl_gset:cx { c_fp_sin ( \l_fp_arg_tl ) _fp }
11849     {
11850         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11851             +
11852         \else:
11853             -
11854     \fi:

```

```

11855         \int_use:N \l_fp_output_integer_int
11856         .
11857         \exp_after:wN \use_none:n
11858         \int_value:w \int_eval:w
11859         \l_fp_output_decimal_int + \c_one_thousand_million
11860         e
11861         \int_use:N \l_fp_output_exponent_int
11862     }
11863 }
11864 \cs_new_protected_nopar:Npn \fp_sin_aux_ii:
11865 {
11866     \if_case:w \l_fp_trig_octant_int
11867     \or:
11868         \exp_after:wN \fp_trig_calc_sin:
11869     \or:
11870         \exp_after:wN \fp_trig_calc_cos:
11871     \or:
11872         \exp_after:wN \fp_trig_calc_cos:
11873     \or:
11874         \exp_after:wN \fp_trig_calc_sin:
11875     \fi:
11876 }

```

(End definition for \fp_sin:Nn and \fp_sin:cn. These functions are documented on page ??.)

```

\fp_cos:Nn Cosine is almost identical, but there is no short cut code here.
\fp_cos:cn
\fp_gcos:Nn
\fp_gcos:cn
\fp_cos_aux:NNn
\fp_cos_aux_i:
\fp_cos_aux_ii:
11877 \cs_new_protected_nopar:Npn \fp_cos:Nn { \fp_cos_aux:NNn \tl_set:Nn }
11878 \cs_new_protected_nopar:Npn \fp_gcos:Nn { \fp_cos_aux:NNn \tl_gset:Nn }
11879 \cs_generate_variant:Nn \fp_cos:Nn { c }
11880 \cs_generate_variant:Nn \fp_gcos:Nn { c }
11881 \cs_new_protected:Npn \fp_cos_aux:NNn #1#2#3
11882 {
11883     \group_begin:
11884     \fp_split:Nn a {#3}
11885     \fp_standardise:NNNN
11886     \l_fp_input_a_sign_int
11887     \l_fp_input_a_integer_int
11888     \l_fp_input_a_decimal_int
11889     \l_fp_input_a_exponent_int
11890     \tl_set:Nx \l_fp_arg_tl
11891     {
11892         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11893         -
11894         \else:
11895         +
11896         \fi:
11897         \int_use:N \l_fp_input_a_integer_int
11898         .
11899         \exp_after:wN \use_none:n
11900         \int_value:w \int_eval:w
11901         \l_fp_input_a_decimal_int + \c_one_thousand_million

```



```

11902         e
11903         \int_use:N \l_fp_input_a_exponent_int
11904     }
11905     \if_cs_exist:w c_fp_cos ( \l_fp_arg_tl ) _fp \cs_end:
11906     \else:
11907         \exp_after:wN \fp_cos_aux_i:
11908     \fi:
11909     \cs_set_protected_nopar:Npx \fp_tmp:w
11910     {
11911         \group_end:
11912         #1 \exp_not:N #2
11913         { \use:c { c_fp_cos ( \l_fp_arg_tl ) _fp } }
11914     }
11915     \fp_tmp:w
11916 }

```

Almost the same as for sine: just a bit of correction for the sign of the output.

```

11917 \cs_new_protected_nopar:Npn \fp_cos_aux_i:
11918 {
11919     \fp_trig_normalise:
11920     \fp_cos_aux_ii:
11921     \if_int_compare:w \l_fp_output_integer_int = \c_one
11922         \l_fp_output_exponent_int \c_zero
11923     \else:
11924         \l_fp_output_integer_int \l_fp_output_decimal_int
11925         \l_fp_output_decimal_int \l_fp_output_extended_int
11926         \l_fp_output_exponent_int -\c_nine
11927     \fi:
11928     \fp_standardise:NNNN
11929     \l_fp_input_a_sign_int
11930     \l_fp_output_integer_int
11931     \l_fp_output_decimal_int
11932     \l_fp_output_exponent_int
11933     \tl_new:c { c_fp_cos ( \l_fp_arg_tl ) _fp }
11934     \tl_gset:cx { c_fp_cos ( \l_fp_arg_tl ) _fp }
11935     {
11936         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11937             +
11938         \else:
11939             -
11940         \fi:
11941         \int_use:N \l_fp_output_integer_int
11942         .
11943         \exp_after:wN \use_none:n
11944         \int_value:w \int_eval:w
11945             \l_fp_output_decimal_int + \c_one_thousand_million
11946         e
11947         \int_use:N \l_fp_output_exponent_int
11948     }
11949 }

```

```

11950 \cs_new_protected_nopar:Npn \fp_cos_aux_ii:
11951 {
11952   \if_case:w \l_fp_trig_octant_int
11953   \or:
11954     \exp_after:wN \fp_trig_calc_cos:
11955   \or:
11956     \exp_after:wN \fp_trig_calc_sin:
11957   \or:
11958     \exp_after:wN \fp_trig_calc_sin:
11959   \or:
11960     \exp_after:wN \fp_trig_calc_cos:
11961   \fi:
11962   \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
11963     \if_int_compare:w \l_fp_trig_octant_int > \c_two
11964       \l_fp_input_a_sign_int \c_minus_one
11965     \fi:
11966   \else:
11967     \if_int_compare:w \l_fp_trig_octant_int > \c_two
11968     \else:
11969       \l_fp_input_a_sign_int \c_one
11970     \fi:
11971   \fi:
11972 }

```

(End definition for \fp_cos:Nn and \fp_cos:cn. These functions are documented on page ??.)

\fp_trig_calc_cos: These functions actually do the calculation for sine and cosine.

\fp_trig_calc_sin: 11973 \cs_new_protected_nopar:Npn \fp_trig_calc_cos:

\fp_trig_calc_Taylor: 11974 {

```

11975   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
11976     \l_fp_output_integer_int \c_one
11977     \l_fp_output_decimal_int \c_zero
11978   \else:
11979     \l_fp_trig_sign_int \c_minus_one
11980     \fp_mul:NNNNNN
11981       \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11982       \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
11983     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11984     \fp_div_integer:NNNNN
11985       \l_fp_trig_decimal_int \l_fp_trig_extended_int
11986       \c_two
11987     \l_fp_trig_decimal_int \l_fp_trig_extended_int
11988     \l_fp_count_int \c_three
11989     \if_int_compare:w \l_fp_trig_extended_int = \c_zero
11990       \if_int_compare:w \l_fp_trig_decimal_int = \c_zero
11991         \l_fp_output_integer_int \c_one
11992         \l_fp_output_decimal_int \c_zero
11993         \l_fp_output_extended_int \c_zero
11994       \else:
11995         \l_fp_output_integer_int \c_zero
11996         \l_fp_output_decimal_int \c_one_thousand_million

```

```

11997         \l_fp_output_extended_int \c_zero
11998     \fi:
11999 \else:
12000     \l_fp_output_integer_int \c_zero
12001     \l_fp_output_decimal_int 999999999 \scan_stop:
12002     \l_fp_output_extended_int \c_one_thousand_million
12003 \fi:
12004 \tex_advance:D \l_fp_output_extended_int -\l_fp_trig_extended_int
12005 \tex_advance:D \l_fp_output_decimal_int -\l_fp_trig_decimal_int
12006 \exp_after:wN \fp_trig_calc_Taylor:
12007 \fi:
12008 }
12009 \cs_new_protected_nopar:Npn \fp_trig_calc_sin:
12010 {
12011     \l_fp_output_integer_int \c_zero
12012     \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12013         \l_fp_output_decimal_int \c_zero
12014     \else:
12015         \l_fp_output_decimal_int \l_fp_input_a_decimal_int
12016         \l_fp_output_extended_int \l_fp_input_a_extended_int
12017         \l_fp_trig_sign_int \c_one
12018         \l_fp_trig_decimal_int \l_fp_input_a_decimal_int
12019         \l_fp_trig_extended_int \l_fp_input_a_extended_int
12020         \l_fp_count_int \c_two
12021         \exp_after:wN \fp_trig_calc_Taylor:
12022     \fi:
12023 }

```

This implements a Taylor series calculation for the trigonometric functions. Lots of shuffling about as T_EX is not exactly a natural choice for this sort of thing.

```

12024 \cs_new_protected_nopar:Npn \fp_trig_calc_Taylor:
12025 {
12026     \l_fp_trig_sign_int -\l_fp_trig_sign_int
12027     \fp_mul:NNNNNN
12028     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12029     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12030     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12031     \fp_mul:NNNNNN
12032     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12033     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12034     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12035     \fp_div_integer:NNNNN
12036     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12037     \l_fp_count_int
12038     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12039     \tex_advance:D \l_fp_count_int \c_one
12040     \fp_div_integer:NNNNN
12041     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12042     \l_fp_count_int
12043     \l_fp_trig_decimal_int \l_fp_trig_extended_int

```

```

12044 \tex_advance:D \l_fp_count_int \c_one
12045 \if_int_compare:w \l_fp_trig_decimal_int > \c_zero
12046 \if_int_compare:w \l_fp_trig_sign_int > \c_zero
12047 \tex_advance:D \l_fp_output_decimal_int \l_fp_trig_decimal_int
12048 \tex_advance:D \l_fp_output_extended_int
12049 \l_fp_trig_extended_int
12050 \if_int_compare:w \l_fp_output_extended_int < \c_one_thousand_million
12051 \else:
12052 \tex_advance:D \l_fp_output_decimal_int \c_one
12053 \tex_advance:D \l_fp_output_extended_int
12054 -\c_one_thousand_million
12055 \fi:
12056 \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12057 \else:
12058 \tex_advance:D \l_fp_output_integer_int \c_one
12059 \tex_advance:D \l_fp_output_decimal_int
12060 -\c_one_thousand_million
12061 \fi:
12062 \else:
12063 \tex_advance:D \l_fp_output_decimal_int -\l_fp_trig_decimal_int
12064 \tex_advance:D \l_fp_output_extended_int
12065 -\l_fp_input_a_extended_int
12066 \if_int_compare:w \l_fp_output_extended_int < \c_zero
12067 \tex_advance:D \l_fp_output_decimal_int \c_minus_one
12068 \tex_advance:D \l_fp_output_extended_int \c_one_thousand_million
12069 \fi:
12070 \if_int_compare:w \l_fp_output_decimal_int < \c_zero
12071 \tex_advance:D \l_fp_output_integer_int \c_minus_one
12072 \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
12073 \fi:
12074 \fi:
12075 \exp_after:wN \fp_trig_calc_Taylor:
12076 \fi:
12077 }

```

(End definition for \fp_trig_calc_cos:. This function is documented on page ??.)

\fp_tan:Nn As might be expected, tangents are calculated from the sine and cosine by division. So
\fp_tan:cn there is a bit of set up, the two subsidiary pieces of work are done and then a division
\fp_gtan:Nn takes place. For small numbers, the same approach is used as for sines, with the input
\fp_gtan:cn value simply returned as is.

```

12078 \cs_new_protected_nopar:Npn \fp_tan:Nn { \fp_tan_aux:NNn \tl_set:Nn }
12079 \cs_new_protected_nopar:Npn \fp_gtan:Nn { \fp_tan_aux:NNn \tl_gset:Nn }
12080 \cs_generate_variant:Nn \fp_tan:Nn { c }
12081 \cs_generate_variant:Nn \fp_gtan:Nn { c }
12082 \cs_new_protected:Npn \fp_tan_aux:NNn #1#2#3
12083 {
12084 \group_begin:
12085 \fp_split:Nn a {#3}
12086 \fp_standardise:NNNN

```

```

12087         \l_fp_input_a_sign_int
12088         \l_fp_input_a_integer_int
12089         \l_fp_input_a_decimal_int
12090         \l_fp_input_a_exponent_int
12091     \tl_set:Nx \l_fp_arg_tl
12092     {
12093         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12094         -
12095         \else:
12096         +
12097         \fi:
12098         \int_use:N \l_fp_input_a_integer_int
12099         .
12100         \exp_after:wN \use_none:n
12101         \int_value:w \int_eval:w
12102         \l_fp_input_a_decimal_int + \c_one_thousand_million
12103         e
12104         \int_use:N \l_fp_input_a_exponent_int
12105     }
12106     \if_int_compare:w \l_fp_input_a_exponent_int < -\c_five
12107     \cs_set_protected_nopar:Npx \fp_tmp:w
12108     {
12109         \group_end:
12110         #1 \exp_not:N #2 { \l_fp_arg_tl }
12111     }
12112     \else:
12113         \if_cs_exist:w
12114             c_fp_tan ( \l_fp_arg_tl ) _fp
12115         \cs_end:
12116         \else:
12117             \exp_after:wN \exp_after:wN \exp_after:wN
12118             \fp_tan_aux_i:
12119         \fi:
12120         \cs_set_protected_nopar:Npx \fp_tmp:w
12121         {
12122             \group_end:
12123             #1 \exp_not:N #2
12124             { \use:c { c_fp_tan ( \l_fp_arg_tl ) _fp } }
12125         }
12126         \fi:
12127     \fp_tmp:w
12128 }

```

The business of the calculation does not check for stored sines or cosines as there would then be an overhead to reading them back in. There is also no need to worry about “small” sine values as these will have been dealt with earlier. There is a two-step lead off so that undefined division is not even attempted.

```

12129 \cs_new_protected_nopar:Npn \fp_tan_aux_i:
12130 {
12131     \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten

```

```

12132     \exp_after:wN \fp_tan_aux_ii:
12133 \else:
12134     \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
12135     \c_zero_fp
12136     \exp_after:wN \fp_trig_overflow_msg:
12137 \fi:
12138 }
12139 \cs_new_protected_nopar:Npn \fp_tan_aux_ii:
12140 {
12141     \fp_trig_normalise:
12142     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12143     \if_int_compare:w \l_fp_trig_octant_int > \c_two
12144     \l_fp_output_sign_int \c_minus_one
12145     \else:
12146     \l_fp_output_sign_int \c_one
12147     \fi:
12148 \else:
12149     \if_int_compare:w \l_fp_trig_octant_int > \c_two
12150     \l_fp_output_sign_int \c_one
12151     \else:
12152     \l_fp_output_sign_int \c_minus_one
12153     \fi:
12154 \fi:
12155 \fp_cos_aux_ii:
12156 \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12157 \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
12158     \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
12159     \c_undefined_fp
12160 \else:
12161     \exp_after:wN \exp_after:wN \exp_after:wN
12162     \fp_tan_aux_iii:
12163 \fi:
12164 \else:
12165     \exp_after:wN \fp_tan_aux_iii:
12166 \fi:
12167 }

```

The division is done here using the same code as the standard division unit, shifting the digits in the calculated sine and cosine to maintain accuracy.

```

12168 \cs_new_protected_nopar:Npn \fp_tan_aux_iii:
12169 {
12170     \l_fp_input_b_integer_int \l_fp_output_decimal_int
12171     \l_fp_input_b_decimal_int \l_fp_output_extended_int
12172     \l_fp_input_b_exponent_int -\c_nine
12173     \fp_standardise:NNNN
12174     \l_fp_input_b_sign_int
12175     \l_fp_input_b_integer_int
12176     \l_fp_input_b_decimal_int
12177     \l_fp_input_b_exponent_int
12178     \fp_sin_aux_ii:

```

```

12179 \l_fp_input_a_integer_int \l_fp_output_decimal_int
12180 \l_fp_input_a_decimal_int \l_fp_output_extended_int
12181 \l_fp_input_a_exponent_int -\c_nine
12182 \fp_standardise:NNNN
12183 \l_fp_input_a_sign_int
12184 \l_fp_input_a_integer_int
12185 \l_fp_input_a_decimal_int
12186 \l_fp_input_a_exponent_int
12187 \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12188 \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
12189 \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
12190 \c_zero_fp
12191 \else:
12192 \exp_after:wN \exp_after:wN \exp_after:wN \fp_tan_aux_iv:
12193 \fi:
12194 \else:
12195 \exp_after:wN \fp_tan_aux_iv:
12196 \fi:
12197 }
12198 \cs_new_protected_nopar:Npn \fp_tan_aux_iv:
12199 {
12200 \l_fp_output_integer_int \c_zero
12201 \l_fp_output_decimal_int \c_zero
12202 \cs_set_eq:NN \fp_div_store: \fp_div_store_integer:
12203 \l_fp_div_offset_int \c_one_hundred_million
12204 \fp_div_loop:
12205 \l_fp_output_exponent_int
12206 \int_eval:w
12207 \l_fp_input_a_exponent_int - \l_fp_input_b_exponent_int
12208 \int_eval_end:
12209 \fp_standardise:NNNN
12210 \l_fp_output_sign_int
12211 \l_fp_output_integer_int
12212 \l_fp_output_decimal_int
12213 \l_fp_output_exponent_int
12214 \tl_new:c { c_fp_tan ( \l_fp_arg_tl ) _fp }
12215 \tl_gset:cx { c_fp_tan ( \l_fp_arg_tl ) _fp }
12216 {
12217 \if_int_compare:w \l_fp_output_sign_int > \c_zero
12218 +
12219 \else:
12220 -
12221 \fi:
12222 \int_use:N \l_fp_output_integer_int
12223 .
12224 \exp_after:wN \use_none:n
12225 \int_value:w \int_eval:w
12226 \l_fp_output_decimal_int + \c_one_thousand_million
12227 e
12228 \int_use:N \l_fp_output_exponent_int

```

```

12229     }
12230 }

```

(End definition for \fp_tan:Nn and \fp_tan:cn. These functions are documented on page ??.)

203.12 Exponent and logarithm functions

\c_fp_exp_1_tl Calculation of exponentials requires a number of precomputed values: first the positive integers.

```

\c_fp_exp_2_tl
\c_fp_exp_3_tl 12231 \tl_const:cn { c_fp_exp_1_tl } { { 2 } { 718281828 } { 459045235 } { 0 } }
\c_fp_exp_4_tl 12232 \tl_const:cn { c_fp_exp_2_tl } { { 7 } { 389056098 } { 930650227 } { 0 } }
\c_fp_exp_5_tl 12233 \tl_const:cn { c_fp_exp_3_tl } { { 2 } { 008553692 } { 318766774 } { 1 } }
\c_fp_exp_6_tl 12234 \tl_const:cn { c_fp_exp_4_tl } { { 5 } { 459815003 } { 314423908 } { 1 } }
\c_fp_exp_7_tl 12235 \tl_const:cn { c_fp_exp_5_tl } { { 1 } { 484131591 } { 025766034 } { 2 } }
\c_fp_exp_8_tl 12236 \tl_const:cn { c_fp_exp_6_tl } { { 4 } { 034287934 } { 927351226 } { 2 } }
\c_fp_exp_9_tl 12237 \tl_const:cn { c_fp_exp_7_tl } { { 1 } { 096633158 } { 428458599 } { 3 } }
\c_fp_exp_10_tl 12238 \tl_const:cn { c_fp_exp_8_tl } { { 2 } { 980957987 } { 041728275 } { 3 } }
\c_fp_exp_20_tl 12239 \tl_const:cn { c_fp_exp_9_tl } { { 8 } { 103083927 } { 575384008 } { 3 } }
\c_fp_exp_30_tl 12240 \tl_const:cn { c_fp_exp_10_tl } { { 2 } { 202646579 } { 480671652 } { 4 } }
\c_fp_exp_40_tl 12241 \tl_const:cn { c_fp_exp_20_tl } { { 4 } { 851651954 } { 097902280 } { 8 } }
\c_fp_exp_50_tl 12242 \tl_const:cn { c_fp_exp_30_tl } { { 1 } { 068647458 } { 152446215 } { 13 } }
\c_fp_exp_60_tl 12243 \tl_const:cn { c_fp_exp_40_tl } { { 2 } { 353852668 } { 370199854 } { 17 } }
\c_fp_exp_70_tl 12244 \tl_const:cn { c_fp_exp_50_tl } { { 5 } { 184705528 } { 587072464 } { 21 } }
\c_fp_exp_80_tl 12245 \tl_const:cn { c_fp_exp_60_tl } { { 1 } { 142007389 } { 815684284 } { 26 } }
\c_fp_exp_90_tl 12246 \tl_const:cn { c_fp_exp_70_tl } { { 2 } { 515438670 } { 919167006 } { 30 } }
\c_fp_exp_100_tl 12247 \tl_const:cn { c_fp_exp_80_tl } { { 5 } { 540622384 } { 393510053 } { 34 } }
\c_fp_exp_200_tl 12248 \tl_const:cn { c_fp_exp_90_tl } { { 1 } { 220403294 } { 317840802 } { 39 } }
12249 \tl_const:cn { c_fp_exp_100_tl } { { 2 } { 688117141 } { 816135448 } { 43 } }
12250 \tl_const:cn { c_fp_exp_200_tl } { { 7 } { 225973768 } { 125749258 } { 86 } }

```

(End definition for \c_fp_exp_1_tl. This function is documented on page ??.)

\c_fp_exp_-1_tl Now the negative integers.

```

\c_fp_exp_-2_tl 12251 \tl_const:cn { c_fp_exp_-1_tl } { { 3 } { 678794411 } { 71442322 } { -1 } }
\c_fp_exp_-3_tl 12252 \tl_const:cn { c_fp_exp_-2_tl } { { 1 } { 353352832 } { 366132692 } { -1 } }
\c_fp_exp_-4_tl 12253 \tl_const:cn { c_fp_exp_-3_tl } { { 4 } { 978706836 } { 786394298 } { -2 } }
\c_fp_exp_-5_tl 12254 \tl_const:cn { c_fp_exp_-4_tl } { { 1 } { 831563888 } { 873418029 } { -2 } }
\c_fp_exp_-6_tl 12255 \tl_const:cn { c_fp_exp_-5_tl } { { 6 } { 737946999 } { 085467097 } { -3 } }
\c_fp_exp_-7_tl 12256 \tl_const:cn { c_fp_exp_-6_tl } { { 2 } { 478752176 } { 666358423 } { -3 } }
\c_fp_exp_-8_tl 12257 \tl_const:cn { c_fp_exp_-7_tl } { { 9 } { 118819655 } { 545162080 } { -4 } }
\c_fp_exp_-9_tl 12258 \tl_const:cn { c_fp_exp_-8_tl } { { 3 } { 354626279 } { 025118388 } { -4 } }
\c_fp_exp_-10_tl 12259 \tl_const:cn { c_fp_exp_-9_tl } { { 1 } { 234098040 } { 866795495 } { -4 } }
\c_fp_exp_-20_tl 12260 \tl_const:cn { c_fp_exp_-10_tl } { { 4 } { 539992976 } { 248451536 } { -5 } }
\c_fp_exp_-30_tl 12261 \tl_const:cn { c_fp_exp_-20_tl } { { 2 } { 061153622 } { 438557828 } { -9 } }
\c_fp_exp_-40_tl 12262 \tl_const:cn { c_fp_exp_-30_tl } { { 9 } { 357622968 } { 840174605 } { -14 } }
\c_fp_exp_-50_tl 12263 \tl_const:cn { c_fp_exp_-40_tl } { { 4 } { 248354255 } { 291588995 } { -18 } }
\c_fp_exp_-60_tl 12264 \tl_const:cn { c_fp_exp_-50_tl } { { 1 } { 928749847 } { 963917783 } { -22 } }
\c_fp_exp_-70_tl 12265 \tl_const:cn { c_fp_exp_-60_tl } { { 8 } { 756510762 } { 696520338 } { -27 } }
\c_fp_exp_-80_tl 12266 \tl_const:cn { c_fp_exp_-70_tl } { { 3 } { 975449735 } { 908646808 } { -31 } }
\c_fp_exp_-90_tl 12267 \tl_const:cn { c_fp_exp_-80_tl } { { 1 } { 804851387 } { 845415172 } { -35 } }
12268 \tl_const:cn { c_fp_exp_-90_tl } { { 8 } { 194012623 } { 990515430 } { -40 } }
\c_fp_exp_-100_tl
\c_fp_exp_-200_tl

```



```

12269 \tl_const:cn { c_fp_exp-100_tl } { { 3 } { 720075976 } { 020835963 } { -44 } }
12270 \tl_const:cn { c_fp_exp-200_tl } { { 1 } { 383896526 } { 736737530 } { -87 } }
      (End definition for \c_fp_exp-1_tl. This function is documented on page ??.)

```

<pre> \fp_exp:Nn \fp_exp:cn \fp_gexp:Nn \fp_gexp:cn \fp_exp_aux:NNn \fp_exp_internal: \fp_exp_aux: \fp_exp_integer: \fp_exp_integer_tens: \fp_exp_integer_units: \fp_exp_integer_const:n \fp_exp_integer_const:nnnn \fp_exp_decimal: \fp_exp_Taylor: \fp_exp_const:Nx \fp_exp_const:cx </pre>	<p>The calculation of an exponent starts off starts in much the same way as the trigonometric functions: normalise the input, look for a pre-defined value and if one is not found hand off to the real workhorse function. The test for a definition of the result is used so that overflows do not result in any outcome being defined.</p> <pre> 12271 \cs_new_protected_nopar:Npn \fp_exp:Nn { \fp_exp_aux:NNn \tl_set:Nn } 12272 \cs_new_protected_nopar:Npn \fp_gexp:Nn { \fp_exp_aux:NNn \tl_gset:Nn } 12273 \cs_generate_variant:Nn \fp_exp:Nn { c } 12274 \cs_generate_variant:Nn \fp_gexp:Nn { c } 12275 \cs_new_protected:Npn \fp_exp_aux:NNn #1#2#3 12276 { 12277 \group_begin: 12278 \fp_split:Nn a {#3} 12279 \fp_standardise:NNNN 12280 \l_fp_input_a_sign_int 12281 \l_fp_input_a_integer_int 12282 \l_fp_input_a_decimal_int 12283 \l_fp_input_a_exponent_int 12284 \l_fp_input_a_extended_int \c_zero 12285 \tl_set:Nx \l_fp_arg_tl 12286 { 12287 \if_int_compare:w \l_fp_input_a_sign_int < \c_zero 12288 - 12289 \else: 12290 + 12291 \fi: 12292 \int_use:N \l_fp_input_a_integer_int 12293 . 12294 \exp_after:wN \use_none:n 12295 \int_value:w \int_eval:w 12296 \l_fp_input_a_decimal_int + \c_one_thousand_million 12297 e 12298 \int_use:N \l_fp_input_a_exponent_int 12299 } 12300 \if_cs_exist:w c_fp_exp (\l_fp_arg_tl) _fp \cs_end: 12301 \else: 12302 \exp_after:wN \fp_exp_internal: 12303 \fi: 12304 \cs_set_protected_nopar:Npx \fp_tmp:w 12305 { 12306 \group_end: 12307 #1 \exp_not:N #2 12308 { 12309 \if_cs_exist:w c_fp_exp (\l_fp_arg_tl) _fp 12310 \cs_end: 12311 \use:c { c_fp_exp (\l_fp_arg_tl) _fp } </pre>
---	--

```

12312         \else:
12313             \c_zero_fp
12314         \fi:
12315     }
12316 }
12317 \fp_tmp:w
12318 }

The first real step is to convert the input into a fixed-point representation for further
calculation: anything which is dropped here as too small would not influence the output
in any case. There are a couple of overflow tests: the maximum

12319 \cs_new_protected_nopar:Npn \fp_exp_internal:
12320 {
12321     \if_int_compare:w \l_fp_input_a_exponent_int < \c_three
12322         \fp_extended_normalise:
12323         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12324             \if_int_compare:w \l_fp_input_a_integer_int < 230 \scan_stop:
12325                 \exp_after:wN \exp_after:wN \exp_after:wN
12326                 \exp_after:wN \exp_after:wN \exp_after:wN
12327                 \exp_after:wN \fp_exp_aux:
12328             \else:
12329                 \exp_after:wN \exp_after:wN \exp_after:wN
12330                 \exp_after:wN \exp_after:wN \exp_after:wN
12331                 \exp_after:wN \fp_exp_overflow_msg:
12332             \fi:
12333         \else:
12334             \if_int_compare:w \l_fp_input_a_integer_int < 230 \scan_stop:
12335                 \exp_after:wN \exp_after:wN \exp_after:wN
12336                 \exp_after:wN \exp_after:wN \exp_after:wN
12337                 \exp_after:wN \fp_exp_aux:
12338             \else:
12339                 \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12340                 { \c_zero_fp }
12341             \fi:
12342         \fi:
12343     \else:
12344         \exp_after:wN \fp_exp_overflow_msg:
12345     \fi:
12346 }

```

The main algorithm makes use of the fact that

$$e^{nmp.q} = e^n e^m e^p e^{0.q}$$

and that there is a Taylor series that can be used to calculate $e^{0.q}$. Thus the approach needed is in three parts. First, the exponent of the integer part of the input is found using the pre-calculated constants. Second, the Taylor series is used to find the exponent for the decimal part of the input. Finally, the two parts are multiplied together to give the result. As the normalisation code will already have dealt with any overflowing values, there are no further checks needed.

```

12347 \cs_new_protected_nopar:Npn \fp_exp_aux:
12348 {
12349   \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
12350     \exp_after:wN \fp_exp_integer:
12351   \else:
12352     \l_fp_output_integer_int \c_one
12353     \l_fp_output_decimal_int \c_zero
12354     \l_fp_output_extended_int \c_zero
12355     \l_fp_output_exponent_int \c_zero
12356     \exp_after:wN \fp_exp_decimal:
12357   \fi:
12358 }

```

The integer part calculation starts with the hundreds. This is set up such that very large negative numbers can short-cut the entire procedure and simply return zero. In other cases, the code either recovers the exponent of the hundreds value or sets the appropriate storage to one (so that multiplication works correctly).

```

12359 \cs_new_protected_nopar:Npn \fp_exp_integer:
12360 {
12361   \if_int_compare:w \l_fp_input_a_integer_int < \c_one_hundred
12362     \l_fp_exp_integer_int \c_one
12363     \l_fp_exp_decimal_int \c_zero
12364     \l_fp_exp_extended_int \c_zero
12365     \l_fp_exp_exponent_int \c_zero
12366     \exp_after:wN \fp_exp_integer_tens:
12367   \else:
12368     \tl_set:Nx \l_fp_tmp_tl
12369     {
12370       \exp_after:wN \use_i:nnn
12371       \int_use:N \l_fp_input_a_integer_int
12372     }
12373     \l_fp_input_a_integer_int
12374     \int_eval:w
12375     \l_fp_input_a_integer_int - \l_fp_tmp_tl 00
12376     \int_eval_end:
12377     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12378       \if_int_compare:w \l_fp_output_integer_int > 200 \scan_stop:
12379       \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12380       { \c_zero_fp }
12381     \else:
12382       \fp_exp_integer_const:n { - \l_fp_tmp_tl 00 }
12383       \exp_after:wN \exp_after:wN \exp_after:wN
12384       \exp_after:wN \exp_after:wN \exp_after:wN
12385       \exp_after:wN \fp_exp_integer_tens:
12386     \fi:
12387   \else:
12388     \fp_exp_integer_const:n { \l_fp_tmp_tl 00 }
12389     \exp_after:wN \exp_after:wN \exp_after:wN
12390     \exp_after:wN \fp_exp_integer_tens:
12391   \fi:

```

```

12392     \fi:
12393 }

```

The tens and units parts are handled in a similar way, with a multiplication step to build up the final value. That also includes a correction step to avoid an overflow of the integer part.

```

12394 \cs_new_protected_nopar:Npn \fp_exp_integer_tens:
12395 {
12396   \l_fp_output_integer_int \l_fp_exp_integer_int
12397   \l_fp_output_decimal_int \l_fp_exp_decimal_int
12398   \l_fp_output_extended_int \l_fp_exp_extended_int
12399   \l_fp_output_exponent_int \l_fp_exp_exponent_int
12400   \if_int_compare:w \l_fp_input_a_integer_int > \c_nine
12401     \tl_set:Nx \l_fp_tmp_tl
12402     {
12403       \exp_after:wN \use_i:nn
12404       \int_use:N \l_fp_input_a_integer_int
12405     }
12406     \l_fp_input_a_integer_int
12407     \int_eval:w
12408     \l_fp_input_a_integer_int - \l_fp_tmp_tl 0
12409     \int_eval_end:
12410     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12411       \fp_exp_integer_const:n { \l_fp_tmp_tl 0 }
12412     \else:
12413       \fp_exp_integer_const:n { - \l_fp_tmp_tl 0 }
12414     \fi:
12415     \fp_mul:NNNNNNNNN
12416     \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12417     \l_fp_output_integer_int \l_fp_output_decimal_int
12418     \l_fp_output_extended_int
12419     \l_fp_output_integer_int \l_fp_output_decimal_int
12420     \l_fp_output_extended_int
12421     \tex_advance:D \l_fp_output_exponent_int \l_fp_exp_exponent_int
12422     \fp_extended_normalise_output:
12423   \fi:
12424   \fp_exp_integer_units:
12425 }
12426 \cs_new_protected_nopar:Npn \fp_exp_integer_units:
12427 {
12428   \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
12429     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12430       \fp_exp_integer_const:n { \int_use:N \l_fp_input_a_integer_int }
12431     \else:
12432       \fp_exp_integer_const:n
12433       { - \int_use:N \l_fp_input_a_integer_int }
12434     \fi:
12435     \fp_mul:NNNNNNNNN
12436     \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12437     \l_fp_output_integer_int \l_fp_output_decimal_int

```

```

12438         \l_fp_output_extended_int
12439         \l_fp_output_integer_int \l_fp_output_decimal_int
12440         \l_fp_output_extended_int
12441         \tex_advance:D \l_fp_output_exponent_int \l_fp_exp_exponent_int
12442         \fp_extended_normalise_output:
12443     \fi:
12444     \fp_exp_decimal:
12445 }

```

Recovery of the stored constant values into the separate registers is done with a simple expansion then assignment.

```

12446 \cs_new_protected:Npn \fp_exp_integer_const:n #1
12447 {
12448     \exp_after:wN \exp_after:wN \exp_after:wN
12449     \fp_exp_integer_const:nnnn
12450     \cs:w c_fp_exp_ #1 _tl \cs_end:
12451 }
12452 \cs_new_protected:Npn \fp_exp_integer_const:nnnn #1#2#3#4
12453 {
12454     \l_fp_exp_integer_int #1 \scan_stop:
12455     \l_fp_exp_decimal_int #2 \scan_stop:
12456     \l_fp_exp_extended_int #3 \scan_stop:
12457     \l_fp_exp_exponent_int #4 \scan_stop:
12458 }

```

Finding the exponential for the decimal part of the number requires a Taylor series calculation. The set up is done here with the loop itself a separate function. Once the decimal part is available this is multiplied by the integer part already worked out to give the final result.

```

12459 \cs_new_protected_nopar:Npn \fp_exp_decimal:
12460 {
12461     \if_int_compare:w \l_fp_input_a_decimal_int > \c_zero
12462     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12463         \l_fp_exp_integer_int \c_one
12464         \l_fp_exp_decimal_int \l_fp_input_a_decimal_int
12465         \l_fp_exp_extended_int \l_fp_input_a_extended_int
12466     \else:
12467         \l_fp_exp_integer_int \c_zero
12468         \if_int_compare:w \l_fp_exp_extended_int = \c_zero
12469             \l_fp_exp_decimal_int
12470             \int_eval:w
12471             \c_one_thousand_million - \l_fp_input_a_decimal_int
12472             \int_eval_end:
12473             \l_fp_exp_extended_int \c_zero
12474         \else:
12475             \l_fp_exp_decimal_int
12476             \int_eval:w
12477             999999999 - \l_fp_input_a_decimal_int
12478             \scan_stop:
12479             \l_fp_exp_extended_int

```

```

12480         \int_eval:w
12481         \c_one_thousand_million - \l_fp_input_a_extended_int
12482         \int_eval_end:
12483     \fi:
12484 \fi:
12485 \l_fp_input_b_sign_int      \l_fp_input_a_sign_int
12486 \l_fp_input_b_decimal_int  \l_fp_input_a_decimal_int
12487 \l_fp_input_b_extended_int \l_fp_input_a_extended_int
12488 \l_fp_count_int \c_one
12489 \fp_exp_Taylor:
12490 \fp_mul:NNNNNNNNN
12491     \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12492     \l_fp_output_integer_int \l_fp_output_decimal_int
12493     \l_fp_output_extended_int
12494     \l_fp_output_integer_int \l_fp_output_decimal_int
12495     \l_fp_output_extended_int
12496 \fi:
12497 \if_int_compare:w \l_fp_output_extended_int < \c_five_hundred_million
12498 \else:
12499     \tex_advance:D \l_fp_output_decimal_int \c_one
12500     \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12501     \else:
12502         \l_fp_output_decimal_int \c_zero
12503         \tex_advance:D \l_fp_output_integer_int \c_one
12504     \fi:
12505 \fi:
12506 \fp_standardise:NNNN
12507     \l_fp_output_sign_int
12508     \l_fp_output_integer_int
12509     \l_fp_output_decimal_int
12510     \l_fp_output_exponent_int
12511 \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12512 {
12513     +
12514     \int_use:N \l_fp_output_integer_int
12515     .
12516     \exp_after:wN \use_none:n
12517     \int_value:w \int_eval:w
12518         \l_fp_output_decimal_int + \c_one_thousand_million
12519     e
12520     \int_use:N \l_fp_output_exponent_int
12521 }
12522 }

```

The Taylor series for $\exp(x)$ is

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

which converges for $-1 < x < 1$. The code above sets up the x part, leaving the loop to multiply the running value by x/n and add it onto the sum. The way that this is done

is that the running total is stored in the `exp` set of registers, while the current item is stored as `input_b`.

```

12523 \cs_new_protected_nopar:Npn \fp_exp_Taylor:
12524 {
12525   \tex_advance:D \l_fp_count_int \c_one
12526   \tex_multiply:D \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12527   \fp_mul:NNNNNN
12528   \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12529   \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12530   \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12531   \fp_div_integer:NNNNN
12532   \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12533   \l_fp_count_int
12534   \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12535   \if_int_compare:w
12536     \int_eval:w
12537       \l_fp_input_b_decimal_int + \l_fp_input_b_extended_int
12538       > \c_zero
12539   \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
12540     \tex_advance:D \l_fp_exp_decimal_int \l_fp_input_b_decimal_int
12541     \tex_advance:D \l_fp_exp_extended_int
12542       \l_fp_input_b_extended_int
12543     \if_int_compare:w \l_fp_exp_extended_int < \c_one_thousand_million
12544   \else:
12545     \tex_advance:D \l_fp_exp_decimal_int \c_one
12546     \tex_advance:D \l_fp_exp_extended_int
12547       -\c_one_thousand_million
12548   \fi:
12549   \if_int_compare:w \l_fp_exp_decimal_int < \c_one_thousand_million
12550   \else:
12551     \tex_advance:D \l_fp_exp_integer_int \c_one
12552     \tex_advance:D \l_fp_exp_decimal_int
12553       -\c_one_thousand_million
12554   \fi:
12555   \else:
12556     \tex_advance:D \l_fp_exp_decimal_int -\l_fp_input_b_decimal_int
12557     \tex_advance:D \l_fp_exp_extended_int
12558       -\l_fp_input_a_extended_int
12559     \if_int_compare:w \l_fp_exp_extended_int < \c_zero
12560     \tex_advance:D \l_fp_exp_decimal_int \c_minus_one
12561     \tex_advance:D \l_fp_exp_extended_int \c_one_thousand_million
12562   \fi:
12563     \if_int_compare:w \l_fp_exp_decimal_int < \c_zero
12564     \tex_advance:D \l_fp_exp_integer_int \c_minus_one
12565     \tex_advance:D \l_fp_exp_decimal_int \c_one_thousand_million
12566   \fi:
12567   \fi:
12568   \exp_after:wN \fp_exp_Taylor:
12569 \fi:

```

This is set up as a function so that the power code can redirect the effect.

Constants for working out logarithms: first those for the powers of ten.

The smaller set for powers of two.

The approach for logarithms is again based on a mix of tables and Taylor series. Here, the initial validation is a bit easier and so it is set up earlier, meaning less need to escape later on.

536


```

12606         > \c_zero
12607         \exp_after:wN \exp_after:wN \exp_after:wN \fp_ln_aux:
12608     \else:
12609         \cs_set_protected:Npx \fp_tmp:w ##1##2
12610         {
12611             \group_end:
12612             ##1 \exp_not:N ##2 { \c_zero_fp }
12613         }
12614         \exp_after:wN \exp_after:wN \exp_after:wN \fp_ln_error_msg:
12615     \fi:
12616 \else:
12617     \cs_set_protected:Npx \fp_tmp:w ##1##2
12618     {
12619         \group_end:
12620         ##1 \exp_not:N ##2 { \c_zero_fp }
12621     }
12622     \exp_after:wN \fp_ln_error_msg:
12623 \fi:
12624 \fp_tmp:w #1 #2
12625 }

```

As the input at this stage meets the validity criteria above, the argument can now be saved for further processing. There is no need to look at the sign of the input as it must be positive. The function here simply sets up to either do the full calculation or recover the stored value, as appropriate.

```

12626 \cs_new_protected_nopar:Npn \fp_ln_aux:
12627 {
12628     \tl_set:Nx \l_fp_arg_tl
12629     {
12630         +
12631         \int_use:N \l_fp_input_a_integer_int
12632         .
12633         \exp_after:wN \use_none:n
12634         \int_value:w \int_eval:w
12635         \l_fp_input_a_decimal_int + \c_one_thousand_million
12636         e
12637         \int_use:N \l_fp_input_a_exponent_int
12638     }
12639     \if_cs_exist:w c_fp_ln ( \l_fp_arg_tl ) _fp \cs_end:
12640 \else:
12641     \exp_after:wN \fp_ln_exponent:
12642 \fi:
12643 \cs_set_protected:Npx \fp_tmp:w ##1##2
12644 {
12645     \group_end:
12646     ##1 \exp_not:N ##2
12647     { \use:c { c_fp_ln ( \l_fp_arg_tl ) _fp } }
12648 }
12649 }

```

The main algorithm here uses the fact the logarithm can be divided up, first taking out

the powers of ten, then powers of two and finally using a Taylor series for the remainder.

$$\ln(10^n \times 2^m \times x) = \ln(10^n) + \ln(2^m) + \ln(x)$$

The second point to remember is that

$$\ln(x^{-1}) = -\ln(x)$$

which means that for the powers of 10 and 2 constants are only needed for positive powers.

The first step is to set up the sign for the output functions and work out the powers of ten in the exponent. First the larger powers are sorted out. The values for the constants are the same as those for the smaller ones, just with a shift in the exponent.

```

12650 \cs_new_protected_nopar:Npn \fp_ln_exponent:
12651 {
12652   \fp_ln_internal:
12653   \if_int_compare:w \l_fp_output_extended_int < \c_five_hundred_million
12654   \else:
12655     \tex_advance:D \l_fp_output_decimal_int \c_one
12656     \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12657     \else:
12658       \l_fp_output_decimal_int \c_zero
12659       \tex_advance:D \l_fp_output_integer_int \c_one
12660     \fi:
12661   \fi:
12662   \fp_standardise:NNNN
12663   \l_fp_output_sign_int
12664   \l_fp_output_integer_int
12665   \l_fp_output_decimal_int
12666   \l_fp_output_exponent_int
12667   \tl_const:cx { c_fp_ln ( \l_fp_arg_tl ) _fp }
12668   {
12669     \if_int_compare:w \l_fp_output_sign_int > \c_zero
12670     +
12671     \else:
12672     -
12673     \fi:
12674     \int_use:N \l_fp_output_integer_int
12675     .
12676     \exp_after:wN \use_none:n
12677     \int_value:w \int_eval:w
12678     \l_fp_output_decimal_int + \c_one_thousand_million
12679     \scan_stop:
12680     e
12681     \int_use:N \l_fp_output_exponent_int
12682   }
12683 }
12684 \cs_new_protected_nopar:Npn \fp_ln_internal:
12685 {
12686   \if_int_compare:w \l_fp_input_a_exponent_int < \c_zero

```

```

12687     \l_fp_input_a_exponent_int -\l_fp_input_a_exponent_int
12688     \l_fp_output_sign_int \c_minus_one
12689   \else:
12690     \l_fp_output_sign_int \c_one
12691   \fi:
12692   \if_int_compare:w \l_fp_input_a_exponent_int > \c_nine
12693     \exp_after:wN \fp_ln_exponent_tens:NN
12694     \int_use:N \l_fp_input_a_exponent_int
12695   \else:
12696     \l_fp_output_integer_int \c_zero
12697     \l_fp_output_decimal_int \c_zero
12698     \l_fp_output_extended_int \c_zero
12699     \l_fp_output_exponent_int \c_zero
12700   \fi:
12701   \fp_ln_exponent_units:
12702 }
12703 \cs_new_protected:Npn \fp_ln_exponent_tens:NN #1 #2
12704 {
12705   \l_fp_input_a_exponent_int #2 \scan_stop:
12706   \fp_ln_const:nn { 10 } { #1 }
12707   \tex_advance:D \l_fp_exp_exponent_int \c_one
12708   \l_fp_output_integer_int \l_fp_exp_integer_int
12709   \l_fp_output_decimal_int \l_fp_exp_decimal_int
12710   \l_fp_output_extended_int \l_fp_exp_extended_int
12711   \l_fp_output_exponent_int \l_fp_exp_exponent_int
12712 }

```

Next the smaller powers of ten, which will need to be combined with the above: always an additive process.

```

12713 \cs_new_protected_nopar:Npn \fp_ln_exponent_units:
12714 {
12715   \if_int_compare:w \l_fp_input_a_exponent_int > \c_zero
12716     \fp_ln_const:nn { 10 } { \int_use:N \l_fp_input_a_exponent_int }
12717     \fp_ln_normalise:
12718     \fp_add:NNNNNNNNN
12719     \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12720     \l_fp_output_integer_int \l_fp_output_decimal_int
12721     \l_fp_output_extended_int
12722     \l_fp_output_integer_int \l_fp_output_decimal_int
12723     \l_fp_output_extended_int
12724   \fi:
12725   \fp_ln_mantissa:
12726 }

```

The smaller table-based parts may need to be exponent shifted so that they stay in line with the larger parts. This is similar to the approach in other places, but here there is a need to watch the extended part of the number. The only case where the new exponent is larger than the old is if there was no previous part. Then simply set the exponent.

```

12727 \cs_new_protected_nopar:Npn \fp_ln_normalise:
12728 {

```

```

12729 \if_int_compare:w \l_fp_exp_exponent_int < \l_fp_output_exponent_int
12730 \tex_advance:D \l_fp_exp_decimal_int \c_one_thousand_million
12731 \exp_after:wN \use_i:nn \exp_after:wN
12732 \fp_ln_normalise_aux:NNNNNNNNN
12733 \int_use:N \l_fp_exp_decimal_int
12734 \exp_after:wN \fp_ln_normalise:
12735 \else:
12736 \l_fp_output_exponent_int \l_fp_exp_exponent_int
12737 \fi:
12738 }
12739 \cs_new_protected:Npn \fp_ln_normalise_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
12740 {
12741 \if_int_compare:w \l_fp_exp_integer_int = \c_zero
12742 \l_fp_exp_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
12743 \else:
12744 \tl_set:Nx \l_fp_tmp_tl
12745 {
12746 \int_use:N \l_fp_exp_integer_int
12747 #1#2#3#4#5#6#7#8
12748 }
12749 \l_fp_exp_integer_int \c_zero
12750 \l_fp_exp_decimal_int \l_fp_tmp_tl \scan_stop:
12751 \fi:
12752 \tex_divide:D \l_fp_exp_extended_int \c_ten
12753 \tl_set:Nx \l_fp_tmp_tl
12754 {
12755 #9
12756 \int_use:N \l_fp_exp_extended_int
12757 }
12758 \l_fp_exp_extended_int \l_fp_tmp_tl \scan_stop:
12759 \tex_advance:D \l_fp_exp_exponent_int \c_one
12760 }

```

The next phase is to decompose the mantissa by division by two to leave a value which is in the range $1 \leq x < 2$. The sum of the two powers needs to take account of the sign of the output: if it is negative then the result gets *smaller* as the mantissa gets *bigger*.

```

12761 \cs_new_protected_nopar:Npn \fp_ln_mantissa:
12762 {
12763 \l_fp_count_int \c_zero
12764 \l_fp_input_a_extended_int \c_zero
12765 \fp_ln_mantissa_aux:
12766 \if_int_compare:w \l_fp_count_int > \c_zero
12767 \fp_ln_const:nn { 2 } { \int_use:N \l_fp_count_int }
12768 \fp_ln_normalise:
12769 \if_int_compare:w \l_fp_output_sign_int > \c_zero
12770 \exp_after:wN \fp_add:NNNNNNNNN
12771 \else:
12772 \exp_after:wN \fp_sub:NNNNNNNNN
12773 \fi:
12774 \l_fp_output_integer_int \l_fp_output_decimal_int

```

```

12775         \l_fp_output_extended_int
12776         \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12777         \l_fp_output_integer_int \l_fp_output_decimal_int
12778         \l_fp_output_extended_int
12779     \fi:
12780     \if_int_compare:w
12781         \int_eval:w
12782         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int > \c_one
12783         \exp_after:wN \fp_ln_Taylor:
12784     \fi:
12785 }
12786 \cs_new_protected_nopar:Npn \fp_ln_mantissa_aux:
12787 {
12788     \if_int_compare:w \l_fp_input_a_integer_int > \c_one
12789         \tex_advance:D \l_fp_count_int \c_one
12790         \fp_ln_mantissa_divide_two:
12791         \exp_after:wN \fp_ln_mantissa_aux:
12792     \fi:
12793 }

```

A fast one-shot division by two.

```

12794 \cs_new_protected_nopar:Npn \fp_ln_mantissa_divide_two:
12795 {
12796     \if_int_odd:w \l_fp_input_a_decimal_int
12797         \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
12798     \fi:
12799     \if_int_odd:w \l_fp_input_a_integer_int
12800         \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
12801     \fi:
12802     \tex_divide:D \l_fp_input_a_integer_int \c_two
12803     \tex_divide:D \l_fp_input_a_decimal_int \c_two
12804     \tex_divide:D \l_fp_input_a_extended_int \c_two
12805 }

```

Recovering constants makes use of the same auxiliary code as for exponents.

```

12806 \cs_new_protected:Npn \fp_ln_const:nn #1#2
12807 {
12808     \exp_after:wN \exp_after:wN \exp_after:wN
12809     \fp_exp_integer_const:nnnn
12810     \cs:w c_fp_ln_ #1 _ #2 _tl \cs_end:
12811 }

```

The Taylor series for the logarithm function is best implemented using the identity

$$\ln(x) = \ln\left(\frac{y+1}{y-1}\right)$$

with

$$y = \frac{x-1}{x+1}$$

This leads to the series

$$\ln(x) = 2y \left(1 + y^2 \left(\frac{1}{3} + y^2 \left(\frac{1}{5} + y^2 \left(\frac{1}{7} + y^2 \left(\frac{1}{9} + \cdots \right) \right) \right) \right) \right)$$

This expansion has the advantage that a lot of the work can be loaded up early by finding y^2 before the loop itself starts. (In practice, the implementation does the multiplication by two at the end of the loop, and expands out the brackets as this is an overall more efficient approach.)

At the implementation level, the code starts by calculating y and storing that in input **a** (which is no longer needed for other purposes). That is done using the full division system avoiding the parsing step. The value is then switched to a fixed-point representation. There is then some shuffling to get all of the working space set up. At this stage, a lot of registers are in use and so the Taylor series is calculated within a group so that the **output** variables can be used to hold the result. The value of y^2 is held in input **b** (there are a few assignments saved by choosing this over **a**), while input **a** is used for the “loop value”.

```

12812 \cs_new_protected_nopar:Npn \fp_ln_Taylor:
12813 {
12814   \group_begin:
12815     \l_fp_input_a_integer_int \c_zero
12816     \l_fp_input_a_exponent_int \c_zero
12817     \l_fp_input_b_integer_int \c_two
12818     \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12819     \l_fp_input_b_exponent_int \c_zero
12820     \fp_div_internal:
12821     \fp_ln_fixed:
12822     \l_fp_input_a_integer_int \l_fp_output_integer_int
12823     \l_fp_input_a_decimal_int \l_fp_output_decimal_int
12824     \l_fp_input_a_extended_int \c_zero
12825     \l_fp_input_a_exponent_int \l_fp_output_exponent_int
12826     \l_fp_output_decimal_int \c_zero %^^A Bug?
12827     \l_fp_output_decimal_int \l_fp_input_a_decimal_int
12828     \l_fp_output_extended_int \l_fp_input_a_extended_int
12829     \fp_mul:NNNNNN
12830     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12831     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12832     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12833     \l_fp_count_int \c_one
12834     \fp_ln_Taylor_aux:
12835     \cs_set_protected_nopar:Npx \fp_tmp:w
12836     {
12837       \group_end:
12838       \l_fp_exp_integer_int \c_zero
12839       \exp_not:N \l_fp_exp_decimal_int
12840       \int_use:N \l_fp_output_decimal_int \scan_stop:
12841       \exp_not:N \l_fp_exp_extended_int
12842       \int_use:N \l_fp_output_extended_int \scan_stop:
12843       \exp_not:N \l_fp_exp_exponent_int

```

```

12844         \int_use:N \l_fp_output_exponent_int \scan_stop:
12845     }
12846     \fp_tmp:w

```

After the loop part of the Taylor series, the factor of 2 needs to be included. The total for the result can then be constructed.

```

12847     \tex_advance:D \l_fp_exp_decimal_int \l_fp_exp_decimal_int
12848     \if_int_compare:w \l_fp_exp_extended_int < \c_five_hundred_million
12849     \else:
12850         \tex_advance:D \l_fp_exp_extended_int -\c_five_hundred_million
12851         \tex_advance:D \l_fp_exp_decimal_int \c_one
12852     \fi:
12853     \tex_advance:D \l_fp_exp_extended_int \l_fp_exp_extended_int
12854     \fp_ln_normalise:
12855     \if_int_compare:w \l_fp_output_sign_int > \c_zero
12856         \exp_after:wN \fp_add:NNNNNNNNN
12857     \else:
12858         \exp_after:wN \fp_sub:NNNNNNNNN
12859     \fi:
12860     \l_fp_output_integer_int \l_fp_output_decimal_int
12861     \l_fp_output_extended_int
12862     \c_zero \l_fp_exp_decimal_int \l_fp_exp_extended_int
12863     \l_fp_output_integer_int \l_fp_output_decimal_int
12864     \l_fp_output_extended_int
12865 }

```

The usual shifts to move to fixed-point working. This is done using the output registers as this saves a reassignment here.

```

12866 \cs_new_protected_nopar:Npn \fp_ln_fixed:
12867 {
12868     \if_int_compare:w \l_fp_output_exponent_int < \c_zero
12869         \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
12870         \exp_after:wN \use_i:nn \exp_after:wN
12871         \fp_ln_fixed_aux:NNNNNNNNN
12872         \int_use:N \l_fp_output_decimal_int
12873         \exp_after:wN \fp_ln_fixed:
12874     \fi:
12875 }
12876 \cs_new_protected:Npn \fp_ln_fixed_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
12877 {
12878     \if_int_compare:w \l_fp_output_integer_int = \c_zero
12879         \l_fp_output_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
12880     \else:
12881         \tl_set:Nx \l_fp_tmp_tl
12882         {
12883             \int_use:N \l_fp_output_integer_int
12884             #1#2#3#4#5#6#7#8
12885         }
12886         \l_fp_output_integer_int \c_zero
12887         \l_fp_output_decimal_int \l_fp_tmp_tl \scan_stop:

```

```

12888 \fi:
12889 \tex_advance:D \l_fp_output_exponent_int \c_one
12890 }

```

The main loop for the Taylor series: unlike some of the other similar functions, the result here is not the final value and is therefore subject to further manipulation outside of the loop.

```

12891 \cs_new_protected_nopar:Npn \fp_ln_Taylor_aux:
12892 {
12893   \tex_advance:D \l_fp_count_int \c_two
12894   \fp_mul:NNNNNN
12895   \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12896   \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12897   \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12898   \if_int_compare:w
12899     \int_eval:w
12900     \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
12901     > \c_zero
12902   \fp_div_integer:NNNNN
12903   \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12904   \l_fp_count_int
12905   \l_fp_exp_decimal_int \l_fp_exp_extended_int
12906   \tex_advance:D \l_fp_output_decimal_int \l_fp_exp_decimal_int
12907   \tex_advance:D \l_fp_output_extended_int \l_fp_exp_extended_int
12908   \if_int_compare:w \l_fp_output_extended_int < \c_one_thousand_million
12909   \else:
12910     \tex_advance:D \l_fp_output_decimal_int \c_one
12911     \tex_advance:D \l_fp_output_extended_int
12912     -\c_one_thousand_million
12913   \fi:
12914   \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12915   \else:
12916     \tex_advance:D \l_fp_output_integer_int \c_one
12917     \tex_advance:D \l_fp_output_decimal_int
12918     -\c_one_thousand_million
12919   \fi:
12920   \exp_after:wN \fp_ln_Taylor_aux:
12921   \fi:
12922 }

```

(End definition for \fp_ln:Nn and \fp_ln:cn. These functions are documented on page ??.)

<pre> \fp_pow:Nn \fp_pow:cn \fp_gpow:Nn \fp_gpow:cn \fp_pow_aux:NNn \fp_pow_aux_i: \fp_pow_positive: \fp_pow_negative: \fp_pow_aux_ii: \fp_pow_aux_iii: \fp_pow_aux_iv: </pre>	<p>The approach used for working out powers is to first filter out the various special cases and then do most of the work using the logarithm and exponent functions. The two storage areas are used in the reverse of the ‘natural’ logic as this avoids some re-assignment in the sanity checking code.</p> <pre> 12923 \cs_new_protected_nopar:Npn \fp_pow:Nn { \fp_pow_aux:NNn \tl_set:Nn } 12924 \cs_new_protected_nopar:Npn \fp_gpow:Nn { \fp_pow_aux:NNn \tl_gset:Nn } 12925 \cs_generate_variant:Nn \fp_pow:Nn { c } 12926 \cs_generate_variant:Nn \fp_gpow:Nn { c } </pre>
--	---


```

12927 \cs_new_protected:Npn \fp_pow_aux:NNn #1#2#3
12928 {
12929   \group_begin:
12930   \fp_read:N #2
12931   \l_fp_input_b_sign_int    \l_fp_input_a_sign_int
12932   \l_fp_input_b_integer_int \l_fp_input_a_integer_int
12933   \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12934   \l_fp_input_b_exponent_int \l_fp_input_a_exponent_int
12935   \fp_split:Nn a {#3}
12936   \fp_standardise:NNNN
12937   \l_fp_input_a_sign_int
12938   \l_fp_input_a_integer_int
12939   \l_fp_input_a_decimal_int
12940   \l_fp_input_a_exponent_int
12941   \if_int_compare:w
12942     \int_eval:w
12943     \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
12944     = \c_zero
12945     \if_int_compare:w
12946       \int_eval:w
12947       \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
12948       = \c_zero
12949       \cs_set_protected:Npx \fp_tmp:w ##1##2
12950       {
12951         \group_end:
12952         ##1 ##2 { \c_undefined_fp }
12953       }
12954     \else:
12955       \cs_set_protected:Npx \fp_tmp:w ##1##2
12956       {
12957         \group_end:
12958         ##1 ##2 { \c_zero_fp }
12959       }
12960     \fi:
12961   \else:
12962     \if_int_compare:w
12963       \int_eval:w
12964       \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
12965       = \c_zero
12966       \cs_set_protected:Npx \fp_tmp:w ##1##2
12967       {
12968         \group_end:
12969         ##1 ##2 { \c_one_fp }
12970       }
12971     \else:
12972       \exp_after:wN \exp_after:wN \exp_after:wN
12973       \fp_pow_aux_i:
12974     \fi:
12975   \fi:
12976   \fp_tmp:w #1 #2

```

```
12977 }
```

Simply using the logarithm function directly will fail when negative numbers are raised to integer powers, which is a mathematically valid operation. So there are some more tests to make, after forcing the power into an integer and decimal parts, if necessary.

