

The L^AT_EX3 Sources

The L^AT_EX3 Project*

February 8, 2012

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>LaTeX3</code> modules	6
III	The <code>l3names</code> package: Namespace for primitives	8
5	Setting up the <code>LaTeX3</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	15
8.5	Deleting control sequences	15
8.6	Showing control sequences	16
8.7	Converting to and from control sequences	16
9	Using or removing tokens and arguments	17
9.1	Selecting tokens from delimited arguments	19
9.2	Decomposing control sequences	19
10	Predicates and conditionals	20
10.1	Tests on control sequences	21
10.2	Testing string equality	22
10.3	Engine-specific conditionals	22
10.4	Primitive conditionals	22

11	Internal kernel functions	23
12	Experimental functions	24
V	The <code>l3expan</code> package: Argument expansion	25
13	Defining new variants	25
14	Methods for defining variants	26
15	Introducing the variants	26
16	Manipulating the first argument	27
17	Manipulating two arguments	28
18	Manipulating three arguments	29
19	Unbraced expansion	30
20	Preventing expansion	30
21	Internal functions and variables	32
VI	The <code>l3prg</code> package: Control structures	33
22	Defining a set of conditional functions	33
23	The boolean data type	35
24	Boolean expressions	36
25	Logical loops	37
26	Switching by case	38
27	Producing n copies	40
28	Detecting <code>T_EX</code> 's mode	40
29	Internal programming functions	41
VII	The <code>l3quark</code> package: Quarks	43
30	Defining quarks	43

31	Quark tests	44
32	Recursion	44
33	Scan marks	45
34	Internal quark functions	46
VIII The l3token package: Token manipulation		47
35	All possible tokens	47
36	Character tokens	48
37	Generic tokens	51
38	Converting tokens	52
39	Token conditionals	52
40	Peeking ahead at the next token	55
41	Decomposing a macro definition	58
42	Experimental token functions	59
IX The l3int package: Integers		61
43	Integer expressions	61
44	Creating and initialising integers	62
45	Setting and incrementing integers	63
46	Using integers	63
47	Integer expression conditionals	64
48	Integer expression loops	64
49	Formatting integers	66
50	Converting from other formats to integers	68
51	Viewing integers	69
52	Constant integers	69

53	Scratch integers	70
54	Internal functions	70
X	The l3skip package: Dimensions and skips	72
55	Creating and initialising dim variables	72
56	Setting dim variables	72
57	Utilities for dimension calculations	73
58	Dimension expression conditionals	74
59	Dimension expression loops	75
60	Using dim expressions and variables	76
61	Viewing dim variables	76
62	Constant dimensions	76
63	Scratch dimensions	77
64	Creating and initialising skip variables	77
65	Setting skip variables	77
66	Skip expression conditionals	78
67	Using skip expressions and variables	78
68	Viewing skip variables	79
69	Constant skips	79
70	Scratch skips	79
71	Creating and initialising muskip variables	80
72	Setting muskip variables	80
73	Using muskip expressions and variables	81
74	Inserting skips into the output	81
75	Viewing muskip variables	81

76	Internal functions	82
77	Experimental skip functions	82
78	Internal functions	83
XI	The l3tl package: Token lists	84
79	Creating and initialising token list variables	84
80	Adding data to token list variables	85
81	Modifying token list variables	85
82	Reassigning token list category codes	86
83	Reassigning token list character codes	87
84	Token list conditionals	87
85	Mapping to token lists	89
86	Using token lists	90
87	Working with the content of token lists	90
88	The first token from a token list	92
89	Viewing token lists	95
90	Constant token lists	95
91	Scratch token lists	95
92	Experimental token list functions	96
93	Internal functions	97
XII	The l3seq package: Sequences and stacks	98
94	Creating and initialising sequences	98
95	Appending data to sequences	99
96	Recovering items from sequences	99
97	Modifying sequences	100

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

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

142	Joining and using coffins	133
143	Measuring coffins	134
144	Coffin diagnostics	134
XVII	The <code>l3color</code> package: Colour support	135
145	Colour in boxes	135
XVIII	The <code>l3io</code> package: Input–output operations	136
146	Managing streams	136
147	Writing to files	137
148	Wrapping lines in output	139
149	Reading from files	140
150	Constants	140
151	Experimental functions	141
152	Internal input–output functions	141
XIX	The <code>l3msg</code> package: Messages	143
153	Creating new messages	143
154	Contextual information for messages	144
155	Issuing messages	145
156	Redirecting messages	146
157	Low-level message functions	147
158	Kernel-specific functions	148
159	Expandable errors	149
160	Internal <code>l3msg</code> functions	150
XX	The <code>l3keys</code> package: Key–value interfaces	151

161	Creating keys	152
162	Sub-dividing keys	156
163	Choice and multiple choice keys	157
164	Setting keys	159
165	Setting known keys only	159
166	Utility functions for keys	160
167	Low-level interface for parsing key–val lists	160
XXI	The <code>l3file</code> package: File operations	162
168	File operation functions	162
169	Internal file functions	163
XXII	The <code>l3fp</code> package: Floating-point operations	165
170	Floating-point variables	165
171	Conversion of floating point values to other formats	166
172	Rounding floating point values	167
173	Floating-point conditionals	168
174	Unary floating-point operations	168
175	Floating-point arithmetic	169
176	Floating-point power operations	169
177	Exponential and logarithm functions	170
178	Trigonometric functions	170
179	Constant floating point values	170
180	Notes on the floating point unit	171
XXIII	The <code>l3luatex</code> package: LuaTeX-specific functions	172

181	Breaking out to Lua	172
182	Category code tables	173
XXIV	Implementation	174
183	l3bootstrap implementation	174
183.1	Format-specific code	174
183.2	Package-specific code	175
183.3	Dealing with package-mode meta-data	177
183.4	The <code>\pdfstrcmp</code> primitive in XeTeX	180
183.5	Engine requirements	180
183.6	The $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}3$ code environment	181
184	l3names implementation	182
185	l3basics implementation	192
185.1	Renaming some $\text{T}_{\text{E}}\text{X}$ primitives (again)	193
185.2	Defining some constants	195
185.3	Defining functions	195
185.4	Selecting tokens	196
185.5	Gobbling tokens from input	197
185.6	Conditional processing and definitions	198
185.7	Dissecting a control sequence	202
185.8	Exist or free	204
185.9	Defining and checking (new) functions	206
185.10	More new definitions	208
185.11	Copying definitions	210
185.12	Undefining functions	210
185.13	Defining functions from a given number of arguments	211
185.14	Using the signature to define functions	212
185.15	Checking control sequence equality	214
185.16	Diagnostic wrapper functions	215
185.17	Engine specific definitions	215
185.18	Doing nothing functions	216
185.19	String comparisons	216
185.20	Breaking out of mapping functions	217
185.21	Deprecated functions	217

186	l3expan implementation	218
186.1	General expansion	219
186.2	Hand-tuned definitions	222
186.3	Definitions with the automated technique	224
186.4	Last-unbraced versions	225
186.5	Preventing expansion	227
186.6	Defining function variants	227
186.7	Variants which cannot be created earlier	230
187	l3prg implementation	231
187.1	Primitive conditionals	231
187.2	Defining a set of conditional functions	231
187.3	The boolean data type	231
187.4	Boolean expressions	233
187.5	Logical loops	238
187.6	Switching by case	239
187.7	Producing n copies	241
187.8	Detecting T _E X’s mode	244
187.9	Internal programming functions	244
187.10	Deprecated functions	246
188	l3quark implementation	249
188.1	Quarks	249
188.2	Scan marks	252
189	l3token implementation	253
189.1	Character tokens	253
189.2	Generic tokens	255
189.3	Token conditionals	256
189.4	Peeking ahead at the next token	266
189.5	Decomposing a macro definition	271
189.6	Experimental token functions	272
189.7	Deprecated functions	273
190	l3int implementation	275
190.1	Integer expressions	276
190.2	Creating and initialising integers	277
190.3	Setting and incrementing integers	279
190.4	Using integers	280
190.5	Integer expression conditionals	280
190.6	Integer expression loops	283
190.7	Formatting integers	284
190.8	Converting from other formats to integers	289
190.9	Viewing integer	292
190.10	Constant integers	293
190.11	Scratch integers	294

190.1	D eprecated functions	294
191	l3skip implementation	295
191.1	Length primitives renamed	295
191.2	Creating and initialising dim variables	295
191.3	Setting dim variables	296
191.4	Utilities for dimension calculations	297
191.5	Dimension expression conditionals	298
191.6	Dimension expression loops	299
191.7	Using dim expressions and variables	301
191.8	Viewing dim variables	301
191.9	Constant dimensions	302
191.1	S cratch dimensions	302
191.1	Creating and initialising skip variables	302
191.1	S etting skip variables	303
191.1	S kip expression conditionals	304
191.1	U sing skip expressions and variables	304
191.1	I nserting skips into the output	305
191.1	V iewing skip variables	305
191.1	C onstant skips	305
191.1	S cratch skips	305
191.1	C reating and initialising muskip variables	306
191.2	S etting muskip variables	306
191.2	Using muskip expressions and variables	307
191.2	V iewing muskip variables	307
191.2	E xperimental skip functions	308
192	l3tl implementation	308
192.1	Functions	308
192.2	Adding to token list variables	310
192.3	Reassigning token list category codes	311
192.4	Reassigning token list character codes	312
192.5	Modifying token list variables	313
192.6	Token list conditionals	315
192.7	Mapping to token lists	318
192.8	Using token lists	319
192.9	Working with the contents of token lists	320
192.1	T he first token from a token list	322
192.1	S pecialist functions for kernel use	326
192.1	V iewing token lists	326
192.1	C onstant token lists	326
192.1	S cratch token lists	327
192.1	E xperimental functions	327
192.1	D eprecated functions	333

193	l3seq implementation	335
193.1	Allocation and initialisation	336
193.2	Appending data to either end	338
193.3	Modifying sequences	338
193.4	Sequence conditionals	339
193.5	Recovering data from sequences	340
193.6	Mapping to sequences	343
193.7	Sequence stacks	345
193.8	Viewing sequences	346
193.9	Experimental functions	346
193.10	Deprecated interfaces	352
194	l3clist implementation	352
194.1	Allocation and initialisation	353
194.2	Removing spaces around items	354
194.3	Adding data to comma lists	355
194.4	Comma lists as stacks	356
194.5	Using comma lists	357
194.6	Modifying comma lists	357
194.7	Comma list conditionals	359
194.8	Mapping to comma lists	360
194.9	Viewing comma lists	362
194.10	Scratch comma lists	362
194.11	Experimental functions	363
194.12	Deprecated interfaces	366
195	l3prop implementation	367
195.1	Allocation and initialisation	368
195.2	Accessing data in property lists	368
195.3	Property list conditionals	371
195.4	Recovering values from property lists with branching	373
195.5	Mapping to property lists	373
195.6	Viewing property lists	374
195.7	Experimental functions	375
195.8	Deprecated interfaces	376

196	l3box implementation	377
196.1	Creating and initialising boxes	377
196.2	Measuring and setting box dimensions	379
196.3	Using boxes	379
196.4	Box conditionals	379
196.5	The last box inserted	380
196.6	Constant boxes	380
196.7	Scratch boxes	381
196.8	Viewing box contents	381
196.9	Horizontal mode boxes	382
196.10	Vertical mode boxes	383
196.11	Affine transformations	385
196.12	Viewing part of a box	393
196.13	Deprecated functions	395
197	l3coffins Implementation	395
197.1	Coffins: data structures and general variables	395
197.2	Basic coffin functions	397
197.3	Measuring coffins	401
197.4	Coffins: handle and pole management	401
197.5	Coffins: calculation of pole intersections	405
197.6	Aligning and typesetting of coffins	408
197.7	Rotating coffins	413
197.8	Resizing coffins	417
197.9	Coffin diagnostics	420
197.10	Messages	425
198	l3color Implementation	426
199	l3io implementation	427
199.1	Primitives	427
199.2	Variables and constants	427
199.3	Stream management	428
199.4	Deferred writing	434
199.5	Immediate writing	434
199.6	Special characters for writing	435
199.7	Hard-wrapping lines based on length	435
199.8	Reading input	440
199.9	Experimental functions	441
199.10	Messages	442
199.11	Deprecated functions	442

200	l3msg implementation	443
200.1	Creating messages	443
200.2	Messages: support functions and text	444
200.3	Showing messages: low level mechanism	445
200.4	Displaying messages	447
200.5	Kernel-specific functions	452
200.6	Expandable errors	457
200.7	Showing variables	459
200.8	Deprecated functions	460
201	l3keys Implementation	462
201.1	Low-level interface	462
201.2	Constants and variables	465
201.3	The key defining mechanism	466
201.4	Turning properties into actions	468
201.5	Creating key properties	472
201.6	Setting keys	476
201.7	Utilities	479
201.8	Messages	479
201.9	Deprecated functions	480
202	l3file implementation	481
203	l3fp Implementation	486
203.1	Constants	486
203.2	Variables	487
203.3	Parsing numbers	489
203.4	Internal utilities	493
203.5	Operations for fp variables	494
203.6	Transferring to other types	499
203.7	Rounding numbers	506
203.8	Unary functions	508
203.9	Basic arithmetic	510
203.10	Arithmetic for internal use	519
203.11	Trigonometric functions	525
203.12	Exponent and logarithm functions	538
203.13	Tests for special values	560
203.14	Floating-point conditionals	560
203.15	Messages	566
204	l3luatex implementation	567
204.1	Category code tables	568
204.2	Deprecated functions	571
	Index	572

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 through 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*: every token in the argument is fully expanded until only unexpandable ones remain. The `\edef` primitive carries out this type of expansion. Functions which feature an **x**-type argument are in general *not* expandable, unless specifically noted.
- f** The **f** specifier stands for *full expansion*, and in contrast to **x** stops at the first non-expandable item (reading the argument from left to right) without trying to expand it. For example, when setting a token list variable (a macro used for storage), the sequence

```
\tl_set:Nn \l_my_a_tl { A }
\tl_set:Nn \l_my_b_tl { B }
\tl_set:Nf \l_my_a_tl { \l_my_b_tl }
```

will leave `\l_my_a_tl` with the content `A\l_my_b_tl`, as `A` cannot be expanded and so terminates expansion before `\l_my_b_tl` is considered.

- 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.

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.

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.

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, **T_EX** 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 **T_EX**’s stomach” (if you are familiar with the **T_EX**book 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 T_EX 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_engine \textit{TF} *` `\xetex_if_engine:TF {\langle true code \rangle} {\langle false code \rangle}`

The underlining and italic of \textit{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 \textit{TF} variant, and so both $\langle true code \rangle$ and $\langle false code \rangle$ will be shown. The two variant forms \textit{T} and \textit{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:

`\token_to_str:N *` `\token_to_str:N \langle token \rangle`

The normal description text.

\TeX hackers note: Detail for the experienced \TeX or $\text{\LaTeX} 2_\epsilon$ programmer. In this case, it would point out that this function is the \TeX primitive `\string`.