```
12978 \cs_new_protected_nopar:Npn \fp_pow_aux_i:
12979 {
12980   \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
12981     \tl_set:Nn \l_fp_sign_tl { + }
12982     \exp_after:wN \fp_pow_aux_ii:
12983   \else:
12984     \l_fp_input_a_extended_int \c_zero
12985     \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
12986       \group_begin:
12987       \fp_extended_normalise:
12988       \if_int_compare:w
12989         \int_eval:w
12990         \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
12991         = \c_zero
12992       \group_end:
12993       \tl_set:Nn \l_fp_sign_tl { - }
12994       \exp_after:wN \exp_after:wN \exp_after:wN
12995       \exp_after:wN \exp_after:wN \exp_after:wN
12996       \exp_after:wN \fp_pow_aux_ii:
12997     \else:
12998       \group_end:
12999       \cs_set_protected:Npx \fp_tmp:w ##1##2
13000       {
13001         \group_end:
13002         ##1 ##2 { \c_undefined_fp }
13003       }
13004     \fi:
13005   \else:
13006     \cs_set_protected:Npx \fp_tmp:w ##1##2
13007     {
13008       \group_end:
13009       ##1 ##2 { \c_undefined_fp }
13010     }
13011   \fi:
13012 \fi:
13013 }
```

The approach used here for powers works well in most cases but gives poorer results for negative integer powers, which often have exact values. So there is some filtering to do. For negative powers where the power is small, an alternative approach is used in which the positive value is worked out and the reciprocal is then taken. The filtering is unfortunately rather long.

```
13014 \cs_new_protected_nopar:Npn \fp_pow_aux_ii:
13015 {
13016   \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
```

```

13017         \exp_after:wN \fp_pow_aux_iv:
13018     \else:
13019         \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
13020             \group_begin:
13021             \l_fp_input_a_extended_int \c_zero
13022             \fp_extended_normalise:
13023             \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
13024                 \if_int_compare:w \l_fp_input_a_integer_int > \c_ten
13025                     \group_end:
13026                     \exp_after:wN \exp_after:wN \exp_after:wN
13027                     \exp_after:wN \exp_after:wN \exp_after:wN
13028                     \exp_after:wN \exp_after:wN \exp_after:wN
13029                     \exp_after:wN \exp_after:wN \exp_after:wN
13030                     \exp_after:wN \exp_after:wN \exp_after:wN
13031                     \fp_pow_aux_iv:
13032                 \else:
13033                     \group_end:
13034                     \exp_after:wN \exp_after:wN \exp_after:wN
13035                     \exp_after:wN \exp_after:wN \exp_after:wN
13036                     \exp_after:wN \exp_after:wN \exp_after:wN
13037                     \exp_after:wN \exp_after:wN \exp_after:wN
13038                     \exp_after:wN \exp_after:wN \exp_after:wN
13039                     \exp_after:wN \fp_pow_aux_iii:
13040                 \fi:
13041             \else:
13042                 \group_end:
13043                 \exp_after:wN \exp_after:wN \exp_after:wN
13044                 \exp_after:wN \exp_after:wN \exp_after:wN
13045                 \exp_after:wN \fp_pow_aux_iv:
13046             \fi:
13047         \else:
13048             \exp_after:wN \exp_after:wN \exp_after:wN
13049             \fp_pow_aux_iv:
13050         \fi:
13051     \fi:
13052 \cs_set_protected:Npx \fp_tmp:w ##1##2
13053 {
13054     \group_end:
13055     ##1 ##2
13056     {
13057         \l_fp_sign_tl
13058         \int_use:N \l_fp_output_integer_int
13059         .
13060         \exp_after:wN \use_none:n
13061         \int_value:w \int_eval:w
13062         \l_fp_output_decimal_int + \c_one_thousand_million
13063         e
13064         \int_use:N \l_fp_output_exponent_int
13065     }
13066 }

```

```
13067 }
```

For the small negative integer powers, the calculation is done for the positive power and the reciprocal is then taken.

```
13068 \cs_new_protected_nopar:Npn \fp_pow_aux_iii:
13069 {
13070   \l_fp_input_a_sign_int \c_one
13071   \fp_pow_aux_iv:
13072   \l_fp_input_a_integer_int \c_one
13073   \l_fp_input_a_decimal_int \c_zero
13074   \l_fp_input_a_exponent_int \c_zero
13075   \l_fp_input_b_integer_int \l_fp_output_integer_int
13076   \l_fp_input_b_decimal_int \l_fp_output_decimal_int
13077   \l_fp_input_b_exponent_int \l_fp_output_exponent_int
13078   \fp_div_internal:
13079 }
```

The business end of the code starts by finding the logarithm of the given base. There is a bit of a shuffle so that this does not have to be re-parsed and so that the output ends up in the correct place. There is also a need to enable using the short-cut for a pre-calculated result. The internal part of the multiplication function can then be used to do the second part of the calculation directly. There is some more set up before doing the exponential: the idea here is to deactivate some internals so that everything works smoothly.

```
13080 \cs_new_protected_nopar:Npn \fp_pow_aux_iv:
13081 {
13082   \group_begin:
13083   \l_fp_input_a_integer_int \l_fp_input_b_integer_int
13084   \l_fp_input_a_decimal_int \l_fp_input_b_decimal_int
13085   \l_fp_input_a_exponent_int \l_fp_input_b_exponent_int
13086   \fp_ln_internal:
13087   \cs_set_protected_nopar:Npx \fp_tmp:w
13088   {
13089     \group_end:
13090     \exp_not:N \l_fp_input_b_sign_int
13091     \int_use:N \l_fp_output_sign_int \scan_stop:
13092     \exp_not:N \l_fp_input_b_integer_int
13093     \int_use:N \l_fp_output_integer_int \scan_stop:
13094     \exp_not:N \l_fp_input_b_decimal_int
13095     \int_use:N \l_fp_output_decimal_int \scan_stop:
13096     \exp_not:N \l_fp_input_b_extended_int
13097     \int_use:N \l_fp_output_extended_int \scan_stop:
13098     \exp_not:N \l_fp_input_b_exponent_int
13099     \int_use:N \l_fp_output_exponent_int \scan_stop:
13100   }
13101   \fp_tmp:w
13102   \l_fp_input_a_extended_int \c_zero
13103   \fp_mul:NNNNNNNNN
13104   \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
13105   \l_fp_input_a_extended_int
```

```

13106     \l_fp_input_b_integer_int \l_fp_input_b_decimal_int
13107     \l_fp_input_b_extended_int
13108     \l_fp_output_integer_int \l_fp_output_decimal_int
13109     \l_fp_output_extended_int
13110     \l_fp_output_exponent_int
13111     \int_eval:w
13112     \l_fp_input_a_exponent_int + \l_fp_input_b_exponent_int
13113     \scan_stop:
13114     \fp_extended_normalise_output:
13115     \tex_multiply:D \l_fp_input_a_sign_int \l_fp_input_b_sign_int
13116     \l_fp_input_a_integer_int \l_fp_output_integer_int
13117     \l_fp_input_a_decimal_int \l_fp_output_decimal_int
13118     \l_fp_input_a_extended_int \l_fp_output_extended_int
13119     \l_fp_input_a_exponent_int \l_fp_output_exponent_int
13120     \l_fp_output_integer_int \c_zero
13121     \l_fp_output_decimal_int \c_zero
13122     \l_fp_output_extended_int \c_zero
13123     \l_fp_output_exponent_int \c_zero
13124     \cs_set_eq:NN \fp_exp_const:Nx \use_none:nn
13125     \fp_exp_internal:
13126 }

```

(End definition for \fp_pow:Nn and \fp_pow:cn. These functions are documented on page ??.)

203.13 Tests for special values

`\fp_if_undefined:N` Testing for an undefined value is easy.

```

13127 \prg_new_conditional:Npnn \fp_if_undefined:N #1 { p , T , F , TF }
13128 {
13129     \if_meaning:w #1 \c_undefined_fp
13130     \prg_return_true:
13131     \else:
13132     \prg_return_false:
13133     \fi:
13134 }

```

(End definition for \fp_if_undefined:N. This function is documented on page 162.)

`\fp_if_zero:N` Testing for a zero fixed-point is also easy.

```

13135 \prg_new_conditional:Npnn \fp_if_zero:N #1 { p , T , F , TF }
13136 {
13137     \if_meaning:w #1 \c_zero_fp
13138     \prg_return_true:
13139     \else:
13140     \prg_return_false:
13141     \fi:
13142 }

```

(End definition for \fp_if_zero:N. This function is documented on page 163.)

203.14 Floating-point conditionals

The idea for the comparisons is to provide two versions: slower and faster. The lead off for both is the same: get the two numbers read and then look for a function to handle the comparison.

```

\fp_compare:nNn
\fp_compare:NNN
\fp_compare_aux:N
  \fp_compare_=:
  \fp_compare_<:
\fp_compare_<_aux:
\fp_compare_absolute_a>b:
\fp_compare_absolute_a<b:
  \fp_compare_>:
13143 \prg_new_protected_conditional:Npnn \fp_compare:nNn #1#2#3 { T , F , TF }
13144 {
13145   \group_begin:
13146   \fp_split:Nn a {#1}
13147   \fp_standardise:NNNN
13148   \l_fp_input_a_sign_int
13149   \l_fp_input_a_integer_int
13150   \l_fp_input_a_decimal_int
13151   \l_fp_input_a_exponent_int
13152   \fp_split:Nn b {#3}
13153   \fp_standardise:NNNN
13154   \l_fp_input_b_sign_int
13155   \l_fp_input_b_integer_int
13156   \l_fp_input_b_decimal_int
13157   \l_fp_input_b_exponent_int
13158   \fp_compare_aux:N #2
13159 }
13160 \prg_new_protected_conditional:Npnn \fp_compare:NNN #1#2#3 { T , F , TF }
13161 {
13162   \group_begin:
13163   \fp_read:N #3
13164   \l_fp_input_b_sign_int    \l_fp_input_a_sign_int
13165   \l_fp_input_b_integer_int \l_fp_input_a_integer_int
13166   \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
13167   \l_fp_input_b_exponent_int \l_fp_input_a_exponent_int
13168   \fp_read:N #1
13169   \fp_compare_aux:N #2
13170 }
13171 \cs_new_protected:Npn \fp_compare_aux:N #1
13172 {
13173   \cs_if_exist:cTF { fp_compare_#1: }
13174   { \use:c { fp_compare_#1: } }
13175   {
13176     \group_end:
13177     \prg_return_false:
13178   }
13179 }

```

For equality, the test is pretty easy as things are either equal or they are not.

```

13180 \cs_new_protected_nopar:cpn { fp_compare_=: }
13181 {
13182   \if_int_compare:w \l_fp_input_a_sign_int = \l_fp_input_b_sign_int
13183   \if_int_compare:w \l_fp_input_a_integer_int = \l_fp_input_b_integer_int
13184   \if_int_compare:w \l_fp_input_a_decimal_int = \l_fp_input_b_decimal_int
13185   \if_int_compare:w

```

```

13186         \l_fp_input_a_exponent_int = \l_fp_input_b_exponent_int
13187     \group_end:
13188     \prg_return_true:
13189 \else:
13190     \group_end:
13191     \prg_return_false:
13192 \fi:
13193 \else:
13194     \group_end:
13195     \prg_return_false:
13196 \fi:
13197 \else:
13198     \group_end:
13199     \prg_return_false:
13200 \fi:
13201 \else:
13202     \group_end:
13203     \prg_return_false:
13204 \fi:
13205 }

```

Comparing two values is quite complex. First, there is a filter step to check if one or other of the given values is zero. If it is then the result is relatively easy to determine.

```

13206 \cs_new_protected_nopar:cpn { fp_compare_>: }
13207 {
13208     \if_int_compare:w \int_eval:w
13209         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
13210         = \c_zero
13211     \if_int_compare:w \int_eval:w
13212         \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
13213         = \c_zero
13214     \group_end:
13215     \prg_return_false:
13216 \else:
13217     \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
13218     \group_end:
13219     \prg_return_false:
13220 \else:
13221     \group_end:
13222     \prg_return_true:
13223 \fi:
13224 \fi:
13225 \else:
13226     \if_int_compare:w \int_eval:w
13227         \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
13228         = \c_zero
13229     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
13230     \group_end:
13231     \prg_return_true:
13232 \else:

```

```

13233         \group_end:
13234         \prg_return_false:
13235         \fi:
13236     \else:
13237         \use:c { fp_compare_>_aux: }
13238     \fi:
13239 \fi:
13240 }

```

Next, check the sign of the input: this again may give an obvious result. If both signs are the same, then hand off to comparing the absolute values.

```

13241 \cs_new_protected_nopar:cpn { fp_compare_>_aux: }
13242 {
13243     \if_int_compare:w \l_fp_input_a_sign_int > \l_fp_input_b_sign_int
13244         \group_end:
13245         \prg_return_true:
13246     \else:
13247         \if_int_compare:w \l_fp_input_a_sign_int < \l_fp_input_b_sign_int
13248             \group_end:
13249             \prg_return_false:
13250         \else:
13251             \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
13252                 \use:c { fp_compare_absolute_a>b: }
13253             \else:
13254                 \use:c { fp_compare_absolute_a<b: }
13255             \fi:
13256         \fi:
13257     \fi:
13258 }

```

Rather long runs of checks, as there is the need to go through each layer of the input and do the comparison. There is also the need to avoid messing up with equal inputs at each stage.

```

13259 \cs_new_protected_nopar:cpn { fp_compare_absolute_a>b: }
13260 {
13261     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
13262         \group_end:
13263         \prg_return_true:
13264     \else:
13265         \if_int_compare:w \l_fp_input_a_exponent_int < \l_fp_input_b_exponent_int
13266             \group_end:
13267             \prg_return_false:
13268         \else:
13269             \if_int_compare:w \l_fp_input_a_integer_int > \l_fp_input_b_integer_int
13270                 \group_end:
13271                 \prg_return_true:
13272             \else:
13273                 \if_int_compare:w
13274                     \l_fp_input_a_integer_int < \l_fp_input_b_integer_int
13275                 \group_end:

```



```

13276         \prg_return_false:
13277     \else:
13278         \if_int_compare:w
13279             \l_fp_input_a_decimal_int > \l_fp_input_b_decimal_int
13280         \group_end:
13281         \prg_return_true:
13282     \else:
13283         \group_end:
13284         \prg_return_false:
13285     \fi:
13286 \fi:
13287 \fi:
13288 \fi:
13289 \fi:
13290 }
13291 \cs_new_protected_nopar:cpn { fp_compare_absolute_a<b: }
13292 {
13293     \if_int_compare:w \l_fp_input_b_exponent_int > \l_fp_input_a_exponent_int
13294     \group_end:
13295     \prg_return_true:
13296 \else:
13297     \if_int_compare:w \l_fp_input_b_exponent_int < \l_fp_input_a_exponent_int
13298     \group_end:
13299     \prg_return_false:
13300 \else:
13301     \if_int_compare:w \l_fp_input_b_integer_int > \l_fp_input_a_integer_int
13302     \group_end:
13303     \prg_return_true:
13304 \else:
13305     \if_int_compare:w
13306         \l_fp_input_b_integer_int < \l_fp_input_a_integer_int
13307     \group_end:
13308     \prg_return_false:
13309 \else:
13310     \if_int_compare:w
13311         \l_fp_input_b_decimal_int > \l_fp_input_a_decimal_int
13312     \group_end:
13313     \prg_return_true:
13314 \else:
13315     \group_end:
13316     \prg_return_false:
13317 \fi:
13318 \fi:
13319 \fi:
13320 \fi:
13321 \fi:
13322 }

```

This is just a case of reversing the two input values and then running the tests already defined.

```

13323 \cs_new_protected_nopar:cpn { fp_compare_<: }
13324 {
13325   \tl_set:Nx \l_fp_tmp_tl
13326   {
13327     \int_set:Nn \exp_not:N \l_fp_input_a_sign_int
13328     { \int_use:N \l_fp_input_b_sign_int }
13329     \int_set:Nn \exp_not:N \l_fp_input_a_integer_int
13330     { \int_use:N \l_fp_input_b_integer_int }
13331     \int_set:Nn \exp_not:N \l_fp_input_a_decimal_int
13332     { \int_use:N \l_fp_input_b_decimal_int }
13333     \int_set:Nn \exp_not:N \l_fp_input_a_exponent_int
13334     { \int_use:N \l_fp_input_b_exponent_int }
13335     \int_set:Nn \exp_not:N \l_fp_input_b_sign_int
13336     { \int_use:N \l_fp_input_a_sign_int }
13337     \int_set:Nn \exp_not:N \l_fp_input_b_integer_int
13338     { \int_use:N \l_fp_input_a_integer_int }
13339     \int_set:Nn \exp_not:N \l_fp_input_b_decimal_int
13340     { \int_use:N \l_fp_input_a_decimal_int }
13341     \int_set:Nn \exp_not:N \l_fp_input_b_exponent_int
13342     { \int_use:N \l_fp_input_a_exponent_int }
13343   }
13344   \l_fp_tmp_tl
13345   \use:c { fp_compare_>: }
13346 }

```

(End definition for \fp_compare:nNn. This function is documented on page ??.)

\fp_compare:n As TEX cannot help out here, a daisy-chain of delimited functions are used. This is very much a first-generation approach: revision will be needed if these functions are really useful.

```

\fp_compare_aux_i:w
\fp_compare_aux_ii:w
\fp_compare_aux_iii:w
\fp_compare_aux_iv:w
\fp_compare_aux_v:w
\fp_compare_aux_vi:w
\fp_compare_aux_vii:w

```

```

13347 \prg_new_protected_conditional:Npnn \fp_compare:n #1 { T , F , TF }
13348 {
13349   \group_begin:
13350   \tl_set:Nx \l_fp_tmp_tl
13351   {
13352     \group_end:
13353     \fp_compare_aux_i:w #1 \exp_not:n { == \q_nil == \q_stop }
13354   }
13355   \l_fp_tmp_tl
13356 }
13357 \cs_new_protected:Npn \fp_compare_aux_i:w #1 == #2 == #3 \q_stop
13358 {
13359   \quark_if_nil:nTF {#2}
13360   { \fp_compare_aux_ii:w #1 != \q_nil != \q_stop }
13361   { \fp_compare:nNnTF {#1} = {#2} \prg_return_true: \prg_return_false: }
13362 }
13363 \cs_new_protected:Npn \fp_compare_aux_ii:w #1 != #2 != #3 \q_stop
13364 {
13365   \quark_if_nil:nTF {#2}
13366   { \fp_compare_aux_iii:w #1 <= \q_nil <= \q_stop }

```

```

13367     { \fp_compare:nNnTF {#1} = {#2} \prg_return_false: \prg_return_true: }
13368   }
13369 \cs_new_protected:Npn \fp_compare_aux_iii:w #1 <= #2 <= #3 \q_stop
13370 {
13371   \quark_if_nil:nTF {#2}
13372   { \fp_compare_aux_iv:w #1 >= \q_nil >= \q_stop }
13373   { \fp_compare:nNnTF {#1} > {#2} \prg_return_false: \prg_return_true: }
13374 }
13375 \cs_new_protected:Npn \fp_compare_aux_iv:w #1 >= #2 >= #3 \q_stop
13376 {
13377   \quark_if_nil:nTF {#2}
13378   { \fp_compare_aux_v:w #1 = \q_nil \q_stop }
13379   { \fp_compare:nNnTF {#1} < {#2} \prg_return_false: \prg_return_true: }
13380 }
13381 \cs_new_protected:Npn \fp_compare_aux_v:w #1 = #2 = #3 \q_stop
13382 {
13383   \quark_if_nil:nTF {#2}
13384   { \fp_compare_aux_vi:w #1 < \q_nil < \q_stop }
13385   { \fp_compare:nNnTF {#1} = {#2} \prg_return_true: \prg_return_false: }
13386 }
13387 \cs_new_protected:Npn \fp_compare_aux_vi:w #1 < #2 < #3 \q_stop
13388 {
13389   \quark_if_nil:nTF {#2}
13390   { \fp_compare_aux_vii:w #1 > \q_nil > \q_stop }
13391   { \fp_compare:nNnTF {#1} < {#2} \prg_return_true: \prg_return_false: }
13392 }
13393 \cs_new_protected:Npn \fp_compare_aux_vii:w #1 > #2 > #3 \q_stop
13394 {
13395   \quark_if_nil:nTF {#2}
13396   { \prg_return_false: }
13397   { \fp_compare:nNnTF {#1} > {#2} \prg_return_true: \prg_return_false: }
13398 }

```

(End definition for \fp_compare:n. This function is documented on page ??.)

203.15 Messages

`\fp_overflow_msg:` A generic overflow message, used whenever there is a possible overflow.

```

13399 \msg_kernel_new:nnnn { fpu } { overflow }
13400 { Number~too~big. }
13401 {
13402   The~input~given~is~too~big~for~the~LaTeX~floating~point~unit. \\
13403   Further~errors~may~well~occur!
13404 }
13405 \cs_new_protected_nopar:Npn \fp_overflow_msg:
13406 { \msg_kernel_error:nn { fpu } { overflow } }

```

(End definition for \fp_overflow_msg:. This function is documented on page ??.)

`\fp_exp_overflow_msg:` A slightly more helpful message for exponent overflows.

```

13407 \msg_kernel_new:nnnn { fpu } { exponent-overflow }

```

```

13408 { Number~too-big-for-exponent-unit. }
13409 {
13410   The~exponent-of~the~input~given-is~too-big-for~the~floating-point~
13411   unit:~the~maximum~input~value~for~an~exponent~is~230.
13412 }
13413 \cs_new_protected_nopar:Npn \fp_exp_overflow_msg:
13414 { \msg_kernel_error:nn { fpu } { exponent-overflow } }
      (End definition for \fp_exp_overflow_msg:.. This function is documented on page ??.)

\fp_ln_error_msg: Logarithms are only valid for positive number
13415 \msg_kernel_new:nnnn { fpu } { logarithm-input-error }
13416 { Invalid~input~to~ln~function. }
13417 { Logarithms~can~only~be~calculated~for~positive~numbers. }
13418 \cs_new_protected_nopar:Npn \fp_ln_error_msg: {
13419   \msg_kernel_error:nn { fpu } { logarithm-input-error }
13420 }
      (End definition for \fp_ln_error_msg:.. This function is documented on page ??.)

\fp_trig_overflow_msg: A slightly more helpful message for trigonometric overflows.
13421 \msg_kernel_new:nnnn { fpu } { trigonometric-overflow }
13422 { Number~too-big-for-trigonometry-unit. }
13423 {
13424   The~trigonometry~code~can~only~work~with~numbers~smaller~
13425   than~1000000000.
13426 }
13427 \cs_new_protected_nopar:Npn \fp_trig_overflow_msg:
13428 { \msg_kernel_error:nn { fpu } { trigonometric-overflow } }
      (End definition for \fp_trig_overflow_msg:.. This function is documented on page ??.)

13429 </initex | package>

```

204 l3luatex implementation

```

13430 <*initex | package>

      Announce and ensure that the required packages are loaded.
13431 <*package>
13432 \ProvidesExplPackage
13433   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
13434 \package_check_loaded_expl:
13435 </package>

      An error message.
13436 \msg_kernel_new:nnnn { luatex } { bad-engine }
13437 { LuaTeX~engine~not~in~use!~Ignoring~#1. }
13438 {
13439   The~feature~you~are~using~is~only~available~
13440   with~the~LuaTeX~engine.~LaTeX3~ignored~‘#1#2’.
13441 }

```

`\lua_now:n` When LuaTeX is in use, this is all a question of primitives with new names. On the other hand, for pdfTeX and XeTeX the argument should be removed from the input stream before issuing an error. This is expandable, using `\msg_expandable_kernel_error:nnn` as done for V-type expansion in `l3expan`.

```

\lua_now:n
\lua_now:x
\lua_shipout_x:n
\lua_shipout_x:x
\lua_shipout:n
\lua_shipout:x
13442 \luatex_if_engine:TF
13443 {
13444     \cs_new_eq:NN \lua_now:x \luatex_directlua:D
13445     \cs_new_eq:NN \lua_shipout_x:n \luatex_latelua:D
13446 }
13447 {
13448     \cs_new:Npn \lua_now:x #1
13449     {
13450         \msg_expandable_kernel_error:nnn
13451         { luatex } { bad-engine } { \lua_now:x }
13452     }
13453     \cs_new_protected:Npn \lua_shipout_x:n #1
13454     {
13455         \msg_expandable_kernel_error:nnn
13456         { luatex } { bad-engine } { \lua_shipout_x:n }
13457     }
13458 }
13459 \cs_new:Npn \lua_now:n #1
13460 { \lua_now:x { \exp_not:n {#1} } }
13461 \cs_generate_variant:Nn \lua_shipout_x:n { x }
13462 \cs_new_protected:Npn \lua_shipout:n #1
13463 { \lua_shipout_x:n { \exp_not:n {#1} } }
13464 \cs_generate_variant:Nn \lua_shipout:n { x }

```

(End definition for `\lua_now:n` and `\lua_now:x`. These functions are documented on page ??.)

204.1 Category code tables

`\g_cctab_allocate_int` To allocate category code tables, both the read-only and stack tables need to be followed.
`\g_cctab_stack_int` There is also a sequence stack for the dynamic tables themselves.
`\g_cctab_stack_seq`

```

13465 \int_new:N \g_cctab_allocate_int
13466 \int_set:Nn \g_cctab_allocate_int { \c_minus_one }
13467 \int_new:N \g_cctab_stack_int
13468 \seq_new:N \g_cctab_stack_seq

```

(End definition for `\g_cctab_allocate_int`. This function is documented on page ??.)

`\cctab_new:N` Creating a new category code table is done slightly differently from other registers. Low-numbered tables are more efficiently-stored than high-numbered ones. There is also a need to have a stack of flexible tables as well as the set of read-only ones. To satisfy both of these requirements, odd numbered tables are used for read-only tables, and even ones for the stack. Here, therefore, the odd numbers are allocated.

```

13469 \cs_new_protected:Npn \cctab_new:N #1
13470 {
13471     \chk_if_free_cs:N #1
13472     \int_gadd:Nn \g_cctab_allocate_int { \c_two }

```

```

13473 \int_compare:nNnTF
13474 \g_cctab_allocate_int < { \c_max_register_int + \c_one }
13475 {
13476   \tex_global:D \tex_chardef:D #1 \g_cctab_allocate_int
13477   \luatex_initcatcodetable:D #1
13478 }
13479 { \msg_kernel_fatal:nxx { alloc } { out-of-registers } { cctab } }
13480 }
13481 \luatex_if_engine:F
13482 {
13483   \cs_set_protected:Npn \cctab_new:N #1
13484   {
13485     \msg_kernel_error:nxx { luatex } { bad-engine }
13486     { \exp_not:N \cctab_new:N }
13487   }
13488 }
13489 <*package>
13490 \luatex_if_engine:T
13491 {
13492   \cs_set_protected:Npn \cctab_new:N #1
13493   {
13494     \chk_if_free_cs:N #1
13495     \newcatcodetable #1
13496     \luatex_initcatcodetable:D #1
13497   }
13498 }
13499 </package>

```

(End definition for \cctab_new:N. This function is documented on page 168.)

\cctab_begin:N The aim here is to ensure that the saved tables are read-only. This is done by using a
 \cctab_end: stack of tables which are not read only, and actually having them as “in use” copies.
 \l_cctab_tmp_tl

```

13500 \cs_new_protected:Npn \cctab_begin:N #1
13501 {
13502   \seq_gpush:Nx \g_cctab_stack_seq { \tex_the:D \luatex_catcodetable:D }
13503   \luatex_catcodetable:D #1
13504   \int_gadd:Nn \g_cctab_stack_int { \c_two }
13505   \int_compare:nNnT \g_cctab_stack_int > \c_max_register_int
13506   { \msg_kernel_fatal:nx { code } { cctab-stack-full } }
13507   \luatex_savecatcodetable:D \g_cctab_stack_int
13508   \luatex_catcodetable:D \g_cctab_stack_int
13509 }
13510 \cs_new_protected_nopar:Npn \cctab_end:
13511 {
13512   \int_gsub:Nn \g_cctab_stack_int { \c_two }
13513   \seq_if_empty:NTF \g_cctab_stack_seq
13514   { \tl_set:Nn \l_cctab_tmp_tl { 0 } }
13515   { \seq_gpop:NN \g_cctab_stack_seq \l_cctab_tmp_tl }
13516   \luatex_catcodetable:D \l_cctab_tmp_tl \scan_stop:
13517 }

```

```

13518 \luatex_if_engine:F
13519 {
13520   \cs_set_protected:Npn \cctab_begin:N #1
13521   {
13522     \msg_kernel_error:nnxx { luatex } { bad-engine }
13523     { \exp_not:N \cctab_begin:N } {#1}
13524   }
13525   \cs_set_protected_nopar:Npn \cctab_end:
13526   {
13527     \msg_kernel_error:nnx { luatex } { bad-engine }
13528     { \exp_not:N \cctab_end: }
13529   }
13530 }
13531 <*package>
13532 \luatex_if_engine:T
13533 {
13534   \cs_set_protected:Npn \cctab_begin:N #1 { \BeginCatcodeRegime #1 }
13535   \cs_set_protected_nopar:Npn \cctab_end: { \EndCatcodeRegime }
13536 }
13537 </package>
13538 \tl_new:N \l_cctab_tmp_tl
      (End definition for \cctab_begin:N. This function is documented on page ??.)

```

`\cctab_gset:Nn` Category code tables are always global, so only one version is needed. The set up here is simple, and means that at the point of use there is no need to worry about escaping category codes.

```

13539 \cs_new_protected:Npn \cctab_gset:Nn #1#2
13540 {
13541   \group_begin:
13542   #2
13543   \luatex_savecatcodetable:D #1
13544   \group_end:
13545 }
13546 \luatex_if_engine:F
13547 {
13548   \cs_set_protected:Npn \cctab_gset:Nn #1#2
13549   {
13550     \msg_kernel_error:nnxx { luatex } { bad-engine }
13551     { \exp_not:N \cctab_gset:Nn } { #1 {#2} }
13552   }
13553 }
      (End definition for \cctab_gset:Nn. This function is documented on page 168.)

```

`\c_code_cctab` Creating category code tables is easy using the function above. The `other` and `string`
`\c_document_cctab` ones are done by completely ignoring the existing codes as this makes life a lot less
`\c_initex_cctab` complex. The table for expl3 category codes is always needed, whereas when in package
`\c_other_cctab` mode the rest can be copied from the existing L^AT_EX 2_ε package `luatex`.
`\c_str_cctab`

```

13554 \luatex_if_engine:T
13555 {

```

```

13556 \cctab_new:N \c_code_cctab
13557 \cctab_gset:Nn \c_code_cctab { }
13558 }
13559 <*package>
13560 \luatex_if_engine:T
13561 {
13562 \cs_new_eq:NN \c_document_cctab \CatcodeTableLaTeX
13563 \cs_new_eq:NN \c_initex_cctab \CatcodeTableIniTeX
13564 \cs_new_eq:NN \c_other_cctab \CatcodeTableOther
13565 \cs_new_eq:NN \c_str_cctab \CatcodeTableString
13566 }
13567 </package>
13568 <*initex>
13569 \luatex_if_engine:T
13570 {
13571 \cctab_new:N \c_document_cctab
13572 \cctab_new:N \c_other_cctab
13573 \cctab_new:N \c_str_cctab
13574 \cctab_gset:Nn \c_document_cctab
13575 {
13576 \char_set_catcode_space:n { 9 }
13577 \char_set_catcode_space:n { 32 }
13578 \char_set_catcode_other:n { 58 }
13579 \char_set_catcode_math_subscript:n { 95 }
13580 \char_set_catcode_active:n { 126 }
13581 }
13582 \cctab_gset:Nn \c_other_cctab
13583 {
13584 \prg_stepwise_inline:nnnn { 0 } { 1 } { 127 }
13585 { \char_set_catcode_other:n {#1} }
13586 }
13587 \cctab_gset:Nn \c_str_cctab
13588 {
13589 \prg_stepwise_inline:nnnn { 0 } { 1 } { 127 }
13590 { \char_set_catcode_other:n {#1} }
13591 \char_set_catcode_space:n { 32 }
13592 }
13593 }
13594 </initex>

```

(End definition for \c_code_cctab. This function is documented on page 169.)

204.2 Deprecated functions

Deprecated 2011-12-21, for removal by 2012-03-31.

\c_string_cctab

```

13595 \cs_new_eq:NN \c_string_cctab \c_str_cctab

```

(End definition for \c_string_cctab. This function is documented on page ??.)

```

13596 </initex | package>

```


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	
\!	8728
\#	5, 8439
\%	8441
\	2309, 2310, 2324, 2325, 8728, 8729
*	2647, 2649, 2653, 2660, 8396, 8397
\,	9313, 9315
\-	348
\.	8676, 8681
\.bool_gset:N	149, 9655
\.bool_gset_inverse:N	149, 9659
\.bool_set:N	148, 9655
\.bool_set_inverse:N	149, 9659
\.choice:	149, 9663
\.choice_code:n	149, 9671
\.choice_code:x	149, 9671
\.choices:nn	149, 9665
\.clist_gset:N	149, 9675
\.clist_gset:c	149, 9675
\.clist_set:N	149, 9675
\.clist_set:c	149, 9675
\.code:n	150, 9667
\.code:x	150, 9667
\.default:V	150, 9683
\.default:n	150, 9683
\.dim_gset:N	150, 9687
\.dim_gset:c	150, 9687
\.dim_set:N	150, 9687
\.dim_set:c	150, 9687
\.fp_gset:N	150, 9695
\.fp_gset:c	150, 9695
\.fp_set:N	150, 9695
\.fp_set:c	150, 9695
\.generate_choices:n	151, 9703
\.int_gset:N	151, 9705
\.int_gset:c	151, 9705
\.int_set:N	151, 9705
\.int_set:c	151, 9705
\.meta:n	151, 9713
\.meta:x	151, 9713
\.multichoice:	151, 9717
\.multichoice:nn	151
\.multichoices:nn	9717
\.skip_gset:N	151, 9721
\.skip_gset:c	151, 9721
\.skip_set:N	151, 9721
\.skip_set:c	151, 9721
\.tl_gset:N	152, 9729
\.tl_gset:c	152, 9729
\.tl_gset_x:N	152, 9729
\.tl_gset_x:c	152, 9729
\.tl_set:N	152, 9729
\.tl_set:c	152, 9729
\.tl_set_x:N	152, 9729
\.tl_set_x:c	152, 9729
\.value_forbidden:	152, 9745
\.value_required:	152, 9745
\/	347
\:	1095, 2739, 2872
\::	31, 1556, 1557, 1558, 1558–1561, 1563, 1565, 1572, 1577, 1583, 1704–1730, 1732, 1737, 1739, 1744, 1749, 1786–1789
\::N	31, 1560, 1560, 1713, 1719, 1720, 1724, 1725
\::V	31, 1577, 1577, 1710
\::V_unbraced	1731, 1739
\::c	31, 1561, 1561, 1705, 1711, 1714, 1721, 1728, 1729
\::f	31, 1565, 1565, 1706–1708, 1788
\::f_unbraced	1731, 1732
\::n	31, 1559, 1559, 1705, 1708–1710, 1715, 1719, 1721, 1722, 1724, 1726, 1727, 1729, 1786
\::o	31, 1563, 1563, 1706, 1709, 1711, 1712, 1716, 1717, 1719, 1720, 1722, 1723, 1725, 1727, 1730, 1787
\::o_unbraced	1731, 1737, 1786–1788
\::v	31, 1577, 1583
\::v_unbraced	1731, 1744
\::x	31, 1572, 1572, 1704, 1713–1718, 1724–1730
\::x_unbraced	1731, 1749, 1789
\;	2739, 2871, 2872
\=	9312, 9314
\?	1862, 2779
\@	1095, 1096, 4431, 4432
\@end	766

\@hyph	769	\atop	469
\@input	770	\atopwithdelims	470
\@italiccorr	771		
\@underline	772	B	
\addtofilelist	10036	\B	4432, 4434
\currname	9964	\badness	617
\filelist	10061	\baselineskip	545
\ifpackageloaded	216	\batchmode	438
\namedef	198	\BeginCatcodeRegime	1353
\nil	193, 201	\begingroup	12, 61, 100, 228, 232, 338, 376
\popfilename	163, 181, 183	\beginL	730
\pushfilename	163, 164, 179	\beginR	732
\tempa	54, 56, 64	\belowdisplaysshortskip	482
\tempboxa	6554	\belowdisplayskip	483
\\	1862,	\binoppenalty	506
	2779, 8354, 8361, 8444, 8454, 8502,	\bool_(:w	1963
	8521, 8638, 8656, 8658, 8663, 8664,	\bool_)_0:w	1963
	8695, 8697, 8703, 8705, 8769, 8845,	\bool_)_1:w	1963
	8853, 9000, 9022, 9049, 9056, 9063,	\bool_8_0:w	1963
	9072, 9080, 9081, 9294, 9886, 9901,	\bool_8_1:w	1963
	9902, 9908, 9921, 9934, 9940, 13402	\bool_:w	1963
{	3, 8438	\bool_choose:Nn	1963, 2052, 2060
}	4, 8440, 8727	\bool_cleanup:N	1963, 2045, 2048, 2050
^	6, 9, 249, 3134, 3141, 8403, 9158	\bool_do_until:cn	2108
_	2324	\bool_do_until:Nn	2108, 2110, 2111, 2113
	3967	\bool_do_until:nm	2114, 2135, 2138
~	8442	\bool_do_while:cn	2108
		\bool_do_while:Nn	2108, 2108, 2109, 2112
		\bool_do_while:nm	2114, 2122, 2125
		\bool_eval_skip_to_end:Nw	1963,
			2072, 2073, 2075, 2077, 2079, 2093
		\bool_eval_skip_to_end_aux:Nw	
			1963, 2081, 2085
		\bool_eval_skip_to_end_aux_ii:Nw	
			1963, 2089, 2091
		\bool_get_next:N	1963, 1974, 1976,
			1996, 2025, 2045, 2047, 2062-2065
A	2738, 2780, 4431, 4433	\bool_get_next:Nn	1996, 2004
\above	467	\bool_get_not_next:N	1986, 1997, 2024
\abovedisplayshortskip	480	\bool_get_not_next:Nn	1997, 2017
\abovedisplayskip	481	\bool_gset:cn	1943
\abovewithdelims	468	\bool_gset:Nn	35, 1943, 1945, 1948
\accent	518	\bool_gset_eq:cc	1935, 1942
\adjdemerits	555	\bool_gset_eq:cN	1935, 1941
\advance	362	\bool_gset_eq:Nc	1935, 1940
\afterassignment	372	\bool_gset_eq:Nn	35, 1935, 1939
\aftergroup	373	\bool_gset_false:c	1923
\alloc_reg:nNn	8181, 8184	\bool_gset_false:N	35, 1923, 1929, 1934
\alloc_setup_type:nmm	8179, 8182	\bool_gset_true:c	1923
\AtBeginDocument	10059	\bool_gset_true:N	35, 1923, 1927, 1933

- \bool_I_0:w [1963](#)
- \bool_I_1:w [1963](#)
- \bool_if:c [1949](#)
- \bool_if:N [1949](#), [1949](#)
- \bool_if:n [1963](#), [1963](#)
- \bool_if:NF
 [294](#), [1959](#), [2105](#), [2111](#), [3794](#), [7990](#), [9848](#)
- \bool_if:nF [2129](#), [2138](#)
- \bool_if:NT [1958](#), [2103](#),
 [2109](#), [3794](#), [3795](#), [7356](#), [9813](#), [10907](#)
- \bool_if:nT [2116](#), [2125](#), [5646](#)
- \bool_if:NTF
 ... [35](#), [1960](#), [3780](#), [8485](#), [9474](#), [10920](#)
- \bool_if:nTF [37](#), [3004](#), [3154](#), [4232](#), [9788](#), [9798](#)
- \bool_if_p:N [1957](#)
- \bool_if_p:n
 ... [1944](#), [1946](#), [1965](#), [1971](#), [2095](#), [2098](#)
- \bool_new:c [1921](#)
- \bool_new:N
 . [35](#), [1921](#), [1921](#), [1922](#), [1961](#), [1962](#),
 [7050](#), [8394](#), [9422](#), [9491](#), [9506](#), [10130](#)
- \bool_Not:N [2006](#), [2026](#)
- \bool_Not:w [1963](#), [2001](#), [2024](#)
- \bool_not_choose:NN [2057](#), [2061](#)
- \bool_not_cleanup:N [2047](#), [2049](#), [2055](#)
- \bool_not_Not:N [2019](#), [2035](#)
- \bool_not_Not:w [2014](#), [2025](#)
- \bool_not_p:n [37](#), [2095](#), [2095](#)
- \bool_p:w [1963](#), [2028](#), [2037](#)
- \bool_S_0:w [1963](#)
- \bool_S_1:w [1963](#)
- \bool_set:cn [1943](#)
- \bool_set:Nn [35](#), [1943](#), [1943](#), [1947](#)
- \bool_set_eq:cc [1935](#), [1938](#)
- \bool_set_eq:cN [1935](#), [1937](#)
- \bool_set_eq:Nc [1935](#), [1936](#)
- \bool_set_eq:NN [35](#), [1935](#), [1935](#)
- \bool_set_false:c [1923](#)
- \bool_set_false:N
 ... [35](#), [309](#), [1923](#), [1925](#), [1932](#), [7352](#),
 [7988](#), [8487](#), [9443](#), [9780](#), [10897](#), [10926](#)
- \bool_set_true:c [1923](#)
- \bool_set_true:N [35](#),
 [323](#), [1923](#), [1923](#), [1931](#), [7370](#), [7385](#),
 [7418](#), [8436](#), [8524](#), [9438](#), [9775](#), [10934](#)
- \bool_until_do:cn [2102](#)
- \bool_until_do:Nn [37](#), [2102](#), [2104](#), [2105](#), [2107](#)
- \bool_until_do:nn .. [38](#), [2114](#), [2127](#), [2132](#)
- \bool_while_do:cn [2102](#)
- \bool_while_do:Nn [37](#), [2102](#), [2102](#), [2103](#), [2106](#)
- \bool_while_do:nn .. [38](#), [2114](#), [2114](#), [2119](#)
- \bool_xor_p:nn [37](#), [2096](#), [2096](#)
- \botmark [453](#)
- \botmarks [679](#)
- \box [661](#)
- \box_clear:c [6461](#)
- \box_clear:N
 [118](#), [6461](#), [6461](#), [6465](#), [7093](#), [7149](#), [7194](#)
- \box_clear_new:c [6467](#)
- \box_clear_new:N .. [118](#), [6467](#), [6467](#), [6479](#)
- \box_clip:c [6960](#)
- \box_clip:N [122](#), [6960](#), [6960](#), [6962](#)
- \box_dp:c [6493](#), [7214](#)
- \box_dp:N [119](#),
 [6493](#), [6494](#), [6497](#), [6500](#), [6703](#), [6816](#),
 [6863](#), [6884](#), [6906](#), [6971](#), [6972](#), [6976](#),
 [7213](#), [7261](#), [7262](#), [7310](#), [7312](#), [7329](#),
 [7343](#), [7505](#), [7523](#), [7671](#), [8060](#), [8074](#)
- \box_gclear:c [6461](#)
- \box_gclear:N [118](#), [6461](#), [6463](#), [6466](#)
- \box_gclear_new:c [6467](#)
- \box_gclear_new:N . [118](#), [6467](#), [6473](#), [6480](#)
- \box_gset_eq:cc [6481](#)
- \box_gset_eq:cN [6481](#)
- \box_gset_eq:Nc [6481](#)
- \box_gset_eq:NN
 ... [118](#), [6464](#), [6476](#), [6481](#), [6483](#), [6486](#)
- \box_gset_eq_clear:cc [6487](#)
- \box_gset_eq_clear:cN [6487](#)
- \box_gset_eq_clear:Nc [6487](#)
- \box_gset_eq_clear:NN [118](#), [6487](#), [6489](#), [6492](#)
- \box_gset_to_last:c [6541](#)
- \box_gset_to_last:N [123](#), [6541](#), [6543](#), [6546](#)
- \box_ht:c [6493](#), [7216](#)
- \box_ht:N [119](#), [6493](#), [6493](#), [6496](#),
 [6502](#), [6702](#), [6815](#), [6862](#), [6883](#), [6905](#),
 [6981](#), [6982](#), [6986](#), [7145](#), [7189](#), [7215](#),
 [7260](#), [7262](#), [7306](#), [7308](#), [7329](#), [7336](#),
 [7504](#), [7522](#), [7668](#), [7670](#), [8058](#), [8073](#)
- \box_if_empty:c [6535](#)
- \box_if_empty:N [6535](#), [6535](#)
- \box_if_empty:NF [6539](#)
- \box_if_empty:NT [6538](#)
- \box_if_empty:NTF [122](#), [6540](#)
- \box_if_empty_p:N [6537](#)
- \box_if_horizontal:c [6523](#)
- \box_if_horizontal:N [6523](#), [6523](#)
- \box_if_horizontal:NF [6529](#)
- \box_if_horizontal:NT [6528](#)
- \box_if_horizontal:NTF [122](#), [6530](#)

<code>\box_if_horizontal_p:N</code>	6527	<code>\box_set_eq:cN</code>	6481
<code>\box_if_vertical:c</code>	6523	<code>\box_set_eq:Nc</code>	6481
<code>\box_if_vertical:N</code>	6523 , 6525	<code>\box_set_eq:NN</code>	118 , 6462 , 6470 , 6481 , 6481 , 6484 , 6485 , 7204 , 7525 , 8063
<code>\box_if_vertical:NF</code>	6533	<code>\box_set_eq_clear:cc</code>	6487
<code>\box_if_vertical:NT</code>	6532	<code>\box_set_eq_clear:cN</code>	6487
<code>\box_if_vertical:NTF</code>	123 , 6534	<code>\box_set_eq_clear:Nc</code>	6487
<code>\box_if_vertical_p:N</code>	6531	<code>\box_set_eq_clear:NN</code>	118 , 6487 , 6487 , 6490 , 6491
<code>\box_move_down:nn</code>	119 , 6512 , 6518 , 6976 , 7003 , 7651	<code>\box_set_ht:cn</code>	6499
<code>\box_move_left:nn</code>	119 , 6512 , 6512	<code>\box_set_ht:Nn</code>	120 , 6499 , 6501 , 6505 , 6728 , 6944 , 6982 , 6989 , 7008 , 7012 , 7504 , 7522 , 7654 , 8057
<code>\box_move_right:nn</code>	119 , 6512 , 6514	<code>\box_set_to_last:c</code>	6541
<code>\box_move_up:nn</code>	119 , 6512 , 6516 , 6986 , 7011 , 7543 , 8055	<code>\box_set_to_last:N</code>	123 , 6541 , 6541 , 6544 , 6545
<code>\box_new:c</code>	6453	<code>\box_set_wd:cn</code>	6499
<code>\box_new:N</code>	118 , 6453 , 6454 , 6460 , 6471 , 6477 , 6551 , 6557 , 6559 , 6677 , 7024 , 7100	<code>\box_set_wd:Nn</code>	120 , 6499 , 6503 , 6507 , 6730 , 6956 , 6965 , 6995 , 7506 , 7524 , 7657 , 8061
<code>\box_resize:cnn</code>	6810	<code>\box_show:c</code>	6560
<code>\box_resize:Nnn</code> 120 , 6810 , 6810 , 6836 , 7766		<code>\box_show:cnn</code>	6570
<code>\box_resize_aux:Nnn</code>	6810 , 6830 , 6832 , 6837 , 6873 , 6893	<code>\box_show:N</code>	123 , 6560 , 6560 , 6569 , 6576
<code>\box_resize_common:N</code> 6855 , 6934 , 6936 , 6936		<code>\box_show:Nnn</code>	127 , 6570 , 6570 , 6579 , 6581
<code>\box_resize_to_ht_plus_dp:cn</code>	6857	<code>\box_show_full:c</code>	6570
<code>\box_resize_to_ht_plus_dp:Nn</code>	121 , 6857 , 6857 , 6877	<code>\box_show_full:N</code>	127 , 6570 , 6580 , 6582
<code>\box_resize_to_wd:cn</code>	6857	<code>\box_trim:cnnnn</code>	6963
<code>\box_resize_to_wd:Nn</code> 121 , 6857 , 6878 , 6897		<code>\box_trim:Nnnnn</code>	122 , 6963 , 6963 , 6992
<code>\box_rotate:Nn</code>	121 , 6683 , 6683 , 7646	<code>\box_use:c</code>	6508
<code>\box_rotate_aux:N</code> 6683 , 6694 , 6696 , 6700		<code>\box_use:N</code>	119 , 6508 , 6509 , 6511 , 6693 , 6717 , 6724 , 6732 , 6829 , 6872 , 6892 , 6912 , 6941 , 6951 , 6957 , 6969 , 6977 , 6987 , 6999 , 7003 , 7011 , 7540 , 7543 , 7621 , 7652 , 7982 , 8052 , 8055
<code>\box_rotate_quadrant_four:</code>	6683 , 6715 , 6797	<code>\box_use_clear:c</code>	6508
<code>\box_rotate_quadrant_one:</code> 6683 , 6709 , 6764		<code>\box_use_clear:N</code>	119 , 6508 , 6508 , 6510
<code>\box_rotate_quadrant_three:</code>	6683 , 6714 , 6786	<code>\box_viewport:cnnnn</code>	6993
<code>\box_rotate_quadrant_two:</code> 6683 , 6710 , 6775		<code>\box_viewport:Nnnnn</code> 122 , 6993 , 6993 , 7015	
<code>\box_rotate_set_sin_cos:</code> 6683 , 6689 , 6734		<code>\box_wd:c</code>	6493 , 7218
<code>\box_rotate_x:nnN</code>	6683 , 6742 , 6770 , 6772 , 6781 , 6783 , 6792 , 6794 , 6803 , 6805	<code>\box_wd:N</code>	120 , 6493 , 6495 , 6498 , 6504 , 6704 , 6817 , 6864 , 6885 , 6907 , 6965 , 7217 , 7263 , 7308 , 7312 , 7318 , 7323 , 7473 , 7506 , 7524 , 7541 , 7670 , 7675 , 7835 , 7842 , 8053 , 8062 , 8075
<code>\box_rotate_y:nnN</code>	6683 , 6753 , 6766 , 6768 , 6777 , 6779 , 6788 , 6790 , 6799 , 6801	<code>\boxmaxdepth</code>	623
<code>\box_scale:cnn</code>	6898	<code>\brokenpenalty</code>	580
<code>\box_scale:Nnn</code> 121 , 6898 , 6898 , 6919 , 7793			
<code>\box_scale_aux:Nnn</code> 6898 , 6913 , 6915 , 6920			
<code>\box_set_dp:cn</code>	6499		
<code>\box_set_dp:Nn</code>	120 , 6499 , 6499 , 6506 , 6729 , 6945 , 6972 , 6979 , 7004 , 7006 , 7505 , 7523 , 7656 , 8059		
<code>\box_set_eq:cc</code>	6481		

C

<code>\C</code>	1868 , 2780
<code>\c_active_char_token</code>	3203 , 3204

<code>\c_alignment_tab_token</code>	3197 , 3198	<code>\c_fp_exp_-40_tl</code>	12251
<code>\c_alignment_token</code>		<code>\c_fp_exp_-4_tl</code>	12251
.	51 , 2644 , 2650 , 2680 , 3198	<code>\c_fp_exp_-50_tl</code>	12251
<code>\c_catcode_active_tl</code>		<code>\c_fp_exp_-5_tl</code>	12251
.	51 , 2659 , 2661 , 2718 , 3204	<code>\c_fp_exp_-60_tl</code>	12251
<code>\c_catcode_letter_token</code>		<code>\c_fp_exp_-6_tl</code>	12251
.	51 , 2644 , 2656 , 2708 , 3200	<code>\c_fp_exp_-70_tl</code>	12251
<code>\c_catcode_other_space_tl</code>		<code>\c_fp_exp_-7_tl</code>	12251
.	136 , 8395 , 8398 , 8408 , 8410 , 8412 , 8445	<code>\c_fp_exp_-80_tl</code>	12251
<code>\c_catcode_other_token</code>		<code>\c_fp_exp_-8_tl</code>	12251
.	51 , 2644 , 2657 , 2713 , 3201	<code>\c_fp_exp_-90_tl</code>	12251
<code>\c_code_cctab</code>	168 , 13554 , 13556 , 13557	<code>\c_fp_exp_-9_tl</code>	12251
<code>\c_coffin_corners_prop</code>		<code>\c_fp_exp_100_tl</code>	12231
.	7028 , 7028-7032 , 7104 , 7232	<code>\c_fp_exp_10_tl</code>	12231
<code>\c_coffin_poles_prop</code>	7033 , 7033 ,	<code>\c_fp_exp_1_tl</code>	12231
.	7035-7037 , 7039-7044 , 7106 , 7234	<code>\c_fp_exp_200_tl</code>	12231
<code>\c_document_cctab</code>		<code>\c_fp_exp_20_tl</code>	12231
.	169 , 13554 , 13562 , 13571 , 13574	<code>\c_fp_exp_2_tl</code>	12231
<code>\c_e_fp</code>	165 , 10089 , 10089	<code>\c_fp_exp_30_tl</code>	12231
<code>\c_eight</code>	68 , 2572 ,	<code>\c_fp_exp_3_tl</code>	12231
.	2604 , 3856 , 3921 , 3926 , 8158 , 10898	<code>\c_fp_exp_40_tl</code>	12231
<code>\c_eleven</code>	68 , 2578 , 2610 , 3921 , 3929 , 8161	<code>\c_fp_exp_4_tl</code>	12231
<code>\c_empty_box</code>	123 , 6462 , 6464 ,	<code>\c_fp_exp_50_tl</code>	12231
.	6470 , 6476 , 6547 , 6548 , 6551 , 7620	<code>\c_fp_exp_5_tl</code>	12231
<code>\c_empty_coffin</code>	7209 , 7209 , 7210 , 7618	<code>\c_fp_exp_60_tl</code>	12231
<code>\c_empty_prop</code>	117 , 6149 , 6149-6155 , 6268	<code>\c_fp_exp_6_tl</code>	12231
<code>\c_empty_tl</code>	93 , 3669 , 4328 , 4343 ,	<code>\c_fp_exp_70_tl</code>	12231
.	4343 , 4345 , 4347 , 4551 , 5372 , 5467	<code>\c_fp_exp_7_tl</code>	12231
<code>\c_false_bool</code>	20 ,	<code>\c_fp_exp_80_tl</code>	12231
.	1014 , 1043 , 1083 , 1084 , 1113 , 1469 ,	<code>\c_fp_exp_8_tl</code>	12231
.	1471 , 1480 , 1492 , 1921 , 1926 , 1930 ,	<code>\c_fp_exp_90_tl</code>	12231
.	2030 , 2041 , 2066 , 2069 , 2070 , 2072 ,	<code>\c_fp_exp_9_tl</code>	12231
.	2075 , 2099 , 2824 , 3769 , 3774 , 3783	<code>\c_fp_ln_10_1_tl</code>	12577
<code>\c_fifteen</code>	68 , 2586 , 2618 , 3921 , 3932 , 8165	<code>\c_fp_ln_10_2_tl</code>	12577
<code>\c_five</code>	68 , 2566 , 2598 ,	<code>\c_fp_ln_10_3_tl</code>	12577
.	3921 , 3925 , 8155 , 10551 , 11808 , 12106	<code>\c_fp_ln_10_4_tl</code>	12577
<code>\c_five_hundred_million</code>	10072 , 10075 ,	<code>\c_fp_ln_10_5_tl</code>	12577
.	10588 , 12497 , 12653 , 12848 , 12850	<code>\c_fp_ln_10_6_tl</code>	12577
<code>\c_forty_four</code>	10072 , 10072 , 10716	<code>\c_fp_ln_10_7_tl</code>	12577
<code>\c_four</code>	68 , 2564 , 2596 , 3921 , 3924 , 8154 ,	<code>\c_fp_ln_10_8_tl</code>	12577
.	8528 , 8534 , 10697 , 10933 , 11745 , 11746	<code>\c_fp_ln_10_9_tl</code>	12577
<code>\c_fourteen</code>	68 , 2584 , 2616 , 3921 , 3931 , 8164	<code>\c_fp_ln_2_1_tl</code>	12586
<code>\c_fp_exp_-100_tl</code>	12251	<code>\c_fp_ln_2_2_tl</code>	12586
<code>\c_fp_exp_-10_tl</code>	12251	<code>\c_fp_ln_2_3_tl</code>	12586
<code>\c_fp_exp_-1_tl</code>	12251	<code>\c_fp_pi_by_four_decimal_int</code>	
<code>\c_fp_exp_-200_tl</code>	12251	10077 , 10077 , 10078 ,
<code>\c_fp_exp_-20_tl</code>	12251	11736 , 11747 , 11759 , 11766 , 11770
<code>\c_fp_exp_-2_tl</code>	12251	<code>\c_fp_pi_by_four_extended_int</code>	
<code>\c_fp_exp_-30_tl</code>	12251	10077 , 10079 ,
<code>\c_fp_exp_-3_tl</code>	12251	10080 , 11736 , 11748 , 11759 , 11771

<code>\c_fp_pi_decimal_int</code>	9725, 9727, 9729, 9731, 9733, 9735, 9737, 9739, 9741, 9743, 9745, 9747
..... 10077 , 10081 , 10082 , 11676	
<code>\c_fp_pi_extended_int</code> 10077 , 10083 , 10084	<code>\c_keys_value_forbidden_tl</code> .. 9416 , 9416
<code>\c_fp_two_pi_decimal_int</code>	<code>\c_keys_value_required_tl</code> ... 9416 , 9417
.. 10077 , 10085 , 10086 , 11672 , 11678	<code>\c_keys_vars_root_tl</code> 9413 , 9414 , 9548 , 9567, 9574, 9577, 9579, 9594–9596, 9599, 9642, 9815, 9817, 9820, 9828
<code>\c_fp_two_pi_extended_int</code>	
.. 10077 , 10087 , 10088 , 11672 , 11678	<code>\c_letter_token</code>
<code>\c_group_begin_token</code>	3197 , 3200
..... 51 , 2644 , 2644 , 2665 , 3006, 3156, 4801, 4835, 6595, 6643	<code>\c_log_iow</code>
<code>\c_group_end_token</code> ... 51 , 2644 , 2645 , 2670, 3007, 3157, 6600, 6601, 6651	8146 , 8146 , 8373 , 8374
<code>\c_initex_cctab</code>	<code>\c_luatex_is_engine_bool</code> 1540 , 1540
169 , 13554 , 13563	<code>\c_math_shift_token</code>
<code>\c_int_from_roman_C_int</code>	3197 , 3199
3857	<code>\c_math_subscript_token</code>
<code>\c_int_from_roman_c_int</code> 51 , 2644 , 2654 , 2698
3857	<code>\c_math_superscript_token</code>
<code>\c_int_from_roman_D_int</code> 51 , 2644 , 2652 , 2693
3857	<code>\c_math_toggle_token</code>
<code>\c_int_from_roman_d_int</code> 51 , 2644 , 2648 , 2675 , 3199
3857	<code>\c_max_const_int</code> . 3351 , 3355 , 3375 , 3379
<code>\c_int_from_roman_I_int</code>	<code>\c_max_dim</code>
3857	75 , 4173 , 4175 , 4176, 4180, 4257, 7720–7723, 7736
<code>\c_int_from_roman_i_int</code>	<code>\c_max_int</code>
3857	68 , 3939 , 3939 , 6581
<code>\c_int_from_roman_L_int</code>	<code>\c_max_register_int</code>
3857 68 , 845 , 846 , 848 , 13474 , 13505
<code>\c_int_from_roman_l_int</code>	<code>\c_max_skip</code>
3857	78 , 4256 , 4257
<code>\c_int_from_roman_M_int</code>	<code>\c_minus_one</code>
3857	68 , 833 , 834, 837, 838, 1196, 3353, 3412, 3921, 4450, 4451, 8146, 8316, 8329, 8401, 8437, 10147, 10255, 10684, 10938, 10939, 11076, 11111, 11354, 11402, 11406, 11571, 11695, 11699, 11964, 11979, 12067, 12071, 12144, 12152, 12560, 12564, 12688, 13466
<code>\c_int_from_roman_m_int</code>	<code>\c_msg_coding_error_text_tl</code>
3857	... 8102 , 8113 , 8635 , 8635 , 9040 , 9048, 9070, 9078, 9087, 9094, 9101, 9108, 9115, 9892, 9899, 9920, 9927
<code>\c_int_from_roman_V_int</code>	<code>\c_msg_continue_text_tl</code> 8635 , 8640 , 8697
3857	<code>\c_msg_critical_text_tl</code> 8635 , 8642 , 8816
<code>\c_int_from_roman_v_int</code>	<code>\c_msg_fatal_text_tl</code> 8635 , 8644 , 8805 , 8947
3857	<code>\c_msg_help_text_tl</code> 8635 , 8646 , 8705
<code>\c_int_from_roman_X_int</code>	<code>\c_msg_hide_tl</code> 8676 , 8678 – 8680 , 8742
3857	<code>\c_msg_hide_tl<dots></code>
<code>\c_int_from_roman_x_int</code>	8676
3857	<code>\c_msg_kernel_bug_more_text_tl</code>
<code>\c_ior_streams_tl</code> 8148 , 8167 , 8200 9280 , 9288 , 9292
<code>\c_ior_term_ior</code>	<code>\c_msg_kernel_bug_text_tl</code> 9280 , 9283 , 9290
138	<code>\c_msg_more_text_prefix_tl</code> 8604 , 8605 , 8621, 8630, 8821, 8829, 8960, 8970
<code>\c_iow_log_iow</code>	<code>\c_msg_no_info_text_tl</code> . 8635 , 8648 , 8695
138	<code>\c_msg_on_line_text_tl</code>
<code>\c_iow_streams_tl</code> 8148 , 8148 , 8167 , 8213	8653 , 8672
<code>\c_iow_term_iow</code>	<code>\c_msg_on_line_tl</code>
138	8635
<code>\c_iow_wrap_end_marker_tl</code> ... 8400 , 8459	
<code>\c_iow_wrap_indent_marker_tl</code> 8400 , 8423	
<code>\c_iow_wrap_marker_tl</code>	
..... 8400 , 8402 , 8409 , 8469 , 8514	
<code>\c_iow_wrap_newline_marker_tl</code> 8400 , 8444	
<code>\c_iow_wrap_unindent_marker_tl</code>	
..... 8400 , 8425	
<code>\c_job_name_tl</code>	
92 , 4879 , 4890	
<code>\c_keys_code_root_tl</code> 9413 , 9413 , 9585 , 9590, 9854, 9856, 9868, 9874, 9879	
<code>\c_keys_props_root_tl</code>	
..... 9415 , 9415 , 9448 , 9478 , 9485, 9655, 9657, 9659, 9661, 9663, 9665, 9667, 9669, 9671, 9673, 9675, 9677, 9679, 9681, 9683, 9685, 9687, 9689, 9691, 9693, 9695, 9697, 9699, 9701, 9703, 9705, 9707, 9709, 9711, 9713, 9715, 9717, 9719, 9721, 9723,	

- \c_msg_return_text_tl [140](#), [8635](#), [8651](#),
8654, 9043, 9051, 9058, 9065, 9296
- \c_msg_text_prefix_tl . . . [8604](#), [8604](#),
8608, 8619, 8628, 8802, 8813, 8826,
8835, 8846, 8854, 8860, 8890, 8943,
8965, 8978, 9001, 9023, 9185, 9216
- \c_msg_trouble_text_tl . [140](#), [8635](#), [8661](#)
- \c_nine [68](#), [2574](#), [2606](#), [3921](#),
3927, 8159, 10260, 11608, 11840,
11926, 12172, 12181, 12400, 12692
- \c_one [68](#), [2558](#), [2590](#),
3410, [3921](#), [3921](#), 4690, 6019, 6575,
8151, 10149, 10160, 10219, 10231,
10279, 10285, 10327, 10352, 10549,
10908, 10912, 10914, 10921, 11061,
11090, 11350, 11387, 11392, 11413,
11536, 11604, 11647, 11661, 11712,
11727, 11762, 11774, 11835, 11921,
11969, 11976, 11991, 12017, 12039,
12044, 12052, 12058, 12146, 12150,
12352, 12362, 12463, 12488, 12499,
12503, 12525, 12545, 12551, 12655,
12659, 12690, 12707, 12759, 12782,
12788, 12789, 12833, 12851, 12889,
12910, 12916, 13070, 13072, 13474
- \c_one_fp . [165](#), 6692, 6826, 6828, 6871,
6891, 6909, 6911, [10090](#), 10090, 12969
- \c_one_hundred
68, [3936](#), 3936, 10282, 10283, 12361
- \c_one_hundred_million
10072, 10074, 11271, 12203
- \c_one_million [10072](#), 10073, 11534
- \c_one_thousand [68](#),
[3936](#), 3937, 11181, 11437, 11481, 11532
- \c_one_thousand_million
10072, 10076, 10222,
10244, 10261, 10271, 10307, 10318,
10332, 10343, 10384, 10426, 10700,
10719, 10838, 10874, 10900, 10951,
10976, 11042, 11059, 11062, 11077,
11086, 11163, 11202, 11210, 11219,
11306, 11319, 11355, 11385, 11388,
11390, 11393, 11403, 11407, 11414,
11415, 11535, 11537, 11549, 11561,
11576, 11609, 11620, 11638, 11696,
11700, 11716, 11720, 11804, 11859,
11901, 11945, 11996, 12002, 12050,
12054, 12056, 12060, 12068, 12072,
12102, 12226, 12296, 12471, 12481,
12500, 12518, 12543, 12547, 12549,
12553, 12561, 12565, 12635, 12656,
12678, 12730, 12797, 12800, 12869,
12908, 12912, 12914, 12918, 13062
- \c_other_cctab
169, [13554](#), 13564, 13572, 13582
- \c_other_char_token [3197](#), 3201
- \c_parameter_token
51, [2644](#), 2651, 2684, 2687
- \c_pdftex_is_engine_bool [1540](#), 1541
- \c_pi_fp . . . [165](#), 6738, 7634, [10091](#), 10091
- \c_seven [68](#), [833](#), 843, 2570, 2602, [3921](#), 8157
- \c_six [68](#), [833](#), 842,
2568, 2600, [3921](#), 8156, 11672, 11678
- \c_sixteen [68](#), [833](#), 840,
1198, 3854, [3921](#), 8145, 8147, 8179,
8182, 8199, 8201, 8212, 8214, 8247,
8265, 8283, 8301, 8318, 8331, 8566
- \c_space_tl [93](#), [4891](#), 4891, 4953, 6012,
6020, 8094, 8095, 8460, 8530, 8673,
9249, 9253, 9254, 9258, 9259, 10040
- \c_space_token [51](#), [2644](#), 2655,
2703, 3008, 3027, 3158, 4802, 4836
- \c_str_cctab
169, [13554](#), 13565, 13573, 13587, 13595
- \c_string_cctab [13595](#), 13595
- \c_ten
2576, 2608, 3694, [3921](#), 3928, 8160,
10272, 10319, 10344, 10496, 10560,
10616, 10718, 10723, 10835, 10871,
10910, 10913, 10923, 11318, 11372,
11548, 11597, 11639, 11651, 11729,
12131, 12752, 12985, 13019, 13024
- \c_ten_thousand [68](#), [3936](#), 3938
- \c_term_ior [8145](#), 8145, 8192, 8321
- \c_term_iow
8146, 8147, 8194, 8334, 8375, 8376
- \c_thirteen [68](#), 2582, 2614, [3921](#), 3930, 8163
- \c_thirty_two [68](#), [3933](#), 3933
- \c_three [68](#), 2562, 2594,
[3921](#), 3923, 8153, 11670, 11988, 12321
- \c_tl_act_lowercase_tl . [5021](#), 5026, 5034
- \c_tl_act_uppercase_tl . [5021](#), 5021, 5032
- \c_tl_rescan_marker_tl
4430, 4438, 4449, 4467
- \c_token_A_int 2869, 2904
- \c_true_bool [20](#), 1014,
1043, [1083](#), 1083, 1117, 1323, 1470,
1481, 1491, 1924, 1928, 1951, 2029,
2032, 2038, 2039, 2067, 2068, 2071,
2073, 2077, 2100, 3769, 3774, 3787

- `\c_twelve` 68, [833](#), 844, 2310,
2325, 2580, 2612, 2781, [3921](#), 8162
- `\c_two` 68,
2560, 2592, 3852, [3921](#), 3922, 8152,
10687, 11675, 11963, 11967, 11986,
12020, 12143, 12149, 12802–12804,
12817, 12893, 13472, 13504, 13512
- `\c_two_hundred_fifty_five` 68, [3934](#), 3934
- `\c_two_hundred_fifty_six` . 68, [3934](#), 3935
- `\c_undefined:D` 1309, 1317
- `\c_undefined_fp`
. 165, [10092](#), 10092, 11249,
12159, 12952, 13002, 13009, 13129
- `\c_xetex_is_engine_bool` [1540](#), 1542
- `\c_zero` . 68, [833](#), 841, 1013, 1021, 1029,
1036, 1042, 1050, 1058, 1065, 1091,
1093, 1498, 1503, 1598, 1607, 2140,
2232–2242, 2250, 2253, 2304, 2306,
2455, 2462, 2479, 2506, 2515, 2556,
2588, 2752, 3284, 3314, 3318, 3319,
3325, 3381, 3382, 3667, [3921](#), 4163,
4224, 4234, 4235, 4872, 4873, 4898,
4945, 5062, 5074, 5549, 5562, 6033,
6048, 6068, 8150, 8179, 8182, 8588,
8590, 9173, 10169, 10180, 10218,
10230, 10232, 10243, 10286–10288,
10390, 10432, 10475, 10478, 10540,
10607, 10742–10750, 10781–10789,
10844, 10880, 10924, 10979, 11017,
11033, 11075, 11079, 11081, 11147,
11151, 11176, 11245, 11255, 11268,
11269, 11290, 11294, 11315, 11324,
11325, 11353, 11401, 11405, 11409,
11410, 11429, 11473, 11547, 11575,
11586, 11594, 11652, 11655, 11662–
11664, 11694, 11698, 11702, 11707,
11730, 11731, 11736, 11755, 11759,
11770, 11795, 11836, 11850, 11892,
11922, 11936, 11962, 11975, 11977,
11989, 11990, 11992, 11993, 11995,
11997, 12000, 12011–12013, 12045,
12046, 12066, 12070, 12093, 12142,
12156, 12157, 12187, 12188, 12200,
12201, 12217, 12284, 12287, 12323,
12349, 12353–12355, 12363–12365,
12377, 12410, 12428, 12429, 12461,
12462, 12467, 12468, 12473, 12502,
12538, 12539, 12559, 12563, 12602,
12606, 12658, 12669, 12686, 12696–
12699, 12715, 12741, 12749, 12763,
12764, 12766, 12769, 12815, 12816,
12819, 12824, 12826, 12838, 12855,
12862, 12868, 12878, 12886, 12901,
12944, 12948, 12965, 12980, 12984,
12991, 13016, 13021, 13023, 13073,
13074, 13102, 13120–13123, 13210,
13213, 13217, 13228, 13229, 13251
- `\c_zero_dim` 75, 4007, 4048, [4173](#),
4174, 4179, 4256, 6624, 6839, 6841,
6842, 6979, 6989, 7001, 7004, 7007,
7012, 7366, 7369, 7372, 7381, 7384,
7387, 7396, 7403, 7469, 7474, 7481
- `\c_zero_fp` 165, 6690, 6706, 6708, 6713,
6922, 6931, 6946, 7782, 7797, 7800,
[10093](#), 10093, 10358, 10368, 10370,
11259, 12135, 12190, 12313, 12340,
12380, 12612, 12620, 12958, 13137
- `\c_zero_muskip` 4272
- `\c_zero_skip`
78, 4195, [4256](#), 4256, 4309, 4310, 6610
- `\catcode` 3–
6, 9, 70–78, 84–91, 101, 281–288, 665
- `\catcodetable` 758
- `\CatcodeTableIniTeX` 13563
- `\CatcodeTableLaTeX` 13562
- `\CatcodeTableOther` 13564
- `\CatcodeTableString` 13565
- `\cctab_begin:N`
168, [13500](#), 13500, 13520, 13523, 13534
- `\cctab_end` 168
- `\cctab_end:`
. . [13500](#), 13510, 13525, 13528, 13535
- `\cctab_gset:Nn` 168, [13539](#), 13539, 13548,
13551, 13557, 13574, 13582, 13587
- `\cctab_new:N` 168, [13469](#), 13469, 13483,
13486, 13492, 13556, 13571–13573
- `\char` 519
- `\char_gset_active:Npn` 59, 3147
- `\char_gset_active:Npx` 3148
- `\char_gset_active_eq:NN` . 59, [3133](#), 3150
- `\char_make_active:N` [3206](#), 3222
- `\char_make_active:n` [3206](#), 3240
- `\char_make_alignment:N` [3206](#)
- `\char_make_alignment:n` [3206](#)
- `\char_make_alignment_tab:N` 3211
- `\char_make_alignment_tab:n` 3229
- `\char_make_begin_group:N` 3208
- `\char_make_begin_group:n` 3226
- `\char_make_comment:N` [3206](#), 3223
- `\char_make_comment:n` [3206](#), 3241

<code>\char_make_end_group:N</code>	3209	<code>\char_set_catcode_comment:N</code>	
<code>\char_make_end_group:n</code>	3227	48, 2555, 2583, 3223	
<code>\char_make_end_line:N</code>	3206, 3212	<code>\char_set_catcode_comment:n</code>	
<code>\char_make_end_line:n</code>	3206, 3230	48, 2587, 2615, 3241	
<code>\char_make_escape:N</code>	3206, 3207	<code>\char_set_catcode_end_line:N</code>	
<code>\char_make_escape:n</code>	3206, 3225	48, 2555, 2565, 3212	
<code>\char_make_group_begin:N</code>	3206	<code>\char_set_catcode_end_line:n</code>	
<code>\char_make_group_begin:n</code>	3206	48, 2587, 2597, 3230	
<code>\char_make_group_end:N</code>	3206	<code>\char_set_catcode_escape:N</code>	
<code>\char_make_group_end:n</code>	3206	48, 2555, 2555, 3207	
<code>\char_make_ignore:N</code>	3206, 3218	<code>\char_set_catcode_escape:n</code>	
<code>\char_make_ignore:n</code>	3206, 3236	48, 2587, 2587, 3225	
<code>\char_make_invalid:N</code>	3206, 3224	<code>\char_set_catcode_group_begin:N</code>	
<code>\char_make_invalid:n</code>	3206, 3242	48, 2555, 2557, 3208	
<code>\char_make_letter:N</code>	3206, 3220	<code>\char_set_catcode_group_begin:n</code>	
<code>\char_make_letter:n</code>	3206, 3238	48, 2587, 2589, 3226	
<code>\char_make_math_shift:N</code>	3210	<code>\char_set_catcode_group_end:N</code>	
<code>\char_make_math_shift:n</code>	3228	48, 2555, 2559, 3209	
<code>\char_make_math_subscript:N</code>	3206, 3216	<code>\char_set_catcode_group_end:n</code>	
<code>\char_make_math_subscript:n</code>	3206, 3234	48, 2587, 2591, 3227	
<code>\char_make_math_superscript:N</code>	3206, 3214	<code>\char_set_catcode_ignore:N</code>	
<code>\char_make_math_superscript:n</code>	3206, 3232	48, 2555, 2573, 3218	
<code>\char_make_math_toggle:N</code>	3206	<code>\char_set_catcode_ignore:n</code>	
<code>\char_make_math_toggle:n</code>	3206	48, 313, 314, 2587, 2605, 3236, 10158	
<code>\char_make_other:N</code>	3206, 3221	<code>\char_set_catcode_invalid:N</code>	
<code>\char_make_other:n</code>	3206, 3239	48, 2555, 2585, 3224	
<code>\char_make_parameter:N</code>	3206, 3213	<code>\char_set_catcode_invalid:n</code>	
<code>\char_make_parameter:n</code>	3206, 3231	48, 2587, 2617, 3242	
<code>\char_make_space:N</code>	3206, 3219	<code>\char_set_catcode_letter:N</code>	
<code>\char_make_space:n</code>	3206, 3237	48, 2555, 2577, 3220, 8676	
<code>\char_set_active:Npn</code>	58, 3133, 3145	<code>\char_set_catcode_letter:n</code>	
<code>\char_set_active:Npx</code>	3133, 3146	48, 317, 319, 2587, 2609, 3238	
<code>\char_set_active_eq:NN</code>	59, 3133, 3149	<code>\char_set_catcode_math_subscript:N</code>	
<code>\char_set_catcode:nn</code>	49, 298–306, 2549, 2549, 2556, 2558, 2560, 2562, 2564, 2566, 2568, 2570, 2572, 2574, 2576, 2578, 2580, 2582, 2584, 2586, 2588, 2590, 2592, 2594, 2596, 2598, 2600, 2602, 2604, 2606, 2608, 2610, 2612, 2614, 2616, 2618, 2781	48, 2555, 2571, 2653, 3217	
<code>\char_set_catcode:w</code>	3169, 3170, 3177, 3179	<code>\char_set_catcode_math_subscript:n</code>	
<code>\char_set_catcode_active:N</code>	48, 2555, 2581, 2660, 3134, 3222, 8729	48, 2587, 2603, 3235, 13579	
<code>\char_set_catcode_active:n</code>	48, 2587, 2613, 3139, 3240, 9312, 9313, 13580	<code>\char_set_catcode_math_superscript:N</code>	
<code>\char_set_catcode_alignment:N</code>	48, 2555, 2563, 2649, 3211	48, 2555, 2569, 3215, 9158	
<code>\char_set_catcode_alignment:n</code>	48, 316, 2587, 2595, 3229	<code>\char_set_catcode_math_superscript:n</code>	
		48, 318, 2587, 2601, 3233	
		<code>\char_set_catcode_math_toggle:N</code>	
		48, 2555, 2561, 2647, 3210	
		<code>\char_set_catcode_math_toggle:n</code>	
		48, 2587, 2593, 3228	
		<code>\char_set_catcode_other:N</code>	
		48, 2555, 2579, 2737, 2738, 2871, 3221, 8396, 8681	
		<code>\char_set_catcode_other:n</code>	
		2587, 2611, 3239, 13578, 13585, 13590	

- \char_set_catcode_parameter:N 1298, 3346, 3361, 4002, 4190, 4266, 4327, 4333, 4338, 6456, 13471, 13494
- \char_set_catcode_parameter:n 537
- \char_set_catcode_space:N 5679, 5680
- \char_set_catcode_space:n .. 48, 2555, 2567, 3213
- \char_set_lccode:nn 104, 5679, 5679, 5831, 6084, 9764
- \char_set_lccode:w 3169, 3172, 3183, 3185
- \char_set_mathcode:nn ... 50, 2619, 2619
- \char_set_mathcode:w 3169, 3171, 3180, 3182
- \char_set_sfcode:nn 5683, 5684
- \char_set_sfcode:w 3169, 3174, 3189, 3191
- \char_set_uccode:nn 104, 5683, 5683
- \char_set_uccode:w 3169, 3173, 3186, 3188
- \char_show_value_catcode:n 49, 2549, 2553
- \char_show_value_catcode:w .. 3176, 3178
- \char_show_value_lccode:n 49, 2619, 2629
- \char_show_value_lccode:w ... 3176, 3184
- \char_show_value_mathcode:n 5695
- \char_show_value_mathcode:w .. 3176, 3181
- \char_show_value_sfcode:n 51, 2619, 2641
- \char_show_value_sfcode:w ... 5695, 5695, 5708, 5758, 5771
- \char_show_value_uccode:n 50, 2619, 2635
- \char_show_value_uccode:w ... 5695, 5696, 5698, 5699
- \char_tmp:NN 6108
- \char_value_catcode:n 6108
- \char_value_catcode:w 111, 6108, 6108, 6110
- \char_value_lccode:n ... 49, 2619, 2627
- \char_value_lccode:w 6108
- \char_value_mathcode:n .. 50, 2619, 2621
- \char_value_mathcode:w 6108
- \char_value_sfcode:n ... 50, 2619, 2639
- \char_value_sfcode:w 6108
- \char_value_uccode:n ... 50, 2619, 2633
- \char_value_uccode:w 6108
- \chardef 5757, 5818
- \chk_if_exist_cs:c 5757, 5820
- \chk_if_exist_cs:N 5757, 5819
- \chk_if_free_cs:c 5757, 5821
- \chk_if_free_cs:N 5757, 5817
- \cleaders 5770
- \clist_clear:c 5770
- \clist_clear:N 5770
- \clist_clear_new:c 5770
- \clist_clear_new:N 5770
- \clist_concat:ccc 5770
- \clist_concat:NNN 5770
- \clist_concat_aux:NNNN 5770
- \clist_const:cn 5770
- \clist_const:cx 5770
- \clist_const:Nn ... 111, 6108, 6108, 6110
- \clist_const:Nx 6108
- \clist_display:c 6133, 6135
- \clist_display:N 6133, 6134
- \clist_gclear:c 5679, 5682
- \clist_gclear:N ... 104, 5679, 5681, 6086
- \clist_gclear_new:c 5683, 5686
- \clist_gclear_new:N ... 104, 5683, 5685
- \clist_gconcat:ccc 5695
- \clist_gconcat:NNN 105, 5695, 5697, 5709, 5760, 5773
- \clist_get:cn 5783, 6127
- \clist_get:NN . 109, 5783, 5783, 5787, 6126
- \clist_get_aux:wN 5783, 5784, 5785
- \clist_gpop:cn 5788
- \clist_gpop:NN ... 110, 5788, 5790, 5805
- \clist_gpush:cn 5806, 5818
- \clist_gpush:co 5806, 5820
- \clist_gpush:cV 5806, 5819
- \clist_gpush:cx 5806, 5821
- \clist_gpush:Nn 110, 5806, 5814
- \clist_gpush:No 5806, 5816
- \clist_gpush:NV 5806, 5815
- \clist_gpush:Nx 5806, 5817
- \clist_gput_left:cn 5757, 5818
- \clist_gput_left:co 5757, 5820
- \clist_gput_left:cV 5757, 5819
- \clist_gput_left:cx 5757, 5821
- \clist_gput_left:Nn 105, 5757, 5759, 5768, 5769, 5814
- \clist_gput_left:No 5757, 5816
- \clist_gput_left:NV 5757, 5815
- \clist_gput_left:Nx 5757, 5817
- \clist_gput_right:cn 5770
- \clist_gput_right:co 5770

\clist_gput_right:cV	5770	\clist_if_in:cV	5879
\clist_gput_right:cx	5770	\clist_if_in:Nn	5879 , 5879
\clist_gput_right:Nn		\clist_if_in:nn	5879 , 5883
..... 105 , 5770 , 5772 , 5781 , 5782		\clist_if_in:NnF	5834 , 5897 , 5898
\clist_gput_right:No	5770	\clist_if_in:nnF	5902
\clist_gput_right:NV	5770	\clist_if_in:NnT	5895 , 5896
\clist_gput_right:Nx	5770	\clist_if_in:nnT	5901
\clist_gremove_all:cn	5841	\clist_if_in:NnTF	107 , 5899 , 5900
\clist_gremove_all:Nn		\clist_if_in:nnTF	5903
..... 106 , 5841 , 5843 , 5871 , 6131		\clist_if_in:No	5879
\clist_gremove_duplicates:c	5825	\clist_if_in:no	5879
\clist_gremove_duplicates:N		\clist_if_in:NV	5879
..... 106 , 5825 , 5827 , 5840		\clist_if_in:nV	5879
\clist_gremove_element:Nn ...	6129 , 6131	\clist_if_in_return:nn	
\clist_gset:cn	5751 5879 , 5881 , 5886 , 5888	
\clist_gset:co	5751	\clist_item:cn	6023
\clist_gset:cV	5751	\clist_item:Nn ...	111 , 6023 , 6023 , 6052
\clist_gset:cx	5751	\clist_item:nn	6053 , 6053
\clist_gset:Nn 105 , 5751 , 5753 , 5756 , 5773		\clist_item_aux:nnNn 6023 , 6025 , 6031 , 6055	
\clist_gset:No	5751 , 6138	\clist_item_n_aux:nw ...	6053 , 6058 , 6061
\clist_gset:NV	5751	\clist_item_n_end:n ...	6053 , 6069 , 6077
\clist_gset:Nx	5751	\clist_item_N_loop:nw	
\clist_gset_eq:cc	5687 , 5694 6023 , 6028 , 6046 , 6050	
\clist_gset_eq:cN	5687 , 5693	\clist_item_n_loop:nw	
\clist_gset_eq:Nc	5687 , 5692 6053 , 6062 , 6063 , 6066 , 6071	
\clist_gset_eq:NN . 104 , 5687 , 5691 , 5828		\clist_item_n_strip:w ..	6053 , 6079 , 6082
\clist_gset_from_seq:cc	6083	\clist_length:c	5998
\clist_gset_from_seq:cN	6083	\clist_length:N 110 , 5998 , 5998 , 6022 , 6026	
\clist_gset_from_seq:Nc	6083	\clist_length:n	5998 , 6007 , 6056
\clist_gset_from_seq:NN		\clist_length_aux:n	6003 , 6006
..... 111 , 6083 , 6085 , 6106 , 6107		\clist_length_aux:w	5998
\clist_gtrim_spaces:c	6137	\clist_length_n_aux:w ..	6012 , 6016 , 6020
\clist_gtrim_spaces:N ..	6137 , 6138 , 6140	\clist_map_aux_unbrace:Nw 5920 , 5929 , 5933	
\clist_if_empty:c	5873 , 5874	\clist_map_break	108
\clist_if_empty:N	5873 , 5873	\clist_map_break:	5976 , 5976
\clist_if_empty:n	6111 , 6111	\clist_map_break:n	109 , 5976 , 5977
\clist_if_empty:NF		\clist_map_function:cN	5904
..... 5704 , 5858 , 5906 , 5936 , 5954		\clist_map_function:NN 107 , 5598 , 5608 ,	
\clist_if_empty:NTF	106 , 9145	5904 , 5904 , 5919 , 5983 , 5991 , 6003	
\clist_if_empty:nTF	111	\clist_map_function:nN	
\clist_if_empty_n_aux:w 5603 , 5613 , 5920 , 5920 , 9552 , 9612	
..... 6111 , 6113 , 6118 , 6121		\clist_map_function_aux:Nw	
\clist_if_empty_n_aux:wNw 6111 , 6122 , 6124	 5904 , 5908 , 5913 , 5917 , 5940	
\clist_if_eq:cc	5875 , 5878	\clist_map_function_n_aux:Nn	
\clist_if_eq:cN	5875 , 5877 5920 , 5922 , 5926 , 5930	
\clist_if_eq:Nc	5875 , 5876	\clist_map_inline:cn	5934
\clist_if_eq:NN	5875 , 5875	\clist_map_inline:Nn	
\clist_if_eq:NNTF	106	... 107 , 5832 , 5934 , 5934 , 5949 , 5951	
\clist_if_in:cn	5879	\clist_map_inline:nn 5934 , 5946 , 9533 , 9627	
\clist_if_in:co	5879	\clist_map_variable:cNn	5952

\clist_map_variable:NNn	\clist_remove_all_aux:NNn
..... 108 , 5952 , 5952 , 5966 , 5975 5841 , 5842 , 5844 , 5845
\clist_map_variable:nNn 5952 , 5963	\clist_remove_all_aux:w 5841 , 5868 , 5869
\clist_map_variable_aux:Nnw	\clist_remove_duplicates:c
..... 5952 , 5957 , 5968 , 5973	5825
\clist_new:c	\clist_remove_duplicates:N
5677 , 5678 106 , 5825 , 5825 , 5839
\clist_new:N	\clist_remove_duplicates_aux:NN
104 , 5677 , 5677 , 5824 , 5872 , 5994 – 5997 5825 , 5826 , 5828 , 5829
\clist_pop:cn	\clist_remove_element:Nn 6129 , 6130
5788	\clist_set:cn
\clist_pop:NN 109 , 5788 , 5788 , 5804	5751
\clist_pop_aux:NNN 5788 , 5789 , 5791 , 5792	\clist_set:co
\clist_pop_aux:NwNNN	5751
5788	\clist_set:cV
\clist_pop_aux:w	5751
5801 , 5803	\clist_set:cx
\clist_pop_aux:wNN	5751
5788	\clist_set:Nn
\clist_pop_aux:wNNN 105 , 5751 , 5751 , 5755 , 5758 ,
5794 , 5796	5760 , 5771 , 5885 , 5948 , 5965 , 5987
\clist_push:cn	\clist_set:No
5806 , 5810	5751 , 6137
\clist_push:co	\clist_set:NV
5806 , 5812	5751
\clist_push:cV	\clist_set:Nx
5806 , 5811	5751
\clist_push:cx	\clist_set_eq:cc
5806 , 5813	5687 , 5690
\clist_push:Nn	\clist_set_eq:cN
110 , 5806 , 5806	5687 , 5689
\clist_push:No	\clist_set_eq:Nc
5806 , 5808	5687 , 5688
\clist_push:NV	\clist_set_eq:NN 104 , 5687 , 5687 , 5826 , 9769
5806 , 5807	\clist_set_from_seq:cc
\clist_push:Nx	6083
5806 , 5809	\clist_set_from_seq:cN
\clist_put_left:cn	6083
5757 , 5810	\clist_set_from_seq:Nc
\clist_put_left:co	6083
5757 , 5812	\clist_set_from_seq:NN
\clist_put_left:cV 111 , 6083 , 6083 , 6104 , 6105
5757 , 5811	\clist_set_from_seq_aux:NNNN
\clist_put_left:cx 6084 , 6086 , 6087
5757 , 5813	\clist_set_from_seq_aux:w ... 6095 , 6103
\clist_put_left:Nn	\clist_show:c
... 105 , 5757 , 5757 , 5766 , 5767 , 5806	5978 , 6135
\clist_put_left:No	\clist_show:N . 110 , 5978 , 5978 , 5993 , 6134
5757 , 5808	\clist_show:n
\clist_put_left:NV	110 , 5978 , 5985
5757 , 5807	\clist_tmp:w
\clist_put_left:Nx	5676 , 5676 ,
5757 , 5809	5710 , 5726 , 5847 , 5868 , 5890 , 5892
\clist_put_left_aux:NNNn	\clist_top:cn
..... 5757 , 5758 , 5760 , 5761	6125 , 6127
\clist_put_right:cn	\clist_top:NN
5770	6125 , 6126
\clist_put_right:co	\clist_trim_spaces:c
5770	6137
\clist_put_right:cV	\clist_trim_spaces:N ... 6137 , 6137 , 6139
5770	\clist_trim_spaces:n
\clist_put_right:cx 111 , 5730 , 5730 , 5752 , 5754 , 6109
5770	\clist_trim_spaces_aux:nn
\clist_put_right:Nn 5730 , 5733 , 5737 , 5743 , 5748
... 105 , 5770 , 5770 , 5779 , 5780 , 5835	\clist_trim_spaces_generic:nw 5710 ,
\clist_put_right:No	5712 , 5732 , 5742 , 5747 , 5922 , 5930
5770	\clist_trim_spaces_generic_aux:w ...
\clist_put_right:NV 5710 , 5721 , 5727
5770	\clist_trim_spaces_generic_aux_ii:nn
\clist_put_right:Nx 5710 , 5728 , 5729
5770 , 9845	
\clist_put_right_aux:NNNn	
..... 5770 , 5771 , 5773 , 5774	
\clist_remove_all:cn	
5841	
\clist_remove_all:Nn	
..... 106 , 5841 , 5841 , 5870 , 6130	
\clist_remove_all_aux:	
..... 5841 , 5851 , 5855 , 5867	

\clist_use:c	5822 , 5823	\coffin_ht:N	131 , 7213 , 7215
\clist_use:N	105 , 5822 , 5822	\coffin_if_exist:NT	
\closein	413	7073 , 7073 , 7091 , 7111 ,
\closeout	408		7128 , 7155 , 7172 , 7202 , 7274 , 7287
\clubpenalties	721	\coffin_join:cnncnnnn	7463
\clubpenalty	548	\coffin_join:cnnNnnnn	7463
\coffin_align:NnnNnnnnN		\coffin_join:Nnnnnnnn	7463
..	7465 , 7502 , 7520 , 7528 , 7528 , 7618	\coffin_join:NnnNnnnn	130 , 7463 , 7463 , 7499
\coffin_attach:cnncnnnn	7500	\coffin_mark_handle:cnnn	7899
\coffin_attach:cnnNnnnn	7500	\coffin_mark_handle:Nnnn	
\coffin_attach:Nnnnnnnn	7500	131 , 7899 , 7899 , 7953
\coffin_attach:NnnNnnnn		\coffin_mark_handle_aux:nnnnNnn	
.....	130 , 7500 , 7500 , 7527	7899 , 7937 , 7942 , 7946
\coffin_attach_mark:NnnNnnnn		\coffin_new:c	7098
.....	7500 , 7518 , 7911 , 7932 , 7948	\coffin_new:N	128 , 7098 , 7098 ,
\coffin_calculate_intersection:Nnn	7348 , 7348 , 7530 , 7533 , 8043		7108 , 7209 , 7211 , 7212 , 7846 – 7848
\coffin_calculate_intersection:nnnnnnnn	7348 , 7354 , 7363 , 7989	\coffin_offset_corner:Nnnnn	7569 , 7571
\coffin_calculate_intersection_aux:nnnnnn	7348 ,	\coffin_offset_corners:Nnn	
	7375 , 7390 , 7399 , 7406 , 7440 , 7449	..	7485 , 7486 , 7492 , 7493 , 7566 , 7566
\coffin_clear:c	7089	\coffin_offset_corners:Nnnnn	7566
\coffin_clear:N	128 , 7089 , 7089 , 7097	\coffin_offset_pole:Nnnnnnn	
\coffin_display_attach:Nnnnn	7547 , 7550 , 7552
.....	7954 , 7994 , 8016 , 8035 , 8041	\coffin_offset_poles:Nnn	7483 , 7484 ,
\coffin_display_handles:cn	131 , 7954		7489 , 7490 , 7512 , 7513 , 7547 , 7547
\coffin_display_handles:Nn		\coffin_reset_structure:N	
.....	7954 , 7954 , 8040	7094 , 7120 , 7138 ,
\coffin_display_handles_aux:nnnn	7954 , 8022 , 8027 , 8033		7162 , 7181 , 7229 , 7229 , 7477 , 7507
\coffin_display_handles_aux:nnnnnn	7954 , 7980 , 7984	\coffin_resize:cn	7763
\coffin_dp:c	7213 , 7214	\coffin_resize:Nnn	129 , 7763 , 7763 , 7775
\coffin_dp:N	131 , 7213 , 7213	\coffin_resize_common:Nnn	
\coffin_end_user_dimensions:	7773 , 7776 , 7776 , 7803
.....	7250 , 7265 , 7282 , 7295 , 7789	\coffin_rotate:cn	7630
\coffin_find_bounding_shift:		\coffin_rotate:Nn	129 , 7630 , 7630 , 7664
.....	7645 , 7734 , 7734	\coffin_rotate_bounding:nnn	
\coffin_find_bounding_shift_aux:nn	7734 , 7738 , 7740	7643 , 7677 , 7677
\coffin_find_corner_maxima:N		\coffin_rotate_corner:Nnnn	
.....	7644 , 7718 , 7718	7638 , 7677 , 7683
\coffin_find_corner_maxima_aux:nn	7718 , 7725 , 7727	\coffin_rotate_pole:Nnnnnn	
\coffin_get_pole:NnN	7640 , 7689 , 7689
...	7219 , 7219 , 7350 , 7351 , 7583 ,	\coffin_rotate_vector:nnNN	
	7584 , 7587 , 7588 , 7968 , 7969 , 7972	..	7679 , 7685 , 7691 , 7692 , 7701 , 7701
\coffin_gset_eq_structure:NN	7236 , 7243	\coffin_saved_Depth:	7069 , 7069 , 7253 , 7268
\coffin_ht:c	7213 , 7216	\coffin_saved_Height:	
		7069 , 7070 , 7252 , 7267
		\coffin_saved_TotalHeight:	
		7069 , 7071 , 7254 , 7269
		\coffin_saved_Width:	7069 , 7072 , 7255 , 7270
		\coffin_scale:cn	7791
		\coffin_scale:Nnn	130 , 7791 , 7791 , 7806
		\coffin_scale_corner:Nnnn	7779 , 7816 , 7816

- \coffin_scale_pole:Nnnnnn [7781](#), [7816](#), [7822](#)
- \coffin_scale_vector:nnNN [7807](#), [7807](#), [7818](#), [7824](#)
- \coffin_set_bounding:N . [7641](#), [7665](#), [7665](#)
- \coffin_set_eq:cc [7200](#)
- \coffin_set_eq:cN [7200](#)
- \coffin_set_eq:Nc [7200](#)