3 Formal language conventions which apply generally

As this is a formal reference guide for $\text{\LaTeX} 3$ 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 \textit{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}{\ExplFileDescription}
```

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 T_EX 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, *⟨code⟩* is therefore used as a shorthand for “replacement text”.

Functions which are not “protected” will be fully expanded inside an **x** expansion. In contrast, “protected” functions are not expanded within **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.

There are three ways to define new functions. All classes define a function to expand to the substitution text. Within the substitution text the actual parameters are substituted for the formal parameters (**#1**, **#2**, ...).

new Create a new function with the **new** primitives, such as `\cs_new:Npn`. The definition is global and will result in an error if it is already defined.

set Create a new function with the **set** primitives, such as `\cs_set:Npn`. The definition is restricted to the current T_EX group and will not result in an error if the function is already defined.

gset Create a new function with the **gset** primitives, such as `\cs_gset:Npn`. The definition is global and will not result in an error if the function is already defined.

Within each set of primitives there are different ways to define a function. The differences depend on restrictions on the actual parameters and the expandability of the resulting function.

nopar Create a new function with the **nopar** primitives, such as `\cs_set_nopar:Npn`. The parameter may not contain `\par` tokens.

protected Create a new function with the **protected** primitives, such as `\cs_set_protected:Npn`. The parameter may contain `\par` tokens but the function will not expand within an **x**-type expansion.

8.2 Defining new functions using primitive parameter text

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

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

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

<hr/> <code>\cs_new_nopar:Npn</code> <code>\cs_new_nopar:(cpn Npx cpx)</code> <hr/>	<code>\cs_new_nopar:Npn <function> <parameters> {<code>}</code> <p>Creates $\langle function \rangle$ 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 by the function. When the $\langle function \rangle$ is used the $\langle parameters \rangle$ absorbed cannot contain <code>\par</code> tokens. The definition is global and an error will result if the $\langle function \rangle$ is already defined.</p>
<hr/> <code>\cs_new_protected:Npn</code> <code>\cs_new_protected:(cpn Npx cpx)</code> <hr/>	<code>\cs_new_protected:Npn <function> <parameters> {<code>}</code> <p>Creates $\langle function \rangle$ 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 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.</p>
<hr/> <code>\cs_new_protected_nopar:Npn</code> <code>\cs_new_protected_nopar:(cpn Npx cpx)</code> <hr/>	<code>\cs_new_protected_nopar:Npn <function> <parameters> {<code>}</code> <p>Creates $\langle function \rangle$ 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 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 x-type argument. The definition is global and an error will result if the $\langle function \rangle$ is already defined.</p>
<hr/> <code>\cs_set:Npn</code> <code>\cs_set:(cpn Npx cpx)</code> <hr/>	<code>\cs_set:Npn <function> <parameters> {<code>}</code> <p>Sets $\langle function \rangle$ 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 by the function. The assignment of a meaning to the $\langle function \rangle$ is restricted to the current TeX group level.</p>
<hr/> <code>\cs_set_nopar:Npn</code> <code>\cs_set_nopar:(cpn Npx cpx)</code> <hr/>	<code>\cs_set_nopar:Npn <function> <parameters> {<code>}</code> <p>Sets $\langle function \rangle$ 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 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 the $\langle function \rangle$ is restricted to the current TeX group level.</p>
<hr/> <code>\cs_set_protected:Npn</code> <code>\cs_set_protected:(cpn Npx cpx)</code> <hr/>	<code>\cs_set_protected:Npn <function> <parameters> {<code>}</code> <p>Sets $\langle function \rangle$ 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 by the function. The assignment of a meaning to the $\langle function \rangle$ is restricted to the current TeX group level. The $\langle function \rangle$ will not expand within an x-type argument.</p>

<code>\cs_set_protected_nopar:Npn</code>	<code>\cs_set_protected_nopar:Npn <function> <parameters> {<code>}</code>
<code>\cs_set_protected_nopar:(cpn Npx cpx)</code>	

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 `\par` tokens. The assignment of a meaning to the $\langle function \rangle$ is restricted to the current TeX group level. The $\langle function \rangle$ will not expand within an x-type argument.

<code>\cs_gset:Npn</code>	<code>\cs_gset:Npn <function> <parameters> {<code>}</code>
<code>\cs_gset:(cpn Npx cpx)</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 the $\langle function \rangle$ is *not* restricted to the current TeX group level: the assignment is global.

<code>\cs_gset_nopar:Npn</code>	<code>\cs_gset_nopar:Npn <function> <parameters> {<code>}</code>
<code>\cs_gset_nopar:(cpn Npx cpx)</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 `\par` tokens. The assignment of a meaning to the $\langle function \rangle$ is *not* restricted to the current TeX group level: the assignment is global.

<code>\cs_gset_protected:Npn</code>	<code>\cs_gset_protected:Npn <function> <parameters> {<code>}</code>
<code>\cs_gset_protected:(cpn Npx cpx)</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 the $\langle function \rangle$ 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.

<code>\cs_gset_protected_nopar:Npn</code>	<code>\cs_gset_protected_nopar:Npn <function> <parameters> {<code>}</code>
<code>\cs_gset_protected_nopar:(cpn Npx cpx)</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 `\par` tokens. The assignment of a meaning to the $\langle function \rangle$ 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

<code>\cs_new:Nn</code>	<code>\cs_new:Nn <function> {<code>}</code>
<code>\cs_new:(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. The definition is global and an error will result if the $\langle function \rangle$ is already defined.

<hr/> <code>\cs_new_nopar:Nn</code> <code>\cs_new_nopar:(cn Nx cx)</code> <hr/>	<code>\cs_new_nopar:Nn <function> {<code>}</code> <p>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 $\langle parameters \rangle$ ($\#1$, $\#2$, <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 definition is global and an error will result if the $\langle function \rangle$ is already defined.</p>
<hr/> <code>\cs_new_protected:Nn</code> <code>\cs_new_protected:(cn Nx cx)</code> <hr/>	<code>\cs_new_protected:Nn <function> {<code>}</code> <p>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 $\langle parameters \rangle$ ($\#1$, $\#2$, <i>etc.</i>) 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.</p>
<hr/> <code>\cs_new_protected_nopar:Nn</code> <code>\cs_new_protected_nopar:(cn Nx cx)</code> <hr/>	<code>\cs_new_protected_nopar:Nn <function> {<code>}</code> <p>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 $\langle parameters \rangle$ ($\#1$, $\#2$, <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 x-type argument. The definition is global and an error will result if the $\langle function \rangle$ is already defined.</p>
<hr/> <code>\cs_set:Nn</code> <code>\cs_set:(cn Nx cx)</code> <hr/>	<code>\cs_set:Nn <function> {<code>}</code> <p>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 assignment of a meaning to the $\langle function \rangle$ is restricted to the current \TeX group level.</p>
<hr/> <code>\cs_set_nopar:Nn</code> <code>\cs_set_nopar:(cn Nx cx)</code> <hr/>	<code>\cs_set_nopar:Nn <function> {<code>}</code> <p>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 the $\langle function \rangle$ is restricted to the current \TeX group level.</p>
<hr/> <code>\cs_set_protected:Nn</code> <code>\cs_set_protected:(cn Nx cx)</code> <hr/>	<code>\cs_set_protected:Nn <function> {<code>}</code> <p>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 x-type argument. The assignment of a meaning to the $\langle function \rangle$ is restricted to the current \TeX group level.</p>

<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 the $\langle function \rangle$ 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 the $\langle function \rangle$ 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, 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 the $\langle function \rangle$ 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, 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 the $\langle function \rangle$ 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, 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 the $\langle function \rangle$ is global.

<code>\cs_generate_from_arg_count:NNnn</code>	<code>\cs_generate_from_arg_count:NNnn</code> $\langle function \rangle$ $\langle creator \rangle$ $\langle number \rangle$
<code>\cs_generate_from_arg_count:(cNnn Ncnn)</code>	$\langle code \rangle$

Updated: 2012-01-14

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

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.

<code>\cs_set_eq:NN</code>	<code>\cs_set_eq:NN</code> $\langle cs\ 1 \rangle$ $\langle cs\ 2 \rangle$
<code>\cs_set_eq:(Nc cN cc)</code>	<code>\cs_set_eq:NN</code> $\langle cs\ 1 \rangle$ $\langle token \rangle$

Sets $\langle control\ sequence\ 1 \rangle$ 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. The assignment of a meaning to the $\langle control\ sequence\ 1 \rangle$ is restricted to the current \TeX group level.

<code>\cs_gset_eq:NN</code>	<code>\cs_gset_eq:NN</code> $\langle cs\ 1 \rangle$ $\langle cs\ 2 \rangle$
<code>\cs_gset_eq:(Nc cN cc)</code>	<code>\cs_gset_eq:NN</code> $\langle cs\ 1 \rangle$ $\langle token \rangle$

Globally sets $\langle control\ sequence\ 1 \rangle$ 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. The assignment of a meaning to the $\langle control\ sequence\ 1 \rangle$ 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.

<code>\cs_undefine:N</code>	<code>\cs_undefine:N</code> $\langle control\ sequence \rangle$
<code>\cs_undefine:c</code>	

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

Updated: 2011-09-15

8.6 Showing control sequences

<code>\cs_meaning:N</code> ★	<code>\cs_meaning:N</code> \langle <i>control sequence</i> \rangle
<code>\cs_meaning:c</code> ★	This function expands to the <i>meaning</i> of the \langle <i>control sequence</i> \rangle control sequence. This will show the \langle <i>replacement text</i> \rangle for a macro.
Updated: 2011-12-22	

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

<code>\cs_show:N</code>	<code>\cs_show:N</code> \langle <i>control sequence</i> \rangle
<code>\cs_show:c</code>	Displays the definition of the \langle <i>control sequence</i> \rangle on the terminal.
Updated: 2011-12-22	

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

8.7 Converting to and from control sequences

<code>\use:c</code> ★	<code>\use:c</code> $\{\langle$ <i>control sequence name</i> $\rangle\}$
Converts the given \langle <i>control sequence name</i> \rangle into a single control sequence token. This process requires two expansions. The content for \langle <i>control sequence name</i> \rangle may be literal material or from other expandable functions. The \langle <i>control sequence name</i> \rangle 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`.

<code>\cs:w</code> ★	<code>\cs:w</code> \langle <i>control sequence name</i> \rangle <code>\cs_end:</code>
<code>\cs_end</code> ★	Converts the given \langle <i>control sequence name</i> \rangle into a single control sequence token. This process requires one expansion. The content for \langle <i>control sequence name</i> \rangle may be literal material or from other expandable functions. The \langle <i>control sequence name</i> \rangle 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.

T_EXhackers note: These are the T_EX 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`.

```
\cs_to_str:N ★ \cs_to_str:N {(control sequence)}
```

Converts the given *<control sequence>* 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 *<control sequence>* 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).

```
\use:n ★ \use:n {<group1>}
\use:(nn|nnn|nnnn) ★ \use:nn {<group1>} {<group2>}
\use:nnn {<group1>} {<group2>} {<group3>}
\use:nnnn {<group1>} {<group2>} {<group3>} {<group4>}
```

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>\use_i:nn {⟨arg₁⟩} {⟨arg₂⟩}</code>
------------------------	---	--

<code>\use_ii:nn</code>	★	These functions absorb two arguments from the input stream. The function <code>\use_i:nn</code> discards the second argument, and leaves the content of the first argument in the input stream. <code>\use_ii:nn</code> discards the first argument and leaves the content of the second argument in the input stream. 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 {⟨arg₁⟩} {⟨arg₂⟩} {⟨arg₃⟩}</code>
-------------------------	---	---

<code>\use_ii:nnn</code>	★	These functions absorb three arguments from the input stream. The function <code>\use_i:nnn</code> discards the second and third arguments, and leaves the content of the first argument in the input stream. <code>\use_ii:nnn</code> and <code>\use_iii:nnn</code> work similarly, leaving the content of second or third arguments in the input stream, respectively. 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 {⟨arg₁⟩} {⟨arg₂⟩} {⟨arg₃⟩} {⟨arg₄⟩}</code>
--------------------------	---	--

<code>\use_ii:nnnn</code>	★	These functions absorb four arguments from the input stream. The function <code>\use_i:nnnn</code> discards the second, third and fourth arguments, and leaves the content of the first argument in the input stream. <code>\use_ii:nnnn</code> , <code>\use_iii:nnnn</code> and <code>\use_iv:nnnn</code> work similarly, leaving the content of second, third or fourth arguments in the input stream, respectively. 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 {⟨arg₁⟩} {⟨arg₂⟩} {⟨arg₃⟩}</code>
----------------------------	---	--

This functions will absorb three arguments and leave the content of the first and second in the input stream. 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>
---------------------	---

Updated: 2011-12-31	Fully expands the <i>⟨expandable tokens⟩</i> and inserts the result into the input stream at the current location. Any hash characters (#) 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_stop:w ⟨balanced text⟩ \q_stop</code>
<code>\use_none_delimit_by_q_recursion_stop:w</code>	★	<code>\use_none_delimit_by_q_recursion_stop:w ⟨balanced text⟩ \q_recursion_stop</code>

Absorb the *⟨balanced text⟩* from 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 {⟨inserted tokens⟩} ⟨balanced text⟩</code>
<code>\use_i_delimit_by_q_stop:nw</code>	★	<code>\q_nil</code>
<code>\use_i_delimit_by_q_recursion_stop:nw</code>	★	<code>\use_i_delimit_by_q_stop:nw {⟨inserted tokens⟩} ⟨balanced text⟩ \q_stop</code>
		<code>\use_i_delimit_by_q_recursion_stop:nw {⟨inserted tokens⟩} ⟨balanced text⟩ \q_recursion_stop</code>

Absorb the *⟨balanced text⟩* from the input stream delimited by the marker given in the function name, leaving *⟨inserted tokens⟩* 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 ⟨function⟩</code>
---	---	--

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

<code>\cs_get_function_name:N</code>	★	<code>\cs_get_function_name:N ⟨function⟩</code>
--------------------------------------	---	---

Splits the *⟨function⟩* into the *⟨name⟩* (i.e. the part before the colon) and the *⟨signature⟩* (i.e. after the colon). The *⟨name⟩* 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 ⟨function⟩</code>
---	---	--

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

`\cs_split_function:NN` ★

`\cs_split_function:NN` $\langle function \rangle$ $\langle processor \rangle$

Splits the $\langle function \rangle$ into the $\langle name \rangle$ (*i.e.* the part before the colon) and the $\langle signature \rangle$ (*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 as the $\langle true\ code \rangle$ or the $\langle false\ code \rangle$. 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
} {\true code} {\false code}

```

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 $\text{\LaTeX 2}_{\epsilon}$. 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

```

\cs_if_eq_p:NN * \cs_if_eq_p:NN {\cs_1} {\cs_2}
\cs_if_eq:NNTF * \cs_if_eq:NNTF {\cs_1} {\cs_2} {\true code} {\false code}

```

Compares the definition of two *control sequences* and is logically `true` the same, *i.e.* if they have exactly the same definition when examined with `\cs_show:N`.

```

\cs_if_exist_p:N * \cs_if_exist_p:N <control sequence>
\cs_if_exist_p:c * \cs_if_exist:NNTF <control sequence> {\true code} {\false code}
\cs_if_exist:NNTF *
\cs_if_exist:cTF *

```

Tests whether the *control sequence* is currently defined (whether as a function or another control sequence type). Any valid definition of *control sequence* will evaluate as `true`.

```

\cs_if_free_p:N * \cs_if_free_p:N <control sequence>
\cs_if_free_p:c * \cs_if_free:NNTF <control sequence> {\true code} {\false code}
\cs_if_free:NNTF *
\cs_if_free:cTF *

```

Tests whether the *control sequence* is currently free to be defined. This test will be `false` if the *control sequence* currently exists (as defined by `\cs_if_exist:N`).

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>	★	

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>	★	

Updated: 2011-09-06

<code>\xetex_if_engine_p:</code>	★	<code>\xetex_if_engine:TF {<true code>} {<false code>}</code>
<code>\xetex_if_engineTF</code>	★	

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
<code>\reverse_if:N</code>	★	the branches of the conditional. <code>\or:</code> is used in case switches, see <code>l3int</code> for more.

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

<code>\if_meaning:w</code>	★	<code>\if_meaning:w <arg₁₂</code>
----------------------------	---	---

`\if_meaning:w` executes *<true code>* when *<arg_{1 and *<arg_{2 are the same, otherwise it executes *<false code>*. *<arg_{1 and *<arg_{2 could be functions, variables, tokens; in all cases the *unexpanded* definitions are compared.}*}*}*}*

T_EXhackers note: This is T_EX's `\ifx`.

<code>\if:w</code>	★	<code>\if:w <token₁₂</code>
<code>\if_catcode:w</code>	★	<code>\if_catcode:w <token₁₂</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>	★	Execute <i><true code></i> if currently in horizontal mode, otherwise execute <i><false code></i> . Similar for the other functions.
<code>\if_mode_math</code>	★	
<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.

```
\chk_if_free_cs:N
\chk_if_free_cs:c
```

```
\chk_if_free_cs:N <cs>
```

This function checks that $\langle cs \rangle$ is free according to the criteria for $\backslash cs_if_free_p:N$, and if not raises a kernel-level error.

12 Experimental functions

```
\cs_if_exist_use:NTF ★
\cs_if_exist_use:cTF ★
```

New: 2011-10-10

```
\cs_if_exist_use:NTF <control sequence> {<true code>} {<false code>}
```

If the $\langle control\ sequence \rangle$ exists, leave it in the input stream, followed by the $\langle true\ code \rangle$ (unbraced). Otherwise, leave the $\langle false \rangle$ 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 $\backslash mypkg_#1:n$ if it exists, and falls back to a default action otherwise. This could also be done (more slowly) using $\backslash prg_case_str:xxn$.

T_EXhackers note: The c variants do not introduce the $\langle control\ sequence \rangle$ 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, while the third argument should be given in braces. Applying `\exp_args:NNo` will expand the content of third argument once before any expansion of the first and second arguments. If `\seq_gpush:No` was not define it 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 `lur`. 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} ...
```

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}
\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/>
<code>\exp_args:NV</code> ★	<code>\exp_args:NV</code> $\langle function \rangle$ $\langle variable \rangle$
	This function absorbs two arguments (the names of the $\langle function \rangle$ and the $\langle variable \rangle$). The content of the $\langle variable \rangle$ are recovered and placed inside braces into the input stream <i>after</i> reinsertion of the $\langle function \rangle$. Thus the $\langle function \rangle$ may take more than one argument: all others will be left unchanged.
<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:Nv</code> ★	<code>\exp_args:Nv</code> $\langle function \rangle$ $\{\langle tokens \rangle\}$
	This function absorbs two arguments (the $\langle function \rangle$ name and the $\langle tokens \rangle$). The $\langle tokens \rangle$ 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 $\langle variable \rangle$. The content of the $\langle variable \rangle$ are recovered and placed inside braces into the input stream <i>after</i> reinsertion of the $\langle function \rangle$. Thus the $\langle function \rangle$ may take more than one argument: all others will be left unchanged.
<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:Nf</code> ★	<code>\exp_args:Nf</code> $\langle function \rangle$ $\{\langle tokens \rangle\}$
	This function absorbs two arguments (the $\langle function \rangle$ name and the $\langle tokens \rangle$). The $\langle tokens \rangle$ 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 $\langle function \rangle$. Thus the $\langle function \rangle$ may take more than one argument: all others will be left unchanged.
<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:Nx</code>	<code>\exp_args:Nx</code> $\langle function \rangle$ $\{\langle tokens \rangle\}$
	This function absorbs two arguments (the $\langle function \rangle$ name and the $\langle tokens \rangle$) and exhaustively expands the $\langle tokens \rangle$ second. The result is inserted in braces into the input stream <i>after</i> reinsertion of the $\langle function \rangle$. Thus the $\langle function \rangle$ may take more than one argument: all others will be left unchanged.

17 Manipulating two arguments

<hr/> <hr/>	<hr/> <hr/>
<code>\exp_args:NNo</code> ★	<code>\exp_args:NNo</code> $\langle token1 \rangle$ $\langle token2 \rangle$ $\{\langle tokens \rangle\}$
<code>\exp_args:(NNo 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/>
<code>\exp_args:Nno</code> ★	<code>\exp_args:Nno</code> $\langle token \rangle$ $\{\langle tokens_1 \rangle\}$ $\{\langle tokens_2 \rangle\}$
<code>\exp_args:(NnV Nnf Noo Nof Noc Nff Nfo Nnc)</code> ★	

Updated: 2012-01-14

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 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

<hr/> <hr/>	
<code>\l_exp_internal_tl</code>	The <code>\exp_</code> 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.
<hr/>	
<code>\exp_eval_register:N</code> ★	<code>\exp_eval_register:N</code> $\langle variable \rangle$
<code>\exp_eval_register:c</code> ★	These functions evaluates a $\langle variable \rangle$ as part of a <code>V</code> or <code>v</code> expansion (respectively), preceded by <code>\c_zero</code> which stops the expansion of a previous <code>\romannumeral</code> . A $\langle variable \rangle$ might exist as one of two things: a parameter-less non-long, non-protected macro or a built-in <code>T_EX</code> register such as <code>\count</code> .
<hr/>	
<code>\::n</code>	<code>\cs_set_nopar:Npn \exp_args:Ncof { \::c \::o \::f \::: }</code>
<code>\::N</code>	Internal forms for the base expansion types. These names do <i>not</i> conform to the general L ^A T _E X3 approach as this makes them more readily visible in the log and so forth.
<code>\::c</code>	
<code>\::o</code>	
<code>\::f</code>	
<code>\::x</code>	
<code>\::v</code>	
<code>\::V</code>	
<code>\:::</code>	
<hr/>	
<code>\cs_generate_internal_variant:n</code>	<code>\cs_generate_internal_variant:n</code> $\langle arg spec \rangle$
	Tests if the function <code>\exp_args:N</code> $\langle arg spec \rangle$ exists, and defines it if it does not. The $\langle arg spec \rangle$ should be a series of one or more of the letters <code>N</code> , <code>c</code> , <code>n</code> , <code>o</code> , <code>V</code> , <code>v</code> , <code>f</code> and <code>x</code> .

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 forms of conditional flow processing based on these states. The first form 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 second form 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
```

Updated: 2012-02-06

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

These functions create a family of conditionals using the same *{⟨code⟩}* to perform the test created. Those conditionals are expandable if *⟨code⟩* is. The **new** versions will check for existing definitions and perform assignments globally (*cf.* `\cs_new:Npn`) whereas the **set** versions do no check and perform assignments locally (*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.

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

Updated: 2012-02-06

These functions create a family of protected conditionals using the same *{⟨code⟩}* to perform the test created. The *⟨code⟩* does not need to be expandable. The **new** version will check for existing definitions and perform assignments globally (*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 (not p).

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. This function will not work properly for `protected` conditionals.
- `\<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.

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` and `\foo_if_bar:NNT` but not `\foo_if_bar:NNF` (because `F` is missing from the `<conditions>` 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_eq_conditional:NNn</code>	<code>\prg_new_eq_conditional:NNn \langle name1 \rangle:\langle arg spec1 \rangle \langle name2 \rangle:\langle arg spec2 \rangle</code>
<code>\prg_set_eq_conditional:NNn</code>	<code>{\langle conditions \rangle}</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 `\langle conditions \rangle`, 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).

<code>\bool_new:N</code>	<code>\bool_new:N \langle boolean \rangle</code>
<code>\bool_new:c</code>	

Creates a new `\langle boolean \rangle` or raises an error if the name is already taken. The declaration is global. The `\langle boolean \rangle` will initially be `false`.

<code>\bool_set_false:N</code>	<code>\bool_set_false:N \langle boolean \rangle</code>
<code>\bool_set_false:c</code>	
<code>\bool_gset_false:N</code>	Sets <code>\langle boolean \rangle</code> logically <code>false</code> .
<code>\bool_gset_false:c</code>	

<code>\bool_set_true:N</code>	<code>\bool_set_true:N \langle boolean \rangle</code>
<code>\bool_set_true:c</code>	
<code>\bool_gset_true:N</code>	Sets <code>\langle boolean \rangle</code> logically <code>true</code> .
<code>\bool_gset_true:c</code>	

<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> .
<code>\bool_gset_eq:NN</code>	
<code>\bool_gset_eq:(cN Nc cc)</code>	
<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
<code>\bool_gset:Nn</code>	<i><boolean></i> variable to the logical truth of this evaluation.
<code>\bool_gset:cn</code>	
<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/>	
<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.
<hr/>	
<code>\g_tmpa_bool</code>	A scratch boolean for global 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.