- \coffin_set_eq:NN [128](#), [7200](#),
[7200](#), [7208](#), [7497](#), [7516](#), [7545](#), [7975](#)
- \coffin_set_eq_structure:NN
..... [7205](#), [7236](#), [7236](#)
- \coffin_set_horizontal_pole:cnn .. [7272](#)
- \coffin_set_horizontal_pole:Nnn
..... [129](#), [7272](#), [7272](#), [7300](#)
- \coffin_set_pole:Nnn ... [7272](#), [7298](#), [7302](#)
- \coffin_set_pole:Nnx
[7142](#), [7185](#), [7272](#), [7277](#), [7290](#), [7559](#),
[7596](#), [7600](#), [7608](#), [7612](#), [7694](#), [7825](#)
- \coffin_set_user_dimensions:N
.. [7250](#), [7250](#), [7276](#), [7289](#), [7765](#), [7794](#)
- \coffin_set_vertical_pole:cnn ... [7272](#)
- \coffin_set_vertical_pole:Nnn
..... [129](#), [7272](#), [7285](#), [7301](#)
- \coffin_shift_corner:Nnnn [7660](#), [7742](#), [7742](#)
- \coffin_shift_pole:Nnnnnn [7662](#), [7742](#), [7750](#)
- \coffin_show_aux:n [8065](#)
- \coffin_show_aux:nn [8082](#), [8092](#)
- \coffin_show_aux:w [8065](#), [8085](#), [8097](#)
- \coffin_show_structure:c [8065](#)
- \coffin_show_structure:N
..... [131](#), [8065](#), [8065](#), [8098](#)
- \coffin_typeset:cnnnn [7616](#)
- \coffin_typeset:Nnnnn [130](#), [7616](#), [7616](#), [7623](#)
- \coffin_update_B:nnnnnnnnN
..... [7581](#), [7589](#), [7604](#)
- \coffin_update_corners:N
.. [7122](#), [7140](#), [7164](#), [7183](#), [7303](#), [7303](#)
- \coffin_update_poles:N .. [7121](#), [7139](#),
[7163](#), [7182](#), [7314](#), [7314](#), [7480](#), [7511](#)
- \coffin_update_T:nnnnnnnnN
..... [7581](#), [7585](#), [7592](#)
- \coffin_update_vertical_poles:NNN ..
..... [7496](#), [7515](#), [7581](#), [7581](#)
- \coffin_wd:c [7213](#), [7218](#)
- \coffin_wd:N [131](#), [7213](#), [7217](#)
- \coffin_x_shift_corner:Nnnn
..... [7785](#), [7831](#), [7831](#)
- \coffin_x_shift_pole:Nnnnnn
..... [7787](#), [7831](#), [7838](#)
- \color [7907](#), [7919](#), [7962](#), [8003](#)
- \color_ensure_current [132](#)
- \color_ensure_current: [7116](#),
[7134](#), [7157](#), [7176](#), [8130](#), [8131](#), [8135](#)
- \color_group_begin [132](#)
- \color_group_begin:
.. [7115](#), [7133](#), [7157](#), [7176](#), [8124](#), [8124](#)
- \color_group_end [132](#)
- \color_group_end:
.. [7118](#), [7136](#), [7160](#), [7179](#), [8124](#), [8125](#)
- \copy [605](#)
- \count [656](#)
- \countdef [355](#)
- \cr [380](#)
- \crr [381](#)
- \cs [46](#)
- \cs:w [16](#), [804](#),
[806](#), [819](#), [878](#), [1143](#), [1171](#), [1372](#),
[1404](#), [1562](#), [1601](#), [1615](#), [1617](#), [1619](#),
[1623](#)–[1625](#), [1660](#), [1666](#), [1686](#), [1688](#),
[1693](#), [1700](#), [1701](#), [1762](#), [1766](#), [1807](#),
[2210](#), [2212](#), [3427](#), [5095](#), [12450](#), [12810](#)
- \cs_end [16](#)
- \cs_end: [804](#), [807](#), [819](#),
[823](#), [878](#), [1137](#), [1143](#), [1165](#), [1171](#),
[1312](#), [1372](#), [1404](#), [1562](#), [1601](#), [1615](#),
[1617](#), [1619](#), [1623](#)–[1625](#), [1660](#), [1666](#),
[1686](#), [1688](#), [1693](#), [1700](#), [1701](#), [1762](#),
[1766](#), [1807](#), [2207](#), [2213](#)–[2216](#), [2218](#),
[2220](#), [2222](#), [2224](#), [2226](#), [2228](#), [2230](#),
[3427](#), [5095](#), [11817](#), [11905](#), [12115](#),
[12300](#), [12310](#), [12450](#), [12639](#), [12810](#)
- \cs_generate_from_arg_count:cNnn ...
.. [1040](#), [1048](#), [1056](#), [1064](#), [1360](#), [1403](#)
- \cs_generate_from_arg_count:NNnn ...
..... [14](#), [1337](#), [1337](#), [1361](#), [1371](#)
- \cs_generate_from_arg_count_aux:nnn
..... [1337](#), [1340](#)–[1349](#), [1358](#)
- \cs_generate_from_arg_count_error_msg:Nn
..... [1337](#), [1351](#), [1362](#)
- \cs_generate_internal_variant:n
..... [31](#), [1849](#), [1888](#), [1888](#)
- \cs_generate_internal_variant_aux:N
..... [1888](#), [1893](#), [1896](#), [1902](#)
- \cs_generate_variant:Nn [25](#), [1808](#), [1808](#),
[1904](#)–[1911](#), [1922](#), [1931](#)–[1934](#), [1947](#),
[1948](#), [1957](#)–[1960](#), [2106](#), [2107](#), [2112](#),
[2113](#), [2178](#), [2201](#), [2468](#), [2469](#), [2499](#)–
[2502](#), [2521](#)–[2524](#), [3350](#), [3371](#), [3383](#),
[3384](#), [3389](#), [3390](#), [3392](#), [3393](#), [3395](#),
[3396](#), [3405](#)–[3408](#), [3417](#)–[3420](#), [3424](#),

3425, 3799, 4006, 4009, 4010, 4014,
 4015, 4017, 4018, 4020, 4021, 4030–
 4033, 4037, 4038, 4042, 4043, 4168,
 4170, 4194, 4197, 4198, 4202, 4203,
 4205, 4206, 4208, 4209, 4213, 4214,
 4218, 4219, 4243, 4250, 4251, 4253,
 4270, 4274, 4275, 4279, 4280, 4282,
 4283, 4285, 4286, 4290, 4291, 4295,
 4296, 4300, 4302, 4330, 4341, 4342,
 4348, 4349, 4354, 4355, 4376–4381,
 4398–4405, 4422–4429, 4470–4473,
 4485–4488, 4531, 4532, 4537, 4538,
 4541–4548, 4557–4560, 4569–4572,
 4592–4595, 4614–4616, 4623–4625,
 4639, 4652, 4668, 4673, 4679, 4691,
 4692, 4751, 4752, 4760, 4761, 4789–
 4792, 4877, 4998, 5003, 5004, 5079,
 5088, 5089, 5128, 5199, 5200, 5205–
 5208, 5213–5216, 5233, 5234, 5259,
 5260, 5282–5287, 5296, 5312, 5313,
 5337, 5364, 5365, 5389, 5418, 5429,
 5430, 5464, 5487–5492, 5521–5532,
 5542, 5566, 5568, 5593, 5594, 5615–
 5620, 5638, 5639, 5708, 5709, 5755,
 5756, 5766–5769, 5779–5782, 5787,
 5804, 5805, 5839, 5840, 5870, 5871,
 5895–5903, 5919, 5951, 5975, 5993,
 6022, 6052, 6104–6107, 6110, 6139,
 6140, 6158, 6161, 6194–6197, 6206,
 6207, 6225–6228, 6243, 6245, 6247,
 6249, 6264, 6265, 6274–6277, 6301–
 6308, 6320–6325, 6340, 6341, 6353,
 6363, 6382–6387, 6402, 6416, 6426,
 6427, 6433–6435, 6438, 6460, 6465,
 6466, 6479, 6480, 6485, 6486, 6491,
 6492, 6496–6498, 6505–6507, 6510,
 6511, 6527–6534, 6537–6540, 6545,
 6546, 6569, 6579, 6582, 6586, 6587,
 6592, 6593, 6598, 6599, 6617, 6618,
 6628, 6629, 6634, 6635, 6640, 6641,
 6646, 6647, 6662, 6663, 6836, 6877,
 6897, 6919, 6962, 6992, 7015, 7097,
 7108, 7125, 7152, 7169, 7199, 7208,
 7300–7302, 7499, 7527, 7623, 7664,
 7775, 7806, 7953, 8040, 8098, 8190,
 8191, 8193, 8195, 8222, 8223, 8338,
 8339, 8366, 8369, 8793–8796, 8923,
 9435, 9600, 9653, 9654, 9757, 9758,
 9771, 9772, 10360, 10366, 10371,
 10372, 10405, 10406, 10453, 10454,
 10469, 10531, 10534, 10601, 10826,
 10829, 10861, 10864, 10945, 10946,
 10970, 10971, 10996, 10997, 11099,
 11100, 11117, 11118, 11230, 11231,
 11782, 11783, 11879, 11880, 12080,
 12081, 12273, 12274, 12576, 12591,
 12592, 12925, 12926, 13461, 13464
 \cs_generate_variant_aux:N
 1811, 1860, 1873
 \cs_generate_variant_aux:NNn
 1808, 1822, 1844
 \cs_generate_variant_aux:nnNNn
 1808, 1812, 1815
 \cs_generate_variant_aux:Nnnw
 1808, 1816, 1817, 1842
 \cs_generate_variant_aux:w
 1860, 1875, 1882
 \cs_get_arg_count_from_signature:c .
 1335, 1405
 \cs_get_arg_count_from_signature:N .
 19, 959,
 968, 975, 985, 1319, 1319, 1336, 1373
 \cs_get_arg_count_from_signature_aux:nnN
 1319, 1320, 1321
 \cs_get_arg_count_from_signature_auxii:w
 1319, 1329, 1334
 \cs_get_function_name:N . 19, 1119, 1119
 \cs_get_function_signature:N
 19, 1119, 1121
 \cs_gnew:cpn 1519
 \cs_gnew:cpx 1523
 \cs_gnew:Npn 1511
 \cs_gnew:Npx 1515
 \cs_gnew_eq:cc 1531
 \cs_gnew_eq:cN 1529
 \cs_gnew_eq:Nc 1530
 \cs_gnew_eq:NN 1528
 \cs_gnew_nopar:cpn 1518
 \cs_gnew_nopar:cpx 1522
 \cs_gnew_nopar:Npn 1510
 \cs_gnew_nopar:Npx 1514
 \cs_gnew_protected:cpn 1521
 \cs_gnew_protected:cpx 1525
 \cs_gnew_protected:Npn 1513
 \cs_gnew_protected:Npx 1517
 \cs_gnew_protected_nopar:cpn 1520
 \cs_gnew_protected_nopar:cpx 1524
 \cs_gnew_protected_nopar:Npn 1512
 \cs_gnew_protected_nopar:Npx 1516
 \cs_gset:cn 1408

\cs_gset:cpn	1274, 1276, 4643, 5939, 6345, 8628, 8630	\cs_gundefine:N	1534
\cs_gset:cpx	1274, 1277	\cs_if_eq:cc	1432
\cs_gset:cx	1408	\cs_if_eq:ccF	1448
\cs_gset:Nn	13, 1367	\cs_if_eq:ccT	1447
\cs_gset:Npn	11, 864, 866, 1260, 1276, 3147, 5393	\cs_if_eq:ccTF	1446
\cs_gset:Npx	864, 868, 1261, 1277, 3148, 5398	\cs_if_eq:cN	1432
\cs_gset:Nx	1367	\cs_if_eq:cnF	1440
\cs_gset_eq:cc	1304, 1307, 1942, 4363	\cs_if_eq:cNT	1439
\cs_gset_eq:cN	1304, 1306, 1317, 1941, 4361, 5402, 6155, 8251, 8287, 9400, 9402	\cs_if_eq:cNTF	1438
\cs_gset_eq:Nc	1304, 1305, 1940, 4362, 5409, 8243, 8258, 8279, 8294	\cs_if_eq:Nc	1432
\cs_gset_eq:NN	15, 1304, 1304–1307, 1309, 1928, 1930, 1939, 3150, 4328, 4360, 6154, 8321, 8334	\cs_if_eq:NcF	1444
\cs_gset_nopar:cn	1408	\cs_if_eq:NcT	1443
\cs_gset_nopar:cpn	1266, 1270	\cs_if_eq:NcTF	1442
\cs_gset_nopar:cpx	1266, 1271, 3045	\cs_if_eq:NN	1432, 1432
\cs_gset_nopar:cx	1408	\cs_if_eq:NNF	1440, 1444, 1448
\cs_gset_nopar:Nn	14, 1367	\cs_if_eq:NNT	1439, 1443, 1447
\cs_gset_nopar:Npn	12, 864, 864, 867, 871, 875, 1258, 1270, 2274	\cs_if_eq:NNTF	21, 1438, 1442, 1446, 8788
\cs_gset_nopar:Npx	864, 865, 869, 873, 877, 1259, 1271, 2280, 4334, 4339, 4371, 4373, 4375, 4391, 4393, 4395, 4397, 4415, 4417, 4419, 4421	\cs_if_eq_p:cc	1445
\cs_gset_nopar:Nx	1367	\cs_if_eq_p:cN	1437
\cs_gset_protected:cn	1408	\cs_if_eq_p:Nc	1441
\cs_gset_protected:cpn	1286, 1288	\cs_if_eq_p:NN	1437, 1441, 1445
\cs_gset_protected:cpx	1286, 1289	\cs_if_exist:c	1123, 1135
\cs_gset_protected:cx	1408	\cs_if_exist:cF	3887, 3894, 3896, 9573
\cs_gset_protected:Nn	14, 1367	\cs_if_exist:cT	8608, 9817
\cs_gset_protected:Npn	12, 864, 874, 1264, 1288	\cs_if_exist:cTF	1188, 1190, 1192, 1194, 7077, 8067, 8242, 8267, 8278, 8303, 8880, 8890, 9448, 9547, 9854, 9868, 9874, 13173
\cs_gset_protected:Npx	864, 876, 1265, 1289	\cs_if_exist:N	1123, 1123
\cs_gset_protected:Nx	1367	\cs_if_exist:Nf	1242, 9491, 9506, 9647
\cs_gset_protected_nopar:cn	1408	\cs_if_exist:NT	1472, 1483, 8314, 8327, 9997, 10015
\cs_gset_protected_nopar:cpn	1280, 1282	\cs_if_exist:NTF	21, 1180, 1182, 1184, 1186, 1451, 2766, 3386, 3388, 4351, 4353, 4676, 6157, 6160, 6469, 6475, 6562, 7075, 8564, 9222
\cs_gset_protected_nopar:cpx	1280, 1283	\cs_if_exist_use:c	1179, 1193
\cs_gset_protected_nopar:cx	1408	\cs_if_exist_use:cF	1189
\cs_gset_protected_nopar:Nn	14, 1367	\cs_if_exist_use:cT	1191
\cs_gset_protected_nopar:Npn	12, 864, 870, 1262, 1282	\cs_if_exist_use:cTF	1187
\cs_gset_protected_nopar:Npx	864, 872, 1263, 1283	\cs_if_exist_use:N	23, 1179, 1185
\cs_gset_protected_nopar:Nx	1367	\cs_if_exist_use:Nf	1181
\cs_gundefine:c	1535	\cs_if_exist_use:NT	1183
		\cs_if_exist_use:NTF	1179
		\cs_if_free:c	1151, 1163
		\cs_if_free:cT	1890
		\cs_if_free:N	1151, 1151
		\cs_if_free:Nf	1219, 1229
		\cs_if_free:NTF	21, 1846, 8378, 8380
		\cs_meaning:c	820, 821

- \cs_meaning:N 15, 804, 808, 828
 \cs_new:cn 1424
 \cs_new:cpn . . . 1274, 1278, 1519, 1998,
 2011, 2044, 2046, 2048, 2049, 2214–
 2217, 2219, 2221, 2223, 2225, 2227,
 2229, 2231, 2233–2242, 3442, 3450,
 3458, 3466, 3474, 3482, 3490, 4085–
 4091, 10508, 10509, 10511, 10513,
 10515, 10517, 10519, 10521, 10523,
 10569, 10571, 10573, 10575, 10577,
 10579, 10581, 10583, 10585, 10631,
 10636, 10641, 10646, 10651, 10656,
 10661, 10666, 10671, 10676, 10681
 \cs_new:cpx 1274, 1279, 1523, 1892
 \cs_new:cx 1424
 \cs_new:Nn 12, 1392
 \cs_new:Npn . . 10, 934, 966, 1250, 1260,
 1278, 1319, 1321, 1334, 1362, 1449,
 1507, 1508, 1511, 1556–1561, 1563,
 1565, 1577, 1583, 1589, 1600, 1602,
 1609, 1610, 1612, 1614, 1616, 1618,
 1620, 1627, 1629, 1634, 1639, 1645,
 1651, 1657, 1663, 1669, 1676, 1683,
 1690, 1697, 1731, 1732, 1737, 1739,
 1744, 1754, 1756, 1758, 1759, 1761,
 1763, 1769, 1775, 1777, 1784, 1790,
 1792, 1794, 1795, 1797, 1802, 1807,
 1815, 1817, 1844, 1873, 1882, 1896,
 1943, 1945, 1971, 1976, 1986, 1996,
 1997, 2024–2026, 2035, 2050, 2055,
 2060, 2061, 2079, 2085, 2091, 2095,
 2096, 2102, 2104, 2108, 2110, 2114,
 2122, 2127, 2135, 2140, 2141, 2146,
 2148, 2154, 2159, 2161, 2167, 2172,
 2179, 2184, 2190, 2195, 2202, 2209,
 2211, 2213, 2243, 2248, 2262, 2314,
 2320, 2329, 2334, 2438, 2444, 2452,
 2459, 2470, 2476, 2541, 2551, 2621,
 2627, 2633, 2639, 2750, 2790, 2797,
 2808, 2819, 2830, 2841, 2849, 2857,
 2865, 2886, 2893, 2894, 2902, 2911,
 2920, 2935, 3016, 3103, 3106, 3115,
 3124, 3137, 3279, 3281, 3289, 3300,
 3311, 3336, 3337, 3427, 3430, 3440,
 3522, 3530, 3538, 3544, 3550, 3558,
 3566, 3572, 3578, 3579, 3593, 3599,
 3631, 3663, 3665, 3671, 3683, 3691,
 3724, 3726, 3728, 3730, 3735, 3740,
 3745, 3765, 3766, 3771, 3776, 3800,
 3808, 3810, 3819, 3821, 3830, 3832,
 3842, 3851, 3853, 3855, 3871, 3880,
 3914, 3916, 3958, 3973, 4044, 4054,
 4056, 4073, 4148, 4150, 4152, 4159,
 4240, 4245, 4248, 4297, 4305, 4466,
 4514, 4515, 4525, 4573, 4626, 4634,
 4672, 4674, 4680, 4685, 4690, 4693,
 4700, 4707, 4710, 4724, 4732, 4737,
 4743, 4753–4755, 4757, 4759, 4762,
 4768, 4770, 4776, 4816, 4824, 4869,
 4896, 4904–4907, 4916, 4917, 4924,
 4935, 4944, 4946, 4953, 4959, 4961,
 4963, 4968, 4977, 4979, 4981, 4987,
 4996, 5005, 5017–5019, 5031, 5033,
 5035, 5042, 5043, 5051, 5056, 5071,
 5127, 5129, 5138, 5187, 5193, 5217,
 5332, 5377, 5383, 5465, 5533, 5541,
 5543, 5559, 5567, 5569, 5578, 5586,
 5633, 5712, 5727, 5729, 5730, 5737,
 5867, 5869, 5904, 5913, 5920, 5926,
 5933, 5998, 6006, 6023, 6031, 6046,
 6053, 6061, 6063, 6077, 6082, 6103,
 6118, 6124, 6180, 6286, 6292, 6326,
 6332, 6388, 6394, 6403, 6410, 8092,
 8097, 8546, 8553, 8763–8768, 8792,
 9177, 9180, 9189, 9194, 9199, 9204,
 9246, 9247, 9251, 9256, 9363, 9489,
 9504, 9611, 9852, 9861, 9878, 10467,
 10470, 10487, 10488, 10494, 10503,
 10524, 10530, 10532, 10535, 10547,
 10558, 10567, 10586, 10594, 10599,
 10602, 10614, 10623, 10625, 10682,
 10695, 10714, 10734, 10740, 10779,
 10818–10820, 10822, 13448, 13459
 \cs_new:Npx 1250, 1261,
 1279, 1515, 6007, 6016, 8421, 9165
 \cs_new:Nx 1392
 \cs_new_eq:cc 990, 1296, 1303, 1531
 \cs_new_eq:cN 1296,
 1301, 1529, 6151, 12134, 12158, 12189
 \cs_new_eq:Nc 1296, 1302, 1530
 \cs_new_eq:NN . . 14, 1296, 1296, 1301–
 1303, 1460–1471, 1506, 1510–1525,
 1528–1531, 1534, 1535, 1538, 1540–
 1542, 1571, 1921, 1935–1942, 2537,
 2643–2645, 2928–2930, 3170–3174,
 3194, 3195, 3198–3201, 3204, 3207–
 3214, 3216, 3218–3232, 3234, 3236–
 3242, 3245–3260, 3269–3274, 3374,
 3378, 3426, 3917, 3918, 3946–3948,
 3996–3998, 4167, 4169, 4179, 4180,

- 4242, 4244, 4247, 4252, 4256, 4257,
 4299, 4301, 4356–4363, 4474, 4475,
 4669–4671, 4878, 5098–5105, 5108–
 5115, 5118–5122, 5125, 5126, 5145–
 5162, 5366–5369, 5431–5456, 5661,
 5662, 5665, 5666, 5677–5694, 5806–
 5823, 5976, 5977, 6126, 6127, 6130,
 6131, 6134, 6135, 6150, 6162–6169,
 6354, 6355, 6418, 6419, 6430, 6493–
 6495, 6508, 6509, 6520–6522, 6548,
 6554, 6600–6607, 6615, 6616, 6653–
 6661, 7016, 7213–7218, 8124, 8144–
 8147, 8167, 8177, 8192, 8194, 8365,
 8382, 8593, 8594, 9262, 9265–9269,
 9842, 9944–9946, 10457–10466,
 13444, 13445, 13562–13565, 13595
 \cs_new_nopar:cn [1424](#)
 \cs_new_nopar:cpn
 .. [1266](#), [1272](#), [1518](#), [2062](#)–[2074](#), [2076](#)
 \cs_new_nopar:cpx [1266](#), [1273](#), [1522](#)
 \cs_new_nopar:cx [1424](#)
 \cs_new_nopar:Nn [12](#), [1392](#)
 \cs_new_nopar:Npn ... [10](#), [1250](#), [1258](#),
[1272](#), [1335](#), [1360](#), [1437](#)–[1448](#), [1458](#),
[1494](#), [1510](#), [1555](#), [1705](#)–[1712](#), [1719](#)–
[1723](#), [1786](#)–[1788](#), [2303](#), [2305](#), [2932](#)–
[2934](#), [2985](#), [2994](#), [3002](#), [3177](#), [3178](#),
[3180](#), [3181](#), [3183](#), [3184](#), [3186](#), [3187](#),
[3189](#), [3190](#), [3750](#)–[3764](#), [3951](#), [4520](#),
[4632](#), [5078](#), [5281](#), [7069](#)–[7072](#), [8381](#),
[8544](#), [8667](#)–[8669](#), [9832](#), [9834](#), [9843](#)
 \cs_new_nopar:Npx
 [1250](#), [1259](#), [1273](#), [1514](#), [1879](#)
 \cs_new_nopar:Nx [1392](#)
 \cs_new_protected:cn [1424](#)
 \cs_new_protected:cpn
 [1286](#), [1290](#), [1521](#), [9655](#),
[9657](#), [9659](#), [9661](#), [9665](#), [9667](#), [9669](#),
[9671](#), [9673](#), [9675](#), [9677](#), [9679](#), [9681](#),
[9683](#), [9685](#), [9687](#), [9689](#), [9691](#), [9693](#),
[9695](#), [9697](#), [9699](#), [9701](#), [9703](#), [9705](#),
[9707](#), [9709](#), [9711](#), [9713](#), [9715](#), [9719](#),
[9721](#), [9723](#), [9725](#), [9727](#), [9729](#), [9731](#),
[9733](#), [9735](#), [9737](#), [9739](#), [9741](#), [9743](#)
 \cs_new_protected:cpx .. [1286](#), [1291](#), [1525](#)
 \cs_new_protected:cx [1424](#)
 \cs_new_protected:Nn [12](#), [1392](#)
 \cs_new_protected:Npn
 .. [10](#), [949](#), [983](#), [1250](#), [1264](#), [1290](#),
[1292](#), [1296](#), [1308](#), [1310](#), [1337](#), [1357](#),
[1513](#), [1572](#), [1749](#), [1808](#), [1888](#), [1921](#),
[1923](#), [1925](#), [1927](#), [1929](#), [2271](#), [2277](#),
[2288](#), [2342](#), [2421](#), [2422](#), [2431](#), [2528](#),
[2549](#), [2553](#), [2555](#), [2557](#), [2559](#), [2561](#),
[2563](#), [2565](#), [2567](#), [2569](#), [2571](#), [2573](#),
[2575](#), [2577](#), [2579](#), [2581](#), [2583](#), [2585](#),
[2587](#), [2589](#), [2591](#), [2593](#), [2595](#), [2597](#),
[2599](#), [2601](#), [2603](#), [2605](#), [2607](#), [2609](#),
[2611](#), [2613](#), [2615](#), [2617](#), [2619](#), [2623](#),
[2625](#), [2629](#), [2631](#), [2635](#), [2637](#), [2641](#),
[2643](#), [2940](#), [2946](#), [2963](#), [2965](#), [2967](#),
[2981](#), [2983](#), [3344](#), [3351](#), [3381](#), [3382](#),
[3385](#), [3387](#), [3391](#), [3394](#), [3397](#), [3399](#),
[3409](#), [3411](#), [3421](#), [3919](#), [4000](#), [4007](#),
[4008](#), [4011](#), [4013](#), [4016](#), [4019](#), [4022](#),
[4024](#), [4026](#), [4028](#), [4034](#), [4036](#), [4039](#),
[4041](#), [4171](#), [4188](#), [4195](#), [4196](#), [4199](#),
[4201](#), [4204](#), [4207](#), [4210](#), [4212](#), [4215](#),
[4217](#), [4254](#), [4264](#), [4271](#), [4273](#), [4276](#),
[4278](#), [4281](#), [4284](#), [4287](#), [4289](#), [4292](#),
[4294](#), [4303](#), [4325](#), [4331](#), [4336](#), [4344](#),
[4346](#), [4350](#), [4352](#), [4364](#), [4366](#), [4368](#),
[4370](#), [4372](#), [4374](#), [4382](#), [4384](#), [4386](#),
[4388](#), [4390](#), [4392](#), [4394](#), [4396](#), [4406](#),
[4408](#), [4410](#), [4412](#), [4414](#), [4416](#), [4418](#),
[4420](#), [4446](#), [4489](#), [4527](#), [4529](#), [4533](#),
[4535](#), [4640](#), [4650](#), [4653](#), [4661](#), [4747](#),
[4749](#), [4876](#), [4999](#), [5001](#), [5083](#), [5094](#),
[5167](#), [5195](#), [5197](#), [5201](#), [5203](#), [5209](#),
[5211](#), [5219](#), [5221](#), [5223](#), [5235](#), [5237](#),
[5239](#), [5288](#), [5294](#), [5301](#), [5307](#), [5314](#),
[5320](#), [5342](#), [5348](#), [5370](#), [5390](#), [5395](#),
[5400](#), [5412](#), [5419](#), [5457](#), [5595](#), [5600](#),
[5605](#), [5610](#), [5626](#), [5644](#), [5654](#), [5676](#),
[5699](#), [5751](#), [5753](#), [5761](#), [5774](#), [5783](#),
[5785](#), [5792](#), [5796](#), [5803](#), [5825](#), [5827](#),
[5829](#), [5841](#), [5843](#), [5845](#), [5888](#), [5934](#),
[5946](#), [5952](#), [5963](#), [5968](#), [5978](#), [5985](#),
[6083](#), [6085](#), [6087](#), [6108](#), [6137](#), [6138](#),
[6150](#)–[6156](#), [6159](#), [6170](#), [6172](#), [6181](#),
[6182](#), [6188](#), [6190](#), [6192](#), [6198](#), [6204](#),
[6208](#), [6214](#), [6220](#), [6229](#)–[6231](#), [6235](#),
[6255](#), [6315](#), [6342](#), [6356](#), [6376](#), [6422](#),
[6424](#), [6454](#), [6461](#), [6463](#), [6467](#), [6473](#),
[6481](#), [6483](#), [6487](#), [6489](#), [6499](#), [6501](#),
[6503](#), [6512](#), [6514](#), [6516](#), [6518](#), [6541](#),
[6543](#), [6560](#), [6570](#), [6580](#), [6583](#)–[6585](#),
[6588](#), [6590](#), [6594](#), [6596](#), [6608](#), [6610](#),
[6611](#), [6613](#), [6619](#)–[6621](#), [6623](#), [6625](#),
[6627](#), [6630](#), [6632](#), [6636](#), [6638](#), [6642](#),

- 6644, 6648, 6664, 6683, 6700, 6734,
 6742, 6753, 6764, 6775, 6786, 6797,
 6810, 6837, 6857, 6878, 6898, 6920,
 6936, 6960, 6963, 6993, 7073, 7089,
 7098, 7109, 7126, 7153, 7170, 7200,
 7219, 7229, 7236, 7243, 7250, 7272,
 7285, 7298, 7303, 7314, 7348, 7363,
 7449, 7463, 7500, 7518, 7528, 7547,
 7552, 7566, 7571, 7581, 7592, 7604,
 7616, 7630, 7665, 7677, 7683, 7689,
 7701, 7718, 7727, 7740, 7742, 7750,
 7763, 7776, 7791, 7807, 7816, 7822,
 7831, 7838, 7899, 7946, 7954, 7984,
 8033, 8041, 8065, 8180, 8183, 8192,
 8194, 8196, 8209, 8224, 8232, 8240,
 8276, 8312, 8325, 8344, 8367, 8371,
 8377, 8379, 8420, 8427, 8466, 8511,
 8578, 8580, 8582, 8584, 8587, 8589,
 8606, 8615, 8617, 8624, 8626, 8633,
 8683, 8692, 8700, 8708, 8710, 8733,
 8748, 8755, 8772, 8867, 8894, 8902,
 8912, 8922, 8924, 8926, 8928, 8930,
 8932, 8934, 8936, 8938, 8950, 8952,
 8954, 8956, 8958, 8985, 8994, 9007,
 9009, 9011, 9013, 9016, 9029, 9031,
 9033, 9035, 9210, 9212, 9220, 9232,
 9272–9278, 9319, 9333, 9345, 9365,
 9370, 9391, 9397, 9427, 9429, 9436,
 9441, 9446, 9455, 9462, 9472, 9487,
 9529, 9545, 9559, 9571, 9582, 9587,
 9592, 9598, 9601, 9606, 9623, 9639,
 9645, 9651, 9749, 9751, 9759, 9761,
 9773, 9778, 9783, 9810, 9983, 9993,
 10027, 10044, 10049, 10142, 10144,
 10155, 10190, 10198, 10200, 10202,
 10208, 10210, 10227, 10239, 10314,
 10339, 10354, 10355, 10361, 10367,
 10369, 10375, 10411, 10830, 10865,
 10904, 10917, 10947, 10972, 10998,
 11101, 11119, 11200, 11212, 11223,
 11232, 11374, 11380, 11396, 11419,
 11463, 11523, 11555, 11565, 11583,
 11616, 11630, 11636, 11686, 11784,
 11881, 12082, 12275, 12446, 12452,
 12571, 12593, 12703, 12739, 12806,
 12876, 12927, 13171, 13357, 13363,
 13369, 13375, 13381, 13387, 13393,
 13453, 13462, 13469, 13500, 13539
 \cs_new_protected:Npx
 1250, 1265, 1291, 1517
 \cs_new_protected:Nx 1392
 \cs_new_protected_nopar:cn 1424
 \cs_new_protected_nopar:cpn
 1280, 1284,
 1520, 9663, 9717, 9745, 9747, 13180,
 13206, 13241, 13259, 13291, 13323
 \cs_new_protected_nopar:cpx
 1280, 1285, 1524
 \cs_new_protected_nopar:cx 1424
 \cs_new_protected_nopar:Nn 13, 1392
 \cs_new_protected_nopar:Npn 11, 1250,
 1262, 1267, 1284, 1293–1295, 1301–
 1307, 1512, 1704, 1713–1718, 1724–
 1730, 1789, 2307, 2936, 2938, 3025,
 3034, 3152, 3163, 3165, 3167, 3401,
 3403, 3413, 3415, 3423, 4440, 4442,
 4444, 4477, 4479, 4481, 4483, 4611–
 4613, 4659, 5092, 5163, 5165, 5297,
 5299, 5338, 5340, 5406, 5621, 5622,
 5624, 5640, 5642, 5650, 5652, 5695,
 5697, 5757, 5759, 5770, 5772, 5788,
 5790, 6251, 6253, 7168, 7198, 7265,
 7734, 8125, 8131, 8135, 8262, 8298,
 8340, 8342, 8370, 8374, 8376, 8473,
 8483, 8499, 8518, 8532, 8538, 8921,
 9519, 9613, 10051, 10165, 10294–
 10296, 10304, 10329, 10373, 10374,
 10407, 10409, 10824, 10827, 10859,
 10862, 10895, 10943, 10944, 10968,
 10969, 10994, 10995, 11011, 11048,
 11065, 11097, 11098, 11115, 11116,
 11170, 11217, 11228, 11229, 11267,
 11313, 11331, 11348, 11359, 11360,
 11365, 11540, 11545, 11573, 11606,
 11649, 11668, 11725, 11743, 11753,
 11780, 11781, 11831, 11864, 11877,
 11878, 11917, 11950, 11973, 12009,
 12024, 12078, 12079, 12129, 12139,
 12168, 12198, 12271, 12272, 12319,
 12347, 12359, 12394, 12426, 12459,
 12523, 12589, 12590, 12626, 12650,
 12684, 12713, 12727, 12761, 12786,
 12794, 12812, 12866, 12891, 12923,
 12924, 12978, 13014, 13068, 13080,
 13405, 13413, 13418, 13427, 13510
 \cs_new_protected_nopar:Npx
 .. 1250, 1263, 1285, 1516, 1877, 8526
 \cs_new_protected_nopar:Nx 1392
 \cs_set:cn 1408
 \cs_set:cpn .. 1274, 1274, 8619, 8621, 9585

- \cs_set:cpx [1274](#), [1275](#),
[2343](#), [2347](#), [2351](#), [2355](#), [2359](#), [2368](#),
[2377](#), [2386](#), [2395](#), [2397](#), [2399](#), [2401](#),
[2403](#), [2405](#), [2407](#), [2409](#), [2411](#), [9590](#)
- \cs_set:cx [1408](#)
- \cs_set:Nn [13](#), [1367](#)
- \cs_set:Npn [11](#), [850](#),
[852](#), [878](#), [887–916](#), [926](#), [957](#), [991](#),
[992](#), [1079–1082](#), [1091](#), [1092](#), [1100](#),
[1106](#), [1116](#), [1119](#), [1121](#), [1179](#), [1181](#),
[1183](#), [1185](#), [1187](#), [1189](#), [1191](#), [1193](#),
[1250](#), [1266](#), [1274](#), [1367](#), [1400](#), [1543–](#)
[1546](#), [2418](#), [2419](#), [3037](#), [3043](#), [3135](#),
[3145](#), [3276](#), [4092](#), [4100](#), [4108](#), [4114](#),
[4120](#), [4128](#), [4136](#), [4142](#), [4619](#), [4708](#),
[5710](#), [5847](#), [5890](#), [8404](#), [8454](#), [10449](#)
- \cs_set:Npx [850](#), [854](#), [1275](#), [3146](#), [4498](#)
- \cs_set:Nx [1367](#)
- \cs_set_eq:cc . [988](#), [1292](#), [1295](#), [1938](#), [4359](#)
- \cs_set_eq:cN [1292](#), [1293](#), [1937](#), [4357](#), [6153](#)
- \cs_set_eq:Nc [1292](#), [1294](#), [1936](#), [4358](#)
- \cs_set_eq:NN [15](#), [1292](#), [1292–1295](#), [1299](#),
[1304](#), [1474–1481](#), [1485–1492](#), [1885](#),
[1924](#), [1926](#), [1935](#), [2684](#), [2944](#), [2948](#),
[2969](#), [2971](#), [3030](#), [3048](#), [3149](#), [4356](#),
[5628](#), [5629](#), [5631](#), [6152](#), [7252–7259](#),
[7267–7270](#), [8187](#), [8188](#), [8444–8446](#),
[9765](#), [9767](#), [11270](#), [11363](#), [12202](#), [13124](#)
- \cs_set_eq:NwN [1547](#), [1547](#)
- \cs_set_nopar:cn [1408](#)
- \cs_set_nopar:cpn [1266](#), [1268](#)
- \cs_set_nopar:cpx [1266](#), [1269](#)
- \cs_set_nopar:cx [1408](#)
- \cs_set_nopar:Nn [13](#), [1367](#)
- \cs_set_nopar:Npn
. . [11](#), [850](#), [850](#), [852–854](#), [856–858](#),
[861](#), [917](#), [919](#), [1085](#), [1215](#), [1268](#), [8670](#)
- \cs_set_nopar:Npx [850](#),
[851](#), [855](#), [859](#), [863](#), [882](#), [1269](#), [1574](#),
[1751](#), [2950](#), [2955](#), [2972](#), [2973](#), [4365](#),
[4367](#), [4369](#), [4383](#), [4385](#), [4387](#), [4389](#),
[4407](#), [4409](#), [4411](#), [4413](#), [8438–8442](#)
- \cs_set_nopar:Nx [1367](#)
- \cs_set_protected:cn [1408](#)
- \cs_set_protected:cpn . . [1286](#), [1286](#), [8775](#)
- \cs_set_protected:cpx . . . [1286](#), [1287](#),
[1369](#), [1402](#), [8777](#), [8779](#), [8781](#), [8783](#)
- \cs_set_protected:cx [1408](#)
- \cs_set_protected:Nn [13](#), [1367](#)
- \cs_set_protected:Npn . . [11](#), [850](#), [860](#),
[879](#), [921](#), [929](#), [937](#), [941](#), [944](#), [952](#),
[961](#), [971](#), [974](#), [978](#), [987](#), [989](#), [993](#),
[1001](#), [1009](#), [1017](#), [1025](#), [1033](#), [1038](#),
[1046](#), [1054](#), [1062](#), [1067](#), [1069](#), [1199](#),
[1211](#), [1213](#), [1217](#), [1227](#), [1240](#), [1252](#),
[1286](#), [5270](#), [6174](#), [7177](#), [8987](#), [8989](#),
[8991](#), [9281](#), [10250](#), [10267](#), [11203](#),
[13483](#), [13492](#), [13520](#), [13534](#), [13548](#)
- \cs_set_protected:Npx [850](#), [862](#),
[1287](#), [8874](#), [11028](#), [11246](#), [11256](#),
[11282](#), [12609](#), [12617](#), [12643](#), [12949](#),
[12955](#), [12966](#), [12999](#), [13006](#), [13052](#)
- \cs_set_protected:Nx [1367](#)
- \cs_set_protected_nopar:cn [1408](#)
- \cs_set_protected_nopar:cpn . [1280](#), [1280](#)
- \cs_set_protected_nopar:cpx . [1280](#), [1281](#)
- \cs_set_protected_nopar:cx [1408](#)
- \cs_set_protected_nopar:Nn [13](#), [1367](#)
- \cs_set_protected_nopar:Npn
. [11](#), [310](#), [850](#), [856](#), [860](#),
[862](#), [866](#), [868](#), [870](#), [872](#), [874](#), [876](#),
[1195](#), [1197](#), [1238](#), [1248](#), [1280](#), [7158](#),
[8373](#), [8375](#), [10241](#), [10258](#), [13525](#), [13535](#)
- \cs_set_protected_nopar:Npx
. . [296](#), [850](#), [858](#), [1281](#), [8743](#), [8869](#),
[10385](#), [10427](#), [10447](#), [10839](#), [10875](#),
[10952](#), [11139](#), [11809](#), [11822](#), [11909](#),
[12107](#), [12120](#), [12304](#), [12835](#), [13087](#)
- \cs_set_protected_nopar:Nx [1367](#)
- \cs_show:c [820](#), [831](#), [9879](#)
- \cs_show:N [15](#), [804](#), [809](#), [832](#), [4876](#)
- \cs_split_function:NN [19](#), [925](#), [933](#), [940](#),
[948](#), [956](#), [965](#), [973](#), [982](#), [1075](#), [1076](#),
[1094](#), [1100](#), [1120](#), [1122](#), [1320](#), [1812](#)
- \cs_split_function_aux:w [1094](#), [1103](#), [1106](#)
- \cs_split_function_auxii:w
. [1094](#), [1114](#), [1116](#)
- \cs_tmp:w [879](#), [882](#), [885](#), [887](#),
[1250](#), [1258–1266](#), [1268–1291](#), [1367](#),
[1376–1400](#), [1408–1431](#), [1848](#), [1885](#)
- \cs_to_str:N
. . [4](#), [17](#), [1085](#), [1085](#), [1104](#), [2318](#), [8382](#)
- \cs_to_str_aux:N . [1085](#), [1089](#), [1091](#), [1092](#)
- \cs_to_str_aux:w [1085](#), [1088](#), [1092](#)
- \cs_undefine:c [1308](#), [1310](#), [1535](#)
- \cs_undefine:N [15](#), [1308](#), [1308](#), [1534](#)
- \csname . [13](#), [32](#), [35](#), [62](#), [80](#), [93](#), [96](#), [167](#),
[170](#), [176](#), [184](#), [189](#), [191](#), [201](#), [204](#),
[214](#), [229](#), [233](#), [269](#), [271](#), [276](#), [278](#), [443](#)

\currentgrouplevel	695	\dim_do_while:nNnn	
\currentgrouptype	696		73, 4112, 4120, 4136, 4140
\currentifbranch	692	\dim_eval:n	75,
\currentiflevel	691		2157, 4148, 4148, 6965, 6968, 6972,
\currentiftype	693		6976, 6982, 6986, 6995, 6998, 7006,
			7011, 7145, 7189, 7279, 7292, 7310,
			7312, 7318, 7329, 7343, 7577, 7578,
			7746, 7747, 7754, 7755, 7835, 7842
D			
\d	1861, 2778	\dim_eval:w	
\dagger	3963, 3969		80, 3996, 3997, 4012, 4035, 4040,
\day	651		4047, 4048, 4051, 4057, 4060, 4065,
\ddagger	3964, 3970		4085–4091, 4149, 4151, 4155, 4172,
\deadcycles	585		6500, 6502, 6504, 6513, 6515, 6517,
\def	54, 56, 98,		6519, 6589, 6609, 6622, 6637, 6665
	104, 106, 107, 109, 112–115, 118,	\dim_eval_end	80
	126, 128–130, 133, 141–144, 147,	\dim_eval_end:	3996, 3998,
	152, 157, 203, 213, 292, 325, 339, 350		4012, 4035, 4040, 4051, 4052, 4057,
\defaultthyphenchar	635		4060, 4066, 4149, 4151, 4155, 4172,
\defaultskewchar	636		6500, 6502, 6504, 6513, 6515, 6517,
\delcode	666		6519, 6589, 6609, 6622, 6637, 6665
\delimiter	460	\dim_gadd:cn	4034
\delimiterfactor	509	\dim_gadd:Nn	71, 4034, 4036, 4038
\delimitershortfall	508	\dim_gset:cn	4011
\deprecated	2421, 2422, 9272–9278	\dim_gset:Nn	71, 4011, 4013, 4015
\Depth	7250, 7253, 7257, 7261, 7268	\dim_gset_eq:cc	4016
\detokenize	32, 35, 80, 93, 96,	\dim_gset_eq:cN	4016
	167, 170, 176, 185, 190, 192, 198,	\dim_gset_eq:Nc	4016
	201, 204, 214, 269, 271, 276, 278, 683	\dim_gset_eq:NN	71, 4016, 4019–4021
\dim_abs:n	72, 4044, 4044	\dim_gset_max:cn	4022
\dim_add:cn	4034	\dim_gset_max:Nn	72, 4022, 4024, 4031
\dim_add:Nn	71, 4034, 4034, 4036, 4037	\dim_gset_min:cn	4022
\dim_compare:n	4063, 4063	\dim_gset_min:Nn	72, 4022, 4028, 4033
\dim_compare:nF	4102, 4117	\dim_gsub:cn	4034
\dim_compare:nNn	4058, 4058	\dim_gsub:Nn	72, 4034, 4041, 4043
\dim_compare:nNnF	4130, 4145	\dim_gzero:c	4007
\dim_compare:nNnT		\dim_gzero:N	71, 4007, 4008, 4010
	4023, 4027, 4122, 4139, 7469, 7474	\dim_new:c	3999
\dim_compare:nNnTF	73,	\dim_new:N	71,
	2163, 6839, 6842, 6971, 6981, 7001,		3999, 4000, 4006, 4174, 4175, 4182–
	7007, 7366, 7369, 7372, 7381, 7384,		4186, 6669–6676, 7025, 7051, 7052,
	7387, 7396, 7403, 7481, 7594, 7606		7057–7060, 7065–7068, 7625–7629,
\dim_compare:nT	4094, 4111		7761, 7762, 7886, 7888, 7889, 10455
\dim_compare:nTF	73	\dim_ratio:nn	72, 4054, 4054
\dim_compare<:nw	4063	\dim_ratio_aux:n	4054, 4055, 4056
\dim_compare=:nw	4063	\dim_set:cn	4011
\dim_compare>:nw	4063	\dim_set:Nn	71, 4011, 4011,
\dim_compare_aux:wNN	4063, 4065, 4073		4013, 4014, 4023, 4027, 4176, 6702–
\dim_compare_p:nNn	4058		6704, 6751, 6762, 6815–6817, 6840,
\dim_do_until:nn	74, 4092, 4114, 4118		6841, 6844, 6846, 6850, 6852, 6862–
\dim_do_until:nNnn	73, 4120, 4142, 4146		6864, 6883–6885, 6905–6907, 6924,
\dim_do_while:nn	74, 4092, 4108		

6925, 6928, 6929, 7132, 7175, 7260– 7263, 7368, 7373, 7383, 7388, 7398, 7405, 7438, 7461, 7472, 7531, 7532, 7534, 7536, 7554, 7555, 7671, 7715, 7716, 7720–7723, 7736, 7811, 7814, 7887, 7992, 7993, 8044–8046, 8048	\dp 664
\dim_set_eq:cc 4016	\driver_box_rotate_begin: 6723
\dim_set_eq:cN 4016	\driver_box_rotate_end: 6725
\dim_set_eq:Nc 4016	\driver_box_scale_begin: 6940
\dim_set_eq:NN 71, 4016, 4016–4018	\driver_box_scale_end: 6942
\dim_set_max:cn 4022	\driver_box_use_clip:N 6961
\dim_set_max:Nn 72, 4022, 4022, 4025, 4030, 7730, 7732	\driver_color_ensure_current: ... 8132
\dim_set_min:cn 4022	\dump 647
\dim_set_min:Nn 72, 4022, 4026, 4029, 4032, 7729, 7731, 7741	E
\dim_show:c 4169	\E 1867, 2780
\dim_show:N 75, 4169, 4169, 4170	\edef 33, 68, 82, 164, 166, 181, 201, 266, 273, 351
\dim_show:n 75, 4171, 4171	\else 14, 22, 63, 117, 137, 172, 187, 207, 404
\dim_strip_bp:n 81, 4150, 4150	\else: 785, 788, 825, 1110, 1127, 1130, 1139, 1145, 1155, 1158, 1167, 1173, 1314, 1325, 1350, 1435, 1499, 1504, 1544, 1595, 1953, 1967, 1981, 1991, 2002, 2005, 2015, 2018, 2031, 2040, 2296, 2298, 2300, 2302, 2448, 2464, 2487, 2495, 2508, 2517, 2666, 2671, 2676, 2681, 2688, 2694, 2699, 2704, 2709, 2714, 2719, 2724, 2729, 2734, 2754, 2762, 2769, 2803, 2814, 2825, 2836, 2898, 2907, 2915, 2924, 2990, 2998, 3020, 3295, 3306, 3316, 3321, 3324, 3327, 3435, 3446, 3454, 3462, 3470, 3478, 3486, 3494, 3502, 3510, 3518, 3721, 4061, 4068, 4226, 4553, 4565, 4578, 4588, 4604, 4785, 4805, 4820, 4828, 4838, 4860, 4900, 6270, 6296, 6524, 6526, 6536, 8557, 8568, 8571, 10148, 10177, 10223, 10234, 10284, 10290, 10300, 10392, 10434, 10477, 10481, 10498, 10542, 10550, 10553, 10562, 10590, 10609, 10618, 10686, 10689, 10706, 10709, 10725, 10728, 10751, 10754, 10757, 10760, 10763, 10766, 10769, 10772, 10775, 10790, 10793, 10796, 10799, 10802, 10805, 10808, 10811, 10814, 10846, 10882, 10911, 10925, 10930, 10981, 11019, 11035, 11060, 11083, 11093, 11153, 11156, 11251, 11261, 11296, 11299, 11335, 11337, 11340, 11386, 11391, 11412, 11588, 11660, 11674, 11709, 11734, 11749, 11764, 11797, 11814, 11818, 11837, 11852, 11894, 11906, 11923, 11938, 11966, 11968, 11978, 11994, 11999, 12014,
\dim_sub:cn 4034	
\dim_sub:Nn 72, 4034, 4039, 4041, 4042	
\dim_until_do:nn ... 74, 4092, 4100, 4105	
\dim_until_do:nNnn .. 74, 4120, 4128, 4133	
\dim_use:c 4167	
\dim_use:N 75, 4046, 4065, 4149, 4155, 4167, 4167, 4168, 7306, 7308, 7312, 7323, 7336, 7562, 7668, 7670, 7673, 7675, 7681, 7687, 7696– 7698, 7820, 7827, 8073–8075, 10419	
\dim_while_do:nn ... 74, 4092, 4092, 4097	
\dim_while_do:nNnn .. 74, 4120, 4120, 4125	
\dim_zero:c 4007	
\dim_zero:N . 71, 4007, 4007–4009, 6705, 6818, 6865, 6886, 6908, 7359, 7360	
\dimen 657	
\dimendef 356	
\dimexpr 710	
\directlua 15, 759	
\discretionary 520	
\displayindent 485	
\displaylimits 495	
\displaystyle 473	
\displaywidowpenalties 723	
\displaywidowpenalty 484	
\displaywidth 486	
\divide 363	
\doublehyphendemerits 553	

12051, 12057, 12062, 12095, 12112, 12116, 12133, 12145, 12148, 12151, 12160, 12164, 12191, 12194, 12219, 12289, 12301, 12312, 12328, 12333, 12338, 12343, 12351, 12367, 12381, 12387, 12412, 12431, 12466, 12474, 12498, 12501, 12544, 12550, 12555, 12608, 12616, 12640, 12654, 12657, 12671, 12689, 12695, 12735, 12743, 12771, 12849, 12857, 12880, 12909, 12915, 12954, 12961, 12971, 12983, 12997, 13005, 13018, 13032, 13041, 13047, 13131, 13139, 13189, 13193, 13197, 13201, 13216, 13220, 13225, 13232, 13236, 13246, 13250, 13253, 13264, 13268, 13272, 13277, 13282, 13296, 13300, 13304, 13309, 13314	\etex_eTeXversion:D 674 \etex_everyeof:D 735, 4449 \etex_firstmarks:D 678 \etex_fontchardp:D 703 \etex_fontcharht:D 702 \etex_fontcharic:D 705 \etex_fontcharwd:D 704 \etex_glueexpr:D . . . 711, 4200, 4211, 4216, 4241, 4246, 4249, 4255, 10414 \etex_glueshrink:D 714, 4315 \etex_glueshrinkorder:D 716, 4235 \etex_gluestretch:D 713, 4314 \etex_gluestretchorder:D 715, 4234 \etex_gluetomu:D 717 \etex_ifcsname:D 672, 800 \etex_ifdefined:D 671, 799, 845 \etex_iffontchar:D 701 \etex_interactionmode:D 699 \etex_interlinepenalties:D 720 \etex_lastlinefit:D 719 \etex_lastnodetype:D 700 \etex_marks:D 676 \etex_middle:D 724 \etex_muexpr:D 712, 4277, 4288, 4293, 4298, 4304 \etex_mutoglu:D 718 \etex_numexpr:D 709, 3270 \etex_pagediscards:D 727 \etex_parshapedimen:D 708 \etex_parshapeindent:D 706 \etex_parshapelength:D 707 \etex_predisplaydirection:D 734 \etex_protected:D 736, 818 \etex_readline:D 686, 8583, 8585 \etex_savinghyphcodes:D 725 \etex_savingvdiscards:D 726 \etex_scantokens:D 684, 4460 \etex_showgroups:D 697 \etex_showifs:D 698 \etex_showtokens:D . 685, 4878, 8084, 9237 \etex_splitbotmarks:D 681 \etex_splitdiscards:D 728 \etex_splitfirstmarks:D 680 \etex_TeXXETstate:D 729 \etex_topmarks:D 677 \etex_tracingassigns:D 687 \etex_tracinggroups:D 694 \etex_tracingifs:D 690 \etex_tracingnesting:D 689 \etex_tracingscantokens:D 688
\emergencystretch 568 \end 442 \EndCatcodeRegime 13535 \endcsname 13, 32, 35, 62, 80, 93, 96, 167, 170, 176, 185, 190, 192, 201, 204, 214, 229, 233, 269, 271, 276, 278, 444 \endgroup . . 12, 61, 111, 120, 228, 232, 377 \endinput 264, 416 \endL 731 \endlinechar 79, 92, 289, 458 \endR 733 \eqno 478 \errhelp 250, 424 \errmessage 418 \ERROR 2418, 2419 \errorcontextlines 425 \errorstopmode 439 \escapechar 457 \etex_beginL:D 730 \etex_beginR:D 732 \etex_botmarks:D 679 \etex_clubpenalties:D 721 \etex_currentgrouplevel:D 695 \etex_currentgrouptype:D 696 \etex_currentifbranch:D 692 \etex_currentiflevel:D 691 \etex_currentifttype:D 693 \etex_detokenize:D 683, 4671, 4672 \etex_dimexpr:D 710, 3997 \etex_displaywidowpenalties:D 723 \etex_endL:D 731 \etex_endR:D 733 \etex_eTeXrevision:D 675	

<code>\etex_unexpanded:D</code>	5469, 5475, 5496, 5503, 5571–5573, 5580, 5581, 5784, 5794, 5855, 5863,
682, 803, 1794, 1796, 1799, 1804, 4712	5868, 6079, 6177, 6335, 6397, 8084,
<code>\etex_unless:D</code>	8085, 8549, 8556, 8559, 8724, 9170–
673, 790	9173, 9184, 9239–9242, 9327, 9329,
<code>\etex_widowpenalties:D</code>	9368, 9476, 9604, 9957, 9969, 10040,
722	10143, 10163, 10168, 10172, 10176,
<code>\eTeXrevision</code>	10179, 10183, 10186, 10205, 10224,
675	10233, 10235, 10245, 10247, 10262,
<code>\eTeXversion</code>	10264, 10276, 10291, 10299, 10301,
674	10308, 10311, 10323, 10333, 10336,
<code>\everycr</code>	10348, 10397, 10418, 10439, 10468,
386	10476, 10479, 10482, 10497, 10499,
<code>\everydisplay</code>	10533, 10541, 10543, 10561, 10563,
487	10600, 10608, 10610, 10617, 10619,
<code>\everyeof</code>	10685, 10688, 10690, 10702, 10721,
735	10836, 10851, 10872, 10887, 10901,
<code>\everyhbox</code>	10940, 10960, 10986, 10991, 10992,
626	11018, 11020, 11040, 11161, 11209,
<code>\everyjob</code>	11220, 11262, 11304, 11320, 11326,
31, 655	11334, 11341–11343, 11550, 11552,
<code>\everymath</code>	11562, 11577, 11580, 11610, 11613,
511	11624, 11627, 11643, 11659, 11665,
<code>\everypar</code>	11673, 11679–11681, 11750, 11763,
574	11775, 11802, 11819, 11857, 11868,
<code>\everyvbox</code>	11870, 11872, 11874, 11899, 11907,
627	11943, 11954, 11956, 11958, 11960,
<code>\exhyphenpenalty</code>	12006, 12021, 12075, 12100, 12117,
550	12132, 12136, 12161, 12165, 12192,
<code>\exp_after:wN</code>	12195, 12224, 12294, 12302, 12325–
29,	12327, 12329–12331, 12335–12337,
801, 801, 819, 824, 826, 883, 918,	12344, 12350, 12356, 12366, 12370,
920, 1004, 1072, 1089, 1093, 1102,	12383–12385, 12389, 12390, 12403,
1103, 1109, 1111, 1138, 1140, 1143,	12448, 12516, 12568, 12607, 12614,
1166, 1168, 1171, 1313, 1315, 1324,	12622, 12633, 12641, 12676, 12693,
1326, 1329, 1372, 1404, 1555, 1562,	12731, 12734, 12770, 12772, 12783,
1564, 1567, 1568, 1575, 1579, 1580,	12791, 12808, 12856, 12858, 12870,
1585, 1586, 1591, 1596, 1598, 1601,	12873, 12920, 12972, 12982, 12994–
1609, 1611, 1613, 1615, 1617, 1619,	12996, 13017, 13026–13030, 13034–
1622–1624, 1628, 1631, 1636, 1641–	13039, 13043–13045, 13048, 13060
1643, 1647–1649, 1653–1655, 1659–	<code>\exp_arg_last_unbraced:nn</code>
1661, 1665–1667, 1671–1674, 1678–	.. 1731, 1731, 1734, 1738, 1741, 1746
1681, 1685–1687, 1692–1695, 1699–	<code>\exp_arg_next:nnn</code>
1702, 1734, 1735, 1738, 1741, 1742,	1556, 1564, 1567, 1575, 1579, 1585
1746, 1747, 1755, 1757, 1758, 1760,	<code>\exp_arg_next_nobrace:nnn</code> 1556, 1557, 1562
1762, 1765, 1766, 1771, 1772, 1776,	<code>\exp_args:cc</code>
1779–1781, 1785, 1791, 1793, 1794,	1614, 1614
1796, 1799, 1804, 1807, 1820, 1826,	<code>\exp_args:Nc</code>
1875, 1900, 2001, 2004, 2006, 2014,	26, 819, 819, 820,
2017, 2019, 2024, 2025, 2028, 2037,	828, 1011, 1019, 1027, 1035, 1239,
2045, 2047–2049, 2052, 2057, 2205,	1249, 1267, 1293, 1301, 1306, 1336,
2316, 2317, 2331, 2441, 2447, 2449,	1361, 1437–1440, 1459, 1614, 4645
2456, 2463, 2465, 2473, 2480, 2747,	
2768, 2787, 2794, 2804, 2815, 2826,	
2837, 2846, 2854, 2862, 2882, 2905,	
2906, 2908, 2914, 2917, 2989, 2991,	
2997, 2999, 3012, 3019, 3021, 3110,	
3119, 3128, 3429, 3432, 3693, 3721,	
3732, 3742, 3875, 4065, 4154, 4458,	
4459, 4509, 4517, 4522, 4563, 4575,	
4576, 4672, 4744, 4748, 4750, 4764,	
4772, 4782, 4798, 4818, 4827, 4830,	
4852–4854, 4872–4874, 4880, 4938,	
4966, 5053, 5054, 5274, 5291, 5304,	
5323, 5324, 5352, 5353, 5374, 5379,	

- \exp_args:Ncc 1295,
1303, 1307, 1445–1448, 1614, 1618
- \exp_args:Nccc 1614, 1620
- \exp_args:Ncco 1676, 1697
- \exp_args:Nccx 1719, 1728
- \exp_args:Ncf 1639, 1663
- \exp_args:NcNc 1676, 1683
- \exp_args:NcNo 1676, 1690
- \exp_args:Ncnx 1719, 1729
- \exp_args:Nco 1639, 1657
- \exp_args:Ncx 1705, 1714
- \exp_args:Nf 27, 1627, 1627,
2144, 2157, 3595, 3664, 3676, 3685,
3778, 3791, 3805, 3815, 3826, 3837,
4965, 5045, 5058, 5076, 5564, 6018,
6037, 6050, 6055, 6071, 9182, 9237
- \exp_args:Nff 1705, 1707
- \exp_args:Nfo 1705, 1706, 6025
- \exp_args:NNc 832, 1075,
1076, 1294, 1302, 1305, 1441–1444,
1614, 1616, 1822, 2273, 2279, 9184
- \exp_args:Nnc 1705, 1705
- \exp_args:NNf 1639, 1639, 2251, 2258, 2267
- \exp_args:Nnf 954, 963, 972, 980, 1705, 1708
- \exp_args:Nnnc 1719, 1721
- \exp_args:NNNo 28, 1609, 1612
- \exp_args:NNno 1719, 1719
- \exp_args:Nnno 1719, 1722
- \exp_args:NNNV 1676, 1676
- \exp_args:NNnx 28, 1719, 1724
- \exp_args:Nnnx 1719, 1726
- \exp_args:NNo 27, 1609,
1610, 3583, 5047, 6171, 8463, 9603
- \exp_args:Nno 27, 1705,
1709, 3102, 4072, 5956, 6186, 9856
- \exp_args:NNoo 28, 1719, 1720
- \exp_args:NNox 1719, 1725
- \exp_args:Nnox 1719, 1727
- \exp_args:NNV 1639, 1651
- \exp_args:NNv 1639, 1645
- \exp_args:NnV 1705, 1710
- \exp_args:NNx 28, 1705, 1713
- \exp_args:Nnx 1705, 1715
- \exp_args:No 26, 1609, 1609, 3583,
3668, 4449, 4611–4613, 4633, 4651,
4660, 4904–4907, 5078, 5190, 5728,
5862, 5881, 5886, 6065, 6069, 8545
- \exp_args:Noc 1705, 1711
- \exp_args:Noo 1705, 1712
- \exp_args:Nooo 1719, 1723
- \exp_args:Noox 1719, 1730
- \exp_args:Nox 1705, 1716
- \exp_args:NV 27, 1627, 1634
- \exp_args:Nv 27, 1627, 1629
- \exp_args:NVV 1639, 1669
- \exp_args:Nx 27, 1627, 1704, 1704
- \exp_args:Nxo 1705, 1717
- \exp_args:Nxx 1705, 1718
- \exp_eval_error_msg:w .. 1589, 1593, 1602
- \exp_eval_register:c 1586, 1589,
1600, 1632, 1649, 1747, 1757, 1805
- \exp_eval_register:N
..... 31, 1580, 1589, 1589,
1601, 1637, 1655, 1673, 1674, 1681,
1742, 1755, 1767, 1773, 1782, 1800
- \exp_last_two_unbraced:Noo
..... 29, 1790, 1790, 7353, 7585, 7589
- \exp_last_two_unbraced_aux:noN
..... 1791, 1792
- \exp_last_unbraced:Nco
..... 1754, 1761, 5940, 6347
- \exp_last_unbraced:NcV 1754, 1763
- \exp_last_unbraced:Nf
..... 29, 1754, 1759, 3674, 6099
- \exp_last_unbraced:Nfo . 1754, 1788, 5545
- \exp_last_unbraced:NNNo 1754, 1784
- \exp_last_unbraced:NNNV 1754, 1777
- \exp_last_unbraced:NNo
.. 1754, 1775, 4952, 5908, 6328, 7559
- \exp_last_unbraced:Nno . 1754, 1786, 6390
- \exp_last_unbraced:NNV 1754, 1769
- \exp_last_unbraced:No
.. 1754, 1758, 7937, 7942, 8021, 8027
- \exp_last_unbraced:Noo
..... 1754, 1787, 6280, 6405
- \exp_last_unbraced:NV 1754, 1754
- \exp_last_unbraced:Nv 1754, 1756
- \exp_last_unbraced:Nx ... 29, 1754, 1789
- \exp_not:c 30, 1807,
1807, 1848, 1898, 2344, 2348, 2353,
2357, 2360, 2362–2364, 2369, 2371–
2373, 2378, 2380–2382, 2387, 2389–
2391, 2396, 2398, 2400, 2402, 2404,
2406, 2408, 2410, 2412–2414, 3049,
8778, 8780, 8782, 8784, 9175, 9352,
9374, 9494, 9496, 9509, 9511, 9649
- \exp_not:f 30, 1794, 1795
- \exp_not:N 30,
801, 802, 1371–1373, 1403–1405,
1555, 1591, 1807, 1848, 2284, 2349,

- 2352, 2356, 2360, 2369, 2378, 2387,
 2455, 2462, 2479, 2506, 2515, 2661,
 2665, 2670, 2675, 2680, 2687, 2693,
 2698, 2703, 2708, 2713, 2718, 2728,
 2733, 2761, 2768, 2952, 2957, 2975,
 2988, 3018, 4159, 4160, 4163, 4449,
 4456, 4466, 4468, 4501, 4502, 4780,
 4782, 4796, 4798, 4826, 4833, 5189,
 5191, 5423, 5656, 6009, 6012, 6019,
 6020, 8461, 8529, 8871, 8879, 8880,
 8882, 9175, 9494, 9496, 9509, 9511,
 9537, 9538, 9563, 9564, 9609, 9631,
 9632, 9649, 10388, 10430, 10449,
 10842, 10878, 10955, 11142, 11249,
 11259, 11812, 11825, 11912, 12110,
 12123, 12307, 12612, 12620, 12646,
 12839, 12841, 12843, 13090, 13092,
 13094, 13096, 13098, 13327, 13329,
 13331, 13333, 13335, 13337, 13339,
 13341, 13486, 13523, 13528, 13551
 \exp_not:n 30, 801,
 803, 1497, 1555, 1751, 2285, 2345,
 2455, 2462, 2479, 2506, 2515, 2953,
 2958, 2972, 2976, 3048, 3050, 4334,
 4365, 4371, 4383, 4391, 4407, 4415,
 4503, 4847, 4914, 5217, 5424, 5630,
 5633, 5636, 5746, 5869, 6013, 6018,
 6096, 6097, 6239, 6240, 6261, 8095,
 8368, 8372, 8588, 8872, 8876, 8883,
 9167, 9249, 9253, 9254, 9258, 9259,
 9540, 9634, 9672, 13353, 13460, 13463
 \exp_not:o 30, 1794, 1794, 4367, 4373,
 4383, 4385, 4387, 4389, 4391, 4393,
 4395, 4397, 4407, 4409, 4411, 4413,
 4415, 4417, 4419, 4421, 4468, 4514,
 4526, 4846, 4909, 4913, 5196, 5198,
 5249, 5703, 5705, 5862, 8744, 9354,
 9376, 9379, 9386, 9395, 9847, 9849
 \exp_not:V
 30, 1794, 1797, 4385, 4393, 4409, 4417
 \exp_not:v 30, 1794, 1802, 9566
 \exp_stop_f 30
 \exp_stop_f: 1565, 1571,
 2318, 5000, 5002, 5048, 5801, 9184
 \expandafter 12, 13, 31, 35,
 61, 62, 64, 96, 136, 138, 166, 169,
 175, 179, 183, 184, 188, 189, 191,
 201, 203, 206, 208, 213, 228, 229,
 232, 233, 264, 268, 270, 275, 277, 374
 \expl_status_pop:w 200
 \ExplFileDate
 . 49, 112, 142, 144, 334, 782, 1552,
 1916, 2428, 2546, 3266, 3993, 4322,
 5135, 5672, 6145, 6450, 7021, 8121,
 8141, 8600, 9303, 9952, 10069, 13433
 \ExplFileDescription
 113, 130, 334, 782, 1552,
 1916, 2428, 2546, 3266, 3993, 4322,
 5135, 5672, 6145, 6450, 7021, 8121,
 8141, 8600, 9303, 9952, 10069, 13433
 \ExplFileName 114, 128, 334, 782, 1552,
 1916, 2428, 2546, 3266, 3993, 4322,
 5135, 5672, 6145, 6450, 7021, 8121,
 8141, 8600, 9303, 9952, 10069, 13433
 \ExplFileVersion
 49, 115, 129, 334, 782, 1552,
 1916, 2428, 2546, 3266, 3993, 4322,
 5135, 5672, 6145, 6450, 7021, 8121,
 8141, 8600, 9303, 9952, 10069, 13433
 \ExplSyntaxNamesOff 6, 266, 273
 \ExplSyntaxNamesOn 6, 266, 266
 \ExplSyntaxOff 3, 6,
 67, 68, 178, 186, 208, 291, 296, 310, 325
 \ExplSyntaxOn 3,
 6, 67, 82, 150, 155, 160, 206, 291, 292

F
 \F 2741, 2775, 2874
 \fam 366
 \fi 22, 44, 65,
 123, 139, 174, 194, 209, 231, 265, 405
 \fi: 785, 789,
 827, 1005, 1073, 1088, 1093, 1112,
 1132, 1133, 1141, 1147, 1160, 1161,
 1169, 1175, 1236, 1316, 1327, 1353,
 1358, 1359, 1435, 1499, 1504, 1543–
 1546, 1594, 1597, 1604, 1605, 1821,
 1901, 1955, 1969, 1981, 1991, 2007,
 2008, 2020, 2021, 2033, 2042, 2296,
 2298, 2300, 2302, 2304, 2306, 2442,
 2450, 2457, 2466, 2474, 2481, 2489,
 2497, 2510, 2519, 2666, 2671, 2676,
 2681, 2688, 2694, 2699, 2704, 2709,
 2714, 2719, 2724, 2729, 2734, 2756,
 2762, 2769, 2806, 2817, 2828, 2839,
 2900, 2909, 2918, 2926, 2992, 3000,
 3022, 3286, 3297, 3308, 3323, 3329,
 3330, 3332, 3437, 3441, 3448, 3456,
 3464, 3472, 3480, 3488, 3496, 3504,
 3512, 3520, 3722, 4050, 4061, 4070,

- 4080, 4228, 4555, 4567, 4580, 4590,
 4607, 4776, 4787, 4807, 4822, 4831,
 4840, 4862, 4866, 4874, 4902, 4939,
 5245, 5248, 5275, 5351, 5355, 5375,
 5470, 6272, 6298, 6336, 6398, 6524,
 6526, 6536, 8560, 8573, 8574, 10150,
 10187, 10188, 10220, 10225, 10236,
 10248, 10265, 10289, 10292, 10302,
 10312, 10337, 10394, 10436, 10474,
 10483, 10484, 10500, 10539, 10544,
 10555, 10556, 10564, 10592, 10606,
 10611, 10620, 10691, 10692, 10708,
 10712, 10727, 10732, 10753, 10756,
 10759, 10762, 10765, 10768, 10771,
 10774, 10777, 10792, 10795, 10798,
 10801, 10804, 10807, 10810, 10813,
 10816, 10837, 10848, 10873, 10884,
 10915, 10927, 10935–10937, 10941,
 10983, 11021, 11037, 11063, 11078,
 11092, 11095, 11155, 11158, 11263,
 11264, 11298, 11301, 11328, 11329,
 11344–11346, 11356, 11389, 11394,
 11404, 11408, 11416, 11417, 11538,
 11553, 11581, 11596, 11614, 11658,
 11666, 11682–11684, 11697, 11701,
 11722, 11723, 11732, 11741, 11751,
 11777, 11778, 11799, 11821, 11828,
 11841, 11854, 11875, 11896, 11908,
 11927, 11940, 11961, 11965, 11970,
 11971, 11998, 12003, 12007, 12022,
 12055, 12061, 12069, 12073, 12074,
 12076, 12097, 12119, 12126, 12137,
 12147, 12153, 12154, 12163, 12166,
 12193, 12196, 12221, 12291, 12303,
 12314, 12332, 12341, 12342, 12345,
 12357, 12386, 12391, 12392, 12414,
 12423, 12434, 12443, 12483, 12484,
 12496, 12504, 12505, 12548, 12554,
 12562, 12566, 12567, 12569, 12615,
 12623, 12642, 12660, 12661, 12673,
 12691, 12700, 12724, 12737, 12751,
 12773, 12779, 12784, 12792, 12798,
 12801, 12852, 12859, 12874, 12888,
 12913, 12919, 12921, 12960, 12974,
 12975, 13004, 13011, 13012, 13040,
 13046, 13050, 13051, 13133, 13141,
 13192, 13196, 13200, 13204, 13223,
 13224, 13235, 13238, 13239, 13255–
 13257, 13285–13289, 13317–13321
 \file_add_path:nN
 158, 9983, 9983, 10022, 10029
 \file_add_path_search:nN 9983, 9987, 9993
 \file_if_exist:n 10020, 10020
 \file_if_exist:nTF 158
 \file_input:n 159, 10027, 10027
 \file_list 159
 \file_list: 10051, 10051
 \file_path_include:n .. 159, 10044, 10044
 \file_path_remove:n ... 159, 10044, 10049
 \finalhyphendemerits 554
 \firstmark 452
 \firstmarks 678
 \floatingpenalty 599
 \font 365
 \fontchardp 703
 \fontcharht 702
 \fontcharic 705
 \fontcharwd 704
 \fontdimen 632
 \fontname 456
 \fp_abs:c 10943
 \fp_abs:N 163, 10943, 10943, 10945
 \fp_abs_aux:NN 10943, 10943, 10944, 10947
 \fp_add:cn 10994
 \fp_add:Nn 164, 6761,
 7427, 7460, 7714, 10994, 10994, 10996
 \fp_add:NNNNNNNN
 .. 11380, 11380, 12718, 12770, 12856
 \fp_add_aux:Nn 10994, 10994, 10995, 10998
 \fp_add_core: . 10994, 11008, 11011, 11112
 \fp_add_difference: . 10994, 11020, 11065
 \fp_add_sum: 10994, 11018, 11048
 \fp_compare:n 13347, 13347
 \fp_compare:NNN 13143, 13160
 \fp_compare:nNn 13143, 13143
 \fp_compare:NNNT 7782
 \fp_compare:NNNTF
 6690, 6692, 6706, 6708,
 6713, 6826, 6828, 6871, 6891, 6909,
 6911, 6922, 6931, 6946, 7797, 7800
 \fp_compare:nNnTF
 163, 7416, 13361, 13367,
 13373, 13379, 13385, 13391, 13397
 \fp_compare:nTF 163
 \fp_compare_<: 13143
 \fp_compare_<_aux: 13143
 \fp_compare_>: 13143
 \fp_compare_absolute_a<b: 13143
 \fp_compare_absolute_a>b: 13143

\fp_compare_aux:N [13143](#), [13158](#), [13169](#), [13171](#)
\fp_compare_aux_i:w . [13347](#), [13353](#), [13357](#)
\fp_compare_aux_ii:w [13347](#), [13360](#), [13363](#)
\fp_compare_aux_iii:w [13347](#), [13366](#), [13369](#)
\fp_compare_aux_iv:w [13347](#), [13372](#), [13375](#)
\fp_compare_aux_v:w . [13347](#), [13378](#), [13381](#)
\fp_compare_aux_vi:w [13347](#), [13384](#), [13387](#)
\fp_compare_aux_vii:w [13347](#), [13390](#), [13393](#)
\fp_const:cn [10361](#)
\fp_const:Nn [160](#), [10361](#), [10361](#), [10366](#)
\fp_cos:cn [11877](#)
\fp_cos:Nn
. [165](#), [6740](#), [7636](#), [11877](#), [11877](#), [11879](#)
\fp_cos_aux:NNn [11877](#), [11877](#), [11878](#), [11881](#)
\fp_cos_aux_i: [11877](#), [11907](#), [11917](#)
\fp_cos_aux_ii: [11877](#), [11920](#), [11950](#), [12155](#)
\fp_div:cn [11228](#)
\fp_div:Nn [164](#), [6737](#), [6821](#), [6825](#),
[6869](#), [6889](#), [7414](#), [7415](#), [7436](#), [7458](#),
[7633](#), [7769](#), [7772](#), [11228](#), [11228](#), [11230](#)
\fp_div_aux:NNn [11228](#), [11228](#), [11229](#), [11232](#)
\fp_div_divide: [11228](#), [11316](#), [11331](#), [11357](#)
\fp_div_divide_aux:
..... [11228](#), [11334](#), [11343](#), [11348](#)
\fp_div_integer:NNNNN . [11523](#), [11523](#),
[11984](#), [12035](#), [12040](#), [12531](#), [12902](#)
\fp_div_internal:
.. [11228](#), [11262](#), [11267](#), [12820](#), [13078](#)
\fp_div_loop:
.. [11228](#), [11272](#), [11313](#), [11327](#), [12204](#)
\fp_div_loop_step:w [11320](#), [11374](#)
\fp_div_store: [11228](#),
[11270](#), [11317](#), [11359](#), [11363](#), [12202](#)
\fp_div_store_decimal: [11228](#), [11363](#), [11365](#)
\fp_div_store_integer:
..... [11228](#), [11270](#), [11360](#), [12202](#)
\fp_exp:cn [12271](#)
\fp_exp:Nn [164](#), [12271](#), [12271](#), [12273](#)
\fp_exp_aux: .. [12271](#), [12327](#), [12337](#), [12347](#)
\fp_exp_aux:NNn [12271](#), [12271](#), [12272](#), [12275](#)
\fp_exp_const:cx [12271](#), [12339](#), [12379](#), [12511](#)
\fp_exp_const:Nx [12271](#), [12571](#), [12576](#), [13124](#)
\fp_exp_decimal: [12271](#), [12356](#), [12444](#), [12459](#)
\fp_exp_integer: ... [12271](#), [12350](#), [12359](#)
\fp_exp_integer_const:n
..... [12271](#), [12382](#), [12388](#),
[12411](#), [12413](#), [12430](#), [12432](#), [12446](#)
\fp_exp_integer_const:nnnn
..... [12271](#), [12449](#), [12452](#), [12809](#)
\fp_exp_integer_tens:
.. [12271](#), [12366](#), [12385](#), [12390](#), [12394](#)
\fp_exp_integer_units: [12271](#), [12424](#), [12426](#)
\fp_exp_internal:
..... [12271](#), [12302](#), [12319](#), [13125](#)
\fp_exp_overflow_msg:
..... [12331](#), [12344](#), [13407](#), [13413](#)
\fp_exp_Taylor: [12271](#), [12489](#), [12523](#), [12568](#)
\fp_extended_normalise: [11540](#),
[11540](#), [11653](#), [12322](#), [12987](#), [13022](#)
\fp_extended_normalise_aux:NNNNNNNN
..... [11540](#)
\fp_extended_normalise_aux_i:
..... [11540](#), [11542](#), [11545](#), [11552](#)
\fp_extended_normalise_aux_i:w
..... [11540](#), [11550](#), [11555](#)
\fp_extended_normalise_aux_ii:
..... [11540](#), [11543](#), [11573](#), [11580](#)
\fp_extended_normalise_aux_ii:w
..... [11540](#), [11562](#), [11565](#)
\fp_extended_normalise_ii_aux:NNNNNNNN
..... [11578](#), [11583](#)
\fp_extended_normalise_output: [11606](#),
[11606](#), [11613](#), [12422](#), [12442](#), [13114](#)
\fp_extended_normalise_output_aux:N
..... [11606](#), [11634](#), [11636](#)
\fp_extended_normalise_output_aux_i:NNNNNNNN
..... [11606](#), [11611](#), [11616](#)
\fp_extended_normalise_output_aux_ii:NNNNNNNN
..... [11606](#), [11627](#), [11630](#)
\fp_gabs:c [10943](#)
\fp_gabs:N [163](#), [10943](#), [10944](#), [10946](#)
\fp_gadd:cn [10994](#)
\fp_gadd:Nn [164](#), [10994](#), [10995](#), [10997](#)
\fp_gcos:cn [11877](#)
\fp_gcos:Nn [165](#), [11877](#), [11878](#), [11880](#)
\fp_gdiv:cn [11228](#)
\fp_gdiv:Nn [164](#), [11228](#), [11229](#), [11231](#)
\fp_gexp:cn [12271](#)
\fp_gexp:Nn [164](#), [12271](#), [12272](#), [12274](#)
\fp_gln:cn [12589](#)
\fp_gln:Nn [165](#), [12589](#), [12590](#), [12592](#)
\fp_gmul:cn [11115](#)
\fp_gmul:Nn [164](#), [11115](#), [11116](#), [11118](#)
\fp_gneg:c [10968](#)
\fp_gneg:N [163](#), [10968](#), [10969](#), [10971](#)
\fp_gpow:cn [12923](#)
\fp_gpow:Nn [164](#), [12923](#), [12924](#), [12926](#)
\fp_ground_figures:cn [10824](#)

- \fp_ground_figures:Nn [162](#), [10824](#), [10827](#), [10829](#)
- \fp_ground_places:cn [10859](#)
- \fp_ground_places:Nn [162](#), [10859](#), [10862](#), [10864](#)
- \fp_gset:cn [10373](#)
- \fp_gset:Nn [161](#), [10364](#), [10373](#), [10374](#), [10406](#)
- \fp_gset_eq:cc [10457](#), [10464](#)
- \fp_gset_eq:cN [10457](#), [10462](#)
- \fp_gset_eq:Nc [10457](#), [10463](#)
- \fp_gset_eq:NN [160](#), [10457](#), [10461](#)
- \fp_gset_from_dim:cn [10407](#)
- \fp_gset_from_dim:Nn [161](#), [10407](#), [10409](#), [10454](#)
- \fp_gsin:cn [11780](#)
- \fp_gsin:Nn [165](#), [11780](#), [11781](#), [11783](#)
- \fp_gsub:cn [11097](#)
- \fp_gsub:Nn [164](#), [11097](#), [11098](#), [11100](#)
- \fp_gtan:cn [12078](#)
- \fp_gtan:Nn [165](#), [12078](#), [12079](#), [12081](#)
- \fp_gzero:c [10367](#)
- \fp_gzero:N [161](#), [10367](#), [10369](#), [10372](#)
- \fp_if_undefined:N [13127](#), [13127](#)
- \fp_if_undefined:NTF [162](#)
- \fp_if_zero:N [13135](#), [13135](#)
- \fp_if_zero:NTF [163](#)
- \fp_level_input_exponents: [10296](#), [10296](#), [11013](#)
- \fp_level_input_exponents_a: [10296](#), [10299](#), [10304](#), [10311](#)
- \fp_level_input_exponents_a:NNNNNNNN [10296](#), [10309](#), [10314](#)
- \fp_level_input_exponents_b: [10296](#), [10301](#), [10329](#), [10336](#)
- \fp_level_input_exponents_b:NNNNNNNN [10296](#), [10334](#), [10339](#)
- \fp_ln:cn [12589](#)
- \fp_ln:Nn [165](#), [12589](#), [12589](#), [12591](#)
- \fp_ln_aux: [12589](#), [12607](#), [12626](#)
- \fp_ln_aux:NNn [12589](#), [12589](#), [12590](#), [12593](#)
- \fp_ln_const:nn [12706](#), [12716](#), [12767](#), [12806](#)
- \fp_ln_error_msg: [12614](#), [12622](#), [13415](#), [13418](#)
- \fp_ln_exponent: ... [12589](#), [12641](#), [12650](#)
- \fp_ln_exponent_tens: [12589](#)
- \fp_ln_exponent_tens:NN .. [12693](#), [12703](#)
- \fp_ln_exponent_units: [12589](#), [12701](#), [12713](#)
- \fp_ln_fixed: . [12589](#), [12821](#), [12866](#), [12873](#)
- \fp_ln_fixed_aux:NNNNNNNN [12589](#), [12871](#), [12876](#)
- \fp_ln_integer_const:nn [12589](#)
- \fp_ln_internal: [12589](#), [12652](#), [12684](#), [13086](#)
- \fp_ln_mantissa: ... [12589](#), [12725](#), [12761](#)
- \fp_ln_mantissa_aux: [12589](#), [12765](#), [12786](#), [12791](#)
- \fp_ln_mantissa_divide_two: [12589](#), [12790](#), [12794](#)
- \fp_ln_normalise: [12589](#), [12717](#), [12727](#), [12734](#), [12768](#), [12854](#)
- \fp_ln_normalise_aux:NNNNNNNN [12732](#), [12739](#)
- \fp_ln_normalise_aux:NNNNNNNN .. [12589](#)
- \fp_ln_Taylor: [12589](#), [12783](#), [12812](#)
- \fp_ln_Taylor_aux: [12589](#), [12834](#), [12891](#), [12920](#)
- \fp_mul:cn [11115](#)
- \fp_mul:Nn [164](#), [6738](#), [6748](#), [6749](#), [6759](#), [6760](#), [7425](#), [7430](#), [7459](#), [7634](#), [7707](#), [7708](#), [7712](#), [7713](#), [7810](#), [7813](#), [11115](#), [11115](#), [11117](#)
- \fp_mul:NNNNNN .. [11419](#), [11419](#), [11980](#), [12027](#), [12031](#), [12527](#), [12829](#), [12894](#)
- \fp_mul:NNNNNNNN [11463](#), [11463](#), [12415](#), [12435](#), [12490](#), [13103](#)
- \fp_mul_aux:NNn [11115](#), [11115](#), [11116](#), [11119](#)
- \fp_mul_end_level: [11115](#), [11186](#), [11190](#), [11193](#), [11197](#), [11217](#), [11443](#), [11448](#), [11452](#), [11457](#), [11459](#), [11460](#), [11489](#), [11496](#), [11502](#), [11509](#), [11513](#), [11516](#), [11520](#)
- \fp_mul_end_level:NNNNNNNN [11115](#), [11221](#), [11223](#)
- \fp_mul_internal: .. [11115](#), [11129](#), [11170](#)
- \fp_mul_product:NN [11178](#)–[11180](#), [11182](#)–[11185](#), [11187](#)–[11189](#), [11191](#), [11192](#), [11196](#), [11212](#), [11431](#)–[11436](#), [11438](#)–[11442](#), [11444](#)–[11447](#), [11449](#)–[11451](#), [11455](#), [11456](#), [11458](#), [11475](#)–[11480](#), [11482](#)–[11488](#), [11490](#)–[11495](#), [11497](#)–[11501](#), [11505](#)–[11508](#), [11510](#)–[11512](#), [11514](#), [11515](#), [11519](#)
- \fp_mul_split:NNNN [11115](#), [11172](#), [11174](#), [11200](#), [11421](#), [11423](#), [11425](#), [11427](#), [11465](#), [11467](#), [11469](#), [11471](#)
- \fp_mul_split:w [11115](#)
- \fp_mul_split_aux:w [11203](#), [11209](#)
- \fp_neg:c [10968](#)
- \fp_neg:N [163](#), [10968](#), [10968](#), [10970](#)
- \fp_neg:NN [10968](#)
- \fp_neg_aux:NN [10968](#), [10969](#), [10972](#)

`\fp_new:c` [10355](#)
`\fp_new:N` [160](#), [6666–6668](#), [6678–6682](#), [6808](#), [6809](#), [7026](#),
[7045–7049](#), [7055](#), [7056](#), [7061–7064](#),
[7759](#), [7760](#), [10355](#), [10355](#), [10360](#), [10363](#)
`\fp_overflow_msg:`
..... [10224](#), [10291](#), [13399](#), [13405](#)
`\fp_pow:cn` [12923](#)
`\fp_pow:Nn` [164](#), [12923](#), [12923](#), [12925](#)
`\fp_pow_aux:NNn` [12923](#), [12923](#), [12924](#), [12927](#)
`\fp_pow_aux_i:` [12923](#), [12973](#), [12978](#)
`\fp_pow_aux_ii:` [12923](#), [12982](#), [12996](#), [13014](#)
`\fp_pow_aux_iii:` ... [12923](#), [13039](#), [13068](#)
`\fp_pow_aux_iv:` [12923](#), [13017](#),
[13031](#), [13045](#), [13049](#), [13071](#), [13080](#)
`\fp_pow_negative:` [12923](#)
`\fp_pow_positive:` [12923](#)
`\fp_read:N` [10142](#), [10142](#), [10833](#),
[10868](#), [10950](#), [10975](#), [11001](#), [11104](#),
[11122](#), [11235](#), [12930](#), [13163](#), [13168](#)
`\fp_read_aux:w` [10142](#), [10143](#), [10144](#)
`\fp_round:` ... [10836](#), [10872](#), [10895](#), [10895](#)
`\fp_round_aux:NNNNNNNN`
..... [10895](#), [10902](#), [10904](#)
`\fp_round_figures:cn` [10824](#)
`\fp_round_figures:Nn`
..... [162](#), [10824](#), [10824](#), [10826](#)
`\fp_round_figures_aux:NNn`
..... [10824](#), [10825](#), [10828](#), [10830](#)
`\fp_round_loop:N` [10895](#), [10906](#), [10917](#), [10940](#)
`\fp_round_places:cn` [10859](#)
`\fp_round_places:Nn`
..... [162](#), [10859](#), [10859](#), [10861](#)
`\fp_round_places_aux:NNn`
..... [10859](#), [10860](#), [10863](#), [10865](#)
`\fp_set:cn` [10373](#)
`\fp_set:Nn` [161](#), [6688](#), [6903](#), [6904](#),
[7632](#), [7795](#), [7796](#), [10373](#), [10373](#), [10405](#)
`\fp_set_aux:NNn` [10373](#), [10373–10375](#)
`\fp_set_eq:cc` [10457](#), [10460](#)
`\fp_set_eq:cN` [10457](#), [10458](#)
`\fp_set_eq:Nc` [10457](#), [10459](#)
`\fp_set_eq:NN` [160](#), [6736](#),
[6746](#), [6747](#), [6757](#), [6758](#), [6870](#), [6890](#),
[7705](#), [7706](#), [7710](#), [7711](#), [10457](#), [10457](#)
`\fp_set_from_dim:cn` [10407](#)
`\fp_set_from_dim:Nn` [161](#), [6744](#),
[6745](#), [6755](#), [6756](#), [6819](#), [6820](#), [6822](#),
[6823](#), [6866](#), [6867](#), [6887](#), [6888](#), [7410–](#)
[7413](#), [7420](#), [7421](#), [7424](#), [7429](#), [7452–](#)
[7456](#), [7703](#), [7704](#), [7767](#), [7768](#), [7770](#),
[7771](#), [7809](#), [7812](#), [10407](#), [10407](#), [10453](#)
`\fp_set_from_dim_aux:NNn`
..... [10407](#), [10408](#), [10410](#), [10411](#)
`\fp_set_from_dim_aux:w`
.. [10407](#), [10418](#), [10447](#), [10449](#), [10452](#)
`\fp_show:c` [10465](#), [10466](#)
`\fp_show:N` [161](#), [10465](#), [10465](#)
`\fp_sin:cn` [11780](#)
`\fp_sin:Nn`
.. [165](#), [6739](#), [7635](#), [11780](#), [11780](#), [11782](#)
`\fp_sin_aux:NNn` [11780](#), [11780](#), [11781](#), [11784](#)
`\fp_sin_aux_i:` [11780](#), [11820](#), [11831](#)
`\fp_sin_aux_ii:` [11780](#), [11834](#), [11864](#), [12178](#)
`\fp_split:Nn` [10155](#),
[10155](#), [10378](#), [10416](#), [11002](#), [11105](#),
[11123](#), [11236](#), [11787](#), [11884](#), [12085](#),
[12278](#), [12596](#), [12935](#), [13146](#), [13152](#)
`\fp_split_aux_i:w` .. [10155](#), [10194](#), [10198](#)
`\fp_split_aux_ii:w` .. [10155](#), [10199](#), [10200](#)
`\fp_split_aux_iii:w` . [10155](#), [10201](#), [10202](#)
`\fp_split_decimal:w` . [10155](#), [10205](#), [10208](#)
`\fp_split_decimal_aux:w`
..... [10155](#), [10209](#), [10210](#)
`\fp_split_exponent:` [10155](#)
`\fp_split_exponent:w` [10163](#), [10190](#)
`\fp_split_sign:`
.. [10155](#), [10161](#), [10165](#), [10176](#), [10186](#)
`\fp_standardise:NNNN` [10227](#),
[10227](#), [10379](#), [10421](#), [11003](#), [11023](#),
[11106](#), [11124](#), [11134](#), [11237](#), [11277](#),
[11788](#), [11842](#), [11885](#), [11928](#), [12086](#),
[12173](#), [12182](#), [12209](#), [12279](#), [12506](#),
[12597](#), [12662](#), [12936](#), [13147](#), [13153](#)
`\fp_standardise_aux:`
..... [10227](#), [10241](#), [10247](#),
[10257](#), [10258](#), [10264](#), [10281](#), [10294](#)
`\fp_standardise_aux:NNNN`
..... [10227](#), [10235](#), [10239](#)
`\fp_standardise_aux:w` [10227](#),
[10245](#), [10251](#), [10263](#), [10268](#), [10295](#)
`\fp_sub:cn` [11097](#)
`\fp_sub:Nn` [164](#), [6750](#), [7422](#), [7432](#),
[7434](#), [7457](#), [7709](#), [11097](#), [11097](#), [11099](#)
`\fp_sub:NNNNNNNN` [11396](#), [11396](#),
[11735](#), [11756](#), [11767](#), [12772](#), [12858](#)
`\fp_sub_aux:NNn` [11097](#), [11097](#), [11098](#), [11101](#)
`\fp_tan:cn` [12078](#)

\fp_tan:Nn [165](#), [12078](#), [12078](#), [12080](#)
\fp_tan_aux:NNn [12078](#), [12078](#), [12079](#), [12082](#)
\fp_tan_aux_i: [12078](#), [12118](#), [12129](#)
\fp_tan_aux_ii: [12078](#), [12132](#), [12139](#)
\fp_tan_aux_iii: [12078](#), [12162](#), [12165](#), [12168](#)
\fp_tan_aux_iv: [12078](#), [12192](#), [12195](#), [12198](#)
\fp_tmp:w [10354](#),
[10354](#), [10385](#), [10403](#), [10427](#), [10445](#),
[10839](#), [10857](#), [10875](#), [10893](#), [10952](#),
[10966](#), [11009](#), [11028](#), [11113](#), [11139](#),
[11168](#), [11246](#), [11256](#), [11265](#), [11282](#),
[11809](#), [11822](#), [11829](#), [11909](#), [11915](#),
[12107](#), [12120](#), [12127](#), [12304](#), [12317](#),
[12609](#), [12617](#), [12624](#), [12643](#), [12835](#),
[12846](#), [12949](#), [12955](#), [12966](#), [12976](#),
[12999](#), [13006](#), [13052](#), [13087](#), [13101](#)
\fp_to_dim:c [10530](#)
\fp_to_dim:N
[161](#), [6751](#), [6762](#), [7439](#), [7461](#), [7715](#),
[7716](#), [7811](#), [7814](#), [10530](#), [10530](#), [10531](#)
\fp_to_int:c [10532](#)
\fp_to_int:N [162](#), [10532](#), [10532](#), [10534](#)
\fp_to_int_aux:w ... [10532](#), [10533](#), [10535](#)
\fp_to_int_large:w .. [10532](#), [10543](#), [10558](#)
\fp_to_int_large_aux:nnn
[10532](#), [10570](#), [10572](#), [10574](#), [10576](#),
[10578](#), [10580](#), [10582](#), [10584](#), [10586](#)
\fp_to_int_large_aux_1:w [10532](#)
\fp_to_int_large_aux_2:w [10532](#)
\fp_to_int_large_aux_3:w [10532](#)
\fp_to_int_large_aux_4:w [10532](#)
\fp_to_int_large_aux_5:w [10532](#)
\fp_to_int_large_aux_6:w [10532](#)
\fp_to_int_large_aux_7:w [10532](#)
\fp_to_int_large_aux_8:w [10532](#)
\fp_to_int_large_aux_i:w
..... [10532](#), [10561](#), [10567](#)
\fp_to_int_large_aux_ii:w
..... [10532](#), [10563](#), [10594](#)
\fp_to_int_none:w [10532](#)
\fp_to_int_small:w .. [10532](#), [10541](#), [10547](#)
\fp_to_tl:c [10599](#)
\fp_to_tl:N [162](#), [10599](#), [10599](#), [10601](#)
\fp_to_tl_aux:w [10599](#), [10600](#), [10602](#)
\fp_to_tl_large:w .. [10599](#), [10610](#), [10614](#)
\fp_to_tl_large_0:w [10599](#)
\fp_to_tl_large_1:w [10599](#)
\fp_to_tl_large_2:w [10599](#)
\fp_to_tl_large_3:w [10599](#)
\fp_to_tl_large_4:w [10599](#)
\fp_to_tl_large_5:w [10599](#)
\fp_to_tl_large_6:w [10599](#)
\fp_to_tl_large_7:w [10599](#)
\fp_to_tl_large_8:w [10599](#)
\fp_to_tl_large_8_aux:w [10599](#)
\fp_to_tl_large_9:w [10599](#)
\fp_to_tl_large_aux_i:w
..... [10599](#), [10617](#), [10623](#)
\fp_to_tl_large_aux_ii:w
..... [10599](#), [10619](#), [10625](#)
\fp_to_tl_large_zeros:NNNNNNNN
..... [10599](#), [10628](#), [10634](#),
[10639](#), [10644](#), [10649](#), [10654](#), [10659](#),
[10664](#), [10669](#), [10679](#), [10737](#), [10740](#)
\fp_to_tl_small:w .. [10599](#), [10608](#), [10682](#)
\fp_to_tl_small_aux:w [10599](#), [10690](#), [10734](#)
\fp_to_tl_small_one:w [10599](#), [10685](#), [10695](#)
\fp_to_tl_small_two:w [10599](#), [10688](#), [10714](#)
\fp_to_tl_small_zeros:NNNNNNNN
..... [10599](#),
[10702](#), [10711](#), [10721](#), [10731](#), [10779](#)
\fp_trig_calc_cos: [11870](#),
[11872](#), [11954](#), [11960](#), [11973](#), [11973](#)
\fp_trig_calc_sin: [11868](#),
[11874](#), [11956](#), [11958](#), [11973](#), [12009](#)
\fp_trig_calc_Taylor:
.. [11973](#), [12006](#), [12021](#), [12024](#), [12075](#)
\fp_trig_normalise:
.. [11649](#), [11649](#), [11833](#), [11919](#), [12141](#)
\fp_trig_normalise_aux:
.. [11649](#), [11654](#), [11668](#), [11673](#), [11681](#)
\fp_trig_octant: ... [11659](#), [11725](#), [11725](#)
\fp_trig_octant_aux_i:
.. [11725](#), [11728](#), [11743](#), [11763](#), [11776](#)
\fp_trig_octant_aux_ii:
..... [11725](#), [11750](#), [11753](#)
\fp_trig_overflow_msg:
..... [11665](#), [12136](#), [13421](#), [13427](#)
\fp_trig_sub:NNN [11649](#), [11671](#), [11677](#), [11686](#)
\fp_use:c [10467](#)
\fp_use:N [161](#), [6845](#), [6847](#),
[6851](#), [6853](#), [10467](#), [10467](#), [10469](#), [10530](#)
\fp_use_aux:w [10467](#), [10468](#), [10470](#)
\fp_use_i_to_iix:NNNNNNNN
..... [10599](#), [10699](#), [10704](#), [10822](#)
\fp_use_i_to_vii:NNNNNNNN
..... [10599](#), [10718](#), [10723](#), [10820](#)
\fp_use_iix_ix:NNNNNNNN
..... [10599](#), [10716](#), [10818](#)
\fp_use_ix:NNNNNNNN [10599](#), [10697](#), [10819](#)

- \fp_use_large:w [10467](#), [10476](#), [10494](#)
 - \fp_use_large_aux_1:w [10467](#)
 - \fp_use_large_aux_2:w [10467](#)
 - \fp_use_large_aux_3:w [10467](#)
 - \fp_use_large_aux_4:w [10467](#)
 - \fp_use_large_aux_5:w [10467](#)
 - \fp_use_large_aux_6:w [10467](#)
 - \fp_use_large_aux_7:w [10467](#)
 - \fp_use_large_aux_8:w [10467](#)
 - \fp_use_large_aux_i:w [10467](#), [10497](#), [10503](#)
 - \fp_use_large_aux_ii:w [10467](#), [10499](#), [10524](#)
 - \fp_use_none:w [10467](#), [10482](#), [10487](#)
 - \fp_use_small:w [10467](#), [10480](#), [10488](#)
 - \fp_zero:c [10367](#)
 - \fp_zero:N [161](#), [10367](#), [10367](#), [10371](#)
 - \frozen@everydisplay [767](#)
 - \frozen@everymath [768](#)
 - \futurelet [361](#)
- G**
- \G [2780](#)
 - \g [2309](#)
 - \g_cctab_allocate_int [13465](#),
[13465](#), [13466](#), [13472](#), [13474](#), [13476](#)
 - \g_cctab_stack_int [13465](#), [13467](#),
[13504](#), [13505](#), [13507](#), [13508](#), [13512](#)
 - \g_cctab_stack_seq
.. [13465](#), [13468](#), [13502](#), [13513](#), [13515](#)
 - \g_file_current_name_tl
.. [158](#), [9955](#), [9955](#),
[9960](#), [9964](#), [9972](#), [10038](#), [10039](#), [10041](#)
 - \g_file_record_seq [159](#), [9967](#),
[9967](#), [9972](#), [10033](#), [10053](#), [10055](#), [10062](#)
 - \g_file_stack_seq
.. [159](#), [9966](#), [9966](#), [10038](#), [10041](#)
 - \g_file_test_ior [9983](#),
[9985](#), [9986](#), [9989](#), [10007](#), [10008](#), [10018](#)
 - \g_ior_streams_prop [8168](#),
[8169](#), [8174](#), [8205](#), [8305](#), [8320](#), [8341](#)
 - \g_ior_tmp_ior [8285](#), [8286](#), [8288](#)
 - \g_ior_tmp_stream [8240](#)
 - \g_iow_streams_prop
.. [8168](#), [8168](#), [8171](#)–[8173](#),
[8218](#), [8226](#), [8234](#), [8269](#), [8333](#), [8343](#)
 - \g_iow_tmp_iow [8249](#), [8250](#), [8252](#)
 - \g_iow_tmp_stream [8240](#)
 - \g_keyval_level_int [9306](#), [9306](#),
[9353](#), [9375](#), [9399](#), [9401](#), [9403](#), [9405](#)
 - \g_peek_token [55](#), [2928](#), [2929](#), [2939](#)
 - \g_prg_map_int [2275](#), [2281](#), [2291](#), [2293](#),
[2341](#), [2341](#), [4642](#), [4643](#), [4646](#), [4648](#),
[5402](#), [5404](#), [5408](#), [5410](#), [5938](#), [5939](#),
[5941](#), [5943](#), [6344](#), [6345](#), [6348](#), [6351](#)
 - \g_scan_marks_tl [2527](#), [2527](#), [2530](#), [2536](#)
 - \g_tmpa_bool [36](#), [1961](#), [1962](#)
 - \g_tmpa_clist [110](#), [5994](#), [5996](#)
 - \g_tmpa_dim [76](#), [4182](#), [4185](#)
 - \g_tmpa_int [69](#), [3940](#), [3943](#)
 - \g_tmpa_skip [78](#), [4258](#), [4261](#)
 - \g_tmpa_tl [93](#), [4892](#), [4892](#)
 - \g_tmpb_clist [110](#), [5994](#), [5997](#)
 - \g_tmpb_dim [76](#), [4182](#), [4186](#)
 - \g_tmpb_int [69](#), [3940](#), [3944](#)
 - \g_tmpb_skip [78](#), [4258](#), [4262](#)
 - \g_tmpb_tl [93](#), [4892](#), [4893](#)
 - \gdef [352](#)
 - \GetIdInfo [6](#), [97](#), [98](#)
 - \GetIdInfoAuxCVS [97](#), [136](#), [141](#)
 - \GetIdInfoAuxI [97](#), [102](#), [104](#)
 - \GetIdInfoAuxII [97](#), [121](#), [126](#)
 - \GetIdInfoAuxIII [97](#), [131](#), [133](#)
 - \GetIdInfoAuxSVN [97](#), [138](#), [143](#)
 - \GetIdInfoFull [97](#)
 - \global [336](#), [367](#)
 - \globaldefs [371](#)
 - \glueexpr [711](#)
 - \glueshrink [714](#)
 - \glueshrinkorder [716](#)
 - \gluestretch [713](#)
 - \gluestretchorder [715](#)
 - \gluetomu [717](#)
 - \group_align_safe_begin [41](#)
 - \group_align_safe_begin: [1973](#),
[2303](#), [2303](#), [2960](#), [2978](#), [4497](#), [4919](#)
 - \group_align_safe_end [41](#)
 - \group_align_safe_end:
.. [2070](#), [2071](#), [2303](#), [2305](#),
[2942](#), [2952](#), [2957](#), [2975](#), [4506](#), [4945](#)
 - \group_begin [9](#)
 - \group_begin: [810](#), [811](#), [832](#), [881](#), [1094](#),
[1860](#), [2308](#), [2323](#), [2646](#), [2659](#), [2683](#),
[2736](#), [2773](#), [2870](#), [3036](#), [3133](#), [3140](#),
[4430](#), [4448](#), [4598](#), [5268](#), [6572](#), [6687](#),
[6814](#), [6861](#), [6882](#), [6902](#), [8124](#), [8395](#),
[8400](#), [8429](#), [8685](#), [8726](#), [9157](#), [9311](#),
[9321](#), [10377](#), [10413](#), [10832](#), [10867](#),
[10949](#), [10974](#), [11000](#), [11103](#), [11121](#),
[11234](#), [11786](#), [11883](#), [12084](#), [12277](#),

12595, 12814, 12929, 12986, 13020,
 13082, 13145, 13162, 13349, 13541
 \group_end 9
 \group_end: 810,
 812, 832, 884, 1099, 1872, 2313,
 2328, 2658, 2662, 2690, 2744, 2784,
 2876, 3101, 3142, 3151, 4437, 4455,
 4602, 4605, 5279, 6577, 6697, 6833,
 6874, 6894, 6916, 8128, 8399, 8419,
 8463, 8690, 8732, 9179, 9318, 9329,
 10387, 10429, 10841, 10877, 10954,
 10991, 11030, 11141, 11248, 11258,
 11284, 11811, 11824, 11911, 12109,
 12122, 12306, 12611, 12619, 12645,
 12837, 12951, 12957, 12968, 12992,
 12998, 13001, 13008, 13025, 13033,
 13042, 13054, 13089, 13176, 13187,
 13190, 13194, 13198, 13202, 13214,
 13218, 13221, 13230, 13233, 13244,
 13248, 13262, 13266, 13270, 13275,
 13280, 13283, 13294, 13298, 13302,
 13307, 13312, 13315, 13352, 13544
 \group_execute_after:N 1538
 \group_insert_after:N . 9, 815, 815, 1538

H

\H 2780
 \halign 378
 \hangafter 556
 \hangindent 557
 \hbadness 618
 \hbox 613
 \hbox:n . 123, 6583, 6583, 6721, 7904, 7959
 \hbox_gset:cn 6584
 \hbox_gset:cw 6594, 6606
 \hbox_gset:Nn 124, 6584, 6585, 6587
 \hbox_gset:Nw . 124, 6594, 6596, 6599, 6605
 \hbox_gset_end 124
 \hbox_gset_end: 6594, 6601, 6607
 \hbox_gset_inline_begin:c ... 6602, 6606
 \hbox_gset_inline_begin:N ... 6602, 6605
 \hbox_gset_inline_end: 6602, 6607
 \hbox_gset_to_wd:cn 6588
 \hbox_gset_to_wd:Nnn 124, 6588, 6590, 6593
 \hbox_overlap_left:n ... 124, 6611, 6611
 \hbox_overlap_right:n 124, 6611, 6613, 6941
 \hbox_set:cn 6584
 \hbox_set:cw 6594, 6603
 \hbox_set:Nn 124,
 6584, 6584-6586, 6685, 6717, 6718,

6812, 6859, 6880, 6900, 6938, 6961,
 6966, 6974, 6984, 6996, 7003, 7010,
 7113, 7210, 7467, 7538, 7647, 8050
 \hbox_set:Nw
 124, 6594, 6594, 6597, 6598, 6602, 7157
 \hbox_set_end 124
 \hbox_set_end: ... 6594, 6600, 6604, 7161
 \hbox_set_inline_begin:c 6602, 6603
 \hbox_set_inline_begin:N 6602, 6602
 \hbox_set_inline_end: 6602, 6604
 \hbox_set_to_wd:cn 6588
 \hbox_set_to_wd:Nnn
 124, 6588, 6588, 6591, 6592
 \hbox_to_wd:nn 124, 6608, 6608, 6948
 \hbox_to_zero:n 124, 6608, 6610, 6612, 6614
 \hbox_unpack:c 6615
 \hbox_unpack:N
 ... 124, 6615, 6615, 6617, 7471, 7620
 \hbox_unpack_clear:c 6615
 \hbox_unpack_clear:N 125, 6615, 6616, 6618
 \hcoffin_set:cn 7109
 \hcoffin_set:cw 7153
 \hcoffin_set:Nn 128, 7109,
 7109, 7125, 7901, 7913, 7956, 7997
 \hcoffin_set:Nw ... 128, 7153, 7153, 7169
 \hcoffin_set_end 128
 \hcoffin_set_end: 7153, 7158, 7168
 \Height 7250, 7252, 7256, 7260, 7267
 \hfil 521
 \hfill 523
 \hfilneg 522
 \hfuzz 620
 \hoffset 595
 \holdinginserts 598
 \hrule 534
 \hsize 559
 \hskip 524
 \hss 525
 \ht 663
 \hyphenation 649
 \hyphenchar 633
 \hyphenpenalty 551

I

\I 2780
 \if 184, 387
 \if:w 22, 785,
 791, 1003, 1071, 1088, 1819, 2913,
 3018, 3434, 10146, 10472, 10537, 10604
 \if_bool:N 42, 1919, 1919

- `\if_box_empty:N` . . . 127, 6520, 6522, 6536
- `\if_case:w` 70, 1339, 3269, 3274, 3694, 11866, 11952
- `\if_catcode:w` 22, 785, 793, 2665, 2670, 2675, 2680, 2687, 2693, 2698, 2703, 2708, 2713, 2718, 2728, 2761, 2987, 4795, 4833, 4851, 8555
- `\if_charcode:w` 22, 785, 792, 2733, 4773, 4779, 4826
- `\if_cs_exist:N` 22, 799, 799, 1128, 1156, 2922
- `\if_cs_exist:w` 799, 800, 823, 1137, 1165, 1312, 11815, 11905, 12113, 12300, 12309, 12639
- `\if_dim:w` 80, 3996, 3996, 4048, 4060, 4085–4091
- `\if_eof:w` 138, 8144, 8144, 8569
- `\if_false` 22
- `\if_false:` 785, 786, 2304, 4866, 4874, 5245, 5248, 5351, 5355
- `\if_hbox:N` 127, 6520, 6520, 6524
- `\if_int_compare:w` 70, 813, 813, 1497, 1503, 2304, 2306, 2454, 2461, 2478, 2505, 2514, 2752, 3269, 3284, 3292, 3303, 3314, 3318, 3319, 3325, 3444, 3452, 3460, 3468, 3476, 3484, 3492, 3500, 4222, 4898, 8566, 10167, 10178, 10213, 10221, 10229, 10243, 10260, 10282, 10283, 10298, 10306, 10331, 10390, 10432, 10475, 10478, 10496, 10540, 10549, 10551, 10560, 10588, 10607, 10616, 10684, 10687, 10697, 10698, 10716, 10717, 10742–10750, 10781–10789, 10835, 10844, 10871, 10880, 10910, 10919, 10923, 10932, 10933, 10939, 10979, 11014, 11033, 11059, 11075, 11079, 11081, 11144, 11148, 11242, 11252, 11287, 11291, 11322, 11325, 11333, 11336, 11338, 11353, 11385, 11390, 11401, 11405, 11409, 11410, 11535, 11547, 11575, 11586, 11608, 11651, 11655, 11670, 11675, 11676, 11694, 11698, 11702, 11704, 11729, 11745, 11755, 11765, 11795, 11808, 11835, 11850, 11892, 11921, 11936, 11962, 11963, 11967, 11975, 11989, 11990, 12012, 12045, 12046, 12050, 12056, 12066, 12070, 12093, 12106, 12131, 12142, 12143, 12149, 12156, 12157, 12187, 12188, 12217, 12287, 12321, 12323, 12324, 12334, 12349, 12361, 12377, 12378, 12400, 12410, 12428, 12429, 12461, 12462, 12468, 12497, 12500, 12535, 12539, 12543, 12549, 12559, 12563, 12602, 12603, 12653, 12656, 12669, 12686, 12692, 12715, 12729, 12741, 12766, 12769, 12780, 12788, 12848, 12855, 12868, 12878, 12898, 12908, 12914, 12941, 12945, 12962, 12980, 12985, 12988, 13016, 13019, 13023, 13024, 13182–13185, 13208, 13211, 13217, 13226, 13229, 13243, 13247, 13251, 13261, 13265, 13269, 13273, 13278, 13293, 13297, 13301, 13305, 13310
- `\if_int_odd:w` 70, 3269, 3273, 3508, 3516, 11733, 12796, 12799
- `\if_meaning:w` 22, 785, 794, 1108, 1125, 1143, 1153, 1171, 1323, 1434, 1591, 1592, 1899, 1951, 1981, 1991, 2000, 2003, 2013, 2016, 2029, 2038, 2440, 2446, 2472, 2485, 2493, 2723, 2768, 2801, 2812, 2823, 2834, 2896, 2996, 3441, 4080, 4551, 4563, 4575, 4586, 4601, 4818, 4937, 5273, 5372, 5467, 6268, 6294, 6334, 6396, 13129, 13137
- `\if_mode_horizontal` 23
- `\if_mode_horizontal:` 795, 796, 2298
- `\if_mode_inner` 23
- `\if_mode_inner:` 795, 798, 2300
- `\if_mode_math` 23
- `\if_mode_math:` 795, 795, 2302
- `\if_mode_vertical` 23
- `\if_mode_vertical:` 795, 797, 2296
- `\if_num:w` 70, 2904, 3269, 3272
- `\if_predicate:w` 41, 1919, 1920, 1965
- `\if_true` 22
- `\if_true:` 785, 785
- `\if_vbox:N` 127, 6520, 6521, 6526
- `\ifcase` 388
- `\ifcat` 389
- `\ifcsname` 672
- `\ifdefined` 671
- `\ifdim` 392
- `\ifeof` 393
- `\iffalse` 398
- `\iffontchar` 701
- `\ifhbox` 394
- `\ifhmode` 400
- `\ifinner` 403

<code>\ifmmode</code>	401	<code>\int_decr:N</code>	62, 3409, 3411, 3416, 3418
<code>\ifnum</code>	390	<code>\int_div_round:nn</code>	60, 3311, 3336
<code>\ifodd</code>	169, 205, 391	<code>\int_div_truncate:nn</code>	
<code>\iftrue</code>	399	61, 3311, 3311, 3340, 3596, 3686
<code>\ifvbox</code>	395	<code>\int_do_until:nn</code> ...	64, 3522, 3544, 3548
<code>\ifvmode</code>	402	<code>\int_do_until:nNnn</code> ..	64, 3550, 3572, 3576
<code>\ifvoid</code>	396	<code>\int_do_while:nn</code>	64, 3522, 3538
<code>\ifx</code>	13, 62, 108, 135, 229, 233, 397	<code>\int_do_while:nNnn</code>	
<code>\ignorespaces</code>	445	63, 3542, 3550, 3566, 3570
<code>\immediate</code>	407	<code>\int_eval:n</code>	60, 1365,
<code>\indent</code>	541	2144, 2260, 2268, 3275, 3276, 3279,	
<code>\initcatcodetable</code>	760	3336, 3578, 3664, 3733, 3743, 3802,	
<code>\input</code>	415	3816, 3820, 3823, 3838, 3847, 4682,	
<code>\input@path</code>	9997, 10000, 10015	4687, 5007, 5060, 5076, 5535, 5547,	
<code>\inputlineno</code>	417	5564, 6000, 6009, 6037, 6050, 6072	
<code>\insert</code>	597	<code>\int_eval:w</code>	
<code>\insertpenalties</code>	600	.	70, 1339, 2206, 2550, 2552, 2554,
<code>\int_abs:n</code>	60, 3281, 3281	2620, 2622, 2624, 2626, 2628, 2630,	
<code>\int_add:cn</code>	3397	2632, 2634, 2636, 2638, 2640, 2642,	
<code>\int_add:Nn</code>	62, 3397,	3269, 3270, 3276, 3279, 3284, 3287,	
3397, 3402, 3405, 8477, 8490, 8528		3291, 3293, 3302, 3304, 3313, 3314,	
<code>\int_compare:n</code>	3428, 3428	3318, 3319, 3325, 3339, 3363, 3398,	
<code>\int_compare:nF</code>	3532, 3547	3400, 3422, 3429, 3444, 3452, 3460,	
<code>\int_compare:nNn</code>	3498, 3498	3468, 3476, 3484, 3492, 3500, 3508,	
<code>\int_compare:nNnF</code>	2264,	3516, 3694, 3721, 3876, 3920, 8548,	
3560, 3575, 8316, 8318, 8329, 8331		10193, 10196, 10214, 10230, 10591,	
<code>\int_compare:nNnT</code>		10699, 10703, 10718, 10722, 10921,	
.....	3552, 3569, 4163, 5062,	10922, 11015, 11041, 11052, 11056,	
5549, 8247, 8265, 8283, 8301, 13505		11068, 11072, 11085, 11089, 11131,	
<code>\int_compare:nNnTF</code>	63,	11145, 11149, 11162, 11215, 11243,	
2098, 2150, 2250, 2253, 3353, 3355,		11253, 11274, 11288, 11292, 11305,	
3581, 3667, 3673, 3820, 3844, 3848,		11323, 11368, 11377, 11382–11384,	
3898, 5074, 5562, 6033, 6035, 6040,		11398–11400, 11410, 11414, 11415,	
6048, 6068, 8201, 8214, 8478, 13473		11527, 11534, 11559, 11569, 11689,	
<code>\int_compare:nT</code>	3524, 3541	11691, 11693, 11705, 11711, 11715,	
<code>\int_compare:nTF</code>	63	11719, 11803, 11858, 11900, 11944,	
<code>\int_compare_<:w</code>	3428	12101, 12206, 12225, 12295, 12374,	
<code>\int_compare_=:w</code>	3428	12407, 12470, 12476, 12480, 12517,	
<code>\int_compare_>:w</code>	3428	12536, 12604, 12634, 12677, 12781,	
<code>\int_compare_aux:Nw</code>	3428, 3432, 3440	12899, 12942, 12946, 12963, 12989,	
<code>\int_compare_aux:nw</code>	3428, 3429, 3430	13061, 13111, 13208, 13211, 13226	
<code>\int_compare_p:nNn</code>	4234, 4235	<code>\int_eval_end</code>	70
<code>\int_const:cn</code>	3351, 3857–3870	<code>\int_eval_end:</code>	1339, 2206, 2550, 2552,
<code>\int_const:Nn</code>	61, 3351,	2554, 2620, 2622, 2624, 2626, 2628,	
3351, 3371, 3921–3939, 10072–10076		2630, 2632, 2634, 2636, 2638, 2640,	
<code>\int_constdef:Nw</code> .	3351, 3362, 3374, 3378	2642, 3269, 3271, 3276, 3279, 3287,	
<code>\int_convert_from_base_ten:nn</code>	3945, 3946	3293, 3298, 3304, 3309, 3334, 3341,	
<code>\int_convert_to_base_ten:nn</code> .	3945, 3948	3363, 3398, 3400, 3422, 3444, 3452,	
<code>\int_convert_to_symbols:nnn</code> .	3945, 3947	3460, 3468, 3476, 3484, 3492, 3500,	
<code>\int_decr:c</code>	3409	3508, 3516, 3694, 3721, 3920, 8551,	

- 10591, 10705, 10724, 11307, 11371,
 11377, 11382–11384, 11398–11400,
 11414, 11415, 11527, 11534, 11689,
 11691, 11693, 11713, 11717, 11721,
 12208, 12376, 12409, 12472, 12482
 \int_from_alph:n 67, 3800, 3800
 \int_from_alph_aux:N ... 3800, 3816, 3819
 \int_from_alph_aux:n ... 3800, 3805, 3808
 \int_from_alph_aux:nN
 3800, 3809, 3810, 3815
 \int_from_base:nn
 67, 3821, 3821, 3852, 3854, 3856, 3948
 \int_from_base_aux:N ... 3821, 3838, 3842
 \int_from_base_aux:nn .. 3821, 3826, 3830
 \int_from_base_aux:nnN
 3821, 3831, 3832, 3837
 \int_from_binary:n 67, 3851, 3851
 \int_from_hexadecimal:n . 67, 3851, 3853
 \int_from_octal:n 67, 3851, 3855
 \int_from_roman:n 67, 3871, 3871
 \int_from_roman_aux:NN
 3871, 3877, 3880, 3905, 3909
 \int_from_roman_clean_up:w
 3871, 3888, 3895, 3897, 3916
 \int_from_roman_end:w .. 3871, 3875, 3914
 \int_gadd:cn 3397
 \int_gadd:Nn
 .. 62, 3397, 3401, 3406, 13472, 13504
 \int_gdecr:c 3409
 \int_gdecr:N 62, 2293, 3409, 3415,
 3420, 4648, 5408, 5943, 6351, 9405
 \int_get_digits:n 69, 3766, 3771, 3805, 3827
 \int_get_sign:n 69, 3766, 3766, 3804, 3825
 \int_get_sign_and_digits_aux:nNNN ..
 3766, 3768, 3773, 3776, 3799
 \int_get_sign_and_digits_aux:oNNN ..
 3766, 3782, 3786, 3792
 \int_gincr:c 3409
 \int_gincr:N 62, 2291, 3409, 3413,
 3419, 4642, 5404, 5938, 6344, 9399
 \int_gset:cn 3421
 \int_gset:Nn 62, 3358, 3368, 3421, 3423, 3425
 \int_gset_eq:cc 3391
 \int_gset_eq:cN 3391
 \int_gset_eq:Nc 3391
 \int_gset_eq:NN 61, 3391, 3394–3396
 \int_gsub:cn 3397
 \int_gsub:Nn .. 62, 3397, 3403, 3408, 13512
 \int_gzero:c 3381
 \int_gzero:N ... 61, 3381, 3382, 3384, 3388
 \int_gzero_new:c 3385
 \int_gzero_new:N ... 61, 3385, 3387, 3390
 \int_if_even:n 3506, 3514
 \int_if_even:nTF 63
 \int_if_odd:n 3506, 3506
 \int_if_odd:nTF 63
 \int_incr:c 3409
 \int_incr:N 62, 3409, 3409, 3414, 3417,
 8264, 8300, 8496, 9542, 9569, 9636
 \int_max:nn 61, 3281, 3289
 \int_min:nn 61, 3281, 3300
 \int_mod:nn 61, 3311, 3337, 3586, 3677
 \int_new:c 3343
 \int_new:N 61, 2341, 3343,
 3344, 3350, 3357, 3367, 3386, 3388,
 3940–3944, 8176, 8383, 8385–8388,
 9306, 9418, 10077, 10079, 10081,
 10083, 10085, 10087, 10095–10123,
 10125–10129, 10132, 10133, 10135,
 10136, 10138–10141, 13465, 13467
 \int_set:cn 3421
 \int_set:Nn 62,
 3421, 3421, 3423, 3424, 6573, 6574,
 8199, 8212, 8228, 8236, 8250, 8286,
 8384, 8430, 8443, 8475, 8503, 8762,
 9538, 9564, 9632, 10078, 10080,
 10082, 10084, 10086, 10088, 10834,
 10869, 13327, 13329, 13331, 13333,
 13335, 13337, 13339, 13341, 13466
 \int_set_eq:cc 3391
 \int_set_eq:cN 3391
 \int_set_eq:Nc 3391
 \int_set_eq:NN
 61, 3391, 3391–3393, 6575, 8401, 8437
 \int_show:c 3917, 3918
 \int_show:N 68, 3917, 3917
 \int_show:n 68, 3919, 3919
 \int_sub:cn 3397
 \int_sub:Nn 62, 3397, 3399, 3404, 3407, 8534
 \int_to_Alph:n 65, 3599, 3631
 \int_to_alph:n 65, 3599, 3599
 \int_to_arabic:n 65, 3578, 3578
 \int_to_base:nn
 66, 3663, 3663, 3725, 3727, 3729, 3946
 \int_to_base_aux_i:nn .. 3663, 3664, 3665
 \int_to_base_aux_ii:nnN
 3663, 3668, 3669, 3671, 3685
 \int_to_base_aux_iii:nnnN 3663, 3676, 3683
 \int_to_binary:n 66, 3724, 3724
 \int_to_hexadecimal:n ... 66, 3724, 3726

- \int_to_letter:n [69](#), [3663](#), [3674](#), [3677](#), [3691](#)
- \int_to_octal:n [66](#), [3724](#), [3728](#)
- \int_to_Roman:n [67](#), [3730](#), [3740](#)
- \int_to_roman:n [67](#), [3730](#), [3730](#)
- \int_to_roman:w
 - [69](#), [813](#), [814](#), [918](#), [920](#), [1087](#), [2053](#),
 - [2058](#), [2204](#), [3269](#), [3433](#), [3733](#), [3743](#)
- \int_to_Roman_aux:N [3742](#), [3745](#), [3748](#)
- \int_to_roman_aux:N [3730](#), [3732](#), [3735](#), [3738](#)
- \int_to_Roman_c:w [3730](#), [3762](#)
- \int_to_roman_c:w [3730](#), [3754](#)
- \int_to_Roman_d:w [3730](#), [3763](#)
- \int_to_roman_d:w [3730](#), [3755](#)
- \int_to_Roman_i:w [3730](#), [3758](#)
- \int_to_roman_i:w [3730](#), [3750](#)
- \int_to_Roman_l:w [3730](#), [3761](#)
- \int_to_roman_l:w [3730](#), [3753](#)
- \int_to_Roman_m:w [3730](#), [3764](#)
- \int_to_roman_m:w [3730](#), [3756](#)
- \int_to_Roman_Q:w [3730](#), [3765](#)
- \int_to_roman_Q:w [3730](#), [3757](#)
- \int_to_Roman_v:w [3730](#), [3759](#)
- \int_to_roman_v:w [3730](#), [3751](#)
- \int_to_Roman_x:w [3730](#), [3760](#)
- \int_to_roman_x:w [3730](#), [3752](#)
- \int_to_symbol:n [3950](#), [3951](#)
- \int_to_symbol_math:n .. [3950](#), [3955](#), [3958](#)
- \int_to_symbol_text:n .. [3950](#), [3956](#), [3973](#)
- \int_to_symbols:nnn .. [66](#), [3579](#), [3579](#),
 - [3595](#), [3601](#), [3633](#), [3947](#), [3960](#), [3975](#)
- \int_to_symbols_aux:nnnn [3583](#), [3593](#)
- \int_until_do:nn ... [64](#), [3522](#), [3530](#), [3535](#)
- \int_until_do:nNnn .. [64](#), [3550](#), [3558](#), [3563](#)
- \int_use:c [3426](#), [3427](#)
- \int_use:N [62](#), [2275](#), [2281](#), [3426](#),
 - [3426](#), [3427](#), [4643](#), [4646](#), [5402](#), [5410](#),
 - [5939](#), [5941](#), [6345](#), [6348](#), [8242](#), [8243](#),
 - [8252](#), [8256](#), [8258](#), [8267](#), [8278](#), [8279](#),
 - [8288](#), [8292](#), [8294](#), [8303](#), [8669](#), [9353](#),
 - [9375](#), [9401](#), [9403](#), [9539](#), [9565](#), [9633](#),
 - [10206](#), [10246](#), [10263](#), [10276](#), [10310](#),
 - [10324](#), [10335](#), [10349](#), [10395](#), [10398](#),
 - [10400](#), [10437](#), [10440](#), [10442](#), [10849](#),
 - [10852](#), [10854](#), [10885](#), [10888](#), [10890](#),
 - [10902](#), [10929](#), [10958](#), [10961](#), [10963](#),
 - [10984](#), [10987](#), [10989](#), [11038](#), [11044](#),