24 Boolean expressions

As we have a boolean datatype and predicate functions returning boolean *<true>* or *<false>* values, it seems only fitting that we also provide a parser for *<boolean expressions>*.

A boolean expression is an expression which given input in the form of predicate functions and boolean variables, return boolean *<true>* or *<false>*. 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/>	<code>\prg_case_dim:nnn {<test dimension expression>}</code> <code>{</code> <code> {<dimexpr case₁>} {<code case₁>}</code> <code> {<dimexpr case₂>} {<code case₂>}</code> <code> ...</code> <code> {<dimexpr case_n>} {<code case_n>}</code> <code>}</code> <code>{<else case>}</code>
--	---

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/>	<code>\prg_case_str:nnn {<test string>}</code> <code>{</code> <code> {<string case₁>} {<code case₁>}</code> <code> {<string case₂>} {<code case₂>}</code> <code> ...</code> <code> {<string case_n>} {<code case_n>}</code> <code>}</code> <code>{<else case>}</code>
--	--

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 fully expands *<strings>* before comparing them, but does not expand the corresponding *<code>*. It 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/>	<code>\prg_case_tl:Nnn <test token list variable></code> <code>{</code> <code> <token list variable case₁> {<code case₁>}</code> <code> <token list variable case₂> {<code case₂>}</code> <code> ...</code> <code> <token list variable case_n> {<code case_n>}</code> <code>}</code> <code>{<else case>}</code>
--	---

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> ★	<code>\prg_replicate:nn {⟨integer expression⟩} {⟨tokens⟩}</code>
----------------------------------	--

Updated: 2011-07-04

Evaluates the $\langle integer\ expression \rangle$ (which should be zero or positive) and creates the resulting number of copies of the $\langle tokens \rangle$. 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 $\langle initial\ value \rangle$, $\langle step \rangle$ and $\langle final\ value \rangle$, all of which should be integer expressions. The $\langle function \rangle$ is then placed in front of each $\langle value \rangle$ from the $\langle initial\ value \rangle$ to the $\langle final\ value \rangle$ in turn (using $\langle step \rangle$ between each $\langle value \rangle$). Thus $\langle function \rangle$ should absorb one numerical argument. For example

```
\cs_set:Npn \my_func:n #1 { [I~saw~#1] \quad }
\prg_stepwise_function:nnnN { 1 } { 1 } { 5 } \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 $\langle initial\ value \rangle$, $\langle step \rangle$ and $\langle final\ value \rangle$, all of which should be integer expressions. The $\langle code \rangle$ is then placed in front of each $\langle value \rangle$ from the $\langle initial\ value \rangle$ to the $\langle final\ value \rangle$ in turn (using $\langle step \rangle$ between each $\langle value \rangle$). Thus the $\langle code \rangle$ 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 $\langle initial\ value \rangle$, $\langle step \rangle$ and $\langle final\ value \rangle$, all of which should be integer expressions. The $\langle code \rangle$ is inserted into the input stream, with the $\langle tl\ var \rangle$ defined as the current $\langle value \rangle$. Thus the $\langle code \rangle$ should make use of the $\langle tl\ var \rangle$.

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 {\langle true code \rangle} {\langle false code \rangle}</code>

Detects if \TeX is currently in inner mode.

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

Detects if \TeX is currently in maths mode.

Updated: 2011-09-05

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

Detects if \TeX 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>

Updated: 2011-08-11

`\group_align_safe_end:`

These functions are used to enclose material in a \TeX 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 \TeX 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 \TeX '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.

\TeX hackers note: This is a protected version of `\prg_do_nothing:`, which therefore stops \TeX '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 $\langle variable \rangle$.

<hr/> <hr/>	<code>\prg_variable_get_type:N</code> ★	<code>\prg_variable_get_type:N</code> $\langle variable \rangle$ Returns the type of $\langle variable \rangle$ (tl, int, etc.)
<hr/> <hr/>	<code>\if_predicate:w</code> ★	<code>\if_predicate:w</code> $\langle predicate \rangle$ $\langle true\ code \rangle$ <code>\else:</code> $\langle false\ code \rangle$ <code>\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 <code>\if_bool:N</code> .)
<hr/> <hr/>	<code>\if_bool:N</code> ★	<code>\if_bool:N</code> $\langle boolean \rangle$ $\langle true\ code \rangle$ <code>\else:</code> $\langle false\ code \rangle$ <code>\fi:</code> This function takes a boolean variable and branches according to the result.
<hr/> <hr/>	<code>\prg_break_point:n</code> ★	<code>\prg_break_point:n</code> $\langle tokens \rangle$ Used to mark the end of a recursion or mapping: the functions <code>\prg_map_break:</code> and <code>\prg_map_break:n</code> use this to break out of the loop. After the loop ends, the $\langle tokens \rangle$ are inserted into the input stream. This occurs even if the the break functions are <i>not</i> applied: <code>\prg_break_point:n</code> is functionally-equivalent in these cases to <code>\use:n</code> .
<hr/> <hr/>	<code>\prg_map_break:n</code> ★	<code>\prg_map_break:n</code> $\{ \langle user\ code \rangle \}$... <code>\prg_break_point:n</code> $\{ \langle ending\ code \rangle \}$ Breaks a recursion in mapping contexts, inserting in the input stream the $\langle user\ code \rangle$ after the $\langle ending\ code \rangle$ for the loop.

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.

Like quarks, they can be used as delimiters in weird functions. 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).

30 Defining quarks

<u><code>\quark_new:N</code></u>	<code>\quark_new:N <quark></code> Creates a new <code><quark></code> which expands only to <code><quark></code> . The <code><quark></code> will be defined globally, and an error message will be raised if the name was already taken.
<u><code>\q_stop</code></u>	Used as a marker for delimited arguments, such as $\cs_set:Npn \tmp:w \#1\#2 \q_stop \{ \#1 \}$
<u><code>\q_mark</code></u>	Used as a marker for delimited arguments when <code>\q_stop</code> 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 <code>\q_stop</code> , which is only ever used as a delimiter).
<u><code>\q_no_value</code></u>	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 <code>\prop_get:NnN</code> if there is no data to return.

31 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:NTF</code>	★	<code>\quark_if_nil:NTF <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:nTF {\token list} {\true code} {\false code}</code>
<code>\quark_if_nil:nTF</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)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:NTF <token> {\true code} {\false code}</code>
<code>\quark_if_no_value:NTF</code>	★	Tests if the $\langle token \rangle$ is equal to <code>\q_no_value</code> .
<code>\quark_if_no_value:cTF</code>	★	

<code>\quark_if_no_value_p:n</code>	★	<code>\quark_if_no_value_p:n {\token list}</code>
<code>\quark_if_no_value:nTF</code>	★	<code>\quark_if_no_value:nTF {\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).

32 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`.

33 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
```

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.

34 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.

35 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
```

36 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 {⟨<i>intexpr</i>₁⟩} {⟨<i>intexpr</i>₂⟩}</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 {⟨<i>integer expression</i>⟩}</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 {⟨<i>integer expression</i>⟩}</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 {⟨<i>intexpr</i>₁⟩} {⟨<i>intexpr</i>₂⟩}</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 {⟨<i>integer expression</i>⟩}</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 {⟨<i>integer expression</i>⟩}</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 {⟨<i>intexpr</i>₁⟩} {⟨<i>intexpr</i>₂⟩}</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 {⟨<i>integer expression</i>⟩}</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 {⟨<i>integer expression</i>⟩}</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 {⟨<i>intexpr</i>₁⟩} {⟨<i>intexpr</i>₂⟩}</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 {⟨<i>integer expression</i>⟩}</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 {⟨<i>integer expression</i>⟩}</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 {⟨<i>intexpr</i>₁⟩} {⟨<i>intexpr</i>₂⟩}</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 {⟨<i>integer expression</i>⟩}</code>
-------------------------------------	---

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

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

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

<code>\l_char_active_seq</code>	Used to track which tokens will require special handling at the document level as they are of category $\langle active \rangle$ (catcode 13). Each entry in the sequence consists of a single active character. Active tokens should be added to the sequence when they are defined for general document use.
---------------------------------	---

New: 2012-01-23

<code>\l_char_special_seq</code>	Used to track which tokens will require special handling when working with verbatim-like material at the document level as they are not of categories $\langle letter \rangle$ (catcode 11) or $\langle other \rangle$ (catcode 12). Each entry in the sequence consists of a single escaped token, for example <code>\</code> for the backslash or <code>\{</code> for an opening brace. Escaped tokens should be added to the sequence when they are defined for general document use.
----------------------------------	--

New: 2012-01-23

37 Generic tokens

<code>\token_new:Nn</code>	<code>\token_new:Nn <token1> {{token2}}</code>
----------------------------	--

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

<code>\c_group_begin_token</code> <code>\c_group_end_token</code> <code>\c_math_toggle_token</code> <code>\c_alignment_token</code> <code>\c_parameter_token</code> <code>\c_math_superscript_token</code> <code>\c_math_subscript_token</code> <code>\c_space_token</code>	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.
--	---

<code>\c_catcode_letter_token</code> <code>\c_catcode_other_token</code>	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.
---	---

<code>\c_catcode_active_tl</code>	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.
-----------------------------------	--

38 Converting tokens

<code>\token_to_meaning:N</code> ★	<code>\token_to_meaning:N <token></code>
------------------------------------	--

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 <token></code>
<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.

\TeX hackers note: `\token_to_str:N` is the \TeX primitive `\string` renamed.

39 Token conditionals

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

Tests if $\langle token \rangle$ has the category code of a begin group token (`{` when normal \TeX 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 <token></code>
<code>\token_if_group_end:NTF</code> ★	<code>\token_if_group_end:NTF <token> {\true code} {\false code}</code>

Tests if $\langle token \rangle$ has the category code of an end group token (`}` when normal \TeX 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 <token></code>
<code>\token_if_math_toggle:NTF</code> ★	<code>\token_if_math_toggle:NTF <token> {\true code} {\false code}</code>

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

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

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

<code>\token_if_parameter_p:N</code>	<code>*</code>	<code>\token_if_parameter_p:N</code>	<code><token></code>
<code>\token_if_parameter:NTF</code>	<code>*</code>	<code>\token_if_alignment:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if `<token>` has the category code of a macro parameter token (`#` when normal `TEX` category codes are in force).

<code>\token_if_math_superscript_p:N</code>	<code>*</code>	<code>\token_if_math_superscript_p:N</code>	<code><token></code>
<code>\token_if_math_superscript:NTF</code>	<code>*</code>	<code>\token_if_math_superscript:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if `<token>` has the category code of a superscript token (`^` when normal `TEX` 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:NTF</code>	<code>*</code>	<code>\token_if_math_subscript:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if `<token>` 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:NTF</code>	<code>*</code>	<code>\token_if_space:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if `<token>` 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:NTF</code>	<code>*</code>	<code>\token_if_letter:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if `<token>` 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:NTF</code>	<code>*</code>	<code>\token_if_other:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if `<token>` 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:NTF</code>	<code>*</code>	<code>\token_if_active:NTF</code>	<code><token> {\true code} {\false code}</code>

Tests if `<token>` 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:NNTF</code>	<code>*</code>	<code>\token_if_eq_catcode:NNTF</code>	<code><token1> <token2> {\true code} {\false code}</code>

Tests if the two `<tokens>` 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:NNTF</code>	<code>*</code>	<code>\token_if_eq_charcode:NNTF</code>	<code><token1> <token2> {\true code} {\false code}</code>

Tests if the two `<tokens>` have the same character code.

<code>\token_if_eq_meaning_p:NN</code>	★	<code>\token_if_eq_meaning_p:NN</code>	$\langle token1 \rangle$	$\langle token2 \rangle$
<code>\token_if_eq_meaning:NNTF</code>	★	<code>\token_if_eq_meaning:NNTF</code>	$\langle token1 \rangle$	$\langle token2 \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

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

<code>\token_if_macro_p:N</code>	★	<code>\token_if_macro_p:N</code>	$\langle token \rangle$
<code>\token_if_macro:NNTF</code>	★	<code>\token_if_macro:NNTF</code>	$\langle token \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

Updated: 2011-05-23

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

<code>\token_if_cs_p:N</code>	★	<code>\token_if_cs_p:N</code>	$\langle token \rangle$
<code>\token_if_cs:NNTF</code>	★	<code>\token_if_cs:NNTF</code>	$\langle token \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

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:NNTF</code>	★	<code>\token_if_expandable:NNTF</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:NNTF</code>	★	<code>\token_if_long_macro:NNTF</code>	$\langle token \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

Updated: 2012-01-20

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

Updated: 2012-01-20

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

Updated: 2012-01-20

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

Updated: 2012-01-20

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

\TeX hackers note: Booleans, boxes and small integer constants are implemented as chardefs.

<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>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Updated: 2012-01-20

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>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Updated: 2012-01-20

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>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Updated: 2012-01-20

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

T_EXhackers note: Constant integers may be implemented as integer registers, chardefs, or mathchardefs depending on their value.

<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>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Updated: 2012-01-20

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>	$\langle token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

Updated: 2012-01-20

Tests if the $\langle token \rangle$ is defined to be a toks register (not used byL^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: 2011-05-23

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

40 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.

\peek_after:Nw**\peek_after:Nw** $\langle function \rangle$ $\langle token \rangle$

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 `\`, `{` or `}` (assuming normal T_EX category codes), *i.e.* it is not necessarily the next argument which would be grabbed by a normal function.

\peek_gafter:Nw**\peek_gafter:Nw** $\langle function \rangle$ $\langle token \rangle$

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 `\`, `{` or `}` (assuming normal T_EX category codes), *i.e.* it is not necessarily the next argument which would be grabbed by a normal function.

\l_peek_tokenToken set by `\peek_after:Nw` and available for testing as described above.

\g_peek_tokenToken set by `\peek_gafter:Nw` and available for testing as described above.

\peek_catcode:NTF**\peek_catcode:NTF** $\langle test token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

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).

\peek_catcode_ignore_spaces:NNTF**\peek_catcode_ignore_spaces:NNTF** $\langle test token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

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).

\peek_catcode_remove:NNTF**\peek_catcode_remove:NNTF** $\langle test token \rangle$ $\{\langle true code \rangle\}$ $\{\langle false code \rangle\}$

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 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_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 *<token>* in the input stream has the same category code as the *<test token>* (as defined by the test `\token_if_eq_catcode:NNTF`). Spaces are ignored by the test and the *<token>* will be removed from the input stream if the test is true. The function will then place either the *<true code>* or *<false code>* 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 *<token>* in the input stream has the same character code as the *<test token>* (as defined by the test `\token_if_eq_charcode:NNTF`). Spaces are respected by the test and the *<token>* will be left in the input stream after the *<true code>* or *<false code>* (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 *<token>* in the input stream has the same character code as the *<test token>* (as defined by the test `\token_if_eq_charcode:NNTF`). Spaces are ignored by the test and the *<token>* will be left in the input stream after the *<true code>* or *<false code>* (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 *<token>* in the input stream has the same character code as the *<test token>* (as defined by the test `\token_if_eq_charcode:NNTF`). Spaces are respected by the test and the *<token>* will be removed from the input stream if the test is true. The function will then place either the *<true code>* or *<false code>* 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 *<token>* in the input stream has the same character code as the *<test token>* (as defined by the test `\token_if_eq_charcode:NNTF`). Spaces are ignored by the test and the *<token>* will be removed from the input stream if the test is true. The function will then place either the *<true code>* or *<false code>* in the input stream (as appropriate to the result of the test).

<code>\peek_meaning:NTF</code>	<code>\peek_meaning:NTF <test token> {{true code}} {{false code}}</code>
--------------------------------	--

Updated: 2011-07-02

Tests if the next *<token>* in the input stream has the same meaning as the *<test token>* (as defined by the test `\token_if_eq_meaning:NNTF`). Spaces are respected by the test and the *<token>* will be left in the input stream after the *<true code>* or *<false code>* (as appropriate to the result of the test).

<code>\peek_meaning_ignore_spaces:NTF</code>	<code>\peek_meaning_ignore_spaces:NTF <test token> {<true code>} {<false code>}</code>
--	--

Updated: 2011-07-02

Tests if the next *<token>* in the input stream has the same meaning as the *<test token>* (as defined by the test `\token_if_eq_meaning:NNTF`). Spaces are ignored by the test and the *<token>* will be left in the input stream after the *<true code>* or *<false code>* (as appropriate to the result of the test).

<code>\peek_meaning_remove:NTF</code>	<code>\peek_meaning_remove:NTF <test token> {<true code>} {<false code>}</code>
---------------------------------------	---

Updated: 2011-07-02

Tests if the next *<token>* in the input stream has the same meaning as the *<test token>* (as defined by the test `\token_if_eq_meaning:NNTF`). Spaces are respected by the test and the *<token>* will be removed from the input stream if the test is true. The function will then place either the *<true code>* or *<false code>* in the input stream (as appropriate to the result of the test).

<code>\peek_meaning_remove_ignore_spaces:NTF</code>	<code>\peek_meaning_remove_ignore_spaces:NTF <test token> {<true code>} {<false code>}</code>
---	---

Updated: 2011-07-02

Tests if the next *<token>* in the input stream has the same meaning as the *<test token>* (as defined by the test `\token_if_eq_meaning:NNTF`). Spaces are ignored by the test and the *<token>* will be removed from the input stream if the test is true. The function will then place either the *<true code>* or *<false code>* in the input stream (as appropriate to the result of the test).

41 Decomposing a macro definition

These functions decompose \TeX macros into their constituent parts: if the *<token>* 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 <token></code>
--------------------------------------	--

If the *<token>* 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 *<token>* 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_spec:N</code>	★	<code>\token_get_replacement_spec:N <token></code>
--	---	--

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 <token></code>
---------------------------------------	---	---

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

42 Experimental token functions

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

`\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.

<code>\char_gset_active:Npn</code>	<code>\char_gset_active:Npn <char> <parameters> {<code>}</code>
------------------------------------	---

`\char_gset_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, 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).

<code>\char_set_active_eq:NN</code>	<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 \TeX group level, and the definition is also local.

<hr/> <code>\char_gset_active_eq:NN</code> <hr/>	<code>\char_gset_active_eq:NN</code> $\langle char \rangle$ $\langle function \rangle$
<code>New: 2011-12-27</code> <hr/>	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 \TeX 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</code> $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$
<code>New: 2011-08-14</code> <hr/>	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`”).

43 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, rounding any remainder. Ties are rounded away from zero. 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.

44 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 $\langle integer \rangle$ 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 $\langle integer2 \rangle$.

45 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	

46 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.

47 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.

48 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/> <hr/> <code>\int_do_until:nNnn</code> ☆	<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/> <hr/> <code>\int_until_do:nNnn</code> ☆	<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/> <hr/> <code>\int_while_do:nNnn</code> ☆	<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/> <hr/> <code>\int_do_while:nn</code> ☆	<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/> <hr/> <code>\int_do_until:nn</code> ☆	<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/> <hr/> <code>\int_until_do:nn</code> ☆	<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**.

49 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).

50 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.

51 Viewing integers

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

52 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/>	

53 Scratch integers

<hr/> <hr/>	
<code>\l_tmpa_int</code>	Scratch integer 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_int</code>	
<code>\l_tmpc_int</code>	
<hr/> <hr/>	
<code>\g_tmpa_int</code>	Scratch integer 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_int</code>	

54 Internal functions

<hr/> <hr/>	
<code>\int_get_digits:n</code> ★	<code>\int_get_digits:n</code> $\langle value \rangle$
	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).
<hr/> <hr/>	
<code>\int_get_sign:n</code> ☆	<code>\int_get_sign:n</code> $\langle value \rangle$
	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).
<hr/> <hr/>	
<code>\int_to_letter:n</code> ★	<code>\int_to_letter:n</code> $\langle integer\ value \rangle$
Updated: 2011-09-17	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, <i>etc.</i>
<hr/> <hr/>	
<code>\int_to_roman:w</code> ★	<code>\int_to_roman:w</code> $\langle integer \rangle$ $\langle space \rangle$ or $\langle non-expandable\ token \rangle$
	Converts $\langle integer \rangle$ to it 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 <i>are</i> 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>

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

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>
		<code>\fi:</code>

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.

55 Creating and initialising dim variables