 - [11159](#), [11165](#), [11209](#), [11221](#), [11302](#),
 - [11309](#), [11321](#), [11551](#), [11563](#), [11579](#),
 - [11591](#), [11601](#), [11612](#), [11625](#), [11644](#),
 - [11800](#), [11806](#), [11855](#), [11861](#), [11897](#),
- [11903](#), [11941](#), [11947](#), [12098](#), [12104](#),
- [12222](#), [12228](#), [12292](#), [12298](#), [12371](#),
- [12404](#), [12430](#), [12433](#), [12514](#), [12520](#),
- [12631](#), [12637](#), [12674](#), [12681](#), [12694](#),
- [12716](#), [12733](#), [12746](#), [12756](#), [12767](#),
- [12840](#), [12842](#), [12844](#), [12872](#), [12883](#),
- [13058](#), [13064](#), [13091](#), [13093](#), [13095](#),
- [13097](#), [13099](#), [13328](#), [13330](#), [13332](#),
- [13334](#), [13336](#), [13338](#), [13340](#), [13342](#)
- \int_value:w [70](#), [1093](#), [2024](#), [2025](#),
 - [2045](#), [2047–2049](#), [2206](#), [3269](#), [3269](#),
 - [3276](#), [3279](#), [3283](#), [3291](#), [3302](#), [3313](#),
 - [3339](#), [3429](#), [3721](#), [3876](#), [4057](#), [7077](#),
 - [7101–7103](#), [7105](#), [7222](#), [7231](#), [7233](#),
 - [7238–7241](#), [7245–7248](#), [7299](#), [7305](#),
 - [7307](#), [7309](#), [7311](#), [7316](#), [7321](#), [7326](#),
 - [7333](#), [7340](#), [7479](#), [7509](#), [7510](#), [7549](#),
 - [7568](#), [7574](#), [7637](#), [7639](#), [7659](#), [7661](#),
 - [7686](#), [7724](#), [7744](#), [7752](#), [7778](#), [7780](#),
 - [7784](#), [7786](#), [7819](#), [7833](#), [7840](#), [7967](#),
 - [8067](#), [8081](#), [8548](#), [10591](#), [10703](#),
 - [10722](#), [11041](#), [11162](#), [11305](#), [11803](#),
 - [11858](#), [11900](#), [11944](#), [12101](#), [12225](#),
 - [12295](#), [12517](#), [12634](#), [12677](#), [13061](#)
- \int_while_do:nn ... [65](#), [3522](#), [3522](#), [3527](#)
- \int_while_do:nNnn .. [64](#), [3550](#), [3550](#), [3555](#)
- \int_zero:c [3381](#)
- \int_zero:N [61](#), [3381](#), [3381](#), [3383](#), [3386](#),
 - [8431](#), [8433](#), [8522](#), [9532](#), [9551](#), [9626](#)
- \int_zero_new:c [3385](#)
- \int_zero_new:N [61](#), [3385](#), [3385](#), [3389](#)
- \interactionmode [699](#)
- \interlinepenalties [720](#)
- \interlinepenalty [579](#)
- \io_new:c [133](#)
- \ior_alloc_read:n [8200](#), [8224](#), [8232](#)
- \ior_close:N
 - .. [134](#), [8198](#), [8312](#), [8338](#), [9989](#), [10018](#)
- \ior_gto:NN [137](#), [8578](#), [8580](#)
- \ior_if_eof:N [8562](#)
- \ior_if_eof:NF [10008](#)
- \ior_if_eof:NTF [138](#), [9986](#)
- \ior_if_eof_p:N [8562](#)
- \ior_list_streams [134](#)
- \ior_list_streams: [8340](#), [8340](#), [8593](#)
- \ior_list_streams_aux:Nn [8341](#), [8343](#), [8344](#)
- \ior_new:c [8192](#)
- \ior_new:N [133](#), [8192](#), [8192](#), [8193](#)
- \ior_open:cn [8196](#)

- \ior_open:Nn
... [133](#), [8196](#), [8196](#), [8222](#), [9985](#), [10007](#)
 - \ior_open_streams: [8592](#), [8593](#)
 - \ior_raw_new:c [8178](#), [8292](#)
 - \ior_raw_new:N
... [138](#), [8178](#), [8180](#), [8188](#), [8190](#), [8285](#)
 - \ior_str_gto:NN [137](#), [8582](#), [8584](#)
 - \ior_str_to:NN [137](#), [8582](#), [8582](#)
 - \ior_stream_alloc:N [8204](#), [8240](#), [8276](#)
 - \ior_stream_alloc_aux:
..... [8240](#), [8282](#), [8298](#), [8306](#), [8308](#)
 - \ior_to:NN [137](#), [8578](#), [8578](#)
 - \iow_alloc_write:n [8213](#), [8224](#), [8224](#)
 - \iow_char:N [135](#), [8382](#), [8382](#)
 - \iow_close:c [8312](#)
 - \iow_close:N .. [134](#), [8211](#), [8312](#), [8325](#), [8339](#)
 - \iow_indent:n [136](#), [8420](#), [8420](#), [8446](#)
 - \iow_indent_expandable:n [8420](#), [8421](#), [8446](#)
 - \iow_list_streams [134](#)
 - \iow_log:n . [134](#), [8373](#), [8374](#), [10054](#)–[10056](#)
 - \iow_log:x [1195](#), [1195](#), [1234](#),
[1852](#), [8373](#), [8373](#), [8750](#), [8752](#), [8753](#)
 - \iow_new:c [8192](#)
 - \iow_new:N [133](#), [8192](#), [8194](#), [8195](#)
 - \iow_newline [135](#)
 - \iow_newline:
.... [8071](#)–[8075](#), [8094](#), [8381](#), [8381](#),
[8454](#), [8737](#), [8739](#), [9249](#), [9253](#), [9258](#)
 - \iow_now:Nn [134](#), [8371](#),
[8371](#), [8374](#), [8376](#), [8378](#), [8588](#), [8590](#)
 - \iow_now:Nx
... [8370](#), [8370](#), [8372](#), [8373](#), [8375](#), [8380](#)
 - \iow_now_buffer_safe:Nn [8586](#), [8587](#)
 - \iow_now_buffer_safe:Nx [8586](#), [8589](#)
 - \iow_now_when_avail:Nn . [134](#), [8377](#), [8377](#)
 - \iow_now_when_avail:Nx [8377](#), [8379](#)
 - \iow_open:cn [8196](#)
 - \iow_open:Nn [133](#), [8196](#), [8209](#), [8223](#)
 - \iow_open_streams: [8592](#), [8594](#)
 - \iow_raw_new:c [8178](#), [8256](#)
 - \iow_raw_new:N
... [138](#), [8178](#), [8183](#), [8187](#), [8191](#), [8249](#)
 - \iow_shipout:Nn ... [134](#), [8367](#), [8367](#), [8369](#)
 - \iow_shipout:Nx [8367](#)
 - \iow_shipout_x:Nn
... [135](#), [8365](#), [8365](#), [8366](#), [8368](#), [8370](#)
 - \iow_shipout_x:Nx [8365](#)
 - \iow_stream_alloc:N [8217](#), [8240](#), [8240](#)
 - \iow_stream_alloc_aux:
..... [8240](#), [8246](#), [8262](#), [8270](#), [8272](#)
 - \iow_term:n [134](#), [8373](#), [8376](#)
 - \iow_term:x
[1195](#), [1197](#), [8069](#), [8077](#), [8088](#), [8373](#),
[8375](#), [8735](#), [8757](#), [8759](#), [8760](#), [9214](#)
 - \iow_wrap:xnnnN
.... [136](#), [8427](#), [8427](#), [8588](#), [8590](#),
[8694](#), [8697](#), [8702](#), [8705](#), [8751](#), [8758](#)
 - \iow_wrap_end: [8538](#)
 - \iow_wrap_end:w [8511](#)
 - \iow_wrap_indent: [8526](#)
 - \iow_wrap_indent:w [8511](#)
 - \iow_wrap_loop:w
..... [8457](#), [8466](#), [8466](#), [8481](#), [8516](#)
 - \iow_wrap_new_marker:n
..... [8400](#), [8404](#), [8415](#)–[8418](#)
 - \iow_wrap_newline: [8518](#)
 - \iow_wrap_newline:w [8511](#)
 - \iow_wrap_special:w [8470](#), [8511](#), [8511](#), [8515](#)
 - \iow_wrap_unindent: [8532](#)
 - \iow_wrap_unindent:w [8511](#)
 - \iow_wrap_word: [8471](#), [8473](#), [8473](#)
 - \iow_wrap_word_fits: ... [8473](#), [8479](#), [8483](#)
 - \iow_wrap_word_newline: [8473](#), [8480](#), [8499](#)
- J**
- \jobname [654](#)
- K**
- \K [2780](#)
 - \kern [532](#)
 - \kernel_register_show:c [1449](#), [1458](#), [3918](#)
 - \kernel_register_show:N [1449](#),
[1449](#), [1459](#), [3917](#), [4169](#), [4252](#), [4301](#)
 - \keys_bool_set:NN [9489](#), [9489](#), [9656](#), [9658](#)
 - \keys_bool_set_inverse:NN
..... [9504](#), [9504](#), [9660](#), [9662](#)
 - \keys_choice_code_store:x
..... [9571](#), [9571](#), [9672](#), [9674](#)
 - \keys_choice_find:n [9522](#), [9612](#), [9861](#), [9861](#)
 - \keys_choice_make: [9492](#),
[9507](#), [9519](#), [9519](#), [9531](#), [9550](#), [9664](#)
 - \keys_choices_generate:n [9545](#), [9545](#), [9704](#)
 - \keys_choices_generate_aux:n
..... [9545](#), [9552](#), [9559](#)
 - \keys_choices_make:nn .. [9529](#), [9529](#), [9666](#)
 - \keys_cmd_set:nn [9497](#), [9512](#), [9521](#), [9523](#),
[9582](#), [9582](#), [9603](#), [9615](#), [9617](#), [9668](#)

- \keys_cmd_set:nx
 9493, 9495, 9508, 9510, 9535, 9561,
 9582, 9587, 9608, 9629, 9648, 9670
 - \keys_cmd_set_aux:n 9582, 9584, 9589, 9592
 - \keys_default_set:n
 .. 9502, 9517, 9598, 9598, 9600, 9684
 - \keys_default_set:V 9598, 9686
 - \keys_define:nn ... 148, 9427, 9427, 9900
 - \keys_define_aux:nnn ... 9427, 9429, 9435
 - \keys_define_aux:onn 9427, 9428
 - \keys_define_elt:n 9432, 9436, 9436
 - \keys_define_elt:nn 9432, 9436, 9441
 - \keys_define_elt_aux:nn
 9436, 9439, 9444, 9446
 - \keys_define_key:n 9449, 9472, 9472
 - \keys_define_key_aux:w . 9472, 9476, 9487
 - \keys_execute: 9807, 9832, 9832
 - \keys_execute:nn
 .. 9832, 9833, 9836, 9852, 9863, 9864
 - \keys_execute_unknown:
 .. 9765, 9767, 9832, 9833, 9834, 9842
 - \keys_execute_unknown_alt:
 9765, 9832, 9843
 - \keys_execute_unknown_std:
 9767, 9832, 9842
 - \keys_if_choice_exist:nnn ... 9872, 9872
 - \keys_if_choice_exist:nnTF 156
 - \keys_if_exist:nn 9866, 9866
 - \keys_if_exist:nnTF 156
 - \keys_if_value:n 9825
 - \keys_if_value_p:n 9790, 9800, 9825
 - \keys_meta_make:n 9601, 9601, 9714
 - \keys_meta_make:x 9601, 9606, 9716
 - \keys_multichoice_find:n 9611, 9611, 9616
 - \keys_multichoice_make:
 9611, 9613, 9625, 9718
 - \keys_multichoices_make:nn
 9611, 9623, 9720
 - \keys_property_find:n . 9447, 9455, 9455
 - \keys_property_find_aux:w
 9455, 9459, 9462, 9468
 - \keys_set:nn
 ... 155, 9604, 9609, 9749, 9749, 9757
 - \keys_set:no 9749
 - \keys_set:nV 9749
 - \keys_set:nv 9749
 - \keys_set_aux:nnn 9749, 9751, 9758
 - \keys_set_aux:onn 9749, 9750
 - \keys_set_elt:n .. 9754, 9766, 9773, 9773
 - \keys_set_elt:nn . 9754, 9766, 9773, 9778
 - \keys_set_elt_aux:nn 9773, 9776, 9781, 9783
 - \keys_set_known:nnN 156, 9759, 9759, 9771
 - \keys_set_known:noN 9759
 - \keys_set_known:nV 9759
 - \keys_set_known:nvN 9759
 - \keys_set_known_aux:nnnN 9759, 9761, 9772
 - \keys_set_known_aux:onnN 9759, 9760
 - \keys_show:nn 156, 9878, 9878
 - \keys_value_or_default:n 9787, 9810, 9810
 - \keys_value_requirement:n
 9639, 9639, 9746, 9748
 - \keys_variable_set:cnN .. 9645, 9678,
 9690, 9698, 9708, 9724, 9732, 9736
 - \keys_variable_set:cnNN . 9645, 9682,
 9694, 9702, 9712, 9728, 9740, 9744
 - \keys_variable_set:NnN
 9645, 9651, 9654, 9676,
 9688, 9696, 9706, 9722, 9730, 9734
 - \keys_variable_set:NnNN
 ... 9645, 9645, 9652, 9653, 9680,
 9692, 9700, 9710, 9726, 9738, 9742
 - \keyval_parse:n 9311, 9319, 9404
 - \keyval_parse:NnN 157, 9397,
 9397, 9432, 9754, 9766, 9944–9946
 - \keyval_parse_elt:w
 9327, 9333, 9333, 9336, 9341
 - \keyval_split_key:w 9347, 9365, 9365
 - \keyval_split_key_value:w 9340, 9345, 9345
 - \keyval_split_key_value_aux:wTF
 9345, 9358, 9363
 - \keyval_split_value:w .. 9359, 9370, 9370
 - \keyval_split_value_aux:w ... 9388, 9391
 - \KV_process_no_space_removal_no_sanitiz:NnN
 9943, 9946
 - \KV_process_space_removal_no_sanitiz:NnN
 9943, 9945
 - \KV_process_space_removal_sanitiz:NnN
 9943, 9944
- L**
- \L 2780
 - \l@expl@log@functions@bool 1226
 - \l_box_angle_fp .. 6666, 6666, 6688, 6736
 - \l_box_bottom_dim .. 6669, 6670, 6703,
 6768, 6772, 6777, 6783, 6788, 6792,
 6801, 6803, 6816, 6824, 6847, 6853,
 6863, 6868, 6884, 6906, 6925, 6928
 - \l_box_bottom_new_dim
 6673, 6674, 6729, 6769, 6780, 6791,
 6802, 6846, 6852, 6925, 6929, 6945

- \l_box_cos_fp [6667](#), [6667](#),
6692, 6708, 6713, 6740, 6748, 6759
- \l_box_left_dim [6669](#), [6671](#), [6705](#),
6768, 6770, 6779, 6783, 6788, 6794,
6799, 6803, 6818, 6865, 6886, 6908
- \l_box_left_new_dim [6673](#), [6675](#),
6720, 6731, 6771, 6782, 6793, 6804
- \l_box_right_dim [6669](#),
6672, 6704, 6766, 6772, 6777, 6781,
6790, 6792, 6801, 6805, 6817, 6820,
6864, 6885, 6888, 6907, 6932, 6933
- \l_box_right_new_dim [6673](#), [6676](#),
6731, 6773, 6784, 6795, 6806, 6840,
6841, 6932, 6933, 6948, 6950, 6956
- \l_box_scale_x_fp [6808](#),
6808, 6819, 6821, 6826, 6870, 6887,
6889–6891, 6903, 6909, 6931, 6946
- \l_box_scale_y_fp
..... [6808](#), 6809, 6822, 6825,
6828, 6845, 6847, 6851, 6853, 6866,
6869–6871, 6890, 6904, 6911, 6922
- \l_box_sin_fp [6667](#),
6668, 6690, 6706, 6739, 6749, 6760
- \l_box_tmp_box [6677](#), [6677](#),
6717, 6718, 6724, 6728–6730, 6732,
6938, 6944, 6945, 6951, 6956, 6957
- \l_box_tmp_fp
..... [6677](#), [6678](#), 6736–6740, 6747, 6749,
6750, 6758, 6760, 6761, 6820, 6821,
6823, 6825, 6867, 6869, 6888, 6889
- \l_box_top_dim [6669](#), [6669](#), [6702](#),
6766, 6770, 6779, 6781, 6790, 6794,
6799, 6805, 6815, 6824, 6845, 6851,
6862, 6868, 6883, 6905, 6924, 6929
- \l_box_top_new_dim
..... [6673](#), [6673](#), 6728, 6767, 6778, 6789,
6800, 6844, 6850, 6924, 6928, 6944
- \l_box_x_fp
.. [6679](#), [6679](#), 6744, 6746, 6755, 6758
- \l_box_x_new_fp
.. [6679](#), [6681](#), 6746, 6748, 6750, 6751
- \l_box_y_fp
.. [6679](#), [6680](#), 6745, 6747, 6756, 6757
- \l_box_y_new_fp
.. [6679](#), [6682](#), 6757, 6759, 6761, 6762
- \l_cctab_tmp_tl [13500](#), 13514–13516, 13538
- \l_clist_if_in_clist [5872](#), 5872, 5885, 5886
- \l_clist_remove_clist
.. [5824](#), 5824, 5831, 5834, 5835, 5837
- \l_clist_tmpa_clist [5675](#), [5675](#),
5763, 5764, 5776, 5777, 5948, 5949,
5965, 5966, 5987, 5989, 5991, 9144
- \l_coffin_aligned_coffin
..... [7209](#), [7211](#), 7466,
7467, 7471, 7477, 7479, 7480, 7496,
7497, 7503–7507, 7509, 7511, 7515,
7516, 7521–7525, 7559, 7574, 7619,
7621, 8050, 8057, 8059, 8061, 8063
- \l_coffin_aligned_internal_coffin ..
..... [7209](#), [7212](#), 7538, 7545
- \l_coffin_bottom_corner_dim
..... [7626](#), [7628](#),
7651, 7655, 7722, 7731, 7747, 7755
- \l_coffin_bounding_prop
..... [7624](#), [7624](#), [7642](#),
7667, 7669, 7672, 7674, 7680, 7737
- \l_coffin_bounding_shift_dim
..... [7625](#), [7625](#), 7649, 7736, 7741
- \l_coffin_calc_a_fp
..... [7045](#), 7045, 7410, 7414, 7421, 7423–
7425, 7428–7430, 7433, 7453, 7457
- \l_coffin_calc_b_fp
..... [7045](#), 7046, 7411, 7414,
7417, 7426, 7434, 7437, 7454, 7460
- \l_coffin_calc_c_fp
.. [7045](#), 7047, 7412, 7415, 7455, 7459
- \l_coffin_calc_d_fp . [7045](#), 7048, 7413,
7415, 7417, 7431, 7435, 7456, 7458
- \l_coffin_calc_result_fp
... [7045](#), 7049, 7420, 7422, 7427,
7432, 7436, 7439, 7452, 7457–7461
- \l_coffin_cos_fp
..... [7055](#), 7056, 7636, 7707, 7712
- \l_coffin_Depth_dim [7065](#), 7065, 7257
- \l_coffin_display_coffin
..... [7846](#), 7846, 7975, 7982,
8052, 8053, 8058, 8060, 8062, 8063
- \l_coffin_display_coord_coffin
..... [7846](#), 7847,
7913, 7933, 7949, 7997, 8017, 8036
- \l_coffin_display_font_tl
.. [7891](#), 7891, 7893, 7896, 7921, 8005
- \l_coffin_display_handles_prop
..... [7849](#), 7849, 7850, 7852, 7854, 7856,
7858, 7860, 7862, 7864, 7866, 7868,
7870, 7872, 7874, 7876, 7878, 7880,
7882, 7884, 7924, 7928, 8008, 8012
- \l_coffin_display_offset_dim . [7886](#),
7886, 7887, 7950, 7951, 8037, 8038

\l_coffin_display_pole_coffin 7929, 7930, 7938, 7943, 8009, 8010, 8013, 8014, 8023, 8028, 8078, 8085
 .. 7846, 7848, 7901, 7912, 7956, 7995
 \l_coffin_display_poles_prop 7890, 7890, 7966, 7971, 7974, 7977, 7979, 7986
 \l_coffin_display_x_dim 7888, 7888, 7992, 8047
 \l_coffin_display_y_dim 7888, 7889, 7993, 8049
 \l_coffin_error_bool 7050, 7050, 7352, 7356, 7370, 7385, 7418, 7988, 7990
 \l_coffin_handles_tmp_prop 7898, 7898, 7976
 \l_coffin_Height_dim ... 7065, 7066, 7256
 \l_coffin_left_corner_dim 7626, 7626, 7650, 7658, 7723, 7729, 7746, 7754
 \l_coffin_offset_x_dim 7051, 7051, 7469, 7470, 7473, 7481, 7483, 7485, 7491, 7494, 7514, 7534, 7542, 8046, 8054
 \l_coffin_offset_y_dim 7051, 7052, 7484, 7486, 7491, 7494, 7514, 7536, 7543, 8048, 8055
 \l_coffin_pole_a_tl 7053, 7053, 7350, 7355, 7583, 7586, 7587, 7590, 7968, 7970, 7973
 \l_coffin_pole_b_tl 7053, 7054, 7351, 7355, 7584, 7586, 7588, 7590, 7969, 7970, 7972, 7973
 \l_coffin_right_corner_dim 7626, 7627, 7658, 7721, 7730
 \l_coffin_scale_x_fp 7759, 7759, 7767, 7769, 7782, 7795, 7800, 7810
 \l_coffin_scale_y_fp 7759, 7760, 7770, 7772, 7796, 7797, 7813
 \l_coffin_scaled_total_height_dim .. 7761, 7761, 7798, 7799, 7804
 \l_coffin_scaled_width_dim 7761, 7762, 7801, 7802, 7804
 \l_coffin_sin_fp 7055, 7055, 7635, 7708, 7713
 \l_coffin_tmp_box 7024, 7024, 7141, 7145, 7149, 7184, 7189, 7194
 \l_coffin_tmp_dim 7024, 7025, 7472, 7474, 7475, 7671, 7673, 7675
 \l_coffin_tmp_fp 7024, 7026, 7632–7636, 7706, 7708, 7709, 7711, 7713, 7714, 7768, 7769, 7771, 7772, 7809–7814
 \l_coffin_tmp_tl .. 7024, 7027, 7034–7044, 7557, 7558, 7560, 7925, 7926, 7929, 7930, 7938, 7943, 8009, 8010, 8013, 8014, 8023, 8028, 8078, 8085
 \l_coffin_top_corner_dim 7626, 7629, 7655, 7720, 7732
 \l_coffin_TotalHeight_dim 7065, 7067, 7258
 \l_coffin_Width_dim 7065, 7068, 7259
 \l_coffin_x_dim 7057, 7057, 7359, 7368, 7388, 7391, 7398, 7405, 7407, 7438, 7441, 7531, 7535, 7554, 7562, 7679, 7681, 7685, 7687, 7691, 7696, 7818, 7820, 7824, 7827, 7992, 8044
 \l_coffin_x_fp 7061, 7061, 7703, 7705, 7711
 \l_coffin_x_prime_dim ... 7057, 7059, 7531, 7535, 7693, 7697, 8044, 8047
 \l_coffin_x_prime_fp 7061, 7063, 7705, 7707, 7709, 7715
 \l_coffin_y_dim 7057, 7058, 7360, 7373, 7376, 7383, 7400, 7442, 7532, 7537, 7555, 7562, 7679, 7681, 7685, 7687, 7691, 7696, 7818, 7820, 7824, 7827, 7993, 8045
 \l_coffin_y_fp 7061, 7062, 7704, 7706, 7710
 \l_coffin_y_prime_dim ... 7057, 7060, 7532, 7537, 7693, 7698, 8045, 8049
 \l_coffin_y_prime_fp 7061, 7064, 7710, 7712, 7714, 7716
 \l_doc_pTF_name_tl 21, 22, 35, 37, 40, 41, 44, 52, 53, 54, 55, 63, 73, 77, 85, 86, 91, 92, 97, 106, 111, 114, 122, 123, 138, 156, 162, 163
 \l_exp_tl 31, 1555, 1555, 1574, 1575, 1751, 1752
 \l_expl_status_bool 96, 294, 309, 323, 327, 328
 \l_expl_status_stack_tl 197
 \l_file_name_tl 159, 9975, 9975, 10022, 10023, 10029, 10030, 10040
 \l_file_search_path_saved_seq 159, 9977, 9978, 9999, 10016
 \l_file_search_path_seq 159, 9976, 9976, 9999, 10001, 10002, 10005, 10016, 10046, 10047, 10050
 \l_file_tmpa_seq 159, 9980, 9981, 10000, 10002, 10061, 10062
 \l_fp_arg_tl 10094, 10094, 11793, 11812, 11816, 11826, 11847, 11848, 11890, 11905, 11913, 11933, 11934, 12091, 12110, 12114, 12124, 12134, 12158, 12189, 12214, 12215, 12285,

12300, 12309, 12311, 12339, 12379,
 12511, 12628, 12639, 12647, 12667
 \l_fp_count_int 10095, 10095,
 11315, 11350, 11362, 11370, 11988,
 12020, 12037, 12039, 12042, 12044,
 12488, 12525, 12533, 12763, 12766,
 12767, 12789, 12833, 12893, 12904
 \l_fp_div_offset_int .. 10096, 10096,
 11271, 11325, 11370, 11372, 12203
 \l_fp_exp_decimal_int . 10097, 10098,
 12363, 12397, 12416, 12436, 12455,
 12464, 12469, 12475, 12491, 12540,
 12545, 12549, 12552, 12556, 12560,
 12563, 12565, 12709, 12719, 12730,
 12733, 12742, 12750, 12776, 12839,
 12847, 12851, 12862, 12905, 12906
 \l_fp_exp_exponent_int
 10097, 10100, 12365,
 12399, 12421, 12441, 12457, 12707,
 12711, 12729, 12736, 12759, 12843
 \l_fp_exp_extended_int 10097,
 10099, 12364, 12398, 12416, 12436,
 12456, 12465, 12468, 12473, 12479,
 12491, 12541, 12543, 12546, 12557,
 12559, 12561, 12710, 12719, 12752,
 12756, 12758, 12776, 12841, 12848,
 12850, 12853, 12862, 12905, 12907
 \l_fp_exp_integer_int
 10097, 10097, 12362, 12396,
 12416, 12436, 12454, 12463, 12467,
 12491, 12551, 12564, 12708, 12719,
 12741, 12746, 12749, 12776, 12838
 \l_fp_input_a_decimal_int
 10101, 10103, 10152, 10343, 10344,
 10349, 10351, 10382, 10384, 10398,
 10424, 10426, 10440, 10838, 10852,
 10874, 10888, 10900, 10902, 10909,
 10913, 10951, 10961, 10976, 10987,
 11057, 11073, 11172, 11254, 11319,
 11321, 11323, 11339, 11352, 11353,
 11355, 11378, 11549, 11551, 11560,
 11568, 11569, 11576, 11579, 11587,
 11595, 11676, 11690, 11691, 11695,
 11698, 11700, 11706, 11714, 11716,
 11729, 11730, 11737, 11739, 11747,
 11757, 11760, 11766, 11768, 11772,
 11791, 11804, 11888, 11901, 11975,
 11981, 11982, 12012, 12015, 12018,
 12029, 12033, 12089, 12102, 12156,
 12180, 12185, 12187, 12282, 12296,
 12461, 12464, 12471, 12477, 12486,
 12528, 12600, 12605, 12635, 12782,
 12796, 12800, 12803, 12818, 12823,
 12827, 12830, 12831, 12895, 12897,
 12900, 12903, 12933, 12939, 12947,
 12964, 12990, 13023, 13073, 13084,
 13104, 13117, 13150, 13166, 13184,
 13209, 13279, 13311, 13331, 13340
 \l_fp_input_a_exponent_int
 10101, 10104, 10153,
 10298, 10306, 10331, 10352, 10383,
 10400, 10425, 10442, 10854, 10870,
 10890, 10914, 10963, 10989, 11022,
 11132, 11275, 11547, 11571, 11575,
 11604, 11651, 11792, 11806, 11808,
 11889, 11903, 12090, 12104, 12106,
 12131, 12181, 12186, 12207, 12283,
 12298, 12321, 12601, 12637, 12686,
 12687, 12692, 12694, 12705, 12715,
 12716, 12816, 12825, 12934, 12940,
 12985, 13019, 13074, 13085, 13112,
 13119, 13151, 13167, 13186, 13261,
 13265, 13293, 13297, 13333, 13342
 \l_fp_input_a_extended_int
 10109, 10109, 11561,
 11563, 11570, 11597, 11601, 11603,
 11652, 11692–11694, 11696, 11706,
 11718, 11720, 11731, 11738, 11740,
 11748, 11758, 11761, 11769, 11773,
 11981, 11982, 12016, 12019, 12029,
 12033, 12065, 12284, 12465, 12481,
 12487, 12528, 12558, 12764, 12797,
 12804, 12824, 12828, 12830, 12831,
 12895, 12897, 12900, 12903, 12984,
 12990, 13021, 13102, 13105, 13118
 \l_fp_input_a_integer_int
 10101, 10102, 10151, 10332, 10335,
 10342, 10381, 10395, 10423, 10437,
 10849, 10885, 10908, 10910, 10912,
 10958, 10984, 11053, 11069, 11182,
 11187, 11191, 11196, 11254, 11318,
 11323, 11333, 11336, 11351, 11354,
 11376, 11377, 11548, 11558, 11559,
 11586, 11591, 11594, 11655, 11657,
 11670, 11675, 11688, 11689, 11699,
 11702, 11708, 11710, 11712, 11737,
 11739, 11755, 11757, 11760, 11768,
 11772, 11790, 11800, 11887, 11897,
 12088, 12098, 12157, 12179, 12184,
 12188, 12281, 12292, 12324, 12334,

12349, 12361, 12371, 12373, 12375,
 12400, 12404, 12406, 12408, 12428,
 12430, 12433, 12599, 12605, 12631,
 12782, 12788, 12799, 12802, 12815,
 12822, 12932, 12938, 12947, 12964,
 13024, 13072, 13083, 13104, 13116,
 13149, 13165, 13183, 13209, 13269,
 13274, 13301, 13306, 13329, 13338
 \l_fp_input_a_sign_int
 10101, 10101, 10147, 10149,
 10380, 10390, 10422, 10432, 10844,
 10880, 10979, 11016, 11050, 11094,
 11146, 11289, 11656, 11661, 11703,
 11789, 11795, 11843, 11850, 11886,
 11892, 11929, 11936, 11962, 11964,
 11969, 12087, 12093, 12142, 12183,
 12280, 12287, 12323, 12377, 12410,
 12429, 12462, 12485, 12526, 12598,
 12602, 12931, 12937, 13016, 13070,
 13115, 13148, 13164, 13182, 13229,
 13243, 13247, 13251, 13327, 13336
 \l_fp_input_b_decimal_int
 10101, 10107, 10318, 10319,
 10324, 10326, 11006, 11057, 11073,
 11109, 11127, 11174, 11240, 11244,
 11339, 11352, 12171, 12176, 12486,
 12529, 12530, 12532, 12534, 12537,
 12540, 12556, 12818, 12832, 12896,
 12933, 12943, 13076, 13084, 13094,
 13106, 13156, 13166, 13184, 13212,
 13227, 13279, 13311, 13332, 13339
 \l_fp_input_b_exponent_int
 10101, 10108, 10298, 10306, 10327,
 10331, 11007, 11110, 11128, 11132,
 11241, 11275, 12172, 12177, 12207,
 12819, 12934, 13077, 13085, 13098,
 13112, 13157, 13167, 13186, 13261,
 13265, 13293, 13297, 13334, 13341
 \l_fp_input_b_extended_int
 10109, 10110, 12487,
 12529, 12530, 12532, 12534, 12537,
 12542, 12832, 12896, 13096, 13107
 \l_fp_input_b_integer_int
 10101, 10106, 10307, 10310,
 10317, 11005, 11053, 11069, 11108,
 11126, 11185, 11189, 11192, 11196,
 11239, 11244, 11333, 11336, 11351,
 12170, 12175, 12817, 12932, 12943,
 13075, 13083, 13092, 13106, 13155,
 13165, 13183, 13212, 13227, 13269,
 13274, 13301, 13306, 13330, 13337
 \l_fp_input_b_sign_int 10101, 10105,
 11004, 11016, 11080, 11107, 11111,
 11125, 11146, 11238, 11289, 12174,
 12485, 12526, 12539, 12931, 12980,
 13090, 13115, 13154, 13164, 13182,
 13217, 13243, 13247, 13328, 13335
 \l_fp_mul_a_i_int 10111, 10111,
 11173, 11178, 11183, 11188, 11192,
 11422, 11431, 11438, 11444, 11449,
 11455, 11458, 11466, 11475, 11483,
 11491, 11498, 11506, 11511, 11515
 \l_fp_mul_a_ii_int
 10111, 10112, 11173, 11179,
 11184, 11189, 11422, 11432, 11439,
 11445, 11450, 11456, 11466, 11476,
 11484, 11492, 11499, 11507, 11512
 \l_fp_mul_a_iii_int 10111,
 10113, 11173, 11180, 11185, 11422,
 11433, 11440, 11446, 11451, 11466,
 11477, 11485, 11493, 11500, 11508
 \l_fp_mul_a_iv_int 10111,
 10114, 11424, 11434, 11441, 11447,
 11468, 11478, 11486, 11494, 11501
 \l_fp_mul_a_v_int
 10111, 10115, 11424, 11435,
 11442, 11468, 11479, 11487, 11495
 \l_fp_mul_a_vi_int 10111, 10116,
 11424, 11436, 11468, 11480, 11488
 \l_fp_mul_b_i_int 10111, 10117,
 11175, 11180, 11184, 11188, 11191,
 11426, 11436, 11442, 11447, 11451,
 11456, 11458, 11470, 11480, 11487,
 11494, 11500, 11507, 11511, 11514
 \l_fp_mul_b_ii_int
 10111, 10118, 11175, 11179,
 11183, 11187, 11426, 11435, 11441,
 11446, 11450, 11455, 11470, 11479,
 11486, 11493, 11499, 11506, 11510
 \l_fp_mul_b_iii_int 10111,
 10119, 11175, 11178, 11182, 11426,
 11434, 11440, 11445, 11449, 11470,
 11478, 11485, 11492, 11498, 11505
 \l_fp_mul_b_iv_int 10111,
 10120, 11428, 11433, 11439, 11444,
 11472, 11477, 11484, 11491, 11497
 \l_fp_mul_b_v_int
 10111, 10121, 11428, 11432,
 11438, 11472, 11476, 11483, 11490

`\l_fp_mul_b_vi_int` [10111](#), [10122](#), [11428](#), [11431](#), [11472](#), [11475](#), [11482](#)
`\l_fp_mul_output_int` [10123](#), [10123](#), [11176](#), [11181](#), [11214](#), [11215](#), [11219](#), [11221](#), [11226](#), [11429](#), [11437](#), [11473](#), [11481](#)
`\l_fp_mul_output_tl` ... [10123](#), [10124](#), [11177](#), [11194](#), [11195](#), [11198](#), [11225](#), [11430](#), [11453](#), [11454](#), [11461](#), [11474](#), [11503](#), [11504](#), [11517](#), [11518](#), [11521](#)
`\l_fp_output_decimal_int`
 [10125](#), [10127](#), [11026](#), [11042](#), [11055](#), [11059](#), [11062](#), [11071](#), [11075](#), [11077](#), [11081](#), [11084](#), [11086](#), [11137](#), [11150](#), [11163](#), [11194](#), [11269](#), [11280](#), [11293](#), [11306](#), [11367](#), [11369](#), [11620](#), [11625](#), [11633](#), [11663](#), [11838](#), [11839](#), [11845](#), [11859](#), [11924](#), [11925](#), [11931](#), [11945](#), [11977](#), [11992](#), [11996](#), [12001](#), [12005](#), [12013](#), [12015](#), [12047](#), [12052](#), [12056](#), [12059](#), [12063](#), [12067](#), [12070](#), [12072](#), [12170](#), [12179](#), [12201](#), [12212](#), [12226](#), [12353](#), [12397](#), [12417](#), [12419](#), [12437](#), [12439](#), [12492](#), [12494](#), [12499](#), [12500](#), [12502](#), [12509](#), [12518](#), [12655](#), [12656](#), [12658](#), [12665](#), [12678](#), [12697](#), [12709](#), [12720](#), [12722](#), [12774](#), [12777](#), [12823](#), [12826](#), [12827](#), [12840](#), [12860](#), [12863](#), [12869](#), [12872](#), [12879](#), [12887](#), [12906](#), [12910](#), [12914](#), [12917](#), [13062](#), [13076](#), [13095](#), [13108](#), [13117](#), [13121](#)
`\l_fp_output_exponent_int` ... [10125](#), [10128](#), [11022](#), [11027](#), [11044](#), [11130](#), [11138](#), [11165](#), [11273](#), [11281](#), [11309](#), [11647](#), [11664](#), [11836](#), [11840](#), [11846](#), [11861](#), [11922](#), [11926](#), [11932](#), [11947](#), [12205](#), [12213](#), [12228](#), [12355](#), [12399](#), [12421](#), [12441](#), [12510](#), [12520](#), [12666](#), [12681](#), [12699](#), [12711](#), [12729](#), [12736](#), [12825](#), [12844](#), [12868](#), [12889](#), [13064](#), [13077](#), [13099](#), [13110](#), [13119](#), [13123](#)
`\l_fp_output_extended_int`
 [10129](#), [10129](#), [11638](#), [11639](#), [11644](#), [11646](#), [11839](#), [11925](#), [11993](#), [11997](#), [12002](#), [12004](#), [12016](#), [12048](#), [12050](#), [12053](#), [12064](#), [12066](#), [12068](#), [12171](#), [12180](#), [12354](#), [12398](#), [12418](#), [12420](#), [12438](#), [12440](#), [12493](#), [12495](#), [12497](#), [12653](#), [12698](#), [12710](#), [12721](#), [12723](#), [12775](#), [12778](#), [12828](#), [12842](#), [12861](#), [12864](#), [12907](#), [12908](#), [12911](#), [13097](#), [13109](#), [13118](#), [13122](#)
`\l_fp_output_integer_int`
 [10125](#), [10126](#), [11025](#), [11038](#), [11051](#), [11061](#), [11067](#), [11076](#), [11079](#), [11082](#), [11088](#), [11090](#), [11136](#), [11150](#), [11159](#), [11198](#), [11268](#), [11279](#), [11293](#), [11302](#), [11362](#), [11608](#), [11609](#), [11612](#), [11619](#), [11662](#), [11835](#), [11838](#), [11844](#), [11855](#), [11921](#), [11924](#), [11930](#), [11941](#), [11976](#), [11991](#), [11995](#), [12000](#), [12011](#), [12058](#), [12071](#), [12200](#), [12211](#), [12222](#), [12352](#), [12378](#), [12396](#), [12417](#), [12419](#), [12437](#), [12439](#), [12492](#), [12494](#), [12503](#), [12508](#), [12514](#), [12659](#), [12664](#), [12674](#), [12696](#), [12708](#), [12720](#), [12722](#), [12774](#), [12777](#), [12822](#), [12860](#), [12863](#), [12878](#), [12883](#), [12886](#), [12916](#), [13058](#), [13075](#), [13093](#), [13108](#), [13116](#), [13120](#)
`\l_fp_output_sign_int`
 [10125](#), [10125](#), [11024](#), [11033](#), [11050](#), [11080](#), [11094](#), [11135](#), [11278](#), [12144](#), [12146](#), [12150](#), [12152](#), [12210](#), [12217](#), [12507](#), [12663](#), [12669](#), [12688](#), [12690](#), [12769](#), [12855](#), [13091](#)
`\l_fp_round_carry_bool` [10130](#), [10130](#), [10897](#), [10907](#), [10920](#), [10926](#), [10934](#)
`\l_fp_round_decimal_tl` [10131](#), [10131](#), [10899](#), [10909](#), [10928](#), [10929](#), [10931](#)
`\l_fp_round_position_int` [10132](#), [10132](#), [10898](#), [10919](#), [10932](#), [10938](#), [10939](#)
`\l_fp_round_target_int`
 [10132](#), [10133](#), [10834](#), [10835](#), [10869](#), [10871](#), [10919](#), [10932](#)
`\l_fp_sign_tl`
 .. [10134](#), [10134](#), [12981](#), [12993](#), [13057](#)
`\l_fp_split_sign_int`
 .. [10135](#), [10135](#), [10160](#), [10162](#), [10175](#)
`\l_fp_tmp_dim` . [10407](#), [10415](#), [10419](#), [10455](#)
`\l_fp_tmp_int` [10136](#), [10136](#), [10204](#), [10206](#), [10921](#)–[10924](#), [10929](#), [11525](#)–[11527](#), [11532](#)–[11534](#)
`\l_fp_tmp_skip` [10407](#), [10414](#), [10415](#), [10456](#)
`\l_fp_tmp_tl`
 [10137](#), [10137](#), [10157](#)–[10159](#), [10163](#), [10168](#), [10170](#), [10173](#), [10179](#), [10181](#), [10184](#), [10273](#), [10278](#), [10320](#), [10326](#), [10345](#), [10351](#), [10977](#), [10992](#), [11589](#), [11595](#), [11598](#), [11603](#), [11621](#), [11628](#), [11640](#), [11646](#), [12368](#), [12375](#), [12382](#),

- 12388, 12401, 12408, 12411, 12413,
12744, 12750, 12753, 12758, 12881,
12887, 13325, 13344, 13350, 13355
- \l_fp_trig_decimal_int
.... 10139, 10140, 11983, 11985,
11987, 11990, 12005, 12018, 12028,
12030, 12032, 12034, 12036, 12038,
12041, 12043, 12045, 12047, 12063
- \l_fp_trig_extended_int 10139, 10141,
11983, 11985, 11987, 11989, 12004,
12019, 12028, 12030, 12032, 12034,
12036, 12038, 12041, 12043, 12049
- \l_fp_trig_octant_int
.... 10138, 10138, 11727, 11733,
11745, 11746, 11762, 11774, 11866,
11952, 11963, 11967, 12143, 12149
- \l_fp_trig_sign_int 10139,
10139, 11979, 12017, 12026, 12046
- \l_ior_stream_int
..... 8176, 8177, 8199, 8201,
8205, 8236, 8278, 8279, 8283, 8286,
8292, 8294, 8300, 8301, 8303, 8305
- \l_iow_current_indentation_int
..... 8386, 8388,
8431, 8491, 8506, 8528, 8534, 8536
- \l_iow_current_indentation_tl 8389,
8391, 8432, 8489, 8509, 8529, 8535
- \l_iow_current_line_int
..... 8386, 8386, 8433,
8477, 8478, 8490, 8496, 8503, 8522
- \l_iow_current_line_tl
..... 8389, 8389, 8434, 8488,
8494, 8502, 8508, 8521, 8523, 8541
- \l_iow_current_word_int
..... 8386, 8387, 8475, 8477, 8505
- \l_iow_current_word_tl .. 8389, 8390,
8468, 8469, 8476, 8489, 8495, 8509
- \l_iow_line_length_int
..... 136, 8383, 8383, 8384, 8430
- \l_iow_line_start_bool
.. 8394, 8394, 8436, 8485, 8487, 8524
- \l_iow_stream_int 8176,
8176, 8177, 8212, 8214, 8218, 8228,
8242, 8243, 8247, 8250, 8252, 8256,
8258, 8264, 8265, 8267, 8269, 8288
- \l_iow_target_length_int
..... 8385, 8385, 8430, 8478
- \l_iow_wrap_tl
.. 8392, 8392, 8435, 8449, 8452, 8458
- \l_iow_wrapped_tl
.. 8393, 8393, 8464, 8501, 8520, 8540
- \l_keys_choice_int .. 154, 9418, 9418,
9532, 9538, 9539, 9542, 9551, 9564,
9565, 9569, 9626, 9632, 9633, 9636
- \l_keys_choice_tl . 154, 9537, 9563, 9631
- \l_keys_choices_tl 9418, 9419
- \l_keys_key_tl 155, 9420,
9420, 9500, 9515, 9785, 9786, 9847
- \l_keys_module_tl
... 9421, 9421, 9428, 9431, 9433,
9457, 9604, 9609, 9750, 9753, 9755,
9760, 9763, 9768, 9786, 9836, 9839
- \l_keys_no_value_bool
... 9422, 9422, 9438, 9443, 9474,
9775, 9780, 9791, 9801, 9813, 9848
- \l_keys_path_tl
.... 155, 9423, 9423, 9452, 9457,
9464, 9467, 9482, 9493, 9495, 9497,
9508, 9510, 9512, 9521, 9523, 9526,
9535, 9548, 9556, 9561, 9567, 9574,
9577, 9579, 9599, 9603, 9608, 9615,
9617, 9620, 9629, 9642, 9648, 9668,
9670, 9786, 9795, 9805, 9815, 9817,
9820, 9828, 9833, 9839, 9863, 9864
- \l_keys_property_tl . 9424, 9424, 9448,
9452, 9470, 9477, 9478, 9481, 9485
- \l_keys_unknown_clist
..... 9425, 9425, 9764, 9769, 9845
- \l_keys_value_tl 155, 9426,
9426, 9805, 9812, 9819, 9849, 9857
- \l_keyval_key_tl
..... 9307, 9307, 9354, 9367, 9376
- \l_keyval_parse_tl .. 9309, 9310, 9326,
9330, 9350, 9372, 9381, 9385, 9394
- \l_keyval_sanitise_tl
..... 9309, 9309, 9322-9325, 9328
- \l_keyval_value_tl 9307,
9308, 9378, 9380, 9383, 9393, 9395
- \l_last_box 7016, 7016
- \l_msg_class_tl 8863, 8864, 8914, 8915, 8918
- \l_msg_current_class_tl
..... 8863, 8865, 8896, 8915
- \l_msg_current_module_tl 8863, 8866, 8897
- \l_msg_redirect_classes_prop 8770, 8770
- \l_msg_redirect_classes_seq
..... 8863, 8863, 8871, 8876, 8879
- \l_msg_redirect_kernel_info_prop . 9015
- \l_msg_redirect_kernel_warning_prop
..... 8993

<code>\l_msg_redirect_names_prop</code>	<code>\let</code>
..... 8770, 8771, 8898, 8929	59, 230, 336, 337, 349
<code>\l_msg_show_tl</code> 9209, 9209, 9234–9236, 9241	<code>\limits</code>
<code>\l_msg_text_tl</code> 8682, 8682, 8709, 8742, 8744	496
<code>\l_msg_tmp_tl</code> 8603, 8603, 8713, 8716, 8724	<code>\linepenalty</code>
<code>\l_peek_search_tl</code>	552
..... 2931, 2931, 2949, 2970, 3013	<code>\lineskip</code>
<code>\l_peek_search_token</code>	546
.. 2930, 2930, 2948, 2969, 2988, 2996	<code>\lineskiplimit</code>
<code>\l_peek_token</code> 55, 2928, 2928, 2937, 2988,	547
2996, 3006–3008, 3027, 3156–3158	<code>\long</code>
<code>\l_seq_remove_seq</code>	33, 339, 368
.. 5218, 5218, 5225, 5228, 5229, 5231	<code>\looseness</code>
<code>\l_seq_tmpa_tl</code> 5143, 5143, 5172, 5178,	564
5183, 5184, 5250, 5255, 5269, 5273	<code>\lower</code>
<code>\l_seq_tmpb_tl</code>	601
.. 5143, 5144, 5246, 5250, 5272, 5273	<code>\lowercase</code>
<code>\l_tl_replace_tl</code>	640
4476, 4476	<code>\lua_now:n</code>
<code>\l_tl_tmpa_tl</code> 4596, 4599, 4601, 4609	167, 13442, 13459
<code>\l_tl_tmpb_tl</code> 4596, 4600, 4601, 4610	<code>\lua_now:x</code>
<code>\l_tmpa_bool</code>	4885,
35, 1961, 1961	13442, 13444, 13448, 13451, 13460
<code>\l_tmpa_box</code>	<code>\lua_shipout:n</code> .. 167, 13442, 13462, 13464
123, 6553, 6554, 6557	<code>\lua_shipout:x</code>
<code>\l_tmpa_clist</code>	13442
110, 5994, 5994	<code>\lua_shipout_x:n</code>
<code>\l_tmpa_dim</code>	168, 13442,
76, 4182, 4182	13445, 13453, 13456, 13461, 13463
<code>\l_tmpa_int</code>	<code>\lua_shipout_x:x</code>
69, 3940, 3940	13442
<code>\l_tmpa_skip</code>	<code>\luaescapestring</code>
78, 4258, 4258	39, 40
<code>\l_tmpa_tl</code>	<code>\luatex_catcodetable:D</code>
4, 93, 4894, 4894	. 758, 773, 13502, 13503, 13508, 13516
<code>\l_tmpb_box</code>	<code>\luatex_directlua:D</code> 759, 1483, 13444
123, 6553, 6559	<code>\luatex_if_engine:</code>
<code>\l_tmpb_clist</code>	1460
110, 5994, 5995	<code>\luatex_if_engine:F</code>
<code>\l_tmpb_dim</code> 1461, 1486, 13481, 13518, 13546
76, 4182, 4183	<code>\luatex_if_engine:T</code> . 1460, 1485, 4883,
<code>\l_tmpb_int</code>	13490, 13532, 13554, 13560, 13569
69, 3940, 3941	<code>\luatex_if_engine:TF</code> .. 1462, 1487, 13442
<code>\l_tmpb_skip</code>	<code>\luatex_if_engine_p:</code> ... 1469, 1491, 1540
78, 4258, 4259	<code>\luatex_if_engineTF</code>
<code>\l_tmpb_tl</code>	21
93, 4894, 4895	<code>\luatex_initcatcodetable:D</code>
<code>\l_tmpc_dim</code>	760, 774, 13477, 13496
76, 4182, 4184	<code>\luatex_latelua:D</code>
<code>\l_tmpc_int</code>	761, 775, 13445
69, 3940, 3942	<code>\luatex luatexversion:D</code>
<code>\l_tmpc_skip</code>	762, 845
78, 4258, 4260	<code>\luatex_savecatcodetable:D</code>
<code>\language</code>	763, 776, 13507, 13543
449	<code>\luatexcatcodetable</code>
<code>\lastbox</code>	773
606	<code>\luatexinitcatcodetable</code>
<code>\lastkern</code>	774
539	<code>\luatexlatelua</code>
<code>\lastlinefit</code>	775
719	<code>\luatexsavecatcodetable</code>
<code>\lastnodetype</code>	776
700	<code>\luatexversion</code>
<code>\lastpenalty</code>	762
645	
<code>\lastskip</code>	
540	
<code>\latelua</code>	
761	
<code>\lccode</code>	
668	
<code>\leaders</code>	
536	
<code>\left</code>	
504	
<code>\lefthyphenmin</code>	
560	
<code>\leftskip</code>	
562	
<code>\leqno</code>	
479	

M

<code>\M</code>	2737, 2780
<code>\m@ne</code>	834
<code>\mag</code>	448
<code>\mark</code>	450
<code>\marks</code>	676
<code>\mathaccent</code>	461
<code>\mathbin</code>	491
<code>\mathchar</code>	462

<code>\mathchardef</code>	359	<code>\msg_critical:nxxxx</code>	8808
<code>\mathchoice</code>	459	<code>\msg_critical:nxxxxx</code>	141, 8808
<code>\mathclose</code>	492	<code>\msg_critical_text:n</code> 140, 8763, 8764, 8811	
<code>\mathcode</code>	670	<code>\msg_direct_interrupt:xxxxx</code> .	9271, 9276
<code>\mathinner</code>	493	<code>\msg_direct_log:xx</code>	9271, 9277
<code>\mathop</code>	494	<code>\msg_direct_term:xx</code>	9271, 9278
<code>\mathopen</code>	498	<code>\msg_error:nn</code>	8819
<code>\mathord</code>	499	<code>\msg_error:nnx</code>	8819
<code>\mathparagraph</code>	3966	<code>\msg_error:nnxx</code>	8819
<code>\mathpunct</code>	500	<code>\msg_error:nnxxx</code>	8819
<code>\mathrel</code>	501	<code>\msg_error:nnxxxx</code>	142, 8819
<code>\mathsection</code>	3965	<code>\msg_error_text:n</code>	
<code>\mathsurround</code>	512	140, 8763, 8765, 8824, 8833, 8963, 8976	
<code>\maxdeadcycles</code>	582	<code>\msg_expandable_error:n</code>	
<code>\maxdepth</code>	583 146, 9157, 9165, 9182	
<code>\maxdimen</code>	4180	<code>\msg_expandable_error_aux:w</code> .	9171, 9177
<code>\meaning</code>	642	<code>\msg_expandable_kernel_error:nn</code>	
<code>\medmuskip</code>	513 2232, 5140, 9180, 9204	
<code>\message</code>	419	<code>\msg_expandable_kernel_error:nnn</code> ...	
<code>\MessageBreak</code>	222, 238–244 1606,	
<code>\middle</code>	724	2255, 4677, 9180, 9199, 13450, 13455	
<code>\mkern</code>	466	<code>\msg_expandable_kernel_error:nnnn</code> ..	
<code>\mode_if_horizontal:</code>	2297, 2297 9180, 9194	
<code>\mode_if_horizontalTF</code>	40	<code>\msg_expandable_kernel_error:nnnnn</code> .	
<code>\mode_if_inner:</code>	2299, 2299 9180, 9189	
<code>\mode_if_innerTF</code>	40	<code>\msg_expandable_kernel_error:nnnnnn</code>	
<code>\mode_if_math:</code>	2301, 2301	146, 9180, 9180, 9191, 9196, 9201, 9206	
<code>\mode_if_math:TF</code>	3954	<code>\msg_fatal:nn</code>	8797
<code>\mode_if_mathTF</code>	40	<code>\msg_fatal:nnx</code>	8797
<code>\mode_if_vertical:</code>	2295, 2295	<code>\msg_fatal:nnxx</code>	8797
<code>\mode_if_verticalTF</code>	41	<code>\msg_fatal:nnxxx</code>	8797
<code>\month</code>	652	<code>\msg_fatal:nnxxxx</code>	141, 8797
<code>\moveleft</code>	602	<code>\msg_fatal_text:n</code>	
<code>\moveright</code>	603 140, 8763, 8763, 8800, 8941	
<code>\msg_aux_show:n</code> 5462, 5983, 5991, 9247, 9247		<code>\msg_generic_new:nn</code>	9271, 9273
<code>\msg_aux_show:nn</code>	6361, 9247, 9251	<code>\msg_generic_new:nnn</code>	9271, 9272
<code>\msg_aux_show:Nnx</code>		<code>\msg_generic_set:nn</code>	9271, 9275
146, 5459, 5980, 5988, 6358, 9220, 9220		<code>\msg_generic_set:nnn</code>	9271, 9274
<code>\msg_aux_show:w</code>	9220, 9240, 9246	<code>\msg_gset:nnn</code>	8606, 8633
<code>\msg_aux_show:x</code> 146, 8348, 9220, 9225, 9232		<code>\msg_gset:nnnn</code> 139, 8606, 8613, 8626, 8634	
<code>\msg_aux_show_unbraced:nn</code> 8349, 9247, 9256		<code>\msg_if_more_text:c</code>	8786
<code>\msg_aux_use:nn</code> ... 146, 8346, 9210, 9210		<code>\msg_if_more_text:cTF</code>	8821, 8960
<code>\msg_aux_use:nnxxxx</code> 9210, 9211, 9212, 9224		<code>\msg_if_more_text:N</code>	8786, 8786
<code>\msg_class_new:nn</code>	9261, 9262	<code>\msg_if_more_text:NF</code>	8795
<code>\msg_class_set:nn</code>		<code>\msg_if_more_text:NT</code>	8794
..... 141, 8772, 8772, 8797, 8808,		<code>\msg_if_more_text:NTF</code>	8796
8819, 8841, 8849, 8857, 8862, 9262		<code>\msg_if_more_text_p:N</code>	8793
<code>\msg_critical:nn</code>	8808	<code>\msg_info:nn</code>	8849
<code>\msg_critical:nnx</code>	8808	<code>\msg_info:nnx</code>	8849
<code>\msg_critical:nnxx</code>	8808	<code>\msg_info:nnxx</code>	8849

- \msg_info:nnxxx [8849](#)
- \msg_info:nnxxxx [142](#), [8849](#)
- \msg_info_text:n [141](#), [8763](#), [8767](#), [8853](#), [9022](#)
- \msg_interrupt:xxx
 - [144](#), [8683](#), [8683](#), [8799](#), [8810](#),
 - [8823](#), [8832](#), [8940](#), [8962](#), [8975](#), [9283](#)
- \msg_interrupt_aux: [8683](#), [8689](#), [8733](#)
- \msg_interrupt_details:xxx
 - [8683](#), [8688](#), [8700](#)
- \msg_interrupt_more_text:n
 - [8683](#), [8696](#), [8704](#), [8710](#)
- \msg_interrupt_no_details:xx
 - [8683](#), [8687](#), [8692](#)
- \msg_interrupt_text:n
 - [8683](#), [8698](#), [8706](#), [8708](#)
- \msg_kernel_bug:x [9280](#), [9281](#)
- \msg_kernel_error:nn
 - ... [1199](#), [1213](#), [7358](#), [8877](#), [8958](#),
 - [8991](#), [9360](#), [13406](#), [13414](#), [13419](#), [13428](#)
- \msg_kernel_error:nnx ... [1199](#), [1211](#),
 - [1454](#), [2532](#), [4493](#), [5373](#), [6565](#), [7080](#),
 - [7085](#), [8886](#), [8958](#), [8989](#), [9228](#), [9460](#),
 - [9499](#), [9514](#), [9555](#), [9794](#), [13485](#), [13527](#)
- \msg_kernel_error:nnxx
 - [1199](#), [1199](#), [1212](#),
 - [1214](#), [1221](#), [1231](#), [1244](#), [1364](#), [7224](#),
 - [8610](#), [8892](#), [8958](#), [8987](#), [9451](#), [9480](#),
 - [9525](#), [9619](#), [9804](#), [9838](#), [13522](#), [13550](#)
- \msg_kernel_error:nnxxx [8958](#), [8985](#)
- \msg_kernel_error:nnxxxx
 - [145](#), [8958](#), [8958](#), [8986](#), [8988](#), [8990](#), [8992](#)
- \msg_kernel_fatal:nn
 - [8202](#), [8215](#), [8938](#), [8956](#), [13506](#)
- \msg_kernel_fatal:nnx . [8938](#), [8954](#), [13479](#)
- \msg_kernel_fatal:nnxx [8938](#), [8952](#)
- \msg_kernel_fatal:nnxxx [8938](#), [8950](#)
- \msg_kernel_fatal:nnxxxx
 - [145](#), [8938](#), [8938](#), [8951](#), [8953](#), [8955](#), [8957](#)
- \msg_kernel_info:nn [8993](#), [9035](#)
- \msg_kernel_info:nnx [8993](#), [9033](#)
- \msg_kernel_info:nnxx [8993](#), [9031](#)
- \msg_kernel_info:nnxxx [8993](#), [9029](#)
- \msg_kernel_info:nnxxxx
 - [145](#), [8993](#), [9016](#), [9030](#), [9032](#), [9034](#), [9036](#)
- \msg_kernel_new:nnn [8930](#),
 - [8932](#), [9119](#), [9121](#), [9123](#), [9125](#), [9127](#),
 - [9134](#), [9141](#), [9149](#), [9151](#), [9153](#), [9155](#)
- \msg_kernel_new:nnnn
 - [145](#), [8099](#), [8107](#), [8110](#), [8351](#), [8358](#),
 - [8930](#), [8930](#), [9037](#), [9045](#), [9053](#), [9060](#),
 - [9067](#), [9075](#), [9084](#), [9091](#), [9098](#), [9105](#),
 - [9112](#), [9407](#), [9880](#), [9883](#), [9889](#), [9896](#),
 - [9905](#), [9911](#), [9917](#), [9924](#), [9931](#), [9937](#),
 - [13399](#), [13407](#), [13415](#), [13421](#), [13436](#)
- \msg_kernel_set:nnn [8930](#), [8936](#)
- \msg_kernel_set:nnnn ... [145](#), [8930](#), [8934](#)
- \msg_kernel_warning:nn [8993](#), [9013](#)
- \msg_kernel_warning:nnx [8993](#), [9011](#)
- \msg_kernel_warning:nnxx [8993](#), [9009](#)
- \msg_kernel_warning:nnxxx ... [8993](#), [9007](#)
- \msg_kernel_warning:nnxxxx
 - [145](#), [8993](#), [8994](#), [9008](#), [9010](#), [9012](#), [9014](#)
- \msg_line_context [140](#)
- \msg_line_context:
 - [1215](#), [1215](#), [1234](#), [8669](#), [8670](#)
- \msg_line_number [140](#)
- \msg_line_number: [8669](#), [8669](#), [8674](#), [9408](#)
- \msg_log:nn [8857](#), [9269](#)
- \msg_log:nnx [8857](#), [9268](#)
- \msg_log:nnxx [8857](#), [9267](#)
- \msg_log:nnxxx [8857](#), [9266](#)
- \msg_log:nnxxxx [142](#), [8857](#), [9265](#)
- \msg_log:x [144](#), [8748](#), [8748](#), [8851](#), [8859](#), [9020](#)
- \msg_new:nnn [8606](#), [8615](#), [8933](#)
- \msg_new:nnnn . [139](#), [8606](#), [8606](#), [8616](#), [8931](#)
- \msg_newline [143](#)
- \msg_newline: [8667](#), [8667](#), [8721](#)
- \msg_no_more_text:xxxx . [8786](#), [8788](#), [8792](#)
- \msg_none:nn [8862](#)
- \msg_none:nnx [8862](#)
- \msg_none:nnxx [8862](#)
- \msg_none:nnxxx [8862](#)
- \msg_none:nnxxxx [142](#), [8862](#)
- \msg_redirect_class:nn . [143](#), [8924](#), [8924](#)
- \msg_redirect_module:nnn [143](#), [8926](#), [8926](#)
- \msg_redirect_name:nnn . [143](#), [8928](#), [8928](#)
- \msg_see_documentation_text:n
 - [8768](#), [8768](#), [8803](#), [8814](#),
 - [8827](#), [8836](#), [8945](#), [8967](#), [8980](#), [9286](#)
- \msg_set:nnn [8606](#), [8624](#), [8937](#)
- \msg_set:nnnn . [139](#), [8606](#), [8617](#), [8625](#), [8935](#)
- \msg_term:x ... [144](#), [8748](#), [8755](#), [8843](#), [8998](#)
- \msg_trace:nn [9264](#), [9269](#)
- \msg_trace:nnx [9264](#), [9268](#)
- \msg_trace:nnxx [9264](#), [9267](#)
- \msg_trace:nnxxx [9264](#), [9266](#)
- \msg_trace:nnxxxx [9264](#), [9265](#)
- \msg_two_newlines [143](#)
- \msg_two_newlines: [8667](#), [8668](#)

\msg_use:nnnnxxx	4271
..... 8776, 8867, 8867, 8996, 9018	
\msg_use_aux:nn	358
..... 8867, 8900, 8902	
\msg_use_aux:nnn	718
..... 8867, 8891, 8894	
\msg_use_code: 8867, 8869, 8909, 8917, 8921	
\msg_use_loop:n	N
..... 8867, 8874, 8922, 8923	
\msg_use_loop:o	\n
..... 8867, 8918	2776
\msg_use_loop_check:nn	\name_primitive:NN
..... 8867, 8899, 8905, 8908, 8912	339, 339, 346-763
\msg_warning:nn	\newbox
..... 8841	6457
\msg_warning:nnx	\newcatcodetable
..... 8841	13495
\msg_warning:nnxx	\newcount
..... 8841	3347
\msg_warning:nnxxx	\newdimen
..... 8841	4003
\msg_warning:nnxxxx	\newlinechar
..... 142, 8841	249, 414
\msg_warning_text:n	\newmuskip
..... 141, 8763, 8766, 8845, 9000	4267
\mskip	\newread
..... 463	8188
\muexpr	\newskip
..... 712	4191
\multiply	\newwrite
..... 364	8187
\muskip	\noalign
..... 660	382
\muskip_add:cn	\noboundary
..... 4287	517
\muskip_add:Nn	\noexpand
..... 79, 4287, 4287, 4289, 4290	35, 39, 40, 166, 169, 172, 174,
\muskip_eval:n	175, 184, 187-189, 191, 193, 194,
..... 79, 4297, 4297	203, 205-209, 268, 270, 275, 277, 375
\muskip_gadd:cn	\noindent
..... 4287	543
\muskip_gadd:Nn	\nolimits
..... 79, 4287, 4289, 4291	497
\muskip_gset:cn	\nonscript
..... 4276	477
\muskip_gset:Nn	\nonstopmode
..... 79, 4276, 4278, 4280	440
\muskip_gset_eq:cc	\nulldelimiterspace
..... 4281	510
\muskip_gset_eq:cN	\nullfont
..... 4281	628
\muskip_gset_eq:Nc	\number
..... 4281	637
\muskip_gset_eq:NN	\numexpr
..... 79, 4281, 4284-4286	709
\muskip_gsub:cn	
..... 4287	O
\muskip_gsub:Nn	\O
..... 79, 4287, 4294, 4296	1865, 2780
\muskip_gzero:c	\omit
..... 4271	383
\muskip_gzero:N	\openin
..... 78, 4271, 4273, 4275	409
\muskip_new:c	\openout
..... 4263	410
\muskip_new:N	\or
..... 78, 4263, 4264, 4270	22, 70, 406
\muskip_set:cn	\or:
..... 4276	785, 787, 1341-1349, 1545,
\muskip_set:Nn	3696-3720, 11867, 11869, 11871,
..... 79, 4276, 4276, 4278, 4279	11873, 11953, 11955, 11957, 11959
\muskip_set_eq:cc	\outer
..... 4281	369
\muskip_set_eq:cN	\output
..... 4281	584
\muskip_set_eq:Nc	\outputpenalty
..... 4281	594
\muskip_set_eq:NN	\over
..... 79, 4281, 4281-4283	471
\muskip_show:c	\overfullrule
..... 4301	622
\muskip_show:N	\overline
..... 80, 4301, 4301, 4302	502
\muskip_show:n	\overwithdelims
..... 80, 4303, 4303	472
\muskip_sub:cn	
..... 4287	P
\muskip_sub:Nn	\P
..... 79, 4287, 4292, 4294, 4295	1863, 2780
\muskip_use:c	
..... 4299	
\muskip_use:N	
..... 79, 4298, 4299, 4299, 4300	

<code>\package_check_loaded_expl:</code>	783, 1553, 1917, 2429, 2547, 3267, 3994, 4323, 5136, 5673, 6146, 6451, 7022, 8122, 8142, 8601, 9304, 9953, 10070, 13434	<code>\pdftex_pdfcompresslevel:D</code>	739
<code>\PackageError</code>	219, 235	<code>\pdftex_pdfcreationdate:D</code>	737
<code>\pagedepth</code>	586	<code>\pdftex_pdfdecimaldigits:D</code>	740
<code>\pagediscards</code>	727	<code>\pdftex_pdfhorigin:D</code>	741
<code>\pagefilllstretch</code>	590	<code>\pdftex_pdfinfo:D</code>	742
<code>\pagefillstretch</code>	589	<code>\pdftex_pdflastxform:D</code>	743
<code>\pagefilstretch</code>	588	<code>\pdftex_pdfliteral:D</code>	744
<code>\pagegoal</code>	592	<code>\pdftex_pdfminorversion:D</code>	745
<code>\pageshrink</code>	591	<code>\pdftex_pdfobjcompresslevel:D</code>	746
<code>\pagestretch</code>	587	<code>\pdftex_pdfoutput:D</code>	747
<code>\pagetotal</code>	593	<code>\pdftex_pdfpkresolution:D</code>	752
<code>\par</code>	542, 6619, 6620, 6622, 6624, 6626, 6631, 6637, 6650	<code>\pdftex_pdfrefxform:D</code>	748
<code>\parfillskip</code>	573	<code>\pdftex_pdfrestore:D</code>	749
<code>\parindent</code>	566	<code>\pdftex_pdfsave:D</code>	750
<code>\parshape</code>	558	<code>\pdftex_pdfsetmatrix:D</code>	751
<code>\parshapedimen</code>	708	<code>\pdftex_pdftextrevision:D</code>	753
<code>\parshapeindent</code>	706	<code>\pdftex_pdfvorigin:D</code>	754
<code>\parshapelength</code>	707	<code>\pdftex_pdfxform:D</code>	755
<code>\parskip</code>	565	<code>\pdftex_strcmp:D</code>	756, 1497, 1503, 2454, 2461, 2478, 2505, 2514, 2752, 4223, 4898, 10167, 10178
<code>\patterns</code>	648	<code>\pdftexrevision</code>	753
<code>\pausing</code>	435	<code>\pdfvorigin</code>	754
<code>\pdf@strcmp</code>	59	<code>\pdfxform</code>	755
<code>\pdfcolorstack</code>	738	<code>\peek_after:NN</code>	3193, 3194
<code>\pdfcompresslevel</code>	739	<code>\peek_after:Nw</code>	55, 2936, 2936, 2961, 2979, 3035, 3194
<code>\pdfcreationdate</code>	737	<code>\peek_catcode:N</code>	3053
<code>\pdfdecimaldigits</code>	740	<code>\peek_catcode:NTF</code>	55
<code>\pdfhorigin</code>	741	<code>\peek_catcode_ignore_spaces:N</code>	3053
<code>\pdfinfo</code>	742	<code>\peek_catcode_ignore_spaces:NTF</code>	55
<code>\pdflastxform</code>	743	<code>\peek_catcode_remove:N</code>	3053
<code>\pdfliteral</code>	744	<code>\peek_catcode_remove:NTF</code>	56
<code>\pdfminorversion</code>	745	<code>\peek_catcode_remove_ignore_spaces:N</code>	3053
<code>\pdfobjcompresslevel</code>	746	<code>\peek_catcode_remove_ignore_spaces:NTF</code>	56
<code>\pdfoutput</code>	747	<code>\peek_charcode:N</code>	3069
<code>\pdfpkresolution</code>	752	<code>\peek_charcode:NTF</code>	56
<code>\pdfrefxform</code>	748	<code>\peek_charcode_ignore_spaces:N</code>	3069
<code>\pdfrestore</code>	749	<code>\peek_charcode_ignore_spaces:NTF</code>	56
<code>\pdfsave</code>	750	<code>\peek_charcode_remove:N</code>	3069
<code>\pdfsetmatrix</code>	751	<code>\peek_charcode_remove:NTF</code>	56
<code>\pdfstrcmp</code>	33, 59, 230, 235, 238, 252, 756	<code>\peek_charcode_remove_ignore_spaces:N</code>	3069
<code>\pdftex_if_engine:</code>	1460	<code>\peek_charcode_remove_ignore_spaces:NTF</code>	56
<code>\pdftex_if_engine:F</code>	1464, 1475, 1489	<code>\peek_def:n</code>	3036, 3037, 3053, 3057, 3061, 3065, 3069, 3073, 3077, 3081, 3085, 3089, 3093, 3097
<code>\pdftex_if_engine:T</code>	1463, 1474, 1488		
<code>\pdftex_if_engine:TF</code>	1465, 1476, 1490, 3372		
<code>\pdftex_if_engine_p:</code>	1470, 1480, 1492, 1541		
<code>\pdftex_if_engineTF</code>	21		
<code>\pdftex_pdfcolorstack:D</code>	738		

- \peek_def_aux:nnnnn [3036](#), [3039–3041](#), [3043](#)
- \peek_execute_branches: [3032](#), [3048](#)
- \peek_execute_branches_catcode:
 - .. [2985](#), [2985](#), [3056](#), [3058](#), [3064](#), [3066](#)
- \peek_execute_branches_charcode: . . .
 - .. [3002](#), [3002](#), [3072](#), [3074](#), [3080](#), [3082](#)
- \peek_execute_branches_charcode:NN [3002](#)
- \peek_execute_branches_charcode_aux:NN
 - [3012](#), [3016](#)
- \peek_execute_branches_meaning: . . .
 - .. [2985](#), [2994](#), [3088](#), [3090](#), [3096](#), [3098](#)
- \peek_execute_branches_N_type: . . .
 - [3152](#), [3152](#), [3164](#), [3166](#), [3168](#)
- \peek_false:w [2932](#), [2934](#), [2955](#),
 - [2973](#), [2991](#), [2999](#), [3010](#), [3021](#), [3160](#)
- \peek_gafter:NN [3193](#), [3195](#)
- \peek_gafter:Nw [55](#), [2938](#), [3195](#)
- \peek_ignore_spaces_execute_branches:
 - [3025](#), [3025](#), [3035](#),
 - [3060](#), [3068](#), [3076](#), [3084](#), [3092](#), [3100](#)
- \peek_ignore_spaces_execute_branches_aux:
 - [3025](#), [3029](#), [3034](#)
- \peek_meaning:N [3085](#)
- \peek_meaning:NTF [57](#)
- \peek_meaning_ignore_spaces:N . . . [3085](#)
- \peek_meaning_ignore_spaces:NTF . . . [57](#)
- \peek_meaning_remove:N [3085](#)
- \peek_meaning_remove:NTF [57](#)
- \peek_meaning_remove_ignore_spaces:N
 - [3085](#)
- \peek_meaning_remove_ignore_spaces:NTF
 - [57](#)
- \peek_N_type: [3152](#)
- \peek_N_type:F [3167](#)
- \peek_N_type:T [3165](#)
- \peek_N_type:TF [3163](#)
- \peek_N_typeTF [59](#)
- \peek_tmp:w [2932](#), [2935](#), [2944](#), [3030](#)
- \peek_token_generic:NN [2946](#)
- \peek_token_generic:NNF [2965](#), [3168](#)
- \peek_token_generic:NNT [2963](#), [3166](#)
- \peek_token_generic:NNTF
 - [2946](#), [2964](#), [2966](#), [3164](#)
- \peek_token_remove_generic:NN . . . [2967](#)
- \peek_token_remove_generic:NNF . . [2983](#)
- \peek_token_remove_generic:NNT . . [2981](#)
- \peek_token_remove_generic:NNTF . . .
 - [2967](#), [2982](#), [2984](#)
- \peek_true:w [2932](#), [2932](#),
 - [2950](#), [2971](#), [2989](#), [2997](#), [3019](#), [3161](#)
- \peek_true_aux:w . . [2932](#), [2933](#), [2943](#), [2972](#)
- \peek_true_remove:w [2940](#), [2940](#), [2971](#)
- \penalty [643](#)
- \postdisplaypenalty [490](#)
- \predisplaydirection [734](#)
- \predisdisplaypenalty [489](#)
- \predisplaysize [488](#)
- \pretolerance [569](#)
- \prevdepth [616](#)
- \prevgraf [575](#)
- \prg_break_point:n
 - . [42](#), [1506](#), [1506–1508](#), [2246](#), [2293](#),
 - [2342](#), [4630](#), [4648](#), [4657](#), [5069](#), [5279](#),
 - [5292](#), [5305](#), [5318](#), [5346](#), [5381](#), [5416](#),
 - [5427](#), [5478](#), [5485](#), [5498](#), [5505](#), [5512](#),
 - [5519](#), [5557](#), [5576](#), [5647](#), [5910](#), [5924](#),
 - [5943](#), [5960](#), [6284](#), [6330](#), [6351](#), [6392](#)
- \prg_case_dim:nnn [39](#), [2154](#), [2154](#)
- \prg_case_dim_aux:nnn . . [2154](#), [2157](#), [2159](#)
- \prg_case_dim_aux:nw [2154](#), [2160](#), [2161](#), [2165](#)
- \prg_case_end:nw [2140](#),
 - [2140](#), [2151](#), [2164](#), [2175](#), [2187](#), [2198](#)
- \prg_case_int:nnn [38](#), [2141](#), [2141](#), [3585](#), [3591](#)
- \prg_case_int_aux:nnn . . [2141](#), [2144](#), [2146](#)
- \prg_case_int_aux:nw [2141](#), [2147](#), [2148](#), [2152](#)
- \prg_case_str:nnn [39](#), [2167](#), [2167](#), [2178](#), [5047](#)
- \prg_case_str:onn [2167](#)
- \prg_case_str:xxn [2167](#), [2179](#)
- \prg_case_str_aux:nw [2167](#), [2170](#), [2172](#), [2176](#)
- \prg_case_str_x_aux:nw
 - [2167](#), [2182](#), [2184](#), [2188](#)
- \prg_case_tl:cnn [2190](#)
- \prg_case_tl:Nnn . . . [39](#), [2190](#), [2190](#), [2201](#)
- \prg_case_tl_aux:Nw [2190](#), [2193](#), [2195](#), [2199](#)
- \prg_conditional_form_F:nnn [1082](#)
- \prg_conditional_form_p:nnn [1079](#)
- \prg_conditional_form_T:nnn [1081](#)
- \prg_conditional_form_TF:nnn [1080](#)
- \prg_define_quicksort:nnn [2342](#), [2342](#), [2417](#)
- \prg_do_nothing [9](#)
- \prg_do_nothing: [1494](#), [1494](#),
 - [4445](#), [4459](#), [4518](#), [4523](#), [5174](#), [5181](#),
 - [6062](#), [6066](#), [6073](#), [9244](#), [10173](#), [10184](#)
- \prg_generate_conditional_aux:nnNNnnnn
 - [925](#),
 - [933](#), [940](#), [948](#), [956](#), [965](#), [973](#), [982](#), [993](#)
- \prg_generate_conditional_aux:nnw . .
 - [995](#), [1001](#), [1007](#)
- \prg_generate_conditional_parm_aux:nnNNnnnn
 - [993](#)

\prg_generate_conditional_parm_aux:nw	1921 , 4596 , 4617 , 5265 , 5472 , 5480 ,
.....	993
\prg_generate_F_form_count:Nnnnn ...	5493 , 5500 , 5507 , 5514 , 5879 , 5883 ,
.....	6309 , 6364 , 6370 , 13143 , 13160 , 13347
\prg_generate_F_form_parm:Nnnnn ...	\prg_quicksort:n 42 , 2417
.....	\prg_quicksort_compare:nnTF 42 , 2418 , 2419
\prg_generate_p_form_count:Nnnnn ...	\prg_quicksort_function:n 42 , 2418 , 2418
.....	\prg_replicate:nn 39 , 2202 ,
\prg_generate_p_form_parm:Nnnnn ...	2202 , 8536 , 9126 , 10491 , 10527 , 10597
.....	\prg_replicate_ 2202 , 2213
\prg_generate_T_form_count:Nnnnn ...	\prg_replicate_0:n 2202
.....	\prg_replicate_1:n 2202
\prg_generate_T_form_parm:Nnnnn ...	\prg_replicate_2:n 2202
.....	\prg_replicate_3:n 2202
\prg_generate_TF_form_count:Nnnnn ...	\prg_replicate_4:n 2202
.....	\prg_replicate_5:n 2202
\prg_generate_TF_form_parm:Nnnnn ...	\prg_replicate_6:n 2202
.....	\prg_replicate_7:n 2202
\prg_get_count_aux:nn	\prg_replicate_8:n 2202
.....	\prg_replicate_9:n 2202
\prg_get_parm_aux:nw	\prg_replicate_aux:N 2202 , 2209 , 2210 , 2212
.....	\prg_replicate_first-:n 2202
\prg_map_break:	\prg_replicate_first_0:n 2202
... 1506 , 1507 , 2256 , 2342 , 2473 ,	\prg_replicate_first_1:n 2202
2480 , 4669 , 5366 , 5368 , 5976 , 6354	\prg_replicate_first_2:n 2202
\prg_map_break:n 42 , 1506 , 1508 ,	\prg_replicate_first_3:n 2202
2342 , 4670 , 5367 , 5369 , 5977 , 6355	\prg_replicate_first_4:n 2202
\prg_new_conditional:Nnn 952 ,	\prg_replicate_first_5:n 2202
961 , 1921 , 2483 , 2491 , 2503 , 2512 , 8562	\prg_replicate_first_6:n 2202
\prg_new_conditional:Npnn	\prg_replicate_first_7:n 2202
.. 32 , 921 , 929 , 1432 , 1495 , 1501 ,	\prg_replicate_first_8:n 2202
1921 , 1949 , 1963 , 2295 , 2297 , 2299 ,	\prg_replicate_first_9:n 2202
2301 , 2663 , 2668 , 2673 , 2678 , 2685 ,	\prg_replicate_first_aux:N 2202 , 2205 , 2211
2691 , 2696 , 2701 , 2706 , 2711 , 2716 ,	\prg_return_false 34
2721 , 2726 , 2731 , 2745 , 2759 , 2764 ,	\prg_return_false:
2785 , 2792 , 2799 , 2810 , 2821 , 2832 ,	.. 917 , 919 , 1126 , 1131 , 1144 , 1149 ,
2843 , 2852 , 2859 , 2877 , 3428 , 3498 ,	1157 , 1174 , 1435 , 1499 , 1504 , 1921 ,
3506 , 3514 , 4058 , 4063 , 4220 , 4230 ,	1954 , 1968 , 2296 , 2298 , 2300 , 2302 ,
4539 , 4561 , 4582 , 4584 , 4777 , 4793 ,	2488 , 2496 , 2509 , 2518 , 2666 , 2671 ,
4809 , 4843 , 4849 , 4864 , 4908 , 4910 ,	2676 , 2681 , 2688 , 2694 , 2699 , 2704 ,
5080 , 6111 , 6266 , 6278 , 6523 , 6525 ,	2709 , 2714 , 2719 , 2724 , 2729 , 2734 ,
6535 , 9825 , 9866 , 9872 , 13127 , 13135	2755 , 2762 , 2769 , 2771 , 2791 , 2798 ,
\prg_new_eq_conditional:NNn	2802 , 2809 , 2813 , 2820 , 2831 , 2835 ,
.....	2842 , 2851 , 2858 , 2867 , 2880 , 2899 ,
34 , 987 , 989 , 1921 ,	2916 , 2925 , 3447 , 3455 , 3461 , 3471 ,
5261 , 5263 , 5873 – 5878 , 6441 – 6444	3479 , 3485 , 3493 , 3503 , 3511 , 3517 ,
\prg_new_map_functions:Nn ... 2420 , 2421	4061 , 4069 , 4227 , 4238 , 4554 , 4566 ,
\prg_new_protected_conditional:Nnn	4579 , 4589 , 4606 , 4621 , 4786 , 4806 ,
.....	4821 , 4829 , 4839 , 4859 , 4873 , 4901 ,
\prg_new_protected_conditional:Npnn	
.....	
33 , 921 , 944 ,	

- 5278, 5468, 5893, 6114, 6271, 6297,
 6313, 6368, 6374, 6524, 6526, 6536,
 8572, 8789, 9830, 9870, 9876, 10024,
 13132, 13140, 13177, 13191, 13195,
 13199, 13203, 13215, 13219, 13234,
 13249, 13267, 13276, 13284, 13299,
 13308, 13316, 13361, 13367, 13373,
 13379, 13385, 13391, 13396, 13397
 \prg_return_true 34
 \prg_return_true: [917](#), 917, 1129, 1146,
 1154, 1159, 1172, 1177, 1435, 1499,
 1504, [1921](#), 1952, 1966, 2296, 2298,
 2300, 2302, 2486, 2494, 2507, 2516,
 2666, 2671, 2676, 2681, 2688, 2694,
 2699, 2704, 2709, 2714, 2719, 2724,
 2729, 2734, 2753, 2762, 2769, 2791,
 2798, 2809, 2820, 2831, 2842, 2851,
 2858, 2867, 2897, 2923, 3445, 3453,
 3463, 3469, 3477, 3487, 3495, 3501,
 3509, 3519, 4061, 4067, 4225, 4237,
 4552, 4564, 4577, 4587, 4603, 4621,
 4784, 4804, 4819, 4837, 4861, 4872,
 4899, 5281, 5476, 5484, 5497, 5504,
 5511, 5518, 5893, 6115, 6269, 6295,
 6318, 6380, 6524, 6526, 6536, 8567,
 8570, 8576, 8790, 9829, 9869, 9875,
 10025, 13130, 13138, 13188, 13222,
 13231, 13245, 13263, 13271, 13281,
 13295, 13303, 13313, 13361, 13367,
 13373, 13379, 13385, 13391, 13397
 \prg_set_conditional:Nnn . [952](#), 952, [1921](#)
 \prg_set_conditional:Npnn
 32, [921](#), 921, 1123,
 1135, 1151, 1163, [1921](#), 4549, 8786
 \prg_set_eq_conditional:NNn
 34, [987](#), 987, [1921](#)
 \prg_set_eq_conditional_aux:NNNn ...
 988, 990, [1067](#), 1067
 \prg_set_eq_conditional_aux:NNNw ...
 [1067](#), 1068, 1069, 1077
 \prg_set_map_functions:Nn ... [2420](#), [2422](#)
 \prg_set_protected_conditional:Nnn .
 [952](#), 971, [1921](#)
 \prg_set_protected_conditional:Npnn
 33, [921](#), 937, [1921](#)
 \prg_stepwise_aux:nnnn
 [2243](#), [2245](#), [2248](#), [2292](#)
 \prg_stepwise_aux:Nnnnn
 [2243](#), [2251](#), [2258](#), [2262](#), [2267](#)
 \prg_stepwise_aux:NNnnnn
 [2271](#), [2273](#), [2279](#), [2288](#)
 \prg_stepwise_function:nnnn
 40, [2243](#), [2243](#)
 \prg_stepwise_inline:nnnn
 40, [2271](#), [2271](#), 13584, 13589
 \prg_stepwise_variable:nnnn
 40, [2271](#), [2277](#)
 \prg_variable_get_scope:N 41, [2308](#), 2314
 \prg_variable_get_scope_aux:w
 [2308](#), 2317, 2320
 \prg_variable_get_type:N . 41, [2308](#), 2329
 \prg_variable_get_type:w [2308](#)
 \prg_variable_get_type_aux:w
 2331, 2334, 2338
 \prop_clear:c [6152](#), [6153](#), 7478
 \prop_clear:N . 112, [6152](#), [6152](#), [6157](#), 7976
 \prop_clear_new:c [6156](#), 7101, 7102, 8774
 \prop_clear_new:N . 112, [6156](#), [6156](#), 6158
 \prop_del:cn [6188](#)
 \prop_del:cV [6188](#)
 \prop_del:Nn 114, [6188](#),
 6188, 6194, 6195, 7971, 7974, 7979
 \prop_del:NV [6188](#)
 \prop_del_aux:NNnnn [6188](#), 6189, 6191, 6192
 \prop_display:c [6417](#), 6419
 \prop_display:N [6417](#), 6418
 \prop_gclear:c [6152](#), 6155
 \prop_gclear:N 112, [6152](#), 6154, 6160
 \prop_gclear_new:c [6156](#)
 \prop_gclear_new:N . 112, [6156](#), 6159, 6161
 \prop_gdel:cn [6188](#)
 \prop_gdel:cV [6188](#)
 \prop_gdel:Nn . 114, [6188](#), 6190, 6196, 6197
 \prop_gdel:NV [6188](#), 8320, 8333
 \prop_get:cn [6403](#)
 \prop_get:cnN [6198](#), [6309](#), 8914
 \prop_get:cnNF 7221
 \prop_get:coN [6309](#)
 \prop_get:cVN [6198](#), [6309](#)
 \prop_get:Nn 117, [6403](#), 6403, 6416
 \prop_get:NnN
 113, [6198](#), 6198, 6206, 6207,
 6309, 6309, 7924, 7928, 8008, 8012
 \prop_get:NnNF 6321, 6324
 \prop_get:NnNT 6320, 6323
 \prop_get:NnNTF 114, 6322, 6325
 \prop_get:NoN [6198](#), [6309](#)
 \prop_get:NVN [6198](#), [6309](#)
 \prop_get_aux:Nnnn [6198](#), 6201, 6204

\prop_get_aux_true:Nnnn	6309 , 6312 , 6315	\prop_if_in:cc	6432
\prop_get_gdel:NnN	6429 , 6430	\prop_if_in:cn	6278
\prop_get_Nn_aux:nwn	6403 , 6405 , 6410 , 6414	\prop_if_in:cnTF	8904 , 8907
\prop_gget:cnN	6421	\prop_if_in:co	6278
\prop_gget:cVN	6421	\prop_if_in:cV	6278
\prop_gget:NnN	6421 , 6422 , 6426 , 6427	\prop_if_in:Nn	6278 , 6278
\prop_gget:NVN	6421	\prop_if_in:NnF	6305 , 6306 , 6434 , 8226 , 8234
\prop_gget_aux:Nnnn	6421 , 6423 , 6424	\prop_if_in:NnT	6303 , 6304 , 6433
\prop_gpop:cnN	6208 , 6364	\prop_if_in:NnTF	114 , 6307 , 6308 , 6435 , 8898
\prop_gpop:coN	6208	\prop_if_in:No	6278
\prop_gpop:NnN	113 , 6208 , 6214 , 6227 , 6228 , 6370 , 6430	\prop_if_in:NV	6278
\prop_gpop:NnNF	6386	\prop_if_in:NVT	8269 , 8305
\prop_gpop:NnNT	6385	\prop_if_in_aux:N	6278 , 6289 , 6292
\prop_gpop:NnNTF	116 , 6387	\prop_if_in_aux:nwn	6278 , 6280 , 6286 , 6290
\prop_gpop:NoN	6208	\prop_if_in_p:Nn	6301 , 6302
\prop_gput:ccx	6437	\prop_map_break	115
\prop_gput:cnn	6229	\prop_map_break:	6299 , 6335 , 6354 , 6354 , 6397
\prop_gput:cno	6229	\prop_map_break:n	116 , 6354 , 6355
\prop_gput:cnV	6229	\prop_map_function:cc	6326
\prop_gput:cnx	6229	\prop_map_function:cN	6326 , 8080
\prop_gput:con	6229	\prop_map_function:Nc	6326
\prop_gput:coo	6229	\prop_map_function:NN	115 , 6326 , 6326 , 6340 , 6341 , 6361 , 8349
\prop_gput:cVn	6229	\prop_map_function_aux:Nwn	6326 , 6328 , 6332 , 6338 , 6347
\prop_gput:cVV	6229	\prop_map_inline:cn	6342 , 7549 , 7568 , 7637 , 7639 , 7659 , 7661 , 7724 , 7778 , 7780 , 7784 , 7786
\prop_gput:Nnn	113 , 6229 , 6230 , 6247 , 6249 , 6438	\prop_map_inline:Nn	115 , 6342 , 6342 , 6353 , 7642 , 7737 , 7977 , 7986
\prop_gput:Nno	6229	\prop_map_tokens:cn	6388
\prop_gput:NnV	6229	\prop_map_tokens:Nn	116 , 6388 , 6388 , 6402
\prop_gput:Nnx	6229	\prop_map_tokens_aux:nwn	6388 , 6390 , 6394 , 6400
\prop_gput:Non	6229	\prop_new:c	6150 , 6151
\prop_gput:Noo	6229	\prop_new:N	112 , 6150 , 6150 , 6157 , 6160 , 7028 , 7033 , 7624 , 7849 , 7890 , 7898 , 8168 , 8169 , 8770 , 8771 , 8993 , 9015
\prop_gput:NVn	6229 , 8205 , 8218	\prop_pop:cnN	6208 , 6364
\prop_gput:NVV	6229	\prop_pop:coN	6208
\prop_gput_if_new:cnn	6251	\prop_pop:NnN	113 , 6208 , 6208 , 6225 , 6226 , 6364 , 6364
\prop_gput_if_new:Nnn	113 , 6251 , 6253 , 6265	\prop_pop:NnNF	6383
\prop_gset_eq:cc	6162 , 6169 , 7245 , 7247	\prop_pop:NnNT	6382
\prop_gset_eq:cN	6162 , 6168 , 7103 , 7105	\prop_pop:NnNTF	115 , 6384
\prop_gset_eq:Nc	6162 , 6167	\prop_pop:NoN	6208
\prop_gset_eq:NN	112 , 6162 , 6166	\prop_pop_aux:NNNnnn	6208 , 6211 , 6217 , 6220
\prop_if_empty:c	6266	\prop_pop_aux_true:NNNnnn	6364 , 6367 , 6373 , 6376
\prop_if_empty:N	6266 , 6266		
\prop_if_empty:Nf	6277		
\prop_if_empty:NT	6276		
\prop_if_empty:NTF	114 , 6275 , 8347 , 9137		
\prop_if_empty_p:N	6274		
\prop_if_eq:cc	6440 , 6444		
\prop_if_eq:cN	6440 , 6442		
\prop_if_eq:Nc	6440 , 6443		
\prop_if_eq:NN	6440 , 6441		

- \prop_put:cnn [6229](#), [7299](#), [8925](#), [8927](#)
 - \prop_put:cno [6229](#)
 - \prop_put:cnV [6229](#)
 - \prop_put:cnx
 - . . . [6229](#), [7305](#), [7307](#), [7309](#), [7311](#),
 - [7316](#), [7321](#), [7326](#), [7333](#), [7340](#), [7573](#),
 - [7686](#), [7744](#), [7752](#), [7819](#), [7833](#), [7840](#)
 - \prop_put:con [6229](#)
 - \prop_put:coo [6229](#)
 - \prop_put:cVn [6229](#)
 - \prop_put:cVV [6229](#)
 - \prop_put:Nnn . . . [113](#), [6229](#), [6229](#), [6243](#),
 - [6245](#), [7029–7032](#), [7850](#), [7852](#), [7854](#),
 - [7856](#), [7858](#), [7860](#), [7862](#), [7864](#), [7866](#),
 - [7868](#), [7870](#), [7872](#), [7874](#), [7876](#), [7878](#),
 - [7880](#), [7882](#), [7884](#), [8171–8174](#), [8929](#)
 - \prop_put:Nno [6229](#), [7035–7037](#), [7039–7044](#)
 - \prop_put:NnV [6229](#)
 - \prop_put:Nnx
 - . . [6229](#), [7667](#), [7669](#), [7672](#), [7674](#), [7680](#)
 - \prop_put:Non [6229](#)
 - \prop_put:Noo [6229](#)
 - \prop_put:NVn [6229](#)
 - \prop_put:NVV [6229](#)
 - \prop_put_aux:NNnn [6229–6231](#)
 - \prop_put_aux:NNnnnnn . . [6229](#), [6233](#), [6235](#)
 - \prop_put_if_new:cnn [6251](#)
 - \prop_put_if_new:Nnn [113](#), [6251](#), [6251](#), [6264](#)
 - \prop_put_if_new_aux:NNnn [6252](#), [6254](#), [6255](#)
 - \prop_set_eq:cc [6162](#), [6165](#), [7238](#), [7240](#), [7508](#)
 - \prop_set_eq:cN . . [6162](#), [6164](#), [7231](#), [7233](#)
 - \prop_set_eq:Nc [6162](#), [6163](#), [7966](#)
 - \prop_set_eq:NN [112](#), [6162](#), [6162](#)
 - \prop_show:c [6356](#), [6419](#)
 - \prop_show:N . . [116](#), [6356](#), [6356](#), [6363](#), [6418](#)
 - \prop_split:Nnn [117](#), [6182](#), [6182](#), [6233](#), [6423](#)
 - \prop_split:NnTF [117](#),
 - [6170](#), [6170](#), [6184](#), [6189](#), [6191](#), [6200](#),
 - [6210](#), [6216](#), [6257](#), [6311](#), [6366](#), [6372](#)
 - \prop_split_aux:nnnn . . [6170](#), [6176](#), [6180](#)
 - \prop_split_aux:NnTF . . [6170](#), [6171](#), [6172](#)
 - \prop_split_aux:w [6170](#), [6174](#), [6177](#), [6181](#)
 - \protect [235](#)
 - \protected [68](#),
 - [82](#), [98](#), [104](#), [126](#), [133](#), [141](#), [143](#), [147](#),
 - [152](#), [157](#), [213](#), [266](#), [273](#), [292](#), [325](#), [736](#)
 - \protected@edef [8452](#), [8716](#)
 - \ProvidesClass [154](#)
 - \ProvidesExplClass [6](#), [146](#), [152](#)
 - \ProvidesExplFile [6](#), [146](#), [157](#)
 - \ProvidesExplPackage
 - [6](#), [146](#), [147](#), [333](#), [781](#), [1551](#),
 - [1915](#), [2427](#), [2545](#), [3265](#), [3992](#), [4321](#),
 - [5134](#), [5671](#), [6144](#), [6449](#), [7020](#), [8120](#),
 - [8140](#), [8599](#), [9302](#), [9951](#), [10068](#), [13432](#)
 - \ProvidesFile [159](#)
 - \ProvidesPackage [47](#), [149](#)
- Q**
- \q [2053](#), [2058](#)
 - \q_mark [43](#), [1877](#), [1879](#), [1883](#), [2432](#), [2433](#),
 - [3438](#), [3440](#), [4501](#), [4510](#), [4514](#), [4525](#),
 - [4696](#), [4697](#), [4700](#), [4703](#), [4704](#), [4714](#),
 - [4717](#), [4718](#), [4724](#), [4728](#), [4730](#), [4732](#),
 - [4758](#), [4759](#), [5717](#), [5718](#), [5734](#), [5743](#),
 - [5748](#), [5850](#), [5856](#), [5869](#), [5923](#), [5931](#),
 - [6114](#), [6115](#), [6124](#), [6175](#), [6177](#), [6178](#)
 - \q_nil [902](#), [905](#), [2345](#), [2349](#),
 - [2432](#), [2432](#), [2485](#), [2506](#), [3809](#), [3831](#),
 - [4563](#), [4575](#), [4576](#), [4716](#), [4720](#), [4737](#),
 - [4740](#), [4743](#), [4782](#), [4798](#), [4818](#), [5716](#),
 - [5720](#), [5727](#), [5794](#), [5803](#), [9328](#), [9359](#),
 - [9379](#), [9386](#), [9391](#), [13353](#), [13360](#),
 - [13366](#), [13372](#), [13378](#), [13384](#), [13390](#)
 - \q_no_value [43](#), [2082](#), [2432](#), [2434](#),
 - [2493](#), [2515](#), [6186](#), [6202](#), [6212](#), [6218](#),
 - [9328](#), [9336](#), [9341](#), [9358](#), [9364](#), [9595](#)
 - \q_prop [117](#),
 - [6148](#), [6148](#), [6149](#), [6175](#), [6176](#), [6178](#),
 - [6240](#), [6261](#), [6282](#), [6286](#), [6294](#), [6329](#),
 - [6332](#), [6350](#), [6391](#), [6394](#), [6407](#), [6410](#)
 - \q_recursion_stop
 - . . . [44](#), [904](#), [907](#), [999](#), [1068](#), [1816](#),
 - [2140](#), [2147](#), [2160](#), [2170](#), [2182](#), [2193](#),
 - [2436](#), [2437](#), [5735](#), [5959](#), [6013](#), [6408](#)
 - \q_recursion_tail
 - [2436](#), [2436](#), [2440](#), [2446](#), [2455](#), [2462](#),
 - [2472](#), [2479](#), [4629](#), [4647](#), [4656](#), [5068](#),
 - [5735](#), [5909](#), [5923](#), [5942](#), [5959](#), [6013](#),
 - [6283](#), [6329](#), [6334](#), [6350](#), [6391](#), [6396](#)
 - \q_stop . [43](#), [903](#), [906](#), [1104](#), [1106](#), [1114](#),
 - [1116](#), [1330](#), [1334](#), [1838](#), [1880](#), [1883](#),
 - [2082](#), [2085](#), [2318](#), [2320](#), [2332](#), [2334](#),
 - [2338](#), [2345](#), [2349](#), [2411](#), [2432](#), [2435](#),
 - [2748](#), [2750](#), [2788](#), [2790](#), [2795](#), [2797](#),
 - [2805](#), [2808](#), [2816](#), [2819](#), [2827](#), [2830](#),
 - [2838](#), [2841](#), [2847](#), [2850](#), [2855](#), [2857](#),
 - [2863](#), [2866](#), [2883](#), [2886](#), [2889](#), [2911](#),
 - [3104](#), [3111](#), [3120](#), [3129](#), [3429](#), [3430](#),
 - [3438](#), [3442](#), [3450](#), [3458](#), [3466](#), [3474](#),

- 3482, 3490, 3877, 3914, 4155, 4160,
 4510, 4525, 4698, 4700, 4705, 4707,
 4722, 4743, 4753, 4754, 4756, 4758,
 4759, 4766, 4774, 4776, 4782, 4798,
 4818, 5127, 5129, 5291, 5294, 5304,
 5307, 5475, 5496, 5503, 5584, 5586,
 5591, 5722, 5727, 5784, 5785, 5794,
 5796, 5856, 6044, 6077, 6116, 6124,
 6175, 6178, 8461, 8550, 9340, 9345,
 9347, 9358, 9363, 9365, 9388, 9391,
 9459, 9462, 9468, 9477, 9487, 10143,
 10144, 10163, 10168, 10173, 10179,
 10184, 10190, 10196, 10198, 10199,
 10202, 10206, 10210, 10246, 10251,
 10468, 10470, 10485, 10487, 10488,
 10494, 10501, 10503, 10506, 10508,
 10509, 10511, 10513, 10515, 10517,
 10519, 10521, 10523, 10524, 10533,
 10535, 10545, 10547, 10558, 10565,
 10567–10569, 10571, 10573, 10575,
 10577, 10579, 10581, 10583, 10585,
 10594, 10600, 10602, 10612, 10614,
 10621, 10623–10625, 10631, 10636,
 10641, 10646, 10651, 10656, 10661,
 10666, 10676, 10681, 10682, 10693,
 10695, 10714, 10734, 11204, 11209,
 11321, 11374, 11551, 11556, 11563,
 11566, 13353, 13357, 13360, 13363,
 13366, 13369, 13372, 13375, 13378,
 13381, 13384, 13387, 13390, 13393
 \q_tl_act_mark
 94, 2525, 2525, 4916, 4920, 4937
 \q_tl_act_stop 94,
 2525, 2526, 4916, 4920, 4924, 4933,
 4935, 4941, 4946, 4949, 4953, 4956
 \quark_if_nil:N 2483, 2483
 \quark_if_nil:n 2503, 2503
 \quark_if_nil:nF 2524
 \quark_if_nil:nT 2352, 2356, 2523
 \quark_if_nil:NTF .. 44, 3812, 3834, 9383
 \quark_if_nil:nTF 44, 2360, 2369, 2378,
 2387, 2522, 5799, 13359, 13365,
 13371, 13377, 13383, 13389, 13395
 \quark_if_nil:o 2503
 \quark_if_nil:oF 9338
 \quark_if_nil:V 2503
 \quark_if_nil_p:n 2521
 \quark_if_no_value:c 2483
 \quark_if_no_value:cF 9815
 \quark_if_no_value:N 2491
 \quark_if_no_value:n 2503, 2512
 \quark_if_no_value:N. 2483
 \quark_if_no_value:Nf 2501
 \quark_if_no_value:NT 2500
 \quark_if_no_value:NTF
 44, 2087, 2502, 7926, 7930, 8010, 8014
 \quark_if_no_value:nTF 44
 \quark_if_no_value_p:N 2499
 \quark_if_recursion_tail_break:N ...
 45, 2470, 2470, 4664
 \quark_if_recursion_tail_break:n ...
 .. 2470, 2476, 4636, 5073, 5915, 5928
 \quark_if_recursion_tail_stop:N
 44, 2438, 2438, 5971
 \quark_if_recursion_tail_stop:n
 45, 2452, 2452, 2468, 5739, 6018
 \quark_if_recursion_tail_stop:o .. 2452
 \quark_if_recursion_tail_stop_do:Nn
 45, 2438, 2444
 \quark_if_recursion_tail_stop_do:nn
 45, 2452, 2459, 2469
 \quark_if_recursion_tail_stop_do:on
 2452
 \quark_new:N
 43, 2431, 2431–2437, 2525, 2526, 6148
- ## R
- \R 1864, 2780
 \radical 464
 \raise 604
 \read 411
 \readline 686
 \relax 3–6, 9, 13,
 62, 70–80, 84–93, 96, 101, 131, 133,
 141, 143, 229, 233, 249, 281–289, 446
 \relpenalty 507
 \RequirePackage 57, 58
 \reverse_if:N 22, 785, 790, 4089–4091, 4773
 \right 505
 \righthyphenmin 561
 \rightskip 563
 \romannumeral 638
 \rule 7908, 7963
- ## S
- \S 2780
 \s_stop 45, 2540, 2540, 2541
 \savecatcodetable 763
 \savinghyphcodes 725
 \savingvdiscards 726

<code>\scan_align_safe_stop</code>	41	<code>\seq_gclear_new:c</code>	5151, 5154
<code>\scan_align_safe_stop:</code> .	2307, 2307, 3953	<code>\seq_gclear_new:N</code>	95, 5151, 5153
<code>\scan_new:N</code>	45, 2528, 2528, 2540	<code>\seq_gconcat:ccc</code>	5195
<code>\scan_stop</code>	9	<code>\seq_gconcat:NNN</code> 96, 5195, 5197, 5200, 10062	
<code>\scan_stop:</code>	308,	<code>\seq_get:cN</code>	5451, 5452
	322, 810, 810, 1071, 1095, 1125,	<code>\seq_get:NN</code>	99, 5451, 5451
	1143, 1153, 1171, 1592, 1861–1869,	<code>\seq_get_left:cN</code> .	5288, 5452, 5472, 5662
	2309, 2324, 2537, 2684, 2761, 3113,	<code>\seq_get_left:NN</code>	96, 5288,
	3122, 3131, 3164, 3166, 3168, 4200,		5288, 5296, 5451, 5472, 5472, 5661
	4211, 4216, 4241, 4246, 4249, 4255,	<code>\seq_get_left:NNF</code>	5488
	4277, 4288, 4293, 4298, 4304, 4314,	<code>\seq_get_left:NNT</code>	5487
	4315, 4431–4434, 4774, 6583, 7904,	<code>\seq_get_left:NNTF</code>	100, 5489
	7959, 8206, 8219, 10151–10153,	<code>\seq_get_left_aux:NnwN</code> .	5288, 5291, 5294
	10193, 10204, 10212, 10217, 10253,	<code>\seq_get_left_aux:Nw</code>	5475
	10254, 10270, 10278, 10317, 10326,	<code>\seq_get_right:cN</code>	5314, 5472
	10342, 10351, 10414, 10909, 10921,	<code>\seq_get_right:NN</code>	
	10922, 11054, 11058, 11070, 11074,		96, 5314, 5314, 5337, 5472, 5480
	11087, 11091, 11133, 11194, 11198,	<code>\seq_get_right:NNF</code>	5491
	11205–11207, 11215, 11226, 11276,	<code>\seq_get_right:NNT</code>	5490
	11378, 11453, 11461, 11503, 11517,	<code>\seq_get_right:NNTF</code>	100, 5492
	11521, 11559, 11560, 11569, 11570,	<code>\seq_get_right_aux:NN</code>	
	11587, 11595, 11603, 11619, 11633,		5314, 5317, 5320, 5483
	11646, 12001, 12324, 12334, 12378,	<code>\seq_get_right_loop:nn</code>	
	12454–12457, 12478, 12679, 12705,		5314, 5323, 5332, 5335, 5352
	12742, 12750, 12758, 12840, 12842,	<code>\seq_gpop:cN</code>	5451, 5456
	12844, 12879, 12887, 13091, 13093,	<code>\seq_gpop:NN</code> .	99, 5451, 5455, 10041, 13515
	13095, 13097, 13099, 13113, 13516	<code>\seq_gpop_left:cN</code>	5297, 5456, 5493
<code>\scantokens</code>	684	<code>\seq_gpop_left:NN</code>	
<code>\scriptfont</code>	630		97, 5297, 5299, 5313, 5455, 5493, 5500
<code>\scriptscriptfont</code>	631	<code>\seq_gpop_left:NNF</code>	5525
<code>\scriptscriptstyle</code>	476	<code>\seq_gpop_left:NNT</code>	5524
<code>\scriptspace</code>	516	<code>\seq_gpop_left:NNTF</code>	100, 5526
<code>\scriptstyle</code>	475	<code>\seq_gpop_right:cN</code>	5338, 5493
<code>\scrollmode</code>	441	<code>\seq_gpop_right:NN</code>	
<code>\seq_break</code>	103		97, 5338, 5340, 5365, 5493, 5514
<code>\seq_break:</code>	5329, 5361, 5366,	<code>\seq_gpop_right:NNF</code>	5531
	5366, 5374, 5469, 5477, 5484, 5497,	<code>\seq_gpop_right:NNT</code>	5530
	5504, 5511, 5518, 5555, 5575, 5583	<code>\seq_gpop_right:NNTF</code>	101, 5532
<code>\seq_break:n</code>		<code>\seq_gpush:cn</code>	5431, 5446
	103, 5278, 5281, 5366, 5367, 5563	<code>\seq_gpush:co</code>	5431, 5449
<code>\seq_clear:c</code>	5147, 5148	<code>\seq_gpush:cV</code>	5431, 5447
<code>\seq_clear:N</code> . . .	95, 5147, 5147, 5225, 8871	<code>\seq_gpush:cv</code>	5431, 5448
<code>\seq_clear_new:c</code>	5151, 5152	<code>\seq_gpush:cx</code>	5431, 5450
<code>\seq_clear_new:N</code>	95, 5151, 5151	<code>\seq_gpush:Nn</code>	100, 5431, 5441
<code>\seq_concat:ccc</code>	5195	<code>\seq_gpush:No</code>	5431, 5444
<code>\seq_concat:NNN</code> 96, 5195, 5195, 5199, 10001		<code>\seq_gpush:Nv</code>	5431, 5442, 10038
<code>\seq_display:c</code>	5664, 5666	<code>\seq_gpush:Nv</code>	5431, 5443
<code>\seq_display:N</code>	5664, 5665	<code>\seq_gpush:Nx</code>	5431, 5445, 13502
<code>\seq_gclear:c</code>	5147, 5150	<code>\seq_gput_left:cn</code>	5209, 5446
<code>\seq_gclear:N</code>	95, 5147, 5149	<code>\seq_gput_left:co</code>	5209, 5449

<code>\seq_gput_left:cV</code>	5209 , 5447	<code>\seq_if_in:co</code>	5265
<code>\seq_gput_left:cv</code>	5209 , 5448	<code>\seq_if_in:cV</code>	5265
<code>\seq_gput_left:cx</code>	5209 , 5450	<code>\seq_if_in:cv</code>	5265
<code>\seq_gput_left:Nn</code>		<code>\seq_if_in:cx</code>	5265
..... 96 , 5209 , 5209 , 5213 , 5214 , 5441		<code>\seq_if_in:Nn</code>	5265 , 5265
<code>\seq_gput_left:No</code>	5209 , 5444	<code>\seq_if_in:NnF</code> ...	5228 , 5284 , 5285 , 10046
<code>\seq_gput_left:Nv</code>	5209 , 5442	<code>\seq_if_in:NnT</code>	5282 , 5283
<code>\seq_gput_left:Nv</code>	5209 , 5443	<code>\seq_if_in:NnTF</code>	98 , 5286 , 5287 , 8876
<code>\seq_gput_left:Nx</code>	5209 , 5445	<code>\seq_if_in:No</code>	5265
<code>\seq_gput_right:cn</code>	5209	<code>\seq_if_in:Nv</code>	5265
<code>\seq_gput_right:co</code>	5209	<code>\seq_if_in:Nv</code>	5265
<code>\seq_gput_right:cV</code>	5209	<code>\seq_if_in:Nx</code>	5265
<code>\seq_gput_right:cv</code>	5209	<code>\seq_if_in_aux:</code>	5265 , 5274 , 5281
<code>\seq_gput_right:cx</code>	5209	<code>\seq_item:cn</code>	5543
<code>\seq_gput_right:Nn</code>		<code>\seq_item:n</code>	102 , 5138 ,
..... 96 , 5209 , 5211 , 5215 , 5216		5138 , 5202 , 5204 , 5210 , 5212 , 5217 ,
<code>\seq_gput_right:No</code>	5209	5270 , 5294 , 5307 , 5393 , 5398 , 5403 ,
<code>\seq_gput_right:Nv</code>	5209 , 9972	5409 , 5628 , 5629 , 5631 , 5636 , 5656
<code>\seq_gput_right:Nv</code>	5209	<code>\seq_item:Nn</code>	101 , 5543 , 5543 , 5566
<code>\seq_gput_right:Nx</code>	5209 , 10033	<code>\seq_item_aux:nnn</code>	5543 , 5545 , 5559 , 5564
<code>\seq_gremove_all:cn</code>	5235	<code>\seq_length:c</code>	5533
<code>\seq_gremove_all:Nn</code> .	97 , 5235 , 5237 , 5260	<code>\seq_length:N</code> .	101 , 5533 , 5533 , 5542 , 5550
<code>\seq_gremove_duplicates:c</code>	5219	<code>\seq_length_aux:n</code>	5533 , 5538 , 5541
<code>\seq_gremove_duplicates:N</code>		<code>\seq_map_break</code>	98
..... 97 , 5219 , 5221 , 5234		<code>\seq_map_break:</code> ..	5366 , 5368 , 5380 , 10011
<code>\seq_greverse:c</code>	5621	<code>\seq_map_break:n</code>	99 , 5366 , 5369
<code>\seq_greverse:N</code> ...	102 , 5621 , 5624 , 5639	<code>\seq_map_function:cN</code>	5377
<code>\seq_gset_eq:cc</code>	5155 , 5162	<code>\seq_map_function:NN</code>	4 ,
<code>\seq_gset_eq:cN</code>	5155 , 5161	98 , 5377 , 5377 , 5389 , 5462 , 5538 , 5567
<code>\seq_gset_eq:Nc</code>	5155 , 5160	<code>\seq_map_function_aux:NNn</code>	
<code>\seq_gset_eq:NN</code>	95 , 5155 , 5159 , 5222	5377 , 5379 , 5383 , 5387
<code>\seq_gset_filter:NNn</code> ...	102 , 5640 , 5642	<code>\seq_map_inline:cn</code>	5412
<code>\seq_gset_from_clist:cc</code>	5595	<code>\seq_map_inline:Nn</code>	98 ,
<code>\seq_gset_from_clist:cN</code>	5595	5226 , 5412 , 5412 , 5418 , 10005 , 10055
<code>\seq_gset_from_clist:cn</code>	5595	<code>\seq_map_variable:ccn</code>	5419
<code>\seq_gset_from_clist:Nc</code>	5595	<code>\seq_map_variable:cNn</code>	5419
<code>\seq_gset_from_clist:NN</code>		<code>\seq_map_variable:Ncn</code>	5419
..... 102 , 5595 , 5605 , 5618 , 5619		<code>\seq_map_variable:NNn</code>	
<code>\seq_gset_from_clist:Nn</code>	5595 , 5610 , 5620	98 , 5419 , 5419 , 5429 , 5430
<code>\seq_gset_map:NNn</code>	102 , 5650 , 5652	<code>\seq_mapthread_function:ccN</code>	5569
<code>\seq_gset_split:Nnn</code>	95 , 5163 , 5165	<code>\seq_mapthread_function:cNN</code>	5569
<code>\seq_if_empty:c</code>	5261 , 5263	<code>\seq_mapthread_function:NcN</code>	5569
<code>\seq_if_empty:N</code>	5261 , 5261	<code>\seq_mapthread_function:NNN</code>	
<code>\seq_if_empty:NtF</code> .	97 , 6089 , 9130 , 13513	101 , 5569 , 5569 , 5593 , 5594
<code>\seq_if_empty_break_return_false:N</code> .		<code>\seq_mapthread_function_aux:NN</code>	
.....	5465 , 5465 ,	5569 , 5571 , 5578
.....	5474 , 5482 , 5495 , 5502 , 5509 , 5516	<code>\seq_mapthread_function_aux:Nnnwnn</code> .	
<code>\seq_if_empty_err_break:N</code>	5569 , 5580 , 5586 , 5591
.....	102 , 5290 , 5303 , 5316 , 5344 , 5370 , 5370	<code>\seq_new:c</code>	5145 , 5146
<code>\seq_if_in:cn</code>	5265		

\seq_new:N	4, 95, <u>5145</u> , 5145, 5218, 8863, 9966, 9967, 9976, 9978, 9981, 13468	\seq_put_left:Nn
\seq_pop:cN <u>5451</u> , 5454	96, <u>5201</u> , 5201, 5205, 5206, 5431
\seq_pop:NN 99, <u>5451</u> , 5453	\seq_put_left:No <u>5201</u> , 5434
\seq_pop_item_def 103	\seq_put_left:Nv <u>5201</u> , 5433
\seq_pop_item_def:	\seq_put_left:Nx <u>5201</u> , 5435
.....	5257, 5328, 5360, <u>5390</u> , 5406, 5416, 5427, 5648, 5658, 6100	\seq_put_right:cn <u>5201</u>
\seq_pop_left:cN <u>5297</u> , 5454, <u>5493</u>	\seq_put_right:co <u>5201</u>
\seq_pop_left:NN	\seq_put_right:cV <u>5201</u>
.....	96, <u>5297</u> , 5297, 5312, 5453, <u>5493</u> , 5493	\seq_put_right:cv <u>5201</u>
\seq_pop_left:NnF 5522	\seq_put_right:cx <u>5201</u>
\seq_pop_left:NNT 5521	\seq_put_right:Nn 96, <u>5201</u> , 5203, 5207, 5208, 5229, 8879, 10047
\seq_pop_left:NNTF 100, 5523	\seq_put_right:No <u>5201</u>
\seq_pop_left_aux:NNN	\seq_put_right:Nv <u>5201</u>
.....	5297, 5298, 5300, 5301	\seq_put_right:Nx <u>5201</u>
\seq_pop_left_aux:NnwNNN	\seq_remove_all:cn <u>5235</u>
.....	5297, 5304, 5307, 5496, 5503	\seq_remove_all:Nn
\seq_pop_right:cN <u>5338</u> , <u>5493</u>	97, <u>5235</u> , 5235, 5259, 10050
\seq_pop_right:NN	\seq_remove_all_aux:NNn
.....	97, <u>5338</u> , 5338, 5364, <u>5493</u> , 5507	<u>5235</u> , 5236, 5238, 5239
\seq_pop_right:NnF 5528	\seq_remove_duplicates:c <u>5219</u>
\seq_pop_right:NNT 5527	\seq_remove_duplicates:N
\seq_pop_right:NNTF 101, 5529	97, <u>5219</u> , 5219, 5233, 10053
\seq_pop_right_aux:NNN	\seq_remove_duplicates_aux:NN
.....	<u>5338</u> , 5339, 5341, 5342	<u>5219</u> , 5220, 5222, 5223
\seq_pop_right_aux_ii:NNN	\seq_reverse:c <u>5621</u>
.....	<u>5338</u> , 5345, 5348, 5510, 5517	\seq_reverse:N 102, <u>5621</u> , 5622, 5638
\seq_push:cn <u>5431</u> , 5436	\seq_reverse_aux:NN	<u>5621</u> , 5623, 5625, 5626
\seq_push:co <u>5431</u> , 5439	\seq_reverse_aux_item:nwn	<u>5621</u> , 5629, 5633
\seq_push:cV <u>5431</u> , 5437	\seq_set_eq:cc 5155, 5158
\seq_push:cv 5438	\seq_set_eq:cN 5155, 5157
\seq_push:cx <u>5431</u> , 5440	\seq_set_eq:Nc 5155, 5156
\seq_push:Nn 100, <u>5431</u> , 5431	\seq_set_eq:NN
\seq_push:No <u>5431</u> , 5434	95, 5155, 5155, 5220, 9999, 10016
\seq_push:Nv <u>5431</u> , 5432	\seq_set_filter:NNn 102, <u>5640</u> , 5640
\seq_push:Nx <u>5431</u> , 5433	\seq_set_filter_aux:NNNn
\seq_push:Nx <u>5431</u> , 5435	<u>5640</u> , 5641, 5643, 5644
\seq_push_item_def:n	\seq_set_from_clist:cc <u>5595</u>
.....	103, 5241, 5322, 5350, <u>5390</u> , 5390, 5414, 5646, 5656, 6092	\seq_set_from_clist:cN <u>5595</u>
\seq_push_item_def:x <u>5390</u> , 5395, 5421	\seq_set_from_clist:cn <u>5595</u>
\seq_push_item_def_aux:	\seq_set_from_clist:Nc <u>5595</u>
.....	<u>5390</u> , 5392, 5397, 5400	\seq_set_from_clist:NN 102, <u>5595</u> , 5595, 5615, 5616, 10000, 10061
\seq_put_left:cn <u>5201</u> , 5436	\seq_set_from_clist:Nn	. <u>5595</u> , 5600, 5617
\seq_put_left:co <u>5201</u> , 5439	\seq_set_map:NNn 102, <u>5650</u> , 5650
\seq_put_left:cV <u>5201</u> , 5437	\seq_set_map_aux:NNNn
\seq_put_left:cv <u>5201</u> , 5438	<u>5650</u> , 5651, 5653, 5654
\seq_put_left:cx <u>5201</u> , 5440	\seq_set_split:Nnn 95, <u>5163</u> , 5163

<code>\seq_set_split_aux:NNnn</code>	<code>\skip_horizontal:N</code>
..... 5163 , 5164 , 5166 , 5167 80 , 4244 , 4244 , 4246 , 4250
<code>\seq_set_split_aux_end:</code>	<code>\skip_horizontal:n</code> 4244 , 4245 , 6968 , 6998
.. 5163 , 5176 , 5180 , 5187 , 5191 , 5193	<code>\skip_if_eq:nn</code>
<code>\seq_set_split_aux_i:w</code>	4220 , 4220
..... 5163 , 5174 , 5181 , 5187	<code>\skip_if_eq:nnTF</code>
<code>\seq_set_split_aux_ii:w</code> 5163 , 5189 , 5193	77
<code>\seq_show:c</code>	<code>\skip_if_infinite_glue:n</code>
5457 , 5666	4230 , 4230
<code>\seq_show:N</code> ... 100 , 5457 , 5457 , 5464 , 5665	<code>\skip_if_infinite_glue:nTF</code>
<code>\seq_tmp:w</code>	77 , 4307
5621 , 5628 , 5631	<code>\skip_new:c</code>
<code>\seq_top:cn</code>	4187
5660 , 5662	<code>\skip_new:N</code>
<code>\seq_top:NN</code>	76 , 4187 , 4188 , 4194 , 4258 – 4262 , 10456
5660 , 5661	<code>\skip_set:cn</code>
<code>\seq_use:c</code>	4199
5567	<code>\skip_set:Nn</code> ... 76 , 4199 , 4199 , 4201 , 4202
<code>\seq_use:N</code>	<code>\skip_set_eq:cc</code>
101 , 5567 , 5567 , 5568	4204
<code>\seq_wrap_item:n</code>	<code>\skip_set_eq:cN</code>
... 5170 , 5194 , 5217 , 5217 , 5253 ,	4204
5350 , 5598 , 5603 , 5608 , 5613 , 5646	<code>\skip_set_eq:Nc</code>
<code>\set@color</code>	4204
8135	<code>\skip_set_eq:NN</code>
<code>\setbox</code>	76 , 4204 , 4204 – 4206
612	<code>\skip_show:c</code>
<code>\setlanguage</code>	4252
370	<code>\skip_show:N</code>
<code>\sfcode</code>	78 , 4252 , 4252 , 4253
667	<code>\skip_show:n</code>
<code>\sffamily</code>	78 , 4254 , 4254
7896	<code>\skip_split_finite_else_action:nnNN</code>
<code>\shipout</code> 81 , 4305 , 4305
577	<code>\skip_sub:cn</code>
<code>\show</code>	4210
420	<code>\skip_sub:Nn</code> ... 77 , 4210 , 4215 , 4217 , 4218
<code>\showbox</code>	<code>\skip_use:c</code>
422	4242
<code>\showboxbreadth</code>	<code>\skip_use:N</code>
436	77 , 4241 , 4242 , 4242 , 4243
<code>\showboxdepth</code>	<code>\skip_vertical:c</code>
437	4244
<code>\showgroups</code>	<code>\skip_vertical:N</code> 80 , 4244 , 4247 , 4249 , 4251
697	<code>\skip_vertical:n</code>
<code>\showifs</code>	4244 , 4248
698	<code>\skip_zero:c</code>
<code>\showlists</code>	4195
423	<code>\skip_zero:N</code>
<code>\showthe</code>	76 , 4195 , 4195 – 4197
421	<code>\skipdef</code>
<code>\showtokens</code>	357
685	<code>\space</code>
<code>\skewchar</code>	49 , 205
634	<code>\spacefactor</code>
<code>\skip</code>	576
658	<code>\spaceskip</code>
<code>\skip_add:cn</code>	571
4210	<code>\span</code>
<code>\skip_add:Nn</code> ... 76 , 4210 , 4210 , 4212 , 4213	384
<code>\skip_eval:n</code>	<code>\special</code>
77 , 4223 , 4240 , 4240	646
<code>\skip_gadd:cn</code>	<code>\splitbotmark</code>
4210	455
<code>\skip_gadd:Nn</code>	<code>\splitbotmarks</code>
76 , 4210 , 4212 , 4214	681
<code>\skip_gset:cn</code>	<code>\splitdiscards</code>
4199	728
<code>\skip_gset:Nn</code>	<code>\splitfirstmark</code>
76 , 4199 , 4201 , 4203	454
<code>\skip_gset_eq:cc</code>	<code>\splitfirstmarks</code>
4204	680
<code>\skip_gset_eq:cN</code>	<code>\splitmaxdepth</code>
4204	624
<code>\skip_gset_eq:Nc</code>	<code>\splittopskip</code>
4204	625
<code>\skip_gset_eq:NN</code> ... 76 , 4204 , 4207 – 4209	<code>\str_head:n</code>
<code>\skip_gsub:cn</code>	91 , 4762 , 4762 , 4783 , 4826
4210	<code>\str_head_aux:w</code>
<code>\skip_gsub:Nn</code>	4762 , 4764 , 4768
77 , 4210 , 4217 , 4219	<code>\str_if_eq:nn</code>
<code>\skip_gzero:c</code>	1495 , 1495
4195	<code>\str_if_eq:nnF</code>