```
\dim_new:N  
\dim_new:c
```

```
\dim_new:N <dimension>
```

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

```
\dim_zero:N  
\dim_zero:c  
\dim_gzero:N  
\dim_gzero:c
```

```
\dim_zero:N <dimension>
```

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

```
\dim_zero_new:N  
\dim_zero_new:c  
\dim_gzero_new:N  
\dim_gzero_new:c
```

```
\dim_zero_new:N <dimension>
```

Ensures that the $\langle dimension \rangle$ exists globally by applying `\dim_new:N` if necessary, then applies `\dim_(g)zero:N` to leave the $\langle dimension \rangle$ set to zero.

New: 2012-01-07

56 Setting dim variables

```
\dim_add:Nn  
\dim_add:cn  
\dim_gadd:Nn  
\dim_gadd:cn
```

```
\dim_add:Nn <dimension> {<dimension expression>}
```

Adds the result of the $\langle dimension expression \rangle$ to the current content of the $\langle dimension \rangle$.

Updated: 2011-10-22

<code>\dim_set:Nn</code>	<code>\dim_set:Nn <dimension> {<dimension expression>}</code>
<code>\dim_set:cn</code>	
<code>\dim_gset:Nn</code>	Sets $\langle dimension \rangle$ to the value of $\langle dimension expression \rangle$, which must evaluate to a length with units.
<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>	
<code>\dim_gset_eq:NN</code>	Sets the content of $\langle dimension1 \rangle$ equal to that of $\langle dimension2 \rangle$.
<code>\dim_gset_eq:(cN Nc cc)</code>	

<code>\dim_set_max:Nn</code>	<code>\dim_set_max:Nn <dimension> {<dimension expression>}</code>
<code>\dim_set_max:cn</code>	
<code>\dim_gset_max:Nn</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.
<code>\dim_gset_max:cn</code>	

Updated: 2012-02-06

<code>\dim_set_min:Nn</code>	<code>\dim_set_min:Nn <dimension> {<dimension expression>}</code>
<code>\dim_set_min:cn</code>	
<code>\dim_gset_min:Nn</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.
<code>\dim_gset_min:cn</code>	

Updated: 2012-02-06

<code>\dim_sub:Nn</code>	<code>\dim_sub:Nn <dimension> {<dimension expression>}</code>
<code>\dim_sub:cn</code>	
<code>\dim_gsub:Nn</code>	Subtracts the result of the $\langle dimension expression \rangle$ to the current content of the $\langle dimension \rangle$.
<code>\dim_gsub:cn</code>	

Updated: 2011-10-22

57 Utilities for dimension calculations

<code>\dim_abs:n</code> ★	<code>\dim_abs:n {<dimexpr>}</code>
Updated: 2011-10-22	Converts the $\langle dimexpr \rangle$ to its absolute value, leaving the result in the input stream as an $\langle dimension denotation \rangle$.

`\dim_ratio:nn` ★ `\dim_ratio:nn {<dimexpr1>} {<dimexpr2>}`

Updated: 2011-10-22 Parses the two *<dimension expressions>* and converts the ratio of the two to a form suitable for use inside a *<dimension expression>*. 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.

58 Dimension expression conditionals

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

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

59 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**.

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

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

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

Places the *<code>* in the input stream for T_EX to process, and then evaluates the relationship between the two *<dimension expressions>* as described for `\dim_compare:nNnTF`. 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**.

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

Evaluates the relationship between the two *<dimension expressions>* as described for `\dim_compare:nTF`, 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:nn</code> ☆	<code>\dim_do_until:nn { <dimexpr₁> <relation> <dimexpr₂> } {<code>}</code>
---------------------------------	---

Evaluates the relationship between the two *<dimension expressions>* as described for `\dim_compare:nTF`, 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**.

<code>\dim_until_do:nn</code> ☆	<code>\dim_until_do:nn { <dimexpr₁> <relation> <dimexpr₂> } {<code>}</code>
---------------------------------	---

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

<code>\dim_while_do:nn</code> ☆	<code>\dim_while_do:nn { <dimexpr1> <relation> <dimexpr2> } {<code>}</code>
---------------------------------	---

Places the *<code>* in the input stream for T_EX to process, and then evaluates the relationship between the two *<dimension expressions>* as described for `\dim_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`.

60 Using dim expressions and variables

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

Updated: 2011-10-22

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

<code>\dim_use:N</code> ★	<code>\dim_use:N <dimension></code>
---------------------------	---

`\dim_use:c` ★

Recovers the content of a *<dimension>* 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 `\dim_eval:n`).

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

61 Viewing dim variables

<code>\dim_show:N</code>	<code>\dim_show:N <dimension></code>
--------------------------	--

`\dim_show:c`

Displays the value of the *<dimension>* 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 *<dimension expression>* on the terminal.

62 Constant dimensions

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

`\c_zero_dim`

A zero length as a dimension or a skip (these are equivalent).

63 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.

64 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.

`\skip_zero_new:N`
`\skip_zero_new:c`
`\skip_gzero_new:N`
`\skip_gzero_new:c`

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

Ensures that the $\langle skip \rangle$ exists globally by applying `\skip_new:N` if necessary, then applies `\skip_(g)zero:N` to leave the $\langle skip \rangle$ set to zero.

New: 2012-01-07

65 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 <skip1> <skip2>

```

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

```

\skip_sub:Nn
\skip_sub:cn
\skip_gsub:Nn
\skip_gsub:cn

```

```

\skip_sub:Nn <skip> {\skip expression}

```

Subtracts the result of the $\langle skip expression \rangle$ to the current content of the $\langle skip \rangle$.

Updated: 2011-10-22

66 Skip expression conditionals

```

\skip_if_eq_p:nn ★
\skip_if_eq:nnTF ★

```

```

\skip_if_eq_p:nn {\skipexpr1} {\skipexpr2}
\dim_compare:nTF
  {\skip expr1} {\skip expr2}
  {\true code} {\false code}

```

This function first evaluates each of the $\langle skip expressions \rangle$ 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.

```

\skip_if_infinite_glue_p:n ★
\skip_if_infinite_glue:nTF ★

```

```

\skip_if_infinite_glue_p:n {\skipexpr}
\skip_if_infinite_glue:nTF {\skipexpr} {\true code} {\false code}

```

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

67 Using skip expressions and variables

```

\skip_eval:n ★

```

Updated: 2011-10-22

```

\skip_eval:n {\skip expression}

```

Evaluates the $\langle skip expression \rangle$, expanding any skips and token list variables within the $\langle expression \rangle$ 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 $\langle glue denotation \rangle$ after two expansions. This will be expressed in points (`pt`), and will require suitable termination if used in a T_EX-style assignment as it is *not* an $\langle internal glue \rangle$.

<code>\skip_use:N</code> ★	<code>\skip_use:N</code> $\langle skip \rangle$
<code>\skip_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>\skip_eval:n</code>).

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

68 Viewing skip variables

<code>\skip_show:N</code>	<code>\skip_show:N</code> $\langle skip \rangle$
<code>\skip_show:c</code>	Displays the value of the $\langle skip \rangle$ on the terminal.

<code>\skip_show:n</code>	<code>\skip_show:n</code> $\langle skip \text{ expression} \rangle$
New: 2011-11-22	Displays the result of evaluating the $\langle skip \text{ expression} \rangle$ on the terminal.

69 Constant skips

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

<code>\c_zero_skip</code>	A zero length as a dimension or a skip (these are equivalent).
---------------------------	--

70 Scratch skips

<code>\l_tmpa_skip</code>	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.
<code>\l_tmpb_skip</code>	
<code>\l_tmpc_skip</code>	

<code>\g_tmpa_skip</code>	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.
<code>\g_tmpb_skip</code>	

71 Creating and initialising muskip variables

`\muskip_new:N`
`\muskip_new:c`

`\muskip_new:N` $\langle muskip \rangle$

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.

`\muskip_zero:N`
`\muskip_zero:c`
`\muskip_gzero:N`
`\muskip_gzero:c`

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

Sets $\langle muskip \rangle$ to 0 mu.

`\muskip_zero_new:N`
`\muskip_zero_new:c`
`\muskip_gzero_new:N`
`\muskip_gzero_new:c`

`\muskip_zero_new:N` $\langle muskip \rangle$

Ensures that the $\langle muskip \rangle$ exists globally by applying `\muskip_new:N` if necessary, then applies `\muskip_(g)zero:N` to leave the $\langle muskip \rangle$ set to zero.

New: 2012-01-07

72 Setting muskip variables

`\muskip_add:Nn`
`\muskip_add:cn`
`\muskip_gadd:Nn`
`\muskip_gadd:cn`

`\muskip_add:Nn` $\langle muskip \rangle$ $\{ \langle muskip \text{ expression} \rangle \}$

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

Updated: 2011-10-22

`\muskip_set:Nn`
`\muskip_set:cn`
`\muskip_gset:Nn`
`\muskip_gset:cn`

`\muskip_set:Nn` $\langle muskip \rangle$ $\{ \langle muskip \text{ expression} \rangle \}$

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

Updated: 2011-10-22

`\muskip_set_eq:NN`
`\muskip_set_eq:(cN|Nc|cc)`
`\muskip_gset_eq:NN`
`\muskip_gset_eq:(cN|Nc|cc)`

`\muskip_set_eq:NN` $\langle muskip1 \rangle$ $\langle muskip2 \rangle$

Sets the content of $\langle muskip1 \rangle$ equal to that of $\langle muskip2 \rangle$.

`\muskip_sub:Nn`
`\muskip_sub:cn`
`\muskip_gsub:Nn`
`\muskip_gsub:cn`

`\muskip_sub:Nn` $\langle muskip \rangle$ $\{ \langle muskip \text{ expression} \rangle \}$

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

Updated: 2011-10-22

73 Using muskip expressions and variables

<hr/> <code>\muskip_eval:n</code> ★ <hr/>	<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 μ , and will require suitable termination if used in a T _E X-style assignment as it is <i>not</i> an $\langle internal muglue \rangle$.

<hr/> <code>\muskip_use:N</code> ★ <code>\muskip_use:c</code> ★ <hr/>	<code>\muskip_use:N ⟨muskip⟩</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>).
--	---

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

74 Inserting skips into the output

<hr/> <code>\skip_horizontal:N</code> <code>\skip_horizontal:(c n)</code> <hr/>	<code>\skip_horizontal:N ⟨skip⟩</code> <code>\skip_horizontal:n {⟨skipexpr⟩}</code>
Updated: 2011-10-22	Inserts a horizontal $\langle skip \rangle$ into the current list.

T_EXhackers note: `\skip_horizontal:N` is the T_EX primitive `\hskip` renamed.

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

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

75 Viewing muskip variables

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

76 Internal functions

```

\if_dim:w <dimen1> <relation> <dimen1>
  <true code>
\else:
  <false>
\fi:

```

Compare two dimensions. The *<relation>* is one of <, = or > with category code 12.

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

```

\dim_eval:w ★ \dim_eval:w <dimexpr> \dim_eval_end:
\dim_eval_end ★

```

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

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

77 Experimental skip functions

```

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

```

Updated: 2011-10-22

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

78 Internal functions

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

New: 2011-11-11

Evaluates the *⟨dimension expression⟩*, expanding any dimensions and token list variables within the *⟨expression⟩* 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 *⟨dimension expression⟩* 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.

79 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>	

80 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>`.

81 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	
$\text{\tl_set:Nn \l_tmpa_tl \{abcccd\} \tl_remove_all:Nn \l_tmpa_tl \{bc\}}$ <p>will result in <code>\l_tmpa_tl</code> containing <code>abcd</code>.</p>	

82 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> .

83 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.

84 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:NN {\tl var_1} {\tl var_2}</code>
<code>\tl_if_eq_p:(Nc cN cc)</code>	★	<code>\tl_if_eq:NNTF {\tl var_1} {\tl var_2} {\true code} {\false code}</code>
<code>\tl_if_eq:NNTF</code>	★	
<code>\tl_if_eq:(Nc cN cc)TF</code>	★	Compares the content of two <i>token list variables</i> 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

```

\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 <token list1> {\token list2} {\true code} {\false code}</code>
		Tests if <i><token list1></i> and <i><token list2></i> are equal, both in respect of character codes and category codes.

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

<code>\tl_if_in:nnTF</code>		<code>\tl_if_in:nnTF {\token list1} {\token list2} {\true code} {\false code}</code>
<code>\tl_if_in:(Vn on no)TF</code>		Tests if <i><token list2></i> is found inside <i><token list1></i> . The <i><token list></i> cannot contain the tokens <code>{</code> , <code>}</code> or <code>#</code> (assuming the usual T _E X category codes apply).

<code>\tl_if_single_p:N</code>	★	<code>\tl_if_single_p:N {\tl var}</code>
<code>\tl_if_single_p:c</code>	★	<code>\tl_if_single:NNTF {\tl var} {\true code} {\false code}</code>
<code>\tl_if_single:NNTF</code>	★	
<code>\tl_if_single:cTF</code>	★	Tests if the content of the <i><tl var></i> 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> .
Updated: 2011-08-13		

<code>\tl_if_single_p:n</code>	★	<code>\tl_if_single_p:n {\token list}</code>
<code>\tl_if_single:nTF</code>	★	<code>\tl_if_single:nTF {\token list} {\true code} {\false code}</code>
Updated: 2011-08-13		
		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> .

<code>\tl_if_single_token_p:n</code>	★	<code>\tl_if_single_token_p:n {\token list}</code>
<code>\tl_if_single_token:nTF</code>	★	<code>\tl_if_single_token:nTF {\token list} {\true code} {\false code}</code>
New: 2011-08-11		
		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.

85 Mapping to token lists

<hr/>	
<code>\tl_map_function:NN</code> ☆	<code>\tl_map_function:NN</code> $\langle tl\ var \rangle$ $\langle function \rangle$
<code>\tl_map_function:cN</code> ☆	
<hr/>	
	Applies $\langle function \rangle$ to every $\langle item \rangle$ in the $\langle tl\ var \rangle$. The $\langle function \rangle$ will receive one argument for each iteration. This may be a number of tokens if the $\langle item \rangle$ was stored within braces. Hence the $\langle function \rangle$ 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</code> $\langle token\ list \rangle$ $\langle function \rangle$
<hr/>	
	Applies $\langle function \rangle$ to every $\langle item \rangle$ in the $\langle token\ list \rangle$, The $\langle function \rangle$ will receive one argument for each iteration. This may be a number of tokens if the $\langle item \rangle$ was stored within braces. Hence the $\langle function \rangle$ 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</code> $\langle tl\ var \rangle$ $\{\langle inline\ function \rangle\}$
<code>\tl_map_inline:cN</code>	
<hr/>	
	Applies the $\langle inline\ function \rangle$ to every $\langle item \rangle$ stored within the $\langle tl\ var \rangle$. 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. See also <code>\tl_map_function:Nn</code> .
<hr/>	
<code>\tl_map_inline:nn</code>	<code>\tl_map_inline:nn</code> $\langle token\ list \rangle$ $\{\langle inline\ function \rangle\}$
<hr/>	
	Applies the $\langle inline\ function \rangle$ to every $\langle item \rangle$ stored within the $\langle token\ list \rangle$. 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. See also <code>\tl_map_function:nn</code> .
<hr/>	
<code>\tl_map_variable:NNn</code>	<code>\tl_map_variable:NNn</code> $\langle tl\ var \rangle$ $\langle variable \rangle$ $\{\langle function \rangle\}$
<code>\tl_map_variable:cNn</code>	
<hr/>	
	Applies the $\langle function \rangle$ to every $\langle item \rangle$ stored within the $\langle tl\ var \rangle$. The $\langle function \rangle$ should consist of code which will receive the $\langle item \rangle$ stored in the $\langle variable \rangle$. 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</code> $\langle token\ list \rangle$ $\langle variable \rangle$ $\{\langle function \rangle\}$
<hr/>	
	Applies the $\langle function \rangle$ to every $\langle item \rangle$ stored within the $\langle token\ list \rangle$. The $\langle function \rangle$ should consist of code which will receive the $\langle item \rangle$ stored in the $\langle variable \rangle$. 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.

86 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.

87 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

`\tl_length:N` ★ `\tl_length:N {\tl var}`

`\tl_length:c` ★

Updated: 2011-08-13

Counts the number of token groups in the $\langle tl\ var \rangle$ and leaves this information in the input stream. Unbraced tokens count as one element as do each token group $\{\dots\}$. This process will ignore any unprotected spaces within $\langle tokens \rangle$. See also `\tl_length:n`. This function requires three expansions, giving an $\langle integer\ denotation \rangle$.

`\tl_reverse:n` ★ `\tl_reverse:n {\token list}`

`\tl_reverse:(V|o)` ★

Updated: 2012-01-08

Reverses the order of the $\langle items \rangle$ in the $\langle token\ list \rangle$, so that $\langle item1 \rangle \langle item2 \rangle \langle item3 \rangle \dots \langle item_n \rangle$ becomes $\langle item_n \rangle \dots \langle item3 \rangle \langle item2 \rangle \langle item1 \rangle$. This process will preserve unprotected space within the $\langle token\ list \rangle$. Tokens are not reversed within braced token groups, which keep their outer set of braces. In situations where performance is important, consider `\tl_reverse_items:n`. See also `\tl_reverse:N`.

T_EXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the token list will not expand further when appearing in an x-type argument expansion.

`\tl_reverse:N` `\tl_reverse:N {\tl var}`

`\tl_reverse:c`

`\tl_greverse:N`

`\tl_greverse:c`

Updated: 2012-01-08

Reverses the order of the $\langle items \rangle$ stored in $\langle tl\ var \rangle$, so that $\langle item1 \rangle \langle item2 \rangle \langle item3 \rangle \dots \langle item_n \rangle$ becomes $\langle item_n \rangle \dots \langle item3 \rangle \langle item2 \rangle \langle item1 \rangle$. This process will preserve unprotected spaces within the $\langle token\ list\ variable \rangle$. Braced token groups are copied without reversing the order of tokens, but keep the outer set of braces. See also `\tl_reverse:n`.

`\tl_reverse_items:n` ★ `\tl_reverse_items:n {\token list}`

New: 2012-01-08

Reverses the order of the $\langle items \rangle$ stored in $\langle tl\ var \rangle$, so that $\{\langle item1 \rangle\} \{\langle item2 \rangle\} \{\langle item3 \rangle\} \dots \{\langle item_n \rangle\}$ becomes $\{\langle item_n \rangle\} \dots \{\langle item3 \rangle\} \{\langle item2 \rangle\} \{\langle item1 \rangle\}$. This process will remove any unprotected space within the $\langle token\ list \rangle$. 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 `\tl_reverse:n` or `\tl_reverse_tokens:n`.

T_EXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the token list will not expand further when appearing in an x-type argument expansion.

`\tl_trim_spaces:n` ★ `\tl_trim_spaces:n \token list`

New: 2011-07-09

Updated: 2011-08-13

Removes any leading and trailing explicit space characters from the $\langle token\ list \rangle$ and leaves the result in the input stream. This process requires two expansions.

T_EXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the token list will not expand further when appearing in an x-type argument expansion.

`\tl_trim_spaces:N`
`\tl_trim_spaces:c`
`\tl_gtrim_spaces:N`
`\tl_gtrim_spaces:c`

New: 2011-07-09

`\tl_trim_spaces:N` $\langle \textit{tl var} \rangle$

Removes any leading and trailing explicit space characters from the content of the $\langle \textit{tl var} \rangle$.

88 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: 2012-01-08

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

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

`\tl_head:n` $\{ \text{ abc } \}$

and

`\tl_head:n` $\{ \sim \text{ abc } \}$

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

T_EXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the token list will not expand further when appearing in an x-type argument expansion.

`\tl_head:w` ★

`\tl_head:w` $\langle \textit{tokens} \rangle$ `\q_stop`

Leaves in the input stream the first non-space token from the $\langle \textit{tokens} \rangle$. An empty list of $\langle \textit{tokens} \rangle$ or one which consists only of space (category code 10) tokens will result in an error, and thus $\langle \textit{tokens} \rangle$ 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.

<hr/>	
<code>\tl_tail:n</code> ★	<code>\tl_tail:n {⟨tokens⟩}</code>
<code>\tl_tail:(V v f)</code> ★	
Updated: 2012-01-08	Discards the all leading space tokens and the first non-space token in the <i>⟨tokens⟩</i> , 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 *⟨tokens⟩* or one which consists only of space (category code 10) tokens will result in `\tl_tail:n` leaving nothing in the input stream.

TeXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the token list will not expand further when appearing in an x-type argument expansion.

<hr/>	
<code>\tl_tail:w</code> ★	<code>\tl_tail:w {⟨tokens⟩} \q_stop</code>
	Discards the all leading space tokens and the first non-space token in the <i>⟨tokens⟩</i> , and leaves the remaining tokens in the input stream. An empty list of <i>⟨tokens⟩</i> or one which consists only of space (category code 10) tokens will result in an error, and thus <i>⟨tokens⟩</i> must <i>not</i> be “blank” as determined by <code>\tl_if_blank:n(TF)</code> . This function requires only a single expansion, and thus is suitable for use within an o-type expansion. In general, <code>\tl_tail:n</code> should be preferred if the number of expansions is not critical.

<hr/>	
<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 <i>⟨tokens⟩</i> into a string, as described for <code>\tl_to_str:n</code> . The <code>\str_head:n</code> function then leaves the first character of this string in the input stream. The <code>\str_tail:n</code> function leaves all characters except the first in the input stream. The first character may be a space. If the <i>⟨tokens⟩</i> argument is entirely empty, nothing is left in the input stream.

<hr/>	
<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>
Updated: 2011-08-10	<code>{⟨true code⟩} {⟨false code⟩}</code>

Tests if the first *⟨token⟩* in the *⟨token list⟩* has the same category code as the *⟨test token⟩*. In the case where *⟨token list⟩* is empty, its head is considered to be `\q_nil`, and the test will be true if *⟨test token⟩* 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 *<token>* in the *<token list>* has the same character code as the *<test token>*. In the case where *<token list>* is empty, its head is considered to be `\q_nil`, and the test will be true if *<test token>* 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 *<token>* in the *<token list>* has the same meaning as the *<test token>*. In the case where *<token list>* is empty, its head is considered to be `\q_nil`, and the test will be true if *<test token>* 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 *<token>* in the *<token list>* is an explicit begin-group character (with category code 1 and any character code), in other words, if the *<token list>* starts with a brace group. In particular, the test is false if the *<token list>* 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.

<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>

New: 2011-08-11

Tests if the first *<token>* in the *<token list>* 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.

<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>

Updated: 2011-08-11

Tests if the first *<token>* in the *<token list>* is an explicit space character (with category code 10 and character code 32). If *<token list>* starts with an implicit token such as `\c_space_token`, 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.

T_EXhackers note: When T_EX 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.

89 Viewing token lists

<code>\tl_show:N</code>	<code>\tl_show:N <tl var></code>
<code>\tl_show:c</code>	Displays the content of the <i><tl var></i> on the terminal.

T_EXhackers note: `\tl_show:N` is the T_EX primitive `\show`.

<code>\tl_show:n</code>	<code>\tl_show:n <token list></code>
	Displays the <i><token list></i> on the terminal.

T_EXhackers note: `\tl_show:n` is the ε -T_EX primitive `\showtokens`.

90 Constant token lists

<code>\c_job_name_tl</code>	Constant that gets the “job name” assigned when T _E X starts.
-----------------------------	--

Updated: 2011-08-18

T_EXhackers note: This is the new name for the primitive `\jobname`. It is a constant that is set by T_EX 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.
--------------------------	--

91 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.
--	--

92 Experimental token list functions

<code>\tl_reverse_tokens:n</code> ★	<code>\tl_reverse_tokens:n {<tokens>}</code>
-------------------------------------	--

New: 2012-01-08

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.

T_EXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the token list will not expand further when appearing in an x-type argument 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: 2012-01-08

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.

T_EXhackers note: Begin-group and end-group characters are normalized and become `{` and `}`, respectively. The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the token list will not expand further when appearing in an x-type argument expansion.

<hr/>	
<code>\tl_item:nn</code> ★	<code>\tl_item:nn {⟨token list⟩} {⟨integer expression⟩}</code>
<code>\tl_item:(Nn cn)</code> ★	
<hr/>	
New: 2011-11-21	
Updated: 2012-01-08	
<hr/>	

Indexing items in the $\langle token list \rangle$ from 0 on the left, this function will evaluate the $\langle integer expression \rangle$ and leave the appropriate item from the $\langle token list \rangle$ in the input stream. If the $\langle integer expression \rangle$ is negative, indexing occurs from the right of the token list, starting at -1 for the right-most item. If the index is out of bounds, then the function expands to nothing.

TeXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the $\langle item \rangle$ will not expand further when appearing in an x-type argument expansion.

93 Internal functions

<hr/>	
<code>\q_tl_act_mark</code>	Quarks which are only used for the particular purposes of <code>\tl_act...</code> functions.
<code>\q_tl_act_stop</code>	
<hr/>	
<hr/>	
<code>\tl_to_str_active_safe:Nx</code>	<code>\tl_to_str_active_safe:Nx ⟨tl var⟩ {⟨tokens⟩}</code>
<hr/>	
New: 2012-01-25	
<hr/>	

Exhaustively-expands the $\langle tokens \rangle$ with the exception of any category $\langle active \rangle$ (cat-code 12) tokens, which are not expanded. The resulting $\langle expanded tokens \rangle$ are then converted to a string as described for `\tl_to_str:n`. The list of $\langle active \rangle$ tokens is taken from `\l_char_active_seq`.

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.

94 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>		

95 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 $\langle sequence \rangle$.

96 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.

```
\seq_gpop_left:NN
\seq_gpop_left:cN
```

```
\seq_gpop_left:NN <sequence> <token list variable>
```

Pops the left-most item from a *<sequence>* into the *<token list variable>*, *i.e.* removes the item from the sequence and stores it in the *<token list variable>*. The *<sequence>* is modified globally, while the assignment of the *<token list variable>* is local. If *<sequence>* is empty an error will be raised.

```
\seq_pop_right:NN
\seq_pop_right:cN
```

```
\seq_pop_right:NN <sequence> <token list variable>
```

Pops the right-most item from a *<sequence>* into the *<token list variable>*, *i.e.* removes the item from the sequence and stores it in the *<token list variable>*. Both of the variables are assigned locally. If *<sequence>* is empty an error will be raised.

```
\seq_gpop_right:NN
\seq_gpop_right:cN
```

```
\seq_gpop_right:NN <sequence> <token list variable>
```

Pops the right-most item from a *<sequence>* into the *<token list variable>*, *i.e.* removes the item from the sequence and stores it in the *<token list variable>*. The *<sequence>* is modified globally, while the assignment of the *<token list variable>* is local. If *<sequence>* is empty an error will be raised.

97 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.

```
\seq_remove_duplicates:N
\seq_remove_duplicates:c
\seq_gremove_duplicates:N
\seq_gremove_duplicates:c
```

```
\seq_remove_duplicates:N <sequence>
```

Removes duplicate items from the *<sequence>*, leaving the left most copy of each item in the *<sequence>*. The *<item>* 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 *<sequence>* and does a comparison with the *<items>* already checked. It is therefore relatively slow with large sequences.

```
\seq_remove_all:Nn
\seq_remove_all:cn
\seq_gremove_all:Nn
\seq_gremove_all:cn
```

```
\seq_remove_all:Nn <sequence> {<item>}
```

Removes every occurrence of *<item>* from the *<sequence>*. The *<item>* comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`.

98 Sequence conditionals

```
\seq_if_empty_p:N *
\seq_if_empty_p:c *
\seq_if_empty:NTF *
\seq_if_empty:cTF *
```

```
\seq_if_empty_p:N <sequence>
```

```
\seq_if_empty:NNTF <sequence> {<true code>} {<false code>}
```

Tests if the *<sequence>* is empty (containing no items).

<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>*.

99 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.

100 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:(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>	<code>\seq_push:Nn <sequence> {(item)}</code>
--	---

Adds the $\{(item)\}$ to the top of the $\langle sequence \rangle$.

101 Viewing sequences

<code>\seq_show:N</code> <code>\seq_show:c</code>	<code>\seq_show:N <sequence></code> Displays the entries in the $\langle sequence \rangle$ in the terminal.
--	--

102 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:cNTF</code>	<code>\seq_get_left:NNTF <sequence> <token list variable> {(true code)} {(false code)}</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:cNTF</code>	<code>\seq_get_right:NNTF <sequence> <token list variable> {(true code)} {(false code)}</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:cNTF</code>	<code>\seq_pop_left:NNTF <sequence> <token list variable> {(true code)} {(false code)}</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:cNTF</code>	<code>\seq_gpop_left:NNTF <sequence> <token list variable> {(true code)} {(false code)}</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.
--	---

<code>\seq_pop_right:NNTF</code>	<code>\seq_pop_right:NNTF <sequence> <token list variable> {(true code)} {(false code)}</code>
<code>\seq_pop_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, pops the right-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$, *i.e.* 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_right:NNTF</code>	<code>\seq_gpop_right:NNTF <sequence> <token list variable></code>
<code>\seq_gpop_right:cNTF</code>	<code>{(true code)} {(false code)}</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 right-most item from a $\langle sequence \rangle$ in the $\langle token list variable \rangle$, *i.e.* 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.

<code>\seq_length:N</code> ☆	<code>\seq_length:N <sequence></code>
<code>\seq_length:c</code> ☆	

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.e.* every item in a $\langle sequence \rangle$ is unique.

<code>\seq_item:Nn</code> ☆	<code>\seq_item:Nn <sequence> {(integer expression)}</code>
<code>\seq_item:cn</code> ☆	

Indexed 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 `\seq_length:N`) then the function will expand to nothing.

T_EXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the $\langle item \rangle$ will not expand further when appearing in an x-type argument expansion.

<code>\seq_use:N</code> ☆	<code>\seq_use:N <sequence></code>
<code>\seq_use:c</code> ☆	

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 `\seq_map_break:` or `\seq_map_break:n`. The $\langle items \rangle$ in the $\langle sequence \rangle$ will be used from left (top) to right (bottom).

<code>\seq_mapthread_function:NNN</code> ☆	<code>\seq_mapthread_function:NNN <seq1> <seq2> <function></code>
<code>\seq_mapthread_function:(NcN cNN ccN)</code> ☆	

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.e.* 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.

103 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.

104 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 <comma list1> <comma list2> <comma list3></code>
<code>\clist_concat:ccc</code>	
<code>\clist_gconcat:NNN</code>	Concatenates the content of <code><comma list2></code> and <code><comma list3></code> together and saves the result in <code><comma list1></code> . The items in <code><comma list2></code> will be placed at the left side of the new comma list.
<code>\clist_gconcat:ccc</code>	

105 Adding data to comma lists

<hr/>	
<code>\clist_set:Nn</code>	<code>\clist_set:Nn <comma list> {<item1>,...,<item_n>}</code>
<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 `<comma list>` to contain the `<items>`, 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 <comma list> {<item1>,...,<item_n>}</code>
<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 `<items>` to the left of the `<comma list>`. Spaces are removed from both sides of each item.

<hr/>	
<code>\clist_put_right:Nn</code>	<code>\clist_put_right:Nn <comma list> {<item1>,...,<item_n>}</code>
<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 `<items>` to the right of the `<comma list>`. Spaces are removed from both sides of each item.

106 Using comma lists

<hr/>	
<code>\clist_use:N</code> ★	<code>\clist_use:N <comma list></code>
<code>\clist_use:c</code> ★	Places the <code><comma list></code> directly into the input stream, including the commas, thus treating it as a <code><token list></code> .

107 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 *<comma list>*, leaving the left most copy of each item in the *<comma list>*. The *<item>* 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 *<comma list>* and does a comparison with the *<items>* already checked. It is therefore relatively slow with large comma lists. Furthermore, it will not work if any of the items in the *<comma list>* 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 *<item>* from the *<comma list>*. The *<item>* comparison takes place on a token basis, as for `\tl_if_eq:nn(TF)`.

Updated: 2011-09-06

T_EXhackers note: The *<item>* may not contain `{`, `}`, or `#` (assuming the usual T_EX category codes apply).

108 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 <i><comma list></i> 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 <i><comma lists></i> 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.

109 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:NnNn</code> <code>\clist_map_variable:(cNn nNn)</code>	<code>\clist_map_variable:NnNn <comma list> <tl var.> {\<function using tl var.>}</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.

110 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.

111 Viewing comma lists

<hr/> <code>\clist_show:N</code> <hr/>	<code>\clist_show:N</code> $\langle comma list \rangle$
<code>\clist_show:c</code> <hr/>	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.

112 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/>	
<code>New: 2011-09-06</code> <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/>	
<code>New: 2011-09-06</code> <hr/>	

113 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> ★ <hr/>	<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.
<code>New: 2011-06-25</code>	
<code>Updated: 2011-09-06</code> <hr/>	

`\clist_item:Nn` ★
`\clist_item:(cn|nn)` ★

Updated: 2012-01-08

`\clist_item:Nn <comma list> {<integer expression>}`

Indexing items in the *<comma list>* from 0 at the top (left), this function will evaluate the *<integer expression>* and leave the appropriate item from the comma list in the input stream. If the *<integer expression>* is negative, indexing occurs from the bottom (right) of the comma list. When the *<integer expression>* is larger than the number of items in the *<comma list>* (as calculated by `\clist_length:N`) then the function will expand to nothing.

T_EXhackers note: The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the *<item>* will not expand further when appearing in an x-type argument expansion.

`\clist_set_from_seq:NN`
`\clist_set_from_seq:(cn|Nc|cc)`
`\clist_gset_from_seq:NN`
`\clist_gset_from_seq:(cn|Nc|cc)`

Updated: 2011-08-31

`\clist_set_from_seq:NN <comma list> <sequence>`

Sets the *<comma list>* to be equal to the content of the *<sequence>*. Items which contain either spaces or commas are surrounded by braces.

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

New: 2011-11-26

`\clist_const:Nn <clist var> {<comma list>}`

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

`\clist_if_empty_p:n` ★
`\clist_if_empty:nTF` ★

New: 2011-12-07

`\clist_if_empty_p:n {<comma list>}`

`\clist_if_empty:nTF {<comma list>} {<true code>} {<false code>}`

Tests if the *<comma list>* is empty (containing no items). The rules for space trimming are as for other n-type comma-list functions, hence the comma list `{~,~,~}` (without outer braces) is empty, while `{~, {}, }` (without outer braces) contains one element, which happens to be empty: the comma-list is not empty.

114 Internal comma-list functions

`\clist_trim_spaces:n` ★

New: 2011-07-09

`\clist_trim_spaces:n {<comma list>}`

Removes leading and trailing spaces from each *<item>* in the *<comma list>*, 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.

115 Creating and initialising property lists

 $\backslash prop_new:N$
 $\backslash prop_new:c$

 $\backslash prop_new:N$ $\langle property\ list \rangle$

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.

 $\backslash prop_clear:N$
 $\backslash prop_clear:c$
 $\backslash prop_gclear:N$
 $\backslash prop_gclear:c$

 $\backslash prop_clear:N$ $\langle property\ list \rangle$

Clears all entries from the $\langle property\ list \rangle$.

 $\backslash prop_clear_new:N$
 $\backslash prop_clear_new:c$
 $\backslash prop_gclear_new:N$
 $\backslash prop_gclear_new:c$

 $\backslash prop_clear_new:N$ $\langle property\ list \rangle$

Ensures that the $\langle property\ list \rangle$ exists globally by applying $\backslash prop_new:N$ if necessary, then applies $\backslash prop_clear:N$ to leave the list empty.

 $\backslash prop_set_eq:NN$
 $\backslash prop_set_eq:(cN|Nc|cc)$
 $\backslash prop_gset_eq:NN$
 $\backslash prop_gset_eq:(cN|Nc|cc)$

 $\backslash prop_set_eq:NN$ $\langle property\ list1 \rangle$ $\langle property\ list2 \rangle$

Sets the content of $\langle property\ list1 \rangle$ equal to that of $\langle property\ list2 \rangle$.

116 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>	

117 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.

118 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.

119 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:NnNTF <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`.

120 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.

121 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.

122 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.

123 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: 2012-01-08

T_EXhackers note: This function is slower than the non-expandable analogue `\prop_get:NnN`. The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the *<value>* will not expand further when appearing in an x-type argument expansion.

124 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_`.

125 Creating and initialising boxes

<code>\box_new:N</code>	<code>\box_new:N <box></code>
<code>\box_new:c</code>	Creates a new <code><box></code> or raises an error if the name is already taken. The declaration is global. The <code><box></code> 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 <code><box></code> 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 <code><box></code> exists globally by applying <code>\box_new:N</code> if necessary, then applies <code>\box_(g)clear:N</code> to leave the <code><box></code> 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 <code><box1></code> equal to that of <code><box2></code> .
<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 <code><box1></code> within the current TeX group equal to that of <code><box2></code> , then clears <code><box2></code> 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 <code><box1></code> equal to that of <code><box2></code> , then clears <code><box2></code> . These assignments are global.

126 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 `\box_use:N <box>` or a “raw” box specification such as `\vbox:n { xyz }`.

<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 `\box_use:N <box>` or a “raw” box specification such as `\vbox:n { xyz }`.

127 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/> <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/> <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	

128 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	
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

129 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

130 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> *	

131 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>	

132 Constant boxes

<code>\c_empty_box</code>	This is a permanently empty box, which is neither set as horizontal nor vertical.
---------------------------	---

133 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>	

134 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`.

135 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/> <code>\hbox_to_wd:nn</code> <hr/>	<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/> <code>\hbox_to_zero:n</code> <hr/>	<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/> <code>\hbox_set:Nn</code> <code>\hbox_set:cn</code> <code>\hbox_gset:Nn</code> <code>\hbox_gset:cn</code> <hr/>	<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/> <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> <hr/>	<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/> <code>\hbox_overlap_right:n</code> <hr/>	<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/> <code>\hbox_overlap_left:n</code> <hr/>	<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/> <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> <hr/>	<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/> <code>\hbox_unpack:N</code> <code>\hbox_unpack:c</code> <hr/>	<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`.

136 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`

`\vbox_gset:Nn`

`\vbox_gset:cn`

Typesets the $\langle contents \rangle$ at natural height and then stores the result inside the $\langle box \rangle$.

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).

TeXhackers note: This is the TeX 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.

TeXhackers note: This is the TeX 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.

TeXhackers note: This is the TeX primitive `\unvbox`.

137 Primitive box conditionals

<code>\if_hbox:N</code> ★	<code>\if_hbox:N <box></code> <code><true code></code> <code>\else:</code> <code><false code></code> <code>\fi:</code> Tests is <code><box></code> is a horizontal box.
---------------------------	--

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

<code>\if_vbox:N</code> ★	<code>\if_vbox:N <box></code> <code><true code></code> <code>\else:</code> <code><false code></code> <code>\fi:</code> Tests is <code><box></code> is a vertical box.
---------------------------	--

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

<code>\if_box_empty:N</code> ★	<code>\if_box_empty:N <box></code> <code><true code></code> <code>\else:</code> <code><false code></code> <code>\fi:</code> Tests is <code><box></code> is an empty (void) box.
--------------------------------	--

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

138 Experimental box functions

<code>\box_show:Nnn</code>	<code>\box_show:Nnn <box> <int 1> <int 2></code>
<code>\box_show:cnn</code>	Display the contents of <code><box></code> in the terminal, showing the first <code><int 1></code> items of the box, and descending into <code><int 1></code> levels of nesting.
New: 2011-11-21	

T_EXhackers note: This is a wrapper around the T_EX primitives `\showbox`, `\showboxbreadth` and `\showboxdepth`.

<code>\box_show_full:N</code>	<code>\box_show_full:N <box></code>
<code>\box_show_full:c</code>	Display the contents of <code><box></code> in the terminal, showing all items in the box.
New: 2011-11-22	

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).

139 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.

140 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.