<code>\skip_gzero:N</code>	1908 , 1909 , 9144
76 , 4195 , 4196 , 4198	<code>\str_if_eq:nnT</code>
<code>\skip_horizontal:c</code>	1906 , 1907 , 5243
4244	<code>\str_if_eq:nnTF</code>
	21 , 1910 , 1911 , 2174 , 3882 , 3885 , 9348

<code>\str_if_eq:no</code>	1904	12546 , 12551 , 12552 , 12556 , 12557 ,
<code>\str_if_eq:nV</code>	1904	12560 , 12561 , 12564 , 12565 , 12655 ,
<code>\str_if_eq:on</code>	1904	12659 , 12707 , 12730 , 12759 , 12789 ,
<code>\str_if_eq:Vn</code>	1904	12797 , 12800 , 12847 , 12850 , 12851 ,
<code>\str_if_eq:VV</code>	1904	12853 , 12869 , 12889 , 12893 , 12906 ,
<code>\str_if_eq:xx</code>	1495 , 1501	12907 , 12910 , 12911 , 12916 , 12917
<code>\str_if_eq:xxTF</code> ..	2186 , 6288 , 6412 , 8514	<code>\tex_afterassignment:D</code>
<code>\str_if_eq_p:nn</code>	1904 , 1905 372 , 2943 , 3029 , 10194
<code>\str_if_eq_return:xx</code>	4845 ,	<code>\tex_aftergroup:D</code>
4896 , 4896 , 4909 , 4913 , 4914 , 5081		373 , 815
<code>\str_length_loop:NNNNNNNN</code>		<code>\tex_atop:D</code>
..... 8544 , 8549 , 8553 , 8559		469
<code>\str_length_skip_spaces:N</code> 8476 , 8544 , 8544		<code>\tex_atopwithdelims:D</code>
<code>\str_length_skip_spaces:n</code> 8544 , 8545 , 8546		470
<code>\str_tail:n</code>	91 , 4762 , 4770	<code>\tex_badness:D</code>
<code>\str_tail_aux:w</code>	4762 , 4772 , 4776	617
<code>\strcmp</code>	230	<code>\tex_baselineskip:D</code>
<code>\string</code>	223 , 238 , 252 , 639	545
		<code>\tex_batchmode:D</code>
		438
		<code>\tex_begingroup:D</code>
		376 , 811
		<code>\tex_belowdisplayshortskip:D</code>
		482
		<code>\tex_belowdisplayskip:D</code>
		483
		<code>\tex_binoppenalty:D</code>
		506
		<code>\tex_botmark:D</code>
		453
		<code>\tex_box:D</code>
		661 , 6488 , 6508
		<code>\tex_boxmaxdepth:D</code>
		623
		<code>\tex_brokenpenalty:D</code>
		580
		<code>\tex_catcode:D</code>
	 665 , 1096 , 1863–1869 , 2310 , 2325 ,
		2550 , 2552 , 2554 , 3170 , 4433 , 4434
		<code>\tex_char:D</code>
		519
		<code>\tex_chardef:D</code>
		354 , 840–
		844 , 846 , 1083 , 1084 , 1944 , 1946 ,
		2869 , 3378 , 3379 , 8181 , 8184 , 13476
		<code>\tex_cleaders:D</code>
		537
		<code>\tex_closein:D</code>
		413 , 8319 , 8332
		<code>\tex_closeout:D</code>
		408
		<code>\tex_clubpenalty:D</code>
		548
		<code>\tex_copy:D</code>
		605 , 6482 , 6509
		<code>\tex_count:D</code>
		656
		<code>\tex_countdef:D</code>
		355 , 837 , 2801
		<code>\tex_cr:D</code>
		380
		<code>\tex_crcr:D</code>
		381
		<code>\tex_csname:D</code>
		443 , 806
		<code>\tex_day:D</code>
		651
		<code>\tex_deadcycles:D</code>
		585
		<code>\tex_def:D</code>
		350 , 819–821 , 831 , 850
		<code>\tex_defaultthyphenchar:D</code>
		635
		<code>\tex_defaultskewchar:D</code>
		636
		<code>\tex_delcode:D</code>
		666
		<code>\tex_delimiter:D</code>
		460
		<code>\tex_delimiterfactor:D</code>
		509
		<code>\tex_delimitershortfall:D</code>
		508
		<code>\tex_dimen:D</code>
		657
		<code>\tex_dimendef:D</code>
		356 , 2823

T

<code>\T</code>	1866 , 2740 , 2774 , 2873
<code>\t</code>	2777
<code>\tabskip</code>	385
<code>\tempa</code>	106 , 108 , 109 , 118 , 124
<code>\tempb</code>	107 , 108
<code>\tex_above:D</code>	467
<code>\tex_abovedisplayshortskip:D</code>	480
<code>\tex_abovedisplayskip:D</code>	481
<code>\tex_abovewithdelims:D</code>	468
<code>\tex_accent:D</code>	518
<code>\tex_adjdemerits:D</code>	555
<code>\tex_advance:D</code>	362 , 3398 , 3400 ,
	3410 , 3412 , 4035 , 4040 , 4211 , 4216 ,
	4288 , 4293 , 10244 , 10255 , 10261 ,
	10271 , 10279 , 10307 , 10318 , 10327 ,
	10332 , 10343 , 10352 , 10384 , 10426 ,
	10838 , 10874 , 10900 , 10908 , 10914 ,
	10938 , 10951 , 10976 , 11061 , 11062 ,
	11076 , 11077 , 11202 , 11210 , 11219 ,
	11319 , 11350–11352 , 11354 , 11355 ,
	11387 , 11388 , 11392 , 11393 , 11402 ,
	11403 , 11406 , 11407 , 11413 , 11536 ,
	11537 , 11549 , 11561 , 11571 , 11576 ,
	11604 , 11609 , 11620 , 11638 , 11647 ,
	11695 , 11696 , 11699 , 11700 , 11762 ,
	11774 , 12004 , 12005 , 12039 , 12044 ,
	12047 , 12048 , 12052 , 12053 , 12058 ,
	12059 , 12063 , 12064 , 12067 , 12068 ,
	12071 , 12072 , 12421 , 12441 , 12499 ,
	12503 , 12525 , 12540 , 12541 , 12545 ,

<code>\tex_discretionary:D</code>	520	3402, 3404, 3414, 3416, 3423, 4008,
<code>\tex_displayindent:D</code>	485	4013, 4019, 4025, 4029, 4036, 4041,
<code>\tex_displaylimits:D</code>	495	4196, 4201, 4207, 4212, 4217, 4273,
<code>\tex_displaystyle:D</code>	473	4278, 4284, 4289, 4294, 5093, 6484,
<code>\tex_displaywidowpenalty:D</code>	484	6490, 6544, 6585, 6591, 6597, 6627,
<code>\tex_displaywidth:D</code>	486	6633, 6639, 6645, 8581, 8585, 13476
<code>\tex_divide:D</code>	363, 10272, 10319, 10344, 10913, 11181, 11372, 11437, 11481, 11526, 11529, 11531, 11533, 11597, 11639, 12752, 12802–12804	<code>\tex_globaldefs:D</code>
<code>\tex_doublehyphendemerits:D</code>	553	371
<code>\tex_dp:D</code>	664, 6494	<code>\tex_halign:D</code>
<code>\tex_dump:D</code>	647	378
<code>\tex_edef:D</code>	351, 851	<code>\tex_hangafter:D</code>
<code>\tex_else:D</code>	404, 788, 847	556
<code>\tex_emergencystretch:D</code>	568	<code>\tex_hangingindent:D</code>
<code>\tex_end:D</code>	442, 766, 1209, 8806, 8948	557
<code>\tex_endcsname:D</code>	444, 807	<code>\tex_hbadness:D</code>
<code>\tex_endgroup:D</code>	377, 764, 812	618
<code>\tex_endinput:D</code>	416, 8817	<code>\tex_hbox:D</code>
<code>\tex_endlinechar:D</code>	307, 308, 322, 458, 4450	613, 6583, 6584, 6589, 6595, 6609, 6610
<code>\tex_eqno:D</code>	478	<code>\tex_hfil:D</code>
<code>\tex_errhelp:D</code>	424, 8724	521
<code>\tex_errmessage:D</code>	418, 1201, 8744	<code>\tex_hfill:D</code>
<code>\tex_errorcontextlines:D</code>	425, 8762	523
<code>\tex_errorstopmode:D</code>	439	<code>\tex_hfilneg:D</code>
<code>\tex_escapechar:D</code>	457, 8401, 8437, 8443	522
<code>\tex_everycr:D</code>	386	<code>\tex_hfuzz:D</code>
<code>\tex_everydisplay:D</code>	487, 767	620
<code>\tex_everyhbox:D</code>	626	<code>\tex_hoffset:D</code>
<code>\tex_everyjob:D</code>	655, 4880, 4882, 9957, 9959, 9969, 9971	595
<code>\tex_everymath:D</code>	511, 768	<code>\tex_holdinginserts:D</code>
<code>\tex_everypar:D</code>	574	598
<code>\tex_everyvbox:D</code>	627	<code>\tex_hrule:D</code>
<code>\tex_exhyphenpenalty:D</code>	550	534
<code>\tex_expandafter:D</code>	374, 801	<code>\tex_hsize:D</code>
<code>\tex_fam:D</code>	366	559, 7132, 7175
<code>\tex_fi:D</code>	405, 789, 849	<code>\tex_hskip:D</code>
<code>\tex_finalhyphendemerits:D</code>	554	524, 4244
<code>\tex_firstmark:D</code>	452	<code>\tex_hss:D</code>
<code>\tex_floatingpenalty:D</code>	599	525, 6612, 6614, 6952
<code>\tex_font:D</code>	365	<code>\tex_ht:D</code>
<code>\tex_fontdimen:D</code>	632	663, 6493
<code>\tex_fontname:D</code>	456	<code>\tex_hyphen:D</code>
<code>\tex_futurelet:D</code>	361, 2937, 2939	348, 769
<code>\tex_gdef:D</code>	352, 864	<code>\tex_hyphenation:D</code>
<code>\tex_global:D</code>	336, 341, 343, 367, 816, 816, 1299, 1304, 1946, 2939, 3362, 3382, 3394,	649
		<code>\tex_hyphenchar:D</code>
		633
		<code>\tex_hyphenpenalty:D</code>
		551
		<code>\tex_if:D</code>
		387, 791, 792
		<code>\tex_ifcase:D</code>
		388, 3274
		<code>\tex_ifcat:D</code>
		389, 793
		<code>\tex_ifdim:D</code>
		392, 3996
		<code>\tex_ifeof:D</code>
		393, 8144
		<code>\tex_iffalse:D</code>
		398, 786
		<code>\tex_ifhbox:D</code>
		394, 6520
		<code>\tex_ifhmode:D</code>
		400, 796
		<code>\tex_ifinner:D</code>
		403, 798
		<code>\tex_ifmmode:D</code>
		401, 795
		<code>\tex_ifnum:D</code>
		390, 813, 3272
		<code>\tex_ifodd:D</code>
		391, 1226, 1919, 1920, 3273
		<code>\tex_iftrue:D</code>
		399, 785
		<code>\tex_ifvbox:D</code>
		395, 6521
		<code>\tex_ifvmode:D</code>
		402, 797
		<code>\tex_ifvoid:D</code>
		396, 6522
		<code>\tex_ifx:D</code>
		397, 794
		<code>\tex_ignorespaces:D</code>
		445
		<code>\tex_immediate:D</code>
		407, 1196, 1198, 8219, 8370
		<code>\tex_indent:D</code>
		541
		<code>\tex_input:D</code>
		415, 770, 10040
		<code>\tex_inputlineno:D</code>
		417, 1216, 1856, 8669

<code>\tex_insert:D</code>	597	<code>\tex_maxdepth:D</code>	583
<code>\tex_insertpenalties:D</code>	600	<code>\tex_meaning:D</code>	642, 804, 808
<code>\tex_interlinepenalty:D</code>	579	<code>\tex_medmuskip:D</code>	513
<code>\tex_italic_correction:D</code>	771	<code>\tex_message:D</code>	419
<code>\tex_italiccor:D</code>	347	<code>\tex_mkern:D</code>	466
<code>\tex_jobname:D</code>	654, 4890, 9960	<code>\tex_month:D</code>	652
<code>\tex_kern:D</code> 532, 6720, 6950, 7470, 7475, 7541, 7542, 7649, 7650, 8053, 8054		<code>\tex_moveleft:D</code>	602, 6513
<code>\tex_language:D</code>	449	<code>\tex_moveright:D</code>	603, 6515
<code>\tex_lastbox:D</code>	606, 6542, 7016	<code>\tex_mskip:D</code>	463
<code>\tex_lastkern:D</code>	539	<code>\tex_multiply:D</code>	364, 11111, 11318, 11532, 11548, 12526, 13115
<code>\tex_lastpenalty:D</code>	645	<code>\tex_muskip:D</code>	660
<code>\tex_lastskip:D</code>	540	<code>\tex_muskipdef:D</code>	358
<code>\tex_lccode:D</code>	668, 1095, 1861, 1862, 2309, 2324, 2626, 2628, 2630, 3172, 4431, 4432	<code>\tex_newlinechar:D</code>	414, 4451
<code>\tex_leaders:D</code>	536	<code>\tex_noalign:D</code>	382
<code>\tex_left:D</code>	504	<code>\tex_noboundary:D</code>	517
<code>\tex_lefthyphenmin:D</code>	560	<code>\tex_noexpand:D</code>	375, 802
<code>\tex_leftskip:D</code>	562	<code>\tex_noindent:D</code>	543
<code>\tex_leqno:D</code>	479	<code>\tex_nolimits:D</code>	497
<code>\tex_let:D</code>	337, 341, 343, 349, 766–776, 785–818, 834, 850, 851, 864, 865, 1292, 1547, 1919, 1920	<code>\tex_nonscript:D</code>	477
<code>\tex_limits:D</code>	496	<code>\tex_nonstopmode:D</code>	440
<code>\tex_linepenalty:D</code>	552	<code>\tex_nulldelimiterspace:D</code>	510
<code>\tex_lineskip:D</code>	546	<code>\tex_nullfont:D</code>	628, 2896
<code>\tex_lineskiplimit:D</code>	547	<code>\tex_number:D</code>	637, 3269
<code>\tex_long:D</code> ... 368, 816, 817, 819, 821, 853, 855, 861, 863, 867, 869, 875, 877		<code>\tex_omit:D</code>	383
<code>\tex_looseness:D</code>	564	<code>\tex_openin:D</code>	409, 8206
<code>\tex_lower:D</code>	601, 6519	<code>\tex_openout:D</code>	410, 8219
<code>\tex_lowercase:D</code> 640, 1097, 1870, 4435, 4474		<code>\tex_or:D</code>	406, 787
<code>\tex_mag:D</code>	448	<code>\tex_outer:D</code>	369
<code>\tex_mark:D</code>	450	<code>\tex_output:D</code>	584
<code>\tex_mathaccent:D</code>	461	<code>\tex_outputpenalty:D</code>	594
<code>\tex_mathbin:D</code>	491	<code>\tex_over:D</code>	471
<code>\tex_mathchar:D</code>	462	<code>\tex_overfullrule:D</code>	622
<code>\tex_mathchardef:D</code> .. 359, 848, 3374, 3375		<code>\tex_overline:D</code>	502
<code>\tex_mathchoice:D</code>	459	<code>\tex_overwithdelims:D</code>	472
<code>\tex_mathclose:D</code>	492	<code>\tex_pagedepth:D</code>	586
<code>\tex_mathcode:D</code> 670, 2620, 2622, 2624, 3171		<code>\tex_pagefilllstretch:D</code>	590
<code>\tex_mathinner:D</code>	493	<code>\tex_pagefillstretch:D</code>	589
<code>\tex_mathop:D</code>	494	<code>\tex_pagefilstretch:D</code>	588
<code>\tex_mathopen:D</code>	498	<code>\tex_pagegoal:D</code>	592
<code>\tex_mathord:D</code>	499	<code>\tex_pageshrink:D</code>	591
<code>\tex_mathpunct:D</code>	500	<code>\tex_pagestretch:D</code>	587
<code>\tex_mathrel:D</code>	501	<code>\tex_pagetotal:D</code>	593
<code>\tex_mathsurround:D</code>	512	<code>\tex_par:D</code>	542, 8127
<code>\tex_maxdeadcycles:D</code>	582	<code>\tex_parfillskip:D</code>	573
		<code>\tex_parindent:D</code>	566
		<code>\tex_parshape:D</code>	558
		<code>\tex_parskip:D</code>	565
		<code>\tex_patterns:D</code>	648
		<code>\tex_pausing:D</code>	435

<code>\tex_penalty:D</code>	643	<code>\tex_spaceskip:D</code>	571
<code>\tex_postdisplaypenalty:D</code>	490	<code>\tex_span:D</code>	384
<code>\tex_predisplaypenalty:D</code>	489	<code>\tex_special:D</code>	646
<code>\tex_predisplaysize:D</code>	488	<code>\tex_splitbotmark:D</code>	455
<code>\tex_pretolerance:D</code>	569	<code>\tex_splitfirstmark:D</code>	454
<code>\tex_prevdepth:D</code>	616	<code>\tex_splitmaxdepth:D</code>	624
<code>\tex_prevgraf:D</code>	575	<code>\tex_splittopskip:D</code>	625
<code>\tex_protected:D</code> 816 , 818 , 831 , 852 , 854 , 856–859, 861 , 863 , 871 , 873 , 875 , 877		<code>\tex_string:D</code>	639, 805
<code>\tex_radical:D</code>	464	<code>\tex_tabskip:D</code>	385
<code>\tex_raise:D</code>	604, 6517	<code>\tex_textfont:D</code>	629
<code>\tex_read:D</code>	411, 8579, 8581	<code>\tex_textstyle:D</code>	474
<code>\tex_relax:D</code>	446, 810, 3271, 3998	<code>\tex_the:D</code>	
<code>\tex_relpenalty:D</code>	507		308, 447, 1216, 1598, 1602, 1856, 2552, 2622, 2628, 2634, 2640, 3177, 3180, 3183, 3186, 3189, 3426, 4167, 4242, 4299, 4882, 9959, 9971, 13502
<code>\tex_right:D</code>	505	<code>\tex_thickmuskip:D</code>	515
<code>\tex_righthyphenmin:D</code>	561	<code>\tex_thinmuskip:D</code>	514
<code>\tex_rightskip:D</code>	563	<code>\tex_time:D</code>	650
<code>\tex_romannumeral:D</code>		<code>\tex_toks:D</code>	659
	638, 814, 1568, 1580, 1586, 1628, 1632, 1637, 1643, 1649, 1655, 1667, 1672, 1674, 1681, 1735, 1742, 1747, 1755, 1757, 1760, 1767, 1773, 1782, 1796, 1800, 1805, 2143, 2156, 2169, 2181, 2192, 4866, 4916, 4970, 4989, 5032, 5034, 5054, 9169	<code>\tex_toksdef:D</code>	360, 2834
<code>\tex_scriptfont:D</code>	630	<code>\tex_tolerance:D</code>	570
<code>\tex_scriptscriptfont:D</code>	631	<code>\tex_topmark:D</code>	451
<code>\tex_scriptscriptstyle:D</code>	476	<code>\tex_topskip:D</code>	581
<code>\tex_scriptspace:D</code>	516	<code>\tex_tracingcommands:D</code>	426
<code>\tex_scriptstyle:D</code>	475	<code>\tex_tracinglostchars:D</code>	427
<code>\tex_scrollmode:D</code>	441	<code>\tex_tracingmacros:D</code>	428
<code>\tex_setbox:D</code>		<code>\tex_tracingonline:D</code>	429, 6575
	612, 6482, 6488, 6542, 6584, 6589, 6595, 6626, 6631, 6637, 6643, 6665	<code>\tex_tracingoutput:D</code>	430
<code>\tex_setlanguage:D</code>	370	<code>\tex_tracingpages:D</code>	431
<code>\tex_sfcode:D</code> . 667 , 2638 , 2640 , 2642 , 3174		<code>\tex_tracingparagraphs:D</code>	432
<code>\tex_shipout:D</code>	577	<code>\tex_tracingrestores:D</code>	433
<code>\tex_show:D</code>	420, 809	<code>\tex_tracingstats:D</code>	434
<code>\tex_showbox:D</code>	422, 6563	<code>\tex_uccode:D</code> . 669 , 2632 , 2634 , 2636 , 3173	
<code>\tex_showboxbreadth:D</code>	436, 6573	<code>\tex_uchyph:D</code>	567
<code>\tex_showboxdepth:D</code>	437, 6574	<code>\tex_undefined:D</code>	336, 343
<code>\tex_showlists:D</code>	423	<code>\tex_underline:D</code>	503, 772
<code>\tex_showthe:D</code> . 421 , 1452 , 2554 , 2624 , 2630, 2636, 2642, 3179, 3182, 3185, 3188, 3191, 3920, 4172, 4255, 4304		<code>\tex_unhbox:D</code>	608, 6616
<code>\tex_skewchar:D</code>	634	<code>\tex_unhcopy:D</code>	609, 6615
<code>\tex_skip:D</code>	658	<code>\tex_unkern:D</code>	533
<code>\tex_skipdef:D</code>	357, 2812	<code>\tex_unpenalty:D</code>	644
<code>\tex_space:D</code>	346	<code>\tex_unskip:D</code>	531
<code>\tex_spacefactor:D</code>	576	<code>\tex_unvbox:D</code>	610, 6661
		<code>\tex_unvcopy:D</code>	611, 6660
		<code>\tex_uppercase:D</code>	641, 4475
		<code>\tex_vadjust:D</code>	544
		<code>\tex_valign:D</code>	379
		<code>\tex_vbadness:D</code>	619
		<code>\tex_vbox:D</code>	
			614, 6619, 6622, 6624, 6626, 6637, 6643

<code>\tex_vcenter:D</code>	465	<code>\tl_act_output:n</code>	
<code>\tex_vfil:D</code>	526	4916 , 4959 , 5042 , 5045 , 5053
<code>\tex_vfill:D</code>	528	<code>\tl_act_result:n</code> ..	4922 , 4944 , 4959 – 4962
<code>\tex_vfilneg:D</code>	527	<code>\tl_act_reverse_group:nn</code>	4968 , 4973 , 4981
<code>\tex_vfuzz:D</code>	621	<code>\tl_act_reverse_group_preserve:nn</code> ..	
<code>\tex_voffset:D</code>	596	4992 , 4996
<code>\tex_vrule:D</code>	535 , 7904 , 7959	<code>\tl_act_reverse_normal:nN</code>	
<code>\tex_vsize:D</code>	578	4968 , 4972 , 4979 , 4991
<code>\tex_vskip:D</code>	529 , 4247	<code>\tl_act_reverse_output:n</code>	
<code>\tex_vsplit:D</code>	607 , 6665	..	4916 , 4961 , 4978 , 4980 , 4984 , 4997
<code>\tex_vss:D</code>	530	<code>\tl_act_reverse_space:n</code>	
<code>\tex_vtop:D</code>	615 , 6620 , 6631	4968 , 4974 , 4977 , 4993
<code>\tex_wd:D</code>	662 , 6495	<code>\tl_act_space:wnnNNN</code> ..	4916 , 4931 , 4953
<code>\tex_widowpenalty:D</code>	549	<code>\tl_clear:c</code>	4344 , 5148 , 5680
<code>\tex_write:D</code>	412 , 1196 , 1198 , 8365	<code>\tl_clear:N</code>	
<code>\tex_xdef:D</code>	353 , 865	..	82 , 4344 , 4344 , 4348 , 4351 , 5147 , 5679 , 8432 , 8434 , 8435 , 8523 , 9322 , 9326 , 9995 , 10899 , 11177 , 11195 , 11430 , 11454 , 11474 , 11504 , 11518
<code>\tex_xleaders:D</code>	538	<code>\tl_clear_new:c</code>	4350 , 5152 , 5684 , 9594 , 9596
<code>\tex_xspaceskip:D</code>	572	<code>\tl_clear_new:N</code>	
<code>\tex_year:D</code>	653	83 , 4350 , 4350 , 4354 , 5151 , 5683
<code>\textasteriskcentered</code>	3977 , 3983	<code>\tl_const:cn</code>	
<code>\textbardbl</code>	3982	4331 , 12231 – 12270 , 12577 – 12588
<code>\textdagger</code>	3978 , 3984	<code>\tl_const:cx</code>	4331 , 8406 , 12667
<code>\textdaggerdbl</code>	3979 , 3985	<code>\tl_const:Nn</code>	82 , 2431 , 2661 , 4331 , 4331 , 4341 , 4343 , 4438 , 4891 , 5021 , 5026 , 6149 , 8148 , 8398 , 8604 , 8605 , 8635 , 8640 , 8642 , 8644 , 8646 , 8648 , 8653 , 8654 , 8661 , 8679 , 9290 , 9292 , 9413 – 9417 , 10089 – 10093
<code>\textfont</code>	629	<code>\tl_const:Nx</code>	
<code>\textparagraph</code>	3981	..	4331 , 4336 , 4342 , 4890 , 6109 , 8402
<code>\textsection</code>	3980	<code>\tl_elt_count:c</code>	5117 , 5122
<code>\textstyle</code>	474	<code>\tl_elt_count:N</code>	5117 , 5121
<code>\TeXETstate</code>	729	<code>\tl_elt_count:n</code>	5117 , 5118
<code>\the</code>	70 – 79 , 447	<code>\tl_elt_count:o</code>	5117 , 5120
<code>\thickmuskip</code>	515	<code>\tl_elt_count:V</code>	5117 , 5119
<code>\thinmuskip</code>	514	<code>\tl_expandable_lowercase:n</code>	94 , 5031 , 5033
<code>\time</code>	650	<code>\tl_expandable_uppercase:n</code>	94 , 5031 , 5031
<code>\tiny</code>	7896	<code>\tl_gclear:c</code>	4344 , 5150 , 5682
<code>\tl_act:NNNnn</code>	4916 , 4916	<code>\tl_gclear:N</code>	
<code>\tl_act_aux:NNNnn</code>	4916 , 4916 , 4917 , 4971 , 4990 , 5009 , 5037	..	82 , 4344 , 4346 , 4349 , 4353 , 5149 , 5681
<code>\tl_act_case_aux:nn</code>	5032 , 5034 , 5035 , 5054	<code>\tl_gclear_new:c</code>	4350 , 5154 , 5686
<code>\tl_act_case_group:nn</code> ..	5031 , 5039 , 5051	<code>\tl_gclear_new:N</code>	
<code>\tl_act_case_normal:nN</code> ..	5031 , 5038 , 5043	83 , 4350 , 4352 , 4355 , 5153 , 5685
<code>\tl_act_case_space:n</code> ..	5031 , 5040 , 5042	<code>\tl_gput_left:cn</code>	4382
<code>\tl_act_end:w</code>	4916	<code>\tl_gput_left:co</code>	4382
<code>\tl_act_end:wn</code>	4938 , 4944	<code>\tl_gput_left:cV</code>	4382
<code>\tl_act_group:wnnNNN</code> ..	4916 , 4930 , 4946	<code>\tl_gput_left:cx</code>	4382
<code>\tl_act_group_recurse:Nnn</code>	4916 , 4963 , 4983		
<code>\tl_act_length_group:nn</code>	5005 , 5011 , 5019		
<code>\tl_act_length_normal:nN</code>	5005 , 5010 , 5017		
<code>\tl_act_length_space:n</code> ..	5005 , 5012 , 5018		
<code>\tl_act_loop:w</code>			
..	4916 , 4920 , 4924 , 4941 , 4949 , 4956		
<code>\tl_act_normal:NnnNNN</code> ..	4916 , 4927 , 4935		

\tl_gput_left:Nn	83, 4382 , 4390 , 4402 , 5210	\tl_gset:Nx
\tl_gput_left:No 4382 , 4394 , 4404		4364 , 4374 , 4380 , 4480 , 4484 , 4750 ,
\tl_gput_left:NV 4382 , 4392 , 4403		5166 , 5198 , 5238 , 5341 , 5517 , 5607 ,
\tl_gput_left:Nx 4382 , 4396 , 4405		5612 , 5625 , 5643 , 5653 , 5698 , 5754 ,
\tl_gput_right:cn 4406		5844 , 6086 , 6230 , 9960 , 10410 , 12574
\tl_gput_right:co 4406	\tl_gset_eq:cc
\tl_gput_right:cV 4406		. 4356 , 4363 , 5162 , 5694 , 6169 , 10464
\tl_gput_right:cx 4406	\tl_gset_eq:cN
\tl_gput_right:Nn 4356 , 4361 , 5161 , 5693 , 6168 , 10462
 83, 2536 , 4406 , 4414 , 4426 , 5212	\tl_gset_eq:Nc
\tl_gput_right:No 4406 , 4418 , 4428		. 4356 , 4362 , 5160 , 5692 , 6167 , 10463
\tl_gput_right:NV 4406 , 4416 , 4427	\tl_gset_eq:NN
\tl_gput_right:Nx	4406 , 4420 , 4429 , 6254	 83, 4347 , 4356 , 4360 , 5159 ,
\tl_gremove_all:cn 4533 , 5115		5691 , 6166 , 9964 , 10358 , 10370 , 10461
\tl_gremove_all:Nn	\tl_gset_rescan:cnn 4440
 84, 4533 , 4535 , 4538 , 5114	\tl_gset_rescan:cno 4440
\tl_gremove_all_in:cn 5107 , 5115	\tl_gset_rescan:cnx 4440
\tl_gremove_all_in:Nn 5107 , 5114	\tl_gset_rescan:Nnn
\tl_gremove_in:cn 5107 , 5111	 84, 4440 , 4442 , 4472 , 4473
\tl_gremove_in:Nn 5107 , 5110	\tl_gset_rescan:Nno 4440
\tl_gremove_once:cn 4527 , 5111	\tl_gset_rescan:Nnx 4440
\tl_gremove_once:Nn	\tl_gtrim_spaces:c 4708
 84, 4527 , 4529 , 4532 , 5110	\tl_gtrim_spaces:N	.. 89, 4708 , 4749 , 4752
\tl_greplace_all:cnn 4477 , 5105	\tl_head:f 4753
\tl_greplace_all:Nnn	\tl_head:n 90, 4753 , 4755 , 4760 , 5125
 84, 4477 , 4483 , 4488 , 4536 , 5104	\tl_head:V 4753
\tl_greplace_all_in:cnn 5097 , 5105	\tl_head:v 4753
\tl_greplace_all_in:Nnn 5097 , 5104	\tl_head:w	90, 4753 , 4753 , 4756 , 4769 ,
\tl_greplace_in:cnn 5097 , 5101		4782 , 4798 , 4818 , 5126 , 10168 , 10179
\tl_greplace_in:Nnn 5097 , 5100	\tl_head_i:n 5124 , 5125
\tl_greplace_once:cnn 4477 , 5101	\tl_head_i:w 5124 , 5126
\tl_greplace_once:Nnn	\tl_head_iii:f 5124
 83, 4477 , 4479 , 4486 , 4530 , 5100	\tl_head_iii:n 5124 , 5127 , 5128
\tl_greverse:c 4999	\tl_head_iii:w 5124 , 5127 , 5129
\tl_greverse:N 89, 4999 , 5001 , 5004	\tl_if_blank:n 4539 , 4539
\tl_gset:cf 4364	\tl_if_blank:nF	.. 3873, 4543 , 4547 , 6019
\tl_gset:cn 4364	\tl_if_blank:nT 4542 , 4546
\tl_gset:co 4364	\tl_if_blank:nTF	... 85, 4544 , 4548 , 6065
\tl_gset:cx 4364 , 11848 , 11934 , 12215	\tl_if_blank:o 4539
\tl_gset:Nc 5091 , 5092	\tl_if_blank:oTF 9335
\tl_gset:Nf 4364 , 5002 , 5791	\tl_if_blank:V 4539
\tl_gset:Nn 83, 4364 ,	\tl_if_blank_p:n 4541 , 4545
	4370 , 4379 , 4381 , 4443 , 5086 , 5300 ,	\tl_if_blank_p_aux:NNw 4539
	5503 , 6191 , 6217 , 6373 , 6425 , 10039 ,	\tl_if_empty:c 4549 , 5263 , 5874
	10374 , 10828 , 10863 , 10944 , 10969 ,	\tl_if_empty:N	... 4549 , 4549 , 5261 , 5873
	10995 , 11098 , 11116 , 11229 , 11781 ,	\tl_if_empty:n 4561 , 4561
	11878 , 12079 , 12272 , 12590 , 12924	\tl_if_empty:NF 4559 , 10030
\tl_gset:No 4364 , 4372	\tl_if_empty:nF 3047 , 4572 , 5929
\tl_gset:Nv 4364	\tl_if_empty:NT 4558 , 9235
\tl_gset:Nv 4364	\tl_if_empty:nT 4571

\tl_if_empty:NTF .. 85, 4560, 9380, 10023	\tl_if_in:nn 4617, 4617
\tl_if_empty:nTF 85, 2791, 2798, 2809, 2820, 2831, 2842, 2851, 2858, 2867, 3915, 4491, 4570, 5169, 5740, 8686, 9488	\tl_if_in:NnF 4612, 4615
\tl_if_empty:o 4573, 4582	\tl_if_in:nnF 4612, 4624
\tl_if_empty:oTF 2888, 4620, 4871, 5891, 6095, 6120	\tl_if_in:NnT 4611, 4614
\tl_if_empty:V 4561	\tl_if_in:nnT 4611, 4623
\tl_if_empty:x 5080, 5080	\tl_if_in:NnTF 86, 2530, 4613, 4616
\tl_if_empty_p:N 4557	\tl_if_in:nnTF 86, 4613, 4625, 7556, 9458, 9465
\tl_if_empty_p:n 4569	\tl_if_in:no 4617
\tl_if_empty_return:o 4540, 4573, 4573, 4583	\tl_if_in:on 4617
\tl_if_eq:cc 4584, 5878, 6444	\tl_if_in:Vn 4617
\tl_if_eq:ccTF 9827	\tl_if_single:N 4904
\tl_if_eq:cN 4584, 5877, 6442	\tl_if_single:n 4908, 4908
\tl_if_eq:Nc 4584, 5876, 6443	\tl_if_single:NF 4906
\tl_if_eq:NN 4584, 4584, 5875, 6441	\tl_if_single:nF 4906
\tl_if_eq:nn 4596, 4596	\tl_if_single:NT 4905
\tl_if_eq:NNF 4595	\tl_if_single:nT 4905
\tl_if_eq:NNT 4594, 5250, 7970, 7973	\tl_if_single:NTF 86, 4907
\tl_if_eq:NNTF . 86, 2197, 4593, 8469, 8915	\tl_if_single:nTF 86, 4907
\tl_if_eq:nnTF 86	\tl_if_single_p:N 4904
\tl_if_eq_p:NN 4592	\tl_if_single_p:n 4904
\tl_if_head_eq_catcode:nN ... 4777, 4793	\tl_if_single_token:n 4910, 4910
\tl_if_head_eq_catcode:nNTF 91	\tl_if_single_token:NTF 86
\tl_if_head_eq_charcode:fN 4777	\tl_item:cn 5056
\tl_if_head_eq_charcode:nN .. 4777, 4777	\tl_item:Nn 5056, 5078, 5079
\tl_if_head_eq_charcode:nNF 4792	\tl_item:nn 94, 5056, 5056, 5078
\tl_if_head_eq_charcode:nNT 4791	\tl_item_aux:nn .. 5056, 5058, 5071, 5076
\tl_if_head_eq_charcode:nNTF 91, 3778, 3791, 4790	\tl_length:c 4680, 5122
\tl_if_head_eq_charcode_p:nN 4789	\tl_length:N ... 89, 4680, 4685, 4692, 5121
\tl_if_head_eq_meaning:nN ... 4777, 4809	\tl_length:n 88, 4680, 4680, 4691, 5063, 5118
\tl_if_head_eq_meaning:nNTF ... 91, 9364	\tl_length:o 4680, 5120
\tl_if_head_eq_meaning_aux_normal:nN 4812, 4816	\tl_length:V 4680, 5119
\tl_if_head_eq_meaning_aux_special:nN 4813, 4824	\tl_length_aux:n . 4680, 4683, 4688, 4690
\tl_if_head_group:n 4849, 4849	\tl_length_tokens:n . 93, 5005, 5005, 5020
\tl_if_head_group:nTF 91, 4800, 4834, 4929	\tl_map_break 88
\tl_if_head_N_type:n 4843, 4843	\tl_map_break: ... 4669, 4669, 8229, 8237
\tl_if_head_N_type:nTF 92, 4781, 4797, 4811, 4912, 4926	\tl_map_break:n 4669, 4670, 5075
\tl_if_head_space:n 4864, 4864	\tl_map_function:cn 4626
\tl_if_head_space:nTF 92	\tl_map_function:NN 87, 4626, 4632, 4639, 4688, 8200, 8213
\tl_if_head_space_aux:w 4864, 4867, 4869	\tl_map_function:nN 87, 4626, 4626, 4633, 4683, 5170
\tl_if_in:cn 4611	\tl_map_function_aux:Nn 4626, 4628, 4634, 4637, 4645
\tl_if_in:Nn 4611	\tl_map_inline:cn 4640
	\tl_map_inline:Nn .. 87, 4640, 4650, 4652
	\tl_map_inline:nn 87, 2780, 4640, 4640, 4651
	\tl_map_variable:cNn 4653
	\tl_map_variable:NNn 87, 4653, 4659, 4668
	\tl_map_variable:nNn 87, 4653, 4653, 4660

- \tl_map_variable_aux:Nnn 4653, 4655, 4661, 4666
- \tl_new:c 4325, 5146, 5678, 9576, 11847, 11933, 12214
- \tl_new:cn 5082
- \tl_new:N 82, 2527, 2931, 4325, 4325, 4330, 4351, 4353, 4476, 4609, 4610, 4892–4895, 5085, 5143–5145, 5675, 5677, 7027, 7053, 7054, 7891, 8389–8393, 8603, 8677, 8682, 8864–8866, 9209, 9307–9310, 9419–9421, 9423–9426, 9955, 9975, 10094, 10124, 10131, 10134, 10137, 10357, 12573, 13538
- \tl_new:Nn 5082, 5083, 5088, 5089
- \tl_new:Nx 5082
- \tl_put_left:cn 4382
- \tl_put_left:co 4382
- \tl_put_left:cV 4382
- \tl_put_left:cx 4382
- \tl_put_left:Nn 83, 4382, 4382, 4398, 5202
- \tl_put_left:No 4382, 4386, 4400
- \tl_put_left:NV 4382, 4384, 4399
- \tl_put_left:Nx 4382, 4388, 4401
- \tl_put_right:cn 4406
- \tl_put_right:co 4406
- \tl_put_right:cV 4406
- \tl_put_right:cx 4406
- \tl_put_right:Nn 83, 4406, 4406, 4422, 5204, 9381
- \tl_put_right:No . 4406, 4410, 4424, 8742
- \tl_put_right:NV 4406, 4408, 4423
- \tl_put_right:Nx 4406, 4412, 4425, 6252, 8488, 8494, 8501, 8520, 8529, 8540, 9350, 9372, 9385, 9394
- \tl_remove_all:cn 4533, 5113
- \tl_remove_all:Nn 84, 4533, 4533, 4537, 5112
- \tl_remove_all_in:cn 5107, 5113
- \tl_remove_all_in:Nn 5107, 5112
- \tl_remove_in:cn 5107, 5109
- \tl_remove_in:Nn 5107, 5108
- \tl_remove_once:cn 4527, 5109
- \tl_remove_once:Nn 84, 4527, 4527, 4531, 5108
- \tl_replace_all:cn 4477, 5103
- \tl_replace_all:Nnn .. 84, 4477, 4481, 4487, 4534, 5102, 5178, 9324, 9325
- \tl_replace_all_aux: 4477, 4482, 4484, 4515, 4518
- \tl_replace_all_in:cn 5097, 5103
- \tl_replace_all_in:Nnn 5097, 5102
- \tl_replace_aux:NNNnn 4477, 4478, 4480, 4482, 4484, 4489
- \tl_replace_aux_ii:w 4477, 4514, 4517, 4522
- \tl_replace_in:cn 5097, 5099
- \tl_replace_in:Nnn 5097, 5098
- \tl_replace_once:cn 4477, 5099
- \tl_replace_once:Nnn 83, 4477, 4477, 4485, 4528, 5098
- \tl_replace_once_aux: 4477, 4478, 4480, 4520
- \tl_replace_once_aux_end:w 4477, 4523, 4525
- \tl_rescan:nn 84, 4440, 4444
- \tl_rescan_aux:w 4440, 4458, 4466
- \tl_reverse:c 4999
- \tl_reverse:N 89, 4999, 4999, 5003
- \tl_reverse:n 89, 4987, 4987, 4998
- \tl_reverse:o 4987, 5000, 5002
- \tl_reverse:V 4987
- \tl_reverse_group_preserve:nn ... 4987
- \tl_reverse_items:n 89, 4693, 4693
- \tl_reverse_items_aux:nwNwn 4693, 4695, 4696, 4700, 4703
- \tl_reverse_items_aux:wn 4693, 4697, 4704, 4707
- \tl_reverse_tokens:n 93, 4968, 4968, 4985
- \tl_set:cf 4364
- \tl_set:cn 4364, 9595, 9599
- \tl_set:co 4364
- \tl_set:cx 4364, 9579
- \tl_set:Nc 5091, 5093, 5094
- \tl_set:Nf 4364, 5000, 5789
- \tl_set:Nn 83, 2284, 2949, 2970, 4364, 4364, 4376, 4378, 4441, 4599, 4600, 4663, 5172, 5246, 5255, 5269, 5272, 5295, 5298, 5310, 5325, 5356, 5423, 5496, 5786, 5798, 5970, 6189, 6202, 6205, 6211, 6212, 6218, 6222, 6317, 6367, 6378, 7034, 7038, 7226, 7557, 7558, 7893, 7896, 8468, 8709, 8896, 8897, 9323, 9433, 9470, 9537, 9563, 9631, 9755, 9768, 9812, 10373, 10825, 10860, 10943, 10968, 10994, 11097, 11115, 11228, 11780, 11877, 12078, 12271, 12589, 12923, 12981, 12993, 13514
- \tl_set:No 4364, 4366, 5095
- \tl_set:NV 4364
- \tl_set:Nv 4364

- \tl_set:Nx [4364](#), [4368](#), [4377](#),
[4478](#), [4482](#), [4748](#), [5164](#), [5183](#), [5196](#),
[5236](#), [5339](#), [5510](#), [5597](#), [5602](#), [5623](#),
[5641](#), [5651](#), [5696](#), [5752](#), [5842](#), [6084](#),
[6229](#), [8078](#), [8449](#), [8508](#), [8535](#), [8713](#),
[9234](#), [9236](#), [9367](#), [9378](#), [9393](#), [9431](#),
[9457](#), [9464](#), [9467](#), [9753](#), [9763](#), [9785](#),
[9786](#), [9990](#), [10010](#), [10157](#), [10170](#),
[10181](#), [10273](#), [10320](#), [10345](#), [10408](#),
[10928](#), [10931](#), [10977](#), [11225](#), [11589](#),
[11598](#), [11621](#), [11640](#), [11793](#), [11890](#),
[12091](#), [12285](#), [12368](#), [12401](#), [12628](#),
[12744](#), [12753](#), [12881](#), [13325](#), [13350](#)
- \tl_set_eq:cc [4356](#),
[4359](#), [5158](#), [5690](#), [6165](#), [9641](#), [10460](#)
- \tl_set_eq:cN
. [4356](#), [4357](#), [5157](#), [5689](#), [6164](#), [10458](#)
- \tl_set_eq:Nc [4356](#),
[4358](#), [5156](#), [5688](#), [6163](#), [9819](#), [10459](#)
- \tl_set_eq:NN [83](#), [4345](#), [4356](#),
[4356](#), [5155](#), [5687](#), [6162](#), [10368](#), [10457](#)
- \tl_set_rescan:cnm [4440](#)
- \tl_set_rescan:cno [4440](#)
- \tl_set_rescan:cnx [4440](#)
- \tl_set_rescan:Nnn
. [84](#), [4440](#), [4440](#), [4470](#), [4471](#)
- \tl_set_rescan:Nno [4440](#), [10158](#)
- \tl_set_rescan:Nnx [4440](#)
- \tl_set_rescan_aux:NNnn
. [4440](#), [4441](#), [4443](#), [4445](#), [4446](#)
- \tl_show:c [4876](#), [10466](#)
- \tl_show:N ... [92](#), [4876](#), [4876](#), [4877](#), [10465](#)
- \tl_show:n [92](#), [4878](#), [4878](#), [8089](#)
- \tl_tail:f [4753](#)
- \tl_tail:n [90](#), [4753](#), [4757](#), [4761](#)
- \tl_tail:V [4753](#)
- \tl_tail:v [4753](#)
- \tl_tail:w .. [90](#), [4753](#), [4754](#), [10173](#), [10184](#)
- \tl_tail_aux:w [4758](#), [4759](#)
- \tl_tmp:w [4498](#),
[4518](#), [4523](#), [4619](#), [4620](#), [4708](#), [4746](#)
- \tl_to_lowercase:n [85](#),
[2311](#), [2326](#), [2742](#), [2782](#), [2875](#), [3142](#),
[4474](#), [4474](#), [8398](#), [8730](#), [9163](#), [9316](#)
- \tl_to_str:c [4672](#)
- \tl_to_str:N [88](#), [4672](#), [4672](#), [4673](#), [8458](#), [8459](#)
- \tl_to_str:n [88](#),
[829](#), [3104](#), [4074](#), [4160](#), [4494](#), [4563](#),
[4576](#), [4671](#), [4671](#), [4765](#), [4774](#), [6171](#),
[6240](#), [6261](#), [6281](#), [6282](#), [6406](#), [6407](#),
[7922](#), [8006](#), [8403](#), [8549](#), [9431](#), [9464](#),
[9753](#), [9763](#), [9785](#), [9863](#), [9879](#), [10450](#)
- \tl_to_uppercase:n [85](#), [4474](#), [4475](#)
- \tl_trim_spaces:c [4708](#)
- \tl_trim_spaces:N .. [89](#), [4708](#), [4747](#), [4751](#)
- \tl_trim_spaces:n ... [89](#), [4708](#), [4710](#),
[4748](#), [4750](#), [5190](#), [6080](#), [9368](#), [9393](#)
- \tl_trim_spaces_aux_i:w
. [4708](#), [4713](#), [4724](#), [4727](#), [5714](#)
- \tl_trim_spaces_aux_ii:w [4718](#), [4732](#), [5718](#)
- \tl_trim_spaces_aux_ii:w\tl_trim_spaces_aux_iii:w
. [4708](#)
- \tl_trim_spaces_aux_iii:w
. [4719](#), [4734](#), [4737](#), [4741](#), [5719](#)
- \tl_trim_spaces_aux_iv:w [4708](#), [4721](#), [4743](#)
- \tl_use:c [4674](#), [5823](#)
- \tl_use:N [88](#), [4674](#), [4674](#), [4679](#), [5822](#)
- \token_get_arg_spec:N ... [58](#), [3102](#), [3115](#)
- \token_get_prefix_arg_replacement_aux:wN
. [3102](#), [3103](#), [3110](#), [3119](#), [3128](#)
- \token_get_prefix_spec:N . [58](#), [3102](#), [3106](#)
- \token_get_replacement_spec:N [3102](#), [3124](#)
- \token_get_replacement_text:N [58](#)
- \token_if_active:N [2716](#), [2716](#)
- \token_if_active:Nf [3259](#)
- \token_if_active:NT [3258](#)
- \token_if_active:NTF [53](#), [3260](#)
- \token_if_active_char:N [3244](#)
- \token_if_active_char:Nf [3259](#)
- \token_if_active_char:NT [3258](#)
- \token_if_active_char:NTF [3260](#)
- \token_if_active_char_p:N [3257](#)
- \token_if_active_p:N [3257](#)
- \token_if_alignment:N [2678](#), [2678](#)
- \token_if_alignment:Nf [3247](#)
- \token_if_alignment:NT [3246](#)
- \token_if_alignment:NTF [52](#), [3248](#)
- \token_if_alignment_p:N [3245](#)
- \token_if_alignment_tab:N [3244](#)
- \token_if_alignment_tab:Nf [3247](#)
- \token_if_alignment_tab:NT [3246](#)
- \token_if_alignment_tab:NTF [3248](#)
- \token_if_alignment_tab_p:N [3245](#)
- \token_if_chardef:N [2773](#), [2785](#)
- \token_if_chardef:NTF [54](#)
- \token_if_chardef_aux:w ... [2787](#), [2790](#)
- \token_if_chardef_p_aux:w [2773](#)
- \token_if_cs:N [2759](#), [2759](#)
- \token_if_cs:NTF [53](#)
- \token_if_dim_register:N [2773](#), [2821](#)

<code>\token_if_dim_register:N</code>	54	<code>\token_if_mathchardef_p_aux:w</code>	2773
<code>\token_if_dim_register_aux:w</code>	2826, 2830	<code>\token_if_other:N</code>	2711, 2711
<code>\token_if_dim_register_p_aux:w</code>	2773	<code>\token_if_other:N</code>	3255
<code>\token_if_eq_catcode:NN</code>	2726, 2726	<code>\token_if_other:NT</code>	3254
<code>\token_if_eq_catcode:NNTF</code>	53	<code>\token_if_other:N</code>	3256
<code>\token_if_eq_catcode_p:NN</code>	3006, 3007, 3156, 3157	<code>\token_if_other_char:N</code>	3244
<code>\token_if_eq_charcode:NN</code>	2731, 2731	<code>\token_if_other_char:N</code>	3255
<code>\token_if_eq_charcode:NNTF</code>	53	<code>\token_if_other_char:NT</code>	3254
<code>\token_if_eq_meaning:NN</code>	2721, 2721	<code>\token_if_other_char:N</code>	3256
<code>\token_if_eq_meaning:NNT</code>	2321	<code>\token_if_other_char_p:N</code>	3253
<code>\token_if_eq_meaning:NNTF</code>	53, 2336, 3027	<code>\token_if_other_p:N</code>	3253
<code>\token_if_eq_meaning_p:NN</code>	3008, 3158	<code>\token_if_parameter:N</code>	2683, 2685
<code>\token_if_expandable:N</code>	2764, 2764	<code>\token_if_parameter:N</code>	52
<code>\token_if_expandable:N</code>	54	<code>\token_if_primitive:N</code>	2869, 2877
<code>\token_if_group_begin:N</code>	2663, 2663	<code>\token_if_primitive:N</code>	55
<code>\token_if_group_begin:N</code>	52	<code>\token_if_primitive_aux:NNw</code>	2869, 2882, 2886
<code>\token_if_group_end:N</code>	2668, 2668	<code>\token_if_primitive_aux_loop:N</code>	2869, 2889, 2902, 2908
<code>\token_if_group_end:N</code>	52	<code>\token_if_primitive_aux_nullfont:N</code>	2869, 2890, 2894
<code>\token_if_int_register:N</code>	2773, 2799	<code>\token_if_primitive_aux_space:w</code>	2869, 2888, 2893
<code>\token_if_int_register:N</code>	54	<code>\token_if_primitive_aux_undefined:N</code>	2869, 2914, 2920
<code>\token_if_int_register_aux:w</code>	2804, 2808	<code>\token_if_primitive_auxii:Nw</code>	2869, 2905, 2911
<code>\token_if_int_register_p_aux:w</code>	2773	<code>\token_if_protected_long_macro:N</code>	2773, 2859
<code>\token_if_letter:N</code>	2706, 2706	<code>\token_if_protected_long_macro:N</code>	54
<code>\token_if_letter:N</code>	53	<code>\token_if_protected_long_macro_aux:w</code>	2862, 2865
<code>\token_if_long_macro:N</code>	2773, 2852	<code>\token_if_protected_long_macro_p_aux:w</code>	2773
<code>\token_if_long_macro:N</code>	54	<code>\token_if_protected_macro:N</code>	2773, 2843
<code>\token_if_long_macro_aux:w</code>	2854, 2857	<code>\token_if_protected_macro:N</code>	54
<code>\token_if_long_macro_p_aux:w</code>	2773	<code>\token_if_protected_macro_aux:w</code>	2846, 2849
<code>\token_if_macro:N</code>	2736, 2745	<code>\token_if_protected_macro_p_aux:w</code>	2773
<code>\token_if_macro:N</code>	53, 2879, 3108, 3117, 3126	<code>\token_if_skip_register:N</code>	2773, 2810
<code>\token_if_macro_p_aux:w</code>	2736, 2747, 2750	<code>\token_if_skip_register:N</code>	54
<code>\token_if_math_shift:N</code>	3244	<code>\token_if_skip_register_aux:w</code>	2815, 2819
<code>\token_if_math_shift:N</code>	3251	<code>\token_if_skip_register_p_aux:w</code>	2773
<code>\token_if_math_shift:NT</code>	3250	<code>\token_if_space:N</code>	2701, 2701
<code>\token_if_math_shift:N</code>	3252	<code>\token_if_space:N</code>	53
<code>\token_if_math_shift_p:N</code>	3249	<code>\token_if_toks_register:N</code>	2773, 2832
<code>\token_if_math_subscript:N</code>	2696, 2696	<code>\token_if_toks_register:N</code>	54
<code>\token_if_math_subscript:N</code>	53	<code>\token_if_toks_register_aux:w</code>	2837, 2841
<code>\token_if_math_superscript:N</code>	2691, 2691	<code>\token_if_toks_register_p_aux:w</code>	2773
<code>\token_if_math_superscript:N</code>	52		
<code>\token_if_math_toggle:N</code>	2673, 2673		
<code>\token_if_math_toggle:N</code>	3251		
<code>\token_if_math_toggle:NT</code>	3250		
<code>\token_if_math_toggle:N</code>	52, 3252		
<code>\token_if_math_toggle_p:N</code>	3249		
<code>\token_if_mathchardef:N</code>	2773, 2792		
<code>\token_if_mathchardef:N</code>	54		
<code>\token_if_mathchardef_aux:w</code>	2794, 2797		

<code>\token_new:Nn</code>	51, 2643, 2643, 2648, 2650–2652, 2654–2657
<code>\token_to_meaning:N</code>	51, 804, 804, 1222, 1232, 1245, 1876, 2748, 2788, 2795, 2805, 2816, 2827, 2838, 2847, 2855, 2863, 2883, 3111, 3120, 3129
<code>\token_to_str:c</code>	820, 820
<code>\token_to_str:N</code>	5, 52, 804, 805, 820, 1088, 1089, 1222, 1232, 1234, 1245, 1365, 1455, 1854, 2332, 2533, 4855, 5373, 6566, 7081, 7086, 7225, 8072, 8077, 8438–8442, 9129, 9136, 9144, 9229
<code>\toks</code>	659
<code>\toksdef</code>	360
<code>\tolerance</code>	570
<code>\topmark</code>	451
<code>\topmarks</code>	677
<code>\topskip</code>	581
<code>\TotalHeight</code>	7250, 7254, 7258, 7262, 7269, 7771, 7798, 7799
<code>\tracingassigns</code>	687
<code>\tracingcommands</code>	426
<code>\tracinggroups</code>	694
<code>\tracingifs</code>	690
<code>\tracinglostchars</code>	427
<code>\tracingmacros</code>	428
<code>\tracingnesting</code>	689
<code>\tracingonline</code>	429
<code>\tracingoutput</code>	430
<code>\tracingpages</code>	431
<code>\tracingparagraphs</code>	432
<code>\tracingrestores</code>	433
<code>\tracingscantokens</code>	688
<code>\tracingstats</code>	434
U	
<code>\U</code>	2780
<code>\uccode</code>	669
<code>\uchyph</code>	567
<code>\underline</code>	503
<code>\unexpanded</code>	179, 183, 682
<code>\unhbox</code>	608
<code>\unhcopy</code>	609
<code>\unkern</code>	533
<code>\unless</code>	673
<code>\unpenalty</code>	644
<code>\unskip</code>	531
<code>\unvbox</code>	610
<code>\unvcopy</code>	611
<code>\uppercase</code>	641
<code>\use:c</code>	16, 878, 878, 1006, 1188, 1190, 1192, 1194, 1978, 1988, 2060, 2061, 3441, 3737, 3747, 3890, 3899, 3901, 3903, 3904, 3908, 4077, 8513, 8802, 8813, 8826, 8829, 8835, 8846, 8854, 8860, 8882, 8943, 8965, 8970, 8978, 9001, 9023, 9216, 9478, 9485, 9647, 9856, 10162, 10192, 10195, 10212, 10215, 10216, 10219, 10222, 10506, 10568, 10624, 10674, 11826, 11913, 12124, 12311, 12647, 13174, 13237, 13252, 13254, 13345
<code>\use:n</code>	17, 888, 888, 1022, 1051, 1313, 1461, 1463, 1467, 1475, 1477, 1485, 1489, 1506, 4445, 4665, 4827, 4846, 5567, 5972, 6186, 6399, 6563, 9175
<code>\use:nn</code>	888, 889, 1571, 3102, 4072, 5956
<code>\use:nnn</code>	888, 890
<code>\use:nnnn</code>	888, 891
<code>\use:x</code>	18, 879, 879, 4157, 4453, 4464, 8455
<code>\use_i:nn</code>	17, 824, 892, 892, 918, 1109, 1138, 1166, 1324, 1465, 1479, 1487, 10262, 10308, 10333, 10901, 11220, 11577, 11610, 12403, 12731, 12870
<code>\use_i:nnn</code>	18, 894, 894, 1120, 1352, 3111, 12370
<code>\use_i:nnnn</code>	18, 894, 898
<code>\use_i_after_else:nw</code>	1543, 1544
<code>\use_i_after_fi:nw</code>	1543, 1543
<code>\use_i_after_or:nw</code>	1543, 1545
<code>\use_i_after_orelse:nw</code>	1543, 1546
<code>\use_i_delimit_by_q_nil:nw</code>	19, 905, 905
<code>\use_i_delimit_by_q_recursion_stop:nw</code>	19, 46, 905, 907, 2447, 2463, 6413
<code>\use_i_delimit_by_q_stop:nw</code>	19, 905, 906, 1826, 6049
<code>\use_i_ii:nnn</code>	18, 894, 897, 1596
<code>\use_ii:nn</code>	17, 826, 892, 893, 920, 1111, 1140, 1168, 1326, 1462, 1468, 1476, 1490, 6178, 9338
<code>\use_ii:nnn</code>	18, 894, 895, 1122, 3120, 9386
<code>\use_ii:nnnn</code>	18, 894, 899
<code>\use_iii:nnn</code>	18, 894, 896, 3129
<code>\use_iii:nnnn</code>	18, 894, 900
<code>\use_iv:nnnn</code>	18, 894, 901
<code>\use_none:n</code>	18, 908, 908, 1022, 1051, 1315, 1460, 1464, 1466, 1474, 1478, 1486, 1488, 1836, 1900, 2449, 2465, 2917, 3668,

- 3783, 3787, 3792, 4540, 4744, 4830,
 4852, 4871, 4874, 4913, 5141, 5324,
 5353, 5385, 5561, 5588, 5589, 5728,
 5863, 6099, 8378, 8380, 9335, 9368,
 10276, 10323, 10348, 10397, 10439,
 10851, 10887, 10960, 10986, 11040,
 11161, 11304, 11624, 11643, 11802,
 11857, 11899, 11943, 12100, 12224,
 12294, 12516, 12633, 12676, 13060
 \use_none:nn 908, 909,
 1835, 4707, 4909, 5251, 6120, 13124
 \use_none:nnn .. 908, 910, 1834, 6258, 9379
 \use_none:nnnn 908, 911, 1833, 10233
 \use_none:nnnnn 908, 912, 1832
 \use_none:nnnnnn 908, 913, 1831
 \use_none:nnnnnnn 908, 914, 1830
 \use_none:nnnnnnnn 908, 915, 1829
 \use_none:nnnnnnnnn
 908, 916, 1330, 1827, 1828
 \use_none_delimit_by_q_nil:w 18, 902, 902
 \use_none_delimit_by_q_recursion_stop:w
 18, 46,
 902, 904, 1004, 1072, 1820, 2441, 2456
 \use_none_delimit_by_q_stop:w
 18, 902, 903, 2352, 2356,
 4502, 5850, 6036, 6042, 8542, 8556
 \use_none_delimit_by_s_stop:w
 46, 2541, 2541
 \usepackage 223
- V**
- \vadjust 544
 \valign 379
 \vbadness 619
 \vbox 614
 \vbox:n 125, 6619, 6619
 \vbox_gset:cn 6625
 \vbox_gset:cw 6642, 6658
 \vbox_gset:Nn 125, 6625, 6627, 6629
 \vbox_gset:Nw . 126, 6642, 6644, 6647, 6657
 \vbox_gset_end 126
 \vbox_gset_end: 6642, 6653, 6659
 \vbox_gset_inline_begin:c ... 6654, 6658
 \vbox_gset_inline_begin:N ... 6654, 6657
 \vbox_gset_inline_end: 6654, 6659
 \vbox_gset_to_ht:cnn 6636
 \vbox_gset_to_ht:Nnn 126, 6636, 6638, 6641
 \vbox_gset_top:cn 6630
 \vbox_gset_top:Nn . 126, 6630, 6632, 6635
 \vbox_set:cn 6625
 \vbox_set:cw 6642, 6655
 \vbox_set:Nn
 ... 125, 6625, 6625, 6627, 6628, 7130
 \vbox_set:Nw
 126, 6642, 6642, 6645, 6646, 6654, 7174
 \vbox_set_end 126
 \vbox_set_end: 6642, 6648, 6653, 6656, 7180
 \vbox_set_inline_begin:c 6654, 6655
 \vbox_set_inline_begin:N 6654, 6654
 \vbox_set_inline_end: 6654, 6656
 \vbox_set_split_to_ht:NNn 126, 6664, 6664
 \vbox_set_to_ht:cnn 6636
 \vbox_set_to_ht:Nnn
 126, 6636, 6636, 6639, 6640
 \vbox_set_top:cn 6630
 \vbox_set_top:Nn
 126, 6630, 6630, 6633, 6634, 7141, 7184
 \vbox_to_ht:nn 125, 6621, 6621
 \vbox_to_zero:n 125, 6621, 6623
 \vbox_top:n 125, 6619, 6620
 \vbox_unpack:c 6660
 \vbox_unpack:N
 ... 126, 6660, 6660, 6662, 7141, 7184
 \vbox_unpack_clear:c 6660
 \vbox_unpack_clear:N 126, 6660, 6661, 6663
 \vcenter 465
 \vcoffin_set:cnn 7126
 \vcoffin_set:cnw 7170
 \vcoffin_set:Nnn .. 129, 7126, 7126, 7152
 \vcoffin_set:Nnw .. 129, 7170, 7170, 7199
 \vcoffin_set_end 129
 \vcoffin_set_end: 7170, 7177, 7198
 \vfil 526
 \vfill 528
 \vfilneg 527
 \vfuzz 621
 \voffset 596
 \voidb@x 6548
 \vrule 535
 \vsize 578
 \vskip 529
 \vsplit 607
 \vss 530
 \vtop 615
- W**
- \wd 662
 \widowpenalties 722
 \widowpenalty 549

<code>\Width</code>	7250, 7255,	<code>\xetex_XeTeXversion:D</code>	757, 1472
	7259, 7263, 7270, 7768, 7801, 7802	<code>\XeTeXversion</code>	757
<code>\write</code>	412	<code>\xleaders</code>	538
X			
<code>\X</code>	2776, 2780	Y	
<code>\xdef</code>	353	<code>\Y</code>	2777, 2780
<code>\xetex_if_engine:</code>	1460	<code>\year</code>	653
<code>\xetex_if_engine:F</code>	1467, 1478	Z	
<code>\xetex_if_engine:T</code>	1466, 1477	<code>\Z</code>	1861, 1869, 2778, 2780
<code>\xetex_if_engine:TF</code>	1468, 1479	<code>\z@</code>	4179
<code>\xetex_if_engine_p:</code>	1471, 1481, 1542		
<code>\xetex_if_engineTF</code>	4, 22		