```
\vcoffin_set:Nnn
\vcoffin_set:cnn
```

New: 2011-08-17
Updated: 2011-09-03

```
\vcoffin_set:Nnn <coffin> {<width>} {<material>}
```

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.

```
\vcoffin_set:Nnw
\vcoffin_set:cnn
\vcoffin_set_end
```

New: 2011-09-10

```
\vcoffin_set:Nnw <coffin> {<width>} <material> \vcoffin_set_end:
```

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.

```
\coffin_set_horizontal_pole:Nnn
\coffin_set_horizontal_pole:cnn
```

New: 2011-08-17

```
\coffin_set_horizontal_pole:Nnn <coffin>
{<pole>} {<offset>}
```

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 $\backslash TotalHeight$, $\backslash Height$, $\backslash Depth$ and $\backslash Width$, which will evaluate to the appropriate dimensions of the $\langle coffin \rangle$.

```
\coffin_set_vertical_pole:Nnn
\coffin_set_vertical_pole:cnn
```

New: 2011-08-17

```
\coffin_set_vertical_pole:Nnn <coffin> {<pole>} {<offset>}
```

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 $\backslash TotalHeight$, $\backslash Height$, $\backslash Depth$ and $\backslash Width$, which will evaluate to the appropriate dimensions of the $\langle coffin \rangle$.

141 Coffin transformations

```
\coffin_resize:Nnn
\coffin_resize:cnn
```

New: 2011-09-02

```
\coffin_resize:Nnn <coffin> {<width>} {<total-height>}
```

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 $\backslash TotalHeight$, $\backslash Height$, $\backslash Depth$ and $\backslash Width$, which will evaluate to the appropriate dimensions of the $\langle coffin \rangle$.

This function is experimental.

```
\coffin_rotate:Nn
\coffin_rotate:cnn
```

New: 2011-09-02

```
\coffin_rotate:Nn <coffin> {<angle>}
```

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.

142 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.

143 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> .

144 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.
Updated: 2012-01-01 <hr/>	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.

145 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.

`\color_group_begin`
`\color_group_end`

New: 2011-09-03

`\color_group_begin:`

`...`

`\color_group_end:`

Creates a colour group: one used to “trap” colour settings.

`\color_ensure_current`

New: 2011-09-03

`\color_ensure_current:`

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 `\color_group_begin: ... \color_group_end: group`.

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).

146 Managing streams

`\ior_new:N`
`\ior_new:c`
`\iow_new:N`
`\io_new:c`

New: 2011-09-26
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`
`\ior_open:cn`

Updated: 2012-01-25

`\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.

T_EXhackers note: The $\langle file\ name \rangle$ may contain both literal items and expandable content, which should on full expansion be the desired file name. Any active characters (as declared in `\l_char_active_seq`) will *not* be expanded, allowing the direct use of these in file names.

`\iow_open:Nn`
`\iow_open:cn`

Updated: 2012-01-25

`\iow_open:Nn <stream> {{file name}}`

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).

T_EXhackers note: The $\langle file\ name \rangle$ may contain both literal items and expandable content, which should on full expansion be the desired file name. Any active characters (as declared in `\l_char_active_seq`) will *not* be expanded, allowing the direct use of these in file names.

`\ior_close:N`
`\ior_close:c`

Updated: 2011-12-27

`\ior_close:N <stream>`

Closes the $\langle stream \rangle$. Streams should always be closed when they are finished with as this ensures that they remain available to other programmer.

`\iow_close:N`
`\iow_close:c`

Updated: 2011-12-27

`\iow_close:N <stream>`

Closes the $\langle stream \rangle$. Streams should always be closed when they are finished with as this ensures that they remain available to other programmer.

`\ior_list_streams`
`\iow_list_streams`

`\ior_list_streams:`
`\iow_list_streams:`

Displays a list of the file names associated with each open stream: intended for tracking down problems.

147 Writing to files

`\iow_now:Nn`
`\iow_now:Nx`

`\iow_now:Nn <stream> {{tokens}}`

This functions writes $\langle tokens \rangle$ to the specified $\langle stream \rangle$ immediately (*i.e.* the write operation is called on expansion of `\iow_now:Nn`).

T_EXhackers note: `\iow_now:Nx` is a protected macro which expands to the two T_EX primitives `\immediate\write`.

`\iow_log:n`
`\iow_log:x`

`\iow_log:n {{tokens}}`

This function writes the given $\langle tokens \rangle$ to the log (transcript) file immediately: it is a dedicated version of `\iow_now:Nn`.

`\iow_term:n`
`\iow_term:x`

`\iow_term:n {{tokens}}`

This function writes the given $\langle tokens \rangle$ to the terminal file immediately: it is a dedicated version of `\iow_now:Nn`.

<hr/> <code>\iow_now_when_avail:Nn</code> <hr/> <code>\iow_now_when_avail:Nx</code> <hr/>	<code>\iow_now_when_avail:Nn <stream> {<tokens>}</code> <p>If $\langle stream \rangle$ is open, writes the $\langle tokens \rangle$ to the $\langle stream \rangle$ in the same manner as <code>\iow_now:Nn</code>. If the $\langle stream \rangle$ is not open, the $\langle tokens \rangle$ are simply thrown away.</p>
<hr/> <code>\iow_shipout:Nn</code> <hr/> <code>\iow_shipout:Nx</code> <hr/>	<code>\iow_shipout:Nn <stream> {<tokens>}</code> <p>This function writes $\langle tokens \rangle$ to the specified $\langle stream \rangle$ when the current page is finalised (<i>i.e.</i> at shipout). The x-type variants expand the $\langle tokens \rangle$ at the point where the function is used but <i>not</i> when the resulting tokens are written to the $\langle stream \rangle$ (<i>cf.</i> <code>\iow_shipout_x:Nn</code>).</p>
<hr/> <code>\iow_shipout_x:Nn</code> <hr/> <code>\iow_shipout_x:Nx</code> <hr/>	<code>\iow_shipout_x:Nn <stream> {<tokens>}</code> <p>This function writes $\langle tokens \rangle$ to the specified $\langle stream \rangle$ when the current page is finalised (<i>i.e.</i> at shipout). The $\langle tokens \rangle$ 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>
<p>T_EXhackers note: <code>\iow_shipout_x:Nn</code> is the T_EX primitive <code>\write</code> renamed.</p>	
<hr/> <code>\iow_char:N</code> ★ <hr/>	<code>\iow_char:N <token></code> <p>Inserts $\langle token \rangle$ into the output stream. Useful when trying to write difficult characters such as %, {, }, <i>etc.</i> in messages, for example:</p> <pre style="margin-left: 40px;"><code>\iow_now:Nx \g_my_iow { \iow_char:N \{ text \iow_char:N \} }</code></pre> <p>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>
<hr/> <code>\iow_newline</code> ★ <hr/>	<code>\iow_newline:</code> <p>Function to add a new line within the $\langle tokens \rangle$ 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>

148 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 <code>x</code>-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	

149 Reading from files

<code>\ior_to:NN</code>	<code>\ior_to:NN</code> $\langle stream \rangle$ $\langle token list variable \rangle$
<code>\ior_gto:NN</code>	

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 in the $\langle token list \rangle$ variable, locally or globally. 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: This protected macro expands to the T_EX primitives `\read` or `\global\read` along with the `to` keyword.

<code>\ior_str_to:NN</code>	<code>\ior_str_to:NN</code> $\langle stream \rangle$ $\langle token list variable \rangle$
<code>\ior_str_gto:NN</code>	

Functions that reads one line from the input $\langle stream \rangle$ and stores the result in the $\langle token list \rangle$ variable, locally or globally. 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: This protected macro expands to the ε -T_EX primitives `\readline` or `\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 `true` 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.

150 Constants

<code>\c_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_log_ior</code>	Constant output streams for writing to the log and to the terminal (plus the log), respectively.
<code>\c_term_ior</code>	

151 Experimental functions

<code>\ior_map_inline:nn</code>	<code>\ior_map_inline:nn {<file name>} {<inline function>}</code>
---------------------------------	---

Applies the *<inline function>* to *<items>* obtained by reading one or more lines (until an equal number of left and right braces are found) from the *<file>*. The *<inline function>* should consist of code which will receive the *<line>* as **#1**. One in line mapping can be nested inside another. If the file is not found, the *<inline function>* is never called.

<code>\ior_str_map_inline:nn</code>	<code>\ior_str_map_inline:nn {<file name>} {<inline function>}</code>
-------------------------------------	---

Applies the *<inline function>* to every *<line>* in the *<file>*. The material is read from the *<file>* as a series of tokens with category code 12 (other), with the exception of space characters which are given category code 10 (space). The *<inline function>* should consist of code which will receive the *<line>* as **#1**. One in line mapping can be nested inside another. If the file is not found, the *<inline function>* is never called.

152 Internal input–output functions

<code>\if_eof:w</code> ★	<code>\if_eof:w <stream></code> <code> <true code></code> <code>\else:</code> <code> <false code></code> <code>\fi:</code>
--------------------------	--

Tests if the *<stream>* returns “end of file”, which is true for non-existent files. The `\else:` branch is optional.

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

<code>\ior_open_unsafe:Nn</code>	<code>\ior_open_unsafe:Nn <stream> {<file name>}</code>
<code>\iow_open_unsafe:Nn</code>	

New: 2012-01-23

These functions have identical syntax to the generally-available versions without the `_unsafe` suffix. However, these functions do not take precautions against active characters in the *<file name>*: they are therefore intended to be used by higher-level functions which have already fully expanded the *<file name>* and which need to perform multiple open or close operations. See for example the implementation of `\file_add_path:Nn`,

<code>\ior_raw_new:N</code>	<code>\ior_raw_new:N <stream></code>
<code>\iow_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_EX level, when a new stream is requested by the stack itself.

<hr/>	
<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.
<hr/>	

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 preceded 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.

<code>\msg_kernel_new:nnnn</code> <code>\msg_kernel_new:nnn</code>	<code>\msg_kernel_new:nnnn {<module>} {<message>} {<text>} {<more text>}</code>
---	---

Updated: 2011-08-16

Creates a kernel `<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.

<code>\msg_kernel_set:nnnn</code> <code>\msg_kernel_set:nnn</code>	<code>\msg_kernel_set:nnnn {<module>} {<message>} {<text>} {<more text>}</code>
---	---

Sets up the text for a kernel `<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.

<code>\msg_kernel_fatal:nnxxxx</code> <code>\msg_kernel_fatal:(nnxxx nnxx nnx nn)</code>	<code>\msg_kernel_fatal:nnxxxx {<module>} {<message>} {<arg one>} {<arg two>} {<arg three>} {<arg four>}</code>
---	---

Issues kernel `<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. Cannot be redirected.

<code>\msg_kernel_error:nnxxxx</code> <code>\msg_kernel_error:(nnxxx nnxx nnx nn)</code>	<code>\msg_kernel_error:nnxxxx {<module>} {<message>} {<arg one>} {<arg two>} {<arg three>} {<arg four>}</code>
---	---

Issues kernel `<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. Cannot be redirected.

<code>\msg_kernel_warning:nnxxxx</code> <code>\msg_kernel_warning:(nnxxx nnxx nnx nn)</code>	<code>\msg_kernel_warning:nnxxxx {<module>} {<message>} {<arg one>} {<arg two>} {<arg three>} {<arg four>}</code>
---	---

Issues kernel `<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_kernel_info:nnxxxx</code> <code>\msg_kernel_info:(nnxxx nnxx nnx nn)</code>	<code>\msg_kernel_info:nnxxxx {<module>} {<message>} {<arg one>} {<arg two>} {<arg three>} {<arg four>}</code>
---	--

Issues kernel `<module>` information `<message>`, passing `<arg one>` to `<arg four>` 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.

<hr/>	
<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>
<hr/>	
	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.

<hr/>	
<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.
<hr/>	

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.

<hr/>	
<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>
<hr/>	

Prints the *<message>* from *<module>* in the terminal, without formatting.

<hr/>	
<code>\msg_aux_show:x</code>	<code>\msg_aux_show:x {<formatted string>}</code>
<hr/>	

Shows the *<formatted string>* on the terminal. The *<formatted string>* must start with `^^J>`, failure to do so causes low-level T_EX errors.

<hr/>	
<code>\msg_aux_show:Nnx</code>	<code>\msg_aux_show:Nnx <variable> {<module>} {<token list>}</code>
<hr/>	

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.

```
.bool_set:N <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/> <div>New: 2011-08-28</div>	<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/> <div>New: 2011-08-21</div>	<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/> <div>New: 2011/09/11</div>	<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/> <div>New: 2011/09/11</div>	<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/> <hr/>	<code><key> .generate_choices:n = {<list>}</code>	
		This property will mark <code><key></code> as a multiple choice key, and will use the <code><list></code> to define the choices. The <code><list></code> should consist of a comma-separated list of choice names. Each choice will be set up to execute <code><code></code> as set using <code>.choice_code:n</code> (or <code>.choice_code:x</code>). Choices are discussed in detail in section 163.
<hr/> <hr/>	<code><key> .int_set:N = <integer></code> <code>.int_set:c</code>	
		Defines <code><key></code> to set <code><integer></code> to <code><value></code> (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 <code><integer></code> will be assigned locally.
<hr/> <hr/>	<code><key> .int_gset:N = <integer></code> <code>.int_gset:c</code>	
		Defines <code><key></code> to set <code><integer></code> to <code><value></code> (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 <code><integer></code> will be assigned globally.
<hr/> <hr/>	<code><key> .meta:n = {<keyval list>}</code> <code>.meta:x</code>	
		Makes <code><key></code> a meta-key, which will set <code><keyval list></code> in one go. If <code><key></code> is given with a value at the time the key is used, then the value will be passed through to the subsidiary <code><keys></code> for processing (as #1).
<hr/> <hr/>	<code><key> .multichoice:</code> New: 2011-08-21	
		Sets <code><key></code> to act as a multiple choice key. Each valid choice for <code><key></code> must then be created, as discussed in section 163.
		This property is experimental.
<hr/> <hr/>	<code><key> .multichoice:nn <choices> <code></code> New: 2011-08-21	
		Sets <code><key></code> to act as a multiple choice key, and defines a series <code><choices></code> which are implemented using the <code><code></code> . Inside <code><code></code> , <code>\l_keys_choice_tl</code> will be the name of the choice made, and <code>\l_keys_choice_int</code> will be the position of the choice in the list of <code><choices></code> (indexed from 0). Choices are discussed in detail in section 163.
		This property is experimental.
<hr/> <hr/>	<code><key> .skip_set:N = <skip></code> <code>.skip_set:c</code>	
		Defines <code><key></code> to set <code><skip></code> to <code><value></code> (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 <code><skip></code> will be assigned locally.
<hr/> <hr/>	<code><key> .skip_gset:N = <skip></code> <code>.skip_gset:c</code>	
		Defines <code><key></code> to set <code><skip></code> to <code><value></code> (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 <code><skip></code> 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>}`

Updated: 2012-01-25

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>`

Updated: 2012-01-25

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. Any active characters (as declared in `\l_char_active_seq`) will *not* be expanded, allowing the direct use of these in file names.

<code>\file_input:n</code>	<code>\file_input:n {<file name>}</code>
----------------------------	--

Updated: 2012-01-25	Searches for $\langle file\ name \rangle$ in the path as detailed for <code>\file_if_exist:nTF</code> , and if found reads in the file as additional L ^A T _E X source. All files read are recorded for information and the file name stack is updated by this function.
---------------------	---

T_EXhackers note: The $\langle file\ name \rangle$ may contain both literal items and expandable content, which should on full expansion be the desired file name. Any active characters (as declared in `\l_char_active_seq`) will *not* be expanded, allowing the direct use of these in file names.

<code>\file_split_path_name_ext:nNNN</code>	<code>\file_split_path_name_ext:nNNN {<name>} <path> <filename> <ext></code>
---	--

New: 2012-01-25

Splits the file $\langle name \rangle$ into any $\langle path \rangle$ (up to the last /), any $\langle ext \rangle$ (from the last . to the end of the $\langle name \rangle$), and the $\langle filename \rangle$ (the rest of the $\langle name \rangle$). Thus for example

`\file_split_path_name_ext:nNNN { figures / example.foo.eps }`

will result in $\langle path \rangle$ `figures/`, $\langle filename \rangle$ `example.foo` and $\langle ext \rangle$ `eps`. The three parts of the $\langle name \rangle$ will be stored in the token list variables given as the $\langle name \rangle$, $\langle path \rangle$ and $\langle filename \rangle$ arguments.

T_EXhackers note: The $\langle name \rangle$ may contain both literal items and expandable content, which should on full expansion be the desired file name. Any active characters (as declared in `\l_char_active_seq`) will *not* be expanded, allowing the direct use of these in file names.

<code>\file_path_include:n</code>	<code>\file_path_include:n {<path>}</code>
-----------------------------------	--

Adds $\langle path \rangle$ 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 $\langle path \rangle$ 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 <code>\file_input:n</code> . This is needed to restore the appropriate file name to <code>\g_file_current_name_tl</code> at the end of each file.
--------------------------------	---

<u>\g_file_record_seq</u>	Stores the name of every file loaded using <code>\file_input:n</code> . In contrast to <code>\g_file_stack_seq</code> , no items are ever removed from this sequence.
<u>\l_file_internal_name_tl</u>	Used to return the full name of a file for internal use.
<u>\l_file_search_path_seq</u>	The sequence of file paths to search when loading a file.
<u>\l_file_internal_saved_path_seq</u>	When loaded on top of L ^A T _E X 2 _ε , there is a need to save the search path so that <code>\input@path</code> can be used as appropriate.
<u>\l_file_internal_seq</u> <small>New: 2011-09-06</small>	When loaded on top of L ^A T _E X 2 _ε , there is a need to convert the comma lists <code>\input@path</code> and <code>\@filelist</code> to sequences.

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 is 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>	

<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>	

<code>\fp_zero_new:N</code>	<code>\fp_zero_new:N</code> <i><floating point variable></i>
<code>\fp_zero_new:c</code>	Ensures that the <i><floating point variable></i> exists globally by applying <code>\fp_new:N</code> if necessary, then applies <code>\fp_(g)zero:N</code> to leave the <i><floating point variable></i> set to zero.
<code>\fp_gzero_new:N</code>	
<code>\fp_gzero_new:c</code>	

New: 2012-01-07

<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>	

<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> in the units points. This means that distances given in other units are first converted to points before being assigned to the <i><floating point variable></i> .
<code>\fp_gset_from_dim:Nn</code>	
<code>\fp_gset_from_dim:cn</code>	

<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 be given as a real number without any exponent part, and will always include a decimal point. For example,

```

\fp_new:Nn \test
\fp_set:Nn \test { 1.234 e 5 }
\fp_use:N \test

```

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.

<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.

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.

<code>\fp_to_dim:N</code> ☆	<code>\fp_to_dim:N</code> \langle <i>floating point variable</i> \rangle
<code>\fp_to_dim:c</code> ☆	Inserts the value of the \langle <i>floating point variable</i> \rangle into the input stream converted into a dimension in points.

<code>\fp_to_int:N</code> ☆	<code>\fp_to_int:N</code> \langle <i>floating point variable</i> \rangle
<code>\fp_to_int:c</code> ☆	Inserts the integer value of the \langle <i>floating point variable</i> \rangle into the input stream. The decimal part of the number will not be included, but will be used to round the integer.

<code>\fp_to_tl:N</code> ☆	<code>\fp_to_tl:N</code> \langle <i>floating point variable</i> \rangle
<code>\fp_to_tl:c</code> ☆	Inserts a representation of the \langle <i>floating point variable</i> \rangle 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.

<code>\fp_round_figures:Nn</code>	<code>\fp_round_figures:Nn</code> \langle <i>floating point variable</i> \rangle $\{\langle$ <i>target</i> $\rangle\}$
<code>\fp_round_figures:cn</code>	
<code>\fp_ground_figures:Nn</code>	Rounds the \langle <i>floating point variable</i> \rangle to the \langle <i>target</i> \rangle number of significant figures (an integer expression).
<code>\fp_ground_figures:cn</code>	

<code>\fp_round_places:Nn</code>	<code>\fp_round_places:Nn</code> \langle <i>floating point variable</i> \rangle $\{\langle$ <i>target</i> $\rangle\}$
<code>\fp_round_places:cn</code>	
<code>\fp_ground_places:Nn</code>	Rounds the \langle <i>floating point variable</i> \rangle to the \langle <i>target</i> \rangle number of decimal places (an integer expression).
<code>\fp_ground_places:cn</code>	

173 Floating-point conditionals

<code>\fp_if_undefined_p:N</code> ★	<code>\fp_if_undefined_p:N</code> $\langle fixed\text{-}point \rangle$
<code>\fp_if_undefined:NTF</code> ★	<code>\fp_if_undefined:NTF</code> $\langle fixed\text{-}point \rangle$ $\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

Tests if $\langle floating\ point \rangle$ is undefined (*i.e.* equal to the special `\c_undefined_fp` 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> <i><floating point variable></i>
<code>\fp_abs:c</code>	
<code>\fp_gabs:N</code>	Converts the <i><floating point variable></i> to its absolute value.
<code>\fp_gabs:c</code>	

<code>\fp_neg:N</code>	<code>\fp_neg:N</code> <i><floating point variable></i>
<code>\fp_neg:c</code>	
<code>\fp_gneg:N</code>	Reverse the sign of the <i><floating point variable></i> .
<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</code> <i><floating point></i> { <i><value></i> }
<code>\fp_add:cn</code>	
<code>\fp_gadd:Nn</code>	Adds the <i><value></i> to the <i><floating point></i> .
<code>\fp_gadd:cn</code>	

<code>\fp_sub:Nn</code>	<code>\fp_sub:Nn</code> <i><floating point></i> { <i><value></i> }
<code>\fp_sub:cn</code>	
<code>\fp_gsub:Nn</code>	Subtracts the <i><value></i> from the <i><floating point></i> .
<code>\fp_gsub:cn</code>	

<code>\fp_mul:Nn</code>	<code>\fp_mul:Nn</code> <i><floating point></i> { <i><value></i> }
<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</code> <i><floating point></i> { <i><value></i> }
<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> group level. If the <i><value></i> is zero, the <i><floating point></i> will be set to <code>\c_undefined_fp</code> .
<code>\fp_gdiv:cn</code>	

176 Floating-point power operations

<code>\fp_pow:Nn</code>	<code>\fp_pow:Nn</code> <i><floating point></i> { <i><value></i> }
<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 <i><value></i> should be either a positive real number or a negative integer. If the <i><floating point></i> is positive, then the <i><value></i> may be any real value. Mathematically invalid operations such as 0^0 will give set the <i><floating point></i> to to <code>\c_undefined_fp</code> .
<code>\fp_gpow:cn</code>	

177 Exponential and logarithm functions

<u>\fp_exp:Nn</u>	<u>\fp_exp:Nn</u> <i><floating point></i> { <i><value></i> }
<u>\fp_exp:cn</u>	
<u>\fp_gexp:Nn</u>	Calculates the exponential of the <i><value></i> and assigns this to the <i><floating point></i> .
<u>\fp_gexp:cn</u>	
<hr/>	
<u>\fp_ln:Nn</u>	<u>\fp_ln:Nn</u> <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>	<u>\fp_sin:Nn</u> <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>	
<hr/>	
<u>\fp_cos:Nn</u>	<u>\fp_cos:Nn</u> <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>	
<hr/>	
<u>\fp_tan:Nn</u>	<u>\fp_tan:Nn</u> <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><code>\c_undefined_fp</code></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><code>\c_zero_fp</code></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 T_EX system. This means that the format is somewhat different from that used in, for example, computer floating point units. Programming in T_EX makes it most convenient to use a radix 10 system, using T_EX `count` registers for storage and taking advantage where possible of delimited arguments.

Part XXIII

The l3_{luatex} 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 TeX. In order to use this within the framework provided here, a family of functions is available. When used with pdfTeX or XeTeX 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 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 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.

TeXhackers 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 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: no TeX expansion of the *⟨Lua input⟩* will occur at this stage.

TeXhackers note: At a TeX level, the *⟨Lua input⟩* is stored as a “whatsit”.

<code>\lua_shipout_x:n</code> <code>\lua_shipout_x:x</code>	<code>\lua_shipout:n {⟨token list⟩}</code> <p>The <i>⟨token list⟩</i> 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 <i>⟨Lua input⟩</i> is passed to the Lua interpreter when the current page is finalised (<i>i.e.</i> at shipout). Each <code>\lua_shipout:n</code> block is treated by Lua as a separate chunk. The Lua interpreter will execute the <i>⟨Lua input⟩</i> during the page-building routine: the <i>⟨Lua input⟩</i> 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).</p>
--	--

\TeX hackers note: `\lua_shipout_x:n` is the \LaTeX 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 \LaTeX engine. In particular, \LaTeX 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 \LaTeX engine.

<code>\cctab_new:N</code>	<code>\cctab_new:N ⟨category code table⟩</code> <p>Creates a new category code table, initially with the codes as used by <code>\InitEX</code>.</p>
---------------------------	--

<code>\cctab_gset:Nn</code>	<code>\cctab_gset:Nn ⟨category code table⟩ {⟨category code set up⟩}</code> <p>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.</p>
-----------------------------	---

<code>\cctab_begin:N</code>	<code>\cctab_begin:N ⟨category code table⟩</code> <p>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>.</p>
-----------------------------	---

<code>\cctab_end</code>	<code>\cctab_end:</code> <p>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.</p>
-------------------------	---

<code>\c_code_cctab</code>	<p>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>.</p>
----------------------------	--

<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{pdfetexcmds}%
59 \let\pdfstrcmp\pdf@strcmp
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` 97 `<*package>`

`\GetIdInfoAuxII` 98 `\protected\def\GetIdInfo`

`\GetIdInfoAuxIII` 99 `{`

`\GetIdInfoAuxCVS` 100 `\begingroup`

`\GetIdInfoAuxSVN` 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 \crrcr	\tex_crrcr: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 \mutoglua	\etex_mutoglua: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 23.)

```

\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 31.)

```

\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 16.)

`\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 70.)

`\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 27.)

`\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 69.)

`\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 69.)

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 864 \tex_let:D \cs_gset_nopar:Npn \tex_gdef:D
\cs_gset:Npn 865 \tex_let:D \cs_gset_nopar:Npx \tex_xdef:D
\cs_gset:Npx 866 \cs_set_protected_nopar:Npn \cs_gset:Npn
\cs_gset_protected_nopar:Npn 867 { \tex_long:D \cs_gset_nopar:Npn }
\cs_gset_protected_nopar:Npx 868 \cs_set_protected_nopar:Npn \cs_gset:Npx
\cs_gset_protected:Npn 869 { \tex_long:D \cs_gset_nopar:Npx }
\cs_gset_protected: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 the reserved `\l_exp_internal_tl` which will be set up in `l3expan`.

```

879 \cs_set_protected:Npn \use:x #1
880 {
881   \cs_set_nopar:Npx \l_exp_internal_tl {#1}
882   \l_exp_internal_tl
883 }

```

(End definition for `\use:x`. This function is documented on page 19.)

`\use:n` These macro grabs its arguments and returns it back to the input (with outer braces removed).

```

\use:nnn 884 \cs_set:Npn \use:n #1 {#1}
\use:nnnn 885 \cs_set:Npn \use:nn #1#2 {#1#2}
886 \cs_set:Npn \use:nnn #1#2#3 {#1#2#3}
887 \cs_set:Npn \use:nnnn #1#2#3#4 {#1#2#3#4}

```

(End definition for `\use:n`. This function is documented on page ??.)

`\use_i:nn` The equivalent to L^AT_EX 2_ε's `\@firstoftwo` and `\@secondoftwo`.

`\use_ii:nn` 888 `\cs_set:Npn \use_i:nn #1#2 {#1}`

889 `\cs_set:Npn \use_ii:nn #1#2 {#2}`

(End definition for `\use_i:nn`. This function is documented on page 18.)

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

`\use_ii:nnn` 890 `\cs_set:Npn \use_i:nnn #1#2#3 {#1}`

`\use_iii:nnn` 891 `\cs_set:Npn \use_ii:nnn #1#2#3 {#2}`

`\use_i_ii:nnn` 892 `\cs_set:Npn \use_iii:nnn #1#2#3 {#3}`

`\use_i:nnnn` 893 `\cs_set:Npn \use_i_ii:nnn #1#2#3 {#1#2}`

`\use_ii:nnnn` 894 `\cs_set:Npn \use_i:nnnn #1#2#3#4 {#1}`

`\use_iii:nnnn` 895 `\cs_set:Npn \use_ii:nnnn #1#2#3#4 {#2}`

`\use_iv:nnnn` 896 `\cs_set:Npn \use_iii:nnnn #1#2#3#4 {#3}`

897 `\cs_set:Npn \use_iv:nnnn #1#2#3#4 {#4}`

(End definition for `\use_i:nnn`. This function is documented on page 18.)

`\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` 898 `\cs_set:Npn \use_none_delimit_by_q_nil:w #1 \q_nil { }`

`\use_none_delimit_by_q_recursion_stop:w` 899 `\cs_set:Npn \use_none_delimit_by_q_stop:w #1 \q_stop { }`

900 `\cs_set:Npn \use_none_delimit_by_q_recursion_stop:w #1 \q_recursion_stop { }`

(End definition for `\use_none_delimit_by_q_nil:w`. This function is documented on page 46.)

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

901 `\cs_set:Npn \use_i_delimit_by_q_nil:nw #1#2 \q_nil {#1}`

902 `\cs_set:Npn \use_i_delimit_by_q_stop:nw #1#2 \q_stop {#1}`

903 `\cs_set:Npn \use_i_delimit_by_q_recursion_stop:nw #1#2 \q_recursion_stop {#1}`

(End definition for `\use_i_delimit_by_q_nil:nw`. This function is documented on page 46.)

185.5 Gobbling tokens from input

`\use_none:n` To gobble tokens from the input we use a standard naming convention: the number of
`\use_none:nn` tokens gobbled is given by the number of `n`'s following the `:` in the name. Although
`\use_none:nnn` defining `\use_none:nnn` and above as separate calls of `\use_none:n` and `\use_none:nn`
`\use_none:nnnn` is slightly faster, this is very non-intuitive to the programmer who will assume that
`\use_none:nnnnn` expanding such a function once will take care of gobbling all the tokens in one go.

`\use_none:nnnnnn` 904 `\cs_set:Npn \use_none:n #1 { }`

`\use_none:nnnnnnnn` 905 `\cs_set:Npn \use_none:nn #1#2 { }`

`\use_none:nnnnnnnnnn` 906 `\cs_set:Npn \use_none:nnn #1#2#3 { }`

`\use_none:nnnnnnnnnnn` 907 `\cs_set:Npn \use_none:nnnn #1#2#3#4 { }`

908 `\cs_set:Npn \use_none:nnnnnn #1#2#3#4#5 { }`

909 `\cs_set:Npn \use_none:nnnnnnnn #1#2#3#4#5#6 { }`

910 `\cs_set:Npn \use_none:nnnnnnnnn #1#2#3#4#5#6#7 { }`

911 `\cs_set:Npn \use_none:nnnnnnnnnn #1#2#3#4#5#6#7#8 { }`

912 `\cs_set:Npn \use_none:nnnnnnnnnnn #1#2#3#4#5#6#7#8#9 { }`

(End definition for `\use_none:n`. This function is documented on page ??.)

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.

```
913 \cs_set_nopar:Npn \prg_return_true:
914 { \exp_after:wN \use_i:nn \int_to_roman:w }
915 \cs_set_nopar:Npn \prg_return_false:
916 { \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.

(End definition for \prg_return_true:. This function is documented on page ??.)

`\prg_set_conditional:Npnn` The user functions for the types using parameter text from the programmer. The various
`\prg_new_conditional:Npnn` functions only differ by which function is used for the assignment. For those `Npnn` type
`\prg_set_protected_conditional:Npnn` functions, we must grab the parameter text, reading everything up to a left brace before
`\prg_new_protected_conditional:Npnn` continuing. Then split the base function into name and signature, and feed `{<name>}`
`\prg_generate_conditional_parm_aux:NNpnn` `{<signature>}` `<boolean>` `<defining function>` `{parm}` `{<parameters>}` `{TF,...}` `{<code>}` to
the auxiliary function responsible for defining all conditionals.

```
917 \cs_set_protected_nopar:Npn \prg_set_conditional:Npnn
918 { \prg_generate_conditional_parm_aux:NNpnn \cs_set:Npn }
919 \cs_set_protected_nopar:Npn \prg_new_conditional:Npnn
920 { \prg_generate_conditional_parm_aux:NNpnn \cs_new:Npn }
921 \cs_set_protected_nopar:Npn \prg_set_protected_conditional:Npnn
922 { \prg_generate_conditional_parm_aux:NNpnn \cs_set_protected:Npn }
923 \cs_set_protected_nopar:Npn \prg_new_protected_conditional:Npnn
924 { \prg_generate_conditional_parm_aux:NNpnn \cs_new_protected:Npn }
925 \cs_set_protected:Npn \prg_generate_conditional_parm_aux:NNpnn #1#2#3#
926 {
927   \cs_split_function:NN #2 \prg_generate_conditional_aux:nnNNnnnn
928   #1 { parm } {#3}
929 }
```

(End definition for `\prg_set_conditional:Npnn` and others. These functions are documented on page ??.)

`\prg_set_conditional:Nnn` The user functions for the types automatically inserting the correct parameter text based on the signature. The various functions only differ by which function is used for the assignment. For those `Nnn` type functions, we calculate the number of arguments. Then split the base function into name and signature, and feed $\{\langle name \rangle\} \{\langle signature \rangle\} \langle boolean \rangle \langle defining function \rangle \{count\} \{\langle arg count \rangle\} \{TF, \dots\} \{\langle code \rangle\}$ to the auxiliary function responsible for defining all conditionals.

```

930 \cs_set_protected_nopar:Npn \prg_set_conditional:Nnn
931   { \prg_generate_conditional_count_aux:NNnn \cs_set:Npn }
932 \cs_set_protected_nopar:Npn \prg_new_conditional:Nnn
933   { \prg_generate_conditional_count_aux:NNnn \cs_new:Npn }
934 \cs_set_protected_nopar:Npn \prg_set_protected_conditional:Nnn
935   { \prg_generate_conditional_count_aux:NNnn \cs_set_protected:Npn }
936 \cs_set_protected_nopar:Npn \prg_new_protected_conditional:Nnn
937   { \prg_generate_conditional_count_aux:NNnn \cs_new_protected:Npn }
938 \cs_set_protected:Npn \prg_generate_conditional_count_aux:NNnn #1#2
939   {
940     \exp_args:Nnf \use:n
941     {
942       \cs_split_function:NN #2 \prg_generate_conditional_aux:nnNNnnnn
943       #1 { count }
944     }
945     { \cs_get_arg_count_from_signature:N #2 }
946   }

```

(End definition for `\prg_set_conditional:Nnn` and others. These functions are documented on page ??.)

`\prg_set_eq_conditional:NNn` The obvious setting-equal functions.
`\prg_new_eq_conditional:NNn`

```

947 \cs_set_protected:Npn \prg_set_eq_conditional:NNn #1#2#3
948   { \prg_set_eq_conditional_aux:NNnn \cs_set_eq:cc #1#2 {#3} }
949 \cs_set_protected:Npn \prg_new_eq_conditional:NNn #1#2#3
950   { \prg_set_eq_conditional_aux:NNnn \cs_new_eq:cc #1#2 {#3} }

```

(End definition for `\prg_set_eq_conditional:NNn` and `\prg_new_eq_conditional:NNn`. These functions are documented on page 35.)

`\prg_generate_conditional_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.

```

951 \cs_set_protected:Npn \prg_generate_conditional_aux:nnNNnnnn #1#2#3#4#5#6#7#8
952   {
953     \prg_generate_conditional_aux:nmw {#5}

```

```

954     {
955       #4 {#1} {#2} {#6} {#8}
956     }
957     #7 , ? , \q_recursion_stop
958   }

```

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.

```

959 \cs_set_protected:Npn \prg_generate_conditional_aux:nnw #1#2#3 ,
960 {
961   \if:w ?#3
962     \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
963   \fi:
964   \use:c { prg_generate_#3_form_#1:Nnnnn } #2
965   \prg_generate_conditional_aux:nnw {#1} {#2}
966 }

```

(End definition for \prg_generate_conditional_aux:nnNNnnnn and \prg_generate_conditional_aux:nnw. These functions are documented on page ??.)

```

\prg_generate_p_form_parm:Nnnnn
\prg_generate_TF_form_parm:Nnnnn
\prg_generate_T_form_parm:Nnnnn
\prg_generate_F_form_parm:Nnnnn

```

How to generate the various forms. The `parm` types here takes the following arguments: 1: how to define (an N-type), 2: name, 3: signature, 4: parameter text (or empty), 5: replacement. Remember that the logic-returning functions expect two arguments to be 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.

```

967 \cs_set_protected:Npn \prg_generate_p_form_parm:Nnnnn #1#2#3#4#5
968 {
969   \exp_args:Nc #1 { #2 _p: #3 } #4
970   {
971     #5 \c_zero
972     \c_true_bool \c_false_bool
973   }
974 }
975 \cs_set_protected:Npn \prg_generate_T_form_parm:Nnnnn #1#2#3#4#5
976 {
977   \exp_args:Nc #1 { #2 : #3 T } #4
978   {
979     #5 \c_zero
980     \use:n \use_none:n
981   }
982 }
983 \cs_set_protected:Npn \prg_generate_F_form_parm:Nnnnn #1#2#3#4#5
984 {
985   \exp_args:Nc #1 { #2 : #3 F } #4
986   {
987     #5 \c_zero
988     { }
989   }
990 }

```



```

991 \cs_set_protected:Npn \prg_generate_TF_form_parm:Nnnnn #1#2#3#4#5
992 {
993   \exp_args:Nc #1 { #2 : #3 TF } #4
994   { #5 \c_zero }
995 }

```

(End definition for \prg_generate_p_form_parm:Nnnnn and others. These functions are documented on page ??.)

```

\prg_generate_p_form_count:Nnnnn
\prg_generate_TF_form_count:Nnnnn
\prg_generate_T_form_count:Nnnnn
\prg_generate_F_form_count:Nnnnn

```

The **count** form is similar, but of course requires a number rather than a primitive argument specification.

```

996 \cs_set_protected:Npn \prg_generate_p_form_count:Nnnnn #1#2#3#4#5
997 {
998   \cs_generate_from_arg_count:cNnn { #2 _p: #3 } #1 {#4}
999   {
1000     #5 \c_zero
1001     \c_true_bool \c_false_bool
1002   }
1003 }
1004 \cs_set_protected:Npn \prg_generate_T_form_count:Nnnnn #1#2#3#4#5
1005 {
1006   \cs_generate_from_arg_count:cNnn { #2 : #3 T } #1 {#4}
1007   {
1008     #5 \c_zero
1009     \use:n \use_none:n
1010   }
1011 }
1012 \cs_set_protected:Npn \prg_generate_F_form_count:Nnnnn #1#2#3#4#5
1013 {
1014   \cs_generate_from_arg_count:cNnn { #2 : #3 F } #1 {#4}
1015   {
1016     #5 \c_zero
1017     { }
1018   }
1019 }
1020 \cs_set_protected:Npn \prg_generate_TF_form_count:Nnnnn #1#2#3#4#5
1021 {
1022   \cs_generate_from_arg_count:cNnn { #2 : #3 TF } #1 {#4}
1023   { #5 \c_zero }
1024 }

```

(End definition for \prg_generate_p_form_count:Nnnnn and others. These functions are documented on page ??.)

```

\prg_set_eq_conditional_aux:NNNn
\prg_set_eq_conditional_aux:NNNw

```

Manual clist loop over argument #4.

```

1025 \cs_set_protected:Npn \prg_set_eq_conditional_aux:NNNn #1#2#3#4
1026 { \prg_set_eq_conditional_aux:NNNw #1#2#3#4 , ? , \q_recursion_stop }
1027 \cs_set_protected:Npn \prg_set_eq_conditional_aux:NNNw #1#2#3#4 ,
1028 {
1029   \if:w ? #4 \scan_stop:
1030   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w

```

```

1031 \fi:
1032 #1
1033 { \exp_args:Nnc \cs_split_function:NN #2 { prg_conditional_form_#4:nnn } }
1034 { \exp_args:Nnc \cs_split_function:NN #3 { prg_conditional_form_#4:nnn } }
1035 \prg_set_eq_conditional_aux:NNNw #1 {#2} {#3}
1036 }
1037 \cs_set:Npn \prg_conditional_form_p:nnn #1#2#3 { #1 _p : #2 }
1038 \cs_set:Npn \prg_conditional_form_TF:nnn #1#2#3 { #1 : #2 TF }
1039 \cs_set:Npn \prg_conditional_form_T:nnn #1#2#3 { #1 : #2 T }
1040 \cs_set:Npn \prg_conditional_form_F:nnn #1#2#3 { #1 : #2 F }

```

(End definition for \prg_set_eq_conditional_aux:NNNn and \prg_set_eq_conditional_aux:NNNw. These functions are documented on page ??.)

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
1041 \tex_chardef:D \c_true_bool = 1~
1042 \tex_chardef:D \c_false_bool = 0~

```

(End definition for \c_true_bool. This function is documented on page 21.)

185.7 Dissecting a control sequence

```

\cs_to_str:N This converts a control sequence into the character string of its name, removing the
\cs_to_str_aux:N leading escape character. This turns out to be a non-trivial matter as there a different
\cs_to_str_aux:w cases:

```

- The usual case of a printable escape character;
- the case of a non-printable escape characters, 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 T_EX 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 a 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.

```

1043 \cs_set_nopar:Npn \cs_to_str:N
1044 {
1045   \int_to_roman:w
1046   \if:w \token_to_str:N \ \cs_to_str_aux:w \fi:
1047   \exp_after:wN \cs_to_str_aux:N \token_to_str:N
1048 }
1049 \cs_set:Npn \cs_to_str_aux:N #1 { \c_zero }
1050 \cs_set:Npn \cs_to_str_aux:w #1 \cs_to_str_aux:N
1051 { - \int_value:w \fi: \exp_after:wN \c_zero }

```

(End definition for \cs_to_str:N. This function is documented on page ??.)

`\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`.

```

1052 \group_begin:
1053 \tex_lccode:D '\@ = '\: \scan_stop:
1054 \tex_catcode:D '\@ = 12~
1055 \tex_lowercase:D
1056 {
1057   \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.

```

1058 \cs_set:Npn \cs_split_function:NN #1#2
1059 {
1060   \exp_after:wN \exp_after:wN
1061   \exp_after:wN \cs_split_function_aux:w
1062   \cs_to_str:N #1 @ a \q_stop #2
1063 }

```

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.

```

1064 \cs_set:Npn \cs_split_function_aux:w #1 @ #2#3 \q_stop #4
1065 {
1066   \if_meaning:w a #2
1067   \exp_after:wN \use_i:nn
1068   \else:
1069   \exp_after:wN\use_ii:nn
1070   \fi:
1071   { #4 {#1} { } \c_false_bool }
1072   { \cs_split_function_auxii:w #2#3 \q_stop #4 {#1} }
1073 }
1074 \cs_set:Npn \cs_split_function_auxii:w #1 @a \q_stop #2#3
1075 { #2{#3}{#1}\c_true_bool }
```

End of lowercase

```

1076 }
      (End definition for \cs_split_function:NN. This function is documented on page ??.)
```

`\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.

```

1077 \cs_set:Npn \cs_get_function_name:N #1
1078 { \cs_split_function:NN #1 \use_i:nnn }
1079 \cs_set:Npn \cs_get_function_signature:N #1
1080 { \cs_split_function:NN #1 \use_ii:nnn }
```

(End definition for `\cs_get_function_name:N` and `\cs_get_function_signature:N`. These functions are documented on page 19.)

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 TeX will only ever skip input in case the token tested against is `\scan_stop:`.

```

1081 \prg_set_conditional:Npnn \cs_if_exist:N #1 { p , T , F , TF }
1082 {
1083   \if_meaning:w #1 \scan_stop:
1084   \prg_return_false:
1085   \else:
1086   \if_cs_exist:N #1
1087   \prg_return_true:
1088   \else:
1089   \prg_return_false:
1090   \fi:
1091   \fi:
1092 }
```

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.

```

1093 \prg_set_conditional:Npnn \cs_if_exist:c #1 { p , T , F , TF }
1094 {
1095   \if_cs_exist:w #1 \cs_end:
1096   \exp_after:wN \use_i:nn
1097   \else:
1098   \exp_after:wN \use_ii:nn
1099   \fi:
1100   {
1101     \exp_after:wN \if_meaning:w \cs:w #1 \cs_end: \scan_stop:
1102     \prg_return_false:
1103     \else:
1104     \prg_return_true:
1105     \fi:
1106   }
1107   \prg_return_false:
1108 }

```

(End definition for `\cs_if_exist:N` and `\cs_if_exist:c`. These functions are documented on page ??.)

`\cs_if_free:N` The logical reversal of the above.

```

\cs_if_free:c
1109 \prg_set_conditional:Npnn \cs_if_free:N #1 { p , T , F , TF }
1110 {
1111   \if_meaning:w #1 \scan_stop:
1112   \prg_return_true:
1113   \else:
1114   \if_cs_exist:N #1
1115   \prg_return_false:
1116   \else:
1117   \prg_return_true:
1118   \fi:
1119   \fi:
1120 }
1121 \prg_set_conditional:Npnn \cs_if_free:c #1 { p , T , F , TF }
1122 {
1123   \if_cs_exist:w #1 \cs_end:
1124   \exp_after:wN \use_i:nn
1125   \else:
1126   \exp_after:wN \use_ii:nn
1127   \fi:
1128   {
1129     \exp_after:wN \if_meaning:w \cs:w #1 \cs_end: \scan_stop:
1130     \prg_return_true:
1131     \else:
1132     \prg_return_false:

```

```

1133         \fi:
1134     }
1135     { \prg_return_true: }
1136 }

```

(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.

```

1137 \cs_set:Npn \cs_if_exist_use:NTF #1#2
1138 { \cs_if_exist:NTF #1 { #1 #2 } }
1139 \cs_set:Npn \cs_if_exist_use:NF #1
1140 { \cs_if_exist:NTF #1 { #1 } }
1141 \cs_set:Npn \cs_if_exist_use:NT #1 #2
1142 { \cs_if_exist:NTF #1 { #1#2 } { } }
1143 \cs_set:Npn \cs_if_exist_use:N #1
1144 { \cs_if_exist:NTF #1 { #1 } { } }
1145 \cs_set:Npn \cs_if_exist_use:cTF #1#2
1146 { \cs_if_exist:cTF {#1} { \use:c {#1} #2 } }
1147 \cs_set:Npn \cs_if_exist_use:cF #1
1148 { \cs_if_exist:cTF {#1} { \use:c {#1} } }
1149 \cs_set:Npn \cs_if_exist_use:cT #1#2
1150 { \cs_if_exist:cTF {#1} { \use:c {#1} #2 } { } }
1151 \cs_set:Npn \cs_if_exist_use:c #1
1152 { \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`.

```

1153 \cs_set_protected_nopar:Npn \iow_log:x
1154 { \tex_immediate:D \tex_write:D \c_minus_one }
1155 \cs_set_protected_nopar:Npn \iow_term:x
1156 { \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 usable if terse error message and halt. This can only happen if a coding error is made by the team, so this is a reasonable response.

```

1157 \cs_set_protected:Npn \msg_kernel_error:nxxx #1#2#3#4
1158 {
1159   \tex_errmessage:D
1160   {
1161     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!~! ^^J
1162     Argh,~internal~LaTeX3~error! ^^J ^^J
1163     Module ~ #1 , ~ message-name~"#2": ^^J
1164     Arguments~'#3'~and~'#4' ^^J ^^J
1165     This~is~one~for~The~LaTeX3~Project:~bailing~out
1166   }
1167   \tex_end:D
1168 }
1169 \cs_set_protected:Npn \msg_kernel_error:nxx #1#2#3
1170 { \msg_kernel_error:nxxx {#1} {#2} {#3} { } }
1171 \cs_set_protected:Npn \msg_kernel_error:nn #1#2
1172 { \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.

```

1173 \cs_set_nopar:Npn \msg_line_context:
1174 { 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 that the argument sequence is not already in use. If it is, an error is signalled. It checks if $\langle csname \rangle$ 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!

```

1175 \cs_set_protected:Npn \chk_if_free_cs:N #1
1176 {
1177   \cs_if_free:NF #1
1178   {
1179     \msg_kernel_error:nxxx { kernel } { command-already-defined }
1180     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1181   }
1182 }
1183 <*package>
1184 \tex_ifodd:D \l@expl@log@functions@bool
1185 \cs_set_protected:Npn \chk_if_free_cs:N #1
1186 {
1187   \cs_if_free:NF #1
1188   {
1189     \msg_kernel_error:nxxx { kernel } { command-already-defined }
1190     { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1191   }

```

```

1192     \iow_log:x { Defining~\token_to_str:N #1~ \msg_line_context: }
1193   }
1194   \fi:
1195 \endpackage
1196 \cs_set_protected_nopar:Npn \chk_if_free_cs:c
1197   { \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.

```

1198 \cs_set_protected:Npn \chk_if_exist_cs:N #1
1199   {
1200     \cs_if_exist:NF #1
1201     {
1202       \msg_kernel_error:nnxx { kernel } { command-not-defined }
1203       { \token_to_str:N #1 } { \token_to_meaning:N #1 }
1204     }
1205   }
1206 \cs_set_protected_nopar:Npn \chk_if_exist_cs:c
1207   { \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 docu-
      mented 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
1208 \cs_set:Npn \cs_tmp:w #1#2
1209   {
1210     \cs_set_protected:Npn #1 ##1
1211     {
1212       \chk_if_free_cs:N ##1
1213       #2 ##1
1214     }
1215   }
1216 \cs_tmp:w \cs_new_nopar:Npn      \cs_gset_nopar:Npn
1217 \cs_tmp:w \cs_new_nopar:Npx      \cs_gset_nopar:Npx
1218 \cs_tmp:w \cs_new:Npn            \cs_gset:Npn
1219 \cs_tmp:w \cs_new:Npx            \cs_gset:Npx
1220 \cs_tmp:w \cs_new_protected_nopar:Npn \cs_gset_protected_nopar:Npn
1221 \cs_tmp:w \cs_new_protected_nopar:Npx \cs_gset_protected_nopar:Npx
1222 \cs_tmp:w \cs_new_protected:Npn      \cs_gset_protected:Npn
1223 \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:cpx`
`\cs_gset_nopar:cpn`
`\cs_gset_nopar:cpx`
`\cs_new_nopar:cpn`
`\cs_new_nopar:cpx`

`\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.

```

1224 \cs_set:Npn \cs_tmp:w #1#2
1225 { \cs_new_protected_nopar:Npn #1 { \exp_args:Nc #2 } }
1226 \cs_tmp:w \cs_set_nopar:cpn \cs_set_nopar:Npn
1227 \cs_tmp:w \cs_set_nopar:cpx \cs_set_nopar:Npx
1228 \cs_tmp:w \cs_gset_nopar:cpn \cs_gset_nopar:Npn
1229 \cs_tmp:w \cs_gset_nopar:cpx \cs_gset_nopar:Npx
1230 \cs_tmp:w \cs_new_nopar:cpn \cs_new_nopar:Npn
1231 \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.

```

\cs_gset:cpn 1232 \cs_tmp:w \cs_set:cpn \cs_set:Npn
\cs_gset:cpx 1233 \cs_tmp:w \cs_set:cpx \cs_set:Npx
\cs_new:cpn 1234 \cs_tmp:w \cs_gset:cpn \cs_gset:Npn
\cs_new:cpx 1235 \cs_tmp:w \cs_gset:cpx \cs_gset:Npx
1236 \cs_tmp:w \cs_new:cpn \cs_new:Npn
1237 \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
`\cs_set_protected_nopar:cpx` the first arguments. We may also do this globally.

```

\cs_gset_protected_nopar:cpn 1238 \cs_tmp:w \cs_set_protected_nopar:cpn \cs_set_protected_nopar:Npn
\cs_gset_protected_nopar:cpx 1239 \cs_tmp:w \cs_set_protected_nopar:cpx \cs_set_protected_nopar:Npx
\cs_new_protected_nopar:cpn 1240 \cs_tmp:w \cs_gset_protected_nopar:cpn \cs_gset_protected_nopar:Npn
\cs_new_protected_nopar:cpx 1241 \cs_tmp:w \cs_gset_protected_nopar:cpx \cs_gset_protected_nopar:Npx
1242 \cs_tmp:w \cs_new_protected_nopar:cpn \cs_new_protected_nopar:Npn
1243 \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
`\cs_set_protected:cpx` arguments. We may also do this globally.

```

\cs_gset_protected:cpn 1244 \cs_tmp:w \cs_set_protected:cpn \cs_set_protected:Npn
\cs_gset_protected:cpx 1245 \cs_tmp:w \cs_set_protected:cpx \cs_set_protected:Npx
\cs_new_protected:cpn 1246 \cs_tmp:w \cs_gset_protected:cpn \cs_gset_protected:Npn
\cs_new_protected:cpx 1247 \cs_tmp:w \cs_gset_protected:cpx \cs_gset_protected:Npx
1248 \cs_tmp:w \cs_new_protected:cpn \cs_new_protected:Npn
1249 \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`
`\cs_set_eq:Nc`
`\cs_set_eq:cc`

The = sign allows us to define funny char tokens like = itself or `_` with this function. For the definition of `\c_space_char{~}` to work we need the ~ after the =.

`\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”.

```
1250 \cs_new_protected:Npn \cs_set_eq:NN #1 { \tex_let:D #1 =~ }
1251 \cs_new_protected_nopar:Npn \cs_set_eq:cN { \exp_args:Nc \cs_set_eq:NN }
1252 \cs_new_protected_nopar:Npn \cs_set_eq:Nc { \exp_args:NNc \cs_set_eq:NN }
1253 \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`
`\cs_new_eq:Nc`
`\cs_new_eq:cc`

```
1254 \cs_new_protected:Npn \cs_new_eq:NN #1
1255 {
1256   \chk_if_free_cs:N #1
1257   \tex_global:D \cs_set_eq:NN #1
1258 }
1259 \cs_new_protected_nopar:Npn \cs_new_eq:cN { \exp_args:Nc \cs_new_eq:NN }
1260 \cs_new_protected_nopar:Npn \cs_new_eq:Nc { \exp_args:NNc \cs_new_eq:NN }
1261 \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`
`\cs_gset_eq:Nc`
`\cs_gset_eq:cc`

```
1262 \cs_new_protected_nopar:Npn \cs_gset_eq:NN { \tex_global:D \cs_set_eq:NN }
1263 \cs_new_protected_nopar:Npn \cs_gset_eq:Nc { \exp_args:NNc \cs_gset_eq:NN }
1264 \cs_new_protected_nopar:Npn \cs_gset_eq:cN { \exp_args:Nc \cs_gset_eq:NN }
1265 \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 Undefining functions

`\cs_undefine:N` The following function is used to free the main memory from the definition of some 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.
`\cs_undefine:c`

```
1266 \cs_new_protected:Npn \cs_undefine:N #1
1267 { \cs_gset_eq:NN #1 \c_undefined:D }
1268 \cs_new_protected:Npn \cs_undefine:c #1
1269 {
1270   \if_cs_exist:w #1 \cs_end:
1271   \exp_after:wN \use:n
1272   \else:
1273   \exp_after:wN \use_none:n
1274   \fi:
```

```

1275 { \cs_gset_eq:cN {#1} \c_undefined:D }
1276 }

```

(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
\cs_get_arg_count_from_signature_aux:nnN
\cs_get_arg_count_from_signature_auxii:w

```

Counting the number of tokens in the signature, i.e., the number of arguments the function should take. If there is no signature, we return that there is -1 arguments to signal 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.

```

1277 \cs_new:Npn \cs_get_arg_count_from_signature:N #1
1278 { \cs_split_function:NN #1 \cs_get_arg_count_from_signature_aux:nnN }
1279 \cs_new:Npn \cs_get_arg_count_from_signature_aux:nnN #1#2#3
1280 {
1281   \if_meaning:w \c_true_bool #3
1282     \exp_after:wN \use_i:nn
1283   \else:
1284     \exp_after:wN \use_ii:nn
1285   \fi:
1286   {
1287     \exp_after:wN \cs_get_arg_count_from_signature_auxii:w
1288     \use_none:nnnnnnnnn #2 9876543210 \q_stop
1289   }
1290   { -1 }
1291 }
1292 \cs_new:Npn \cs_get_arg_count_from_signature_auxii:w #1#2 \q_stop {#1}

```

A variant form we need right away.

```

1293 \cs_new_nopar:Npn \cs_get_arg_count_from_signature:c
1294 { \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:cNnn
\cs_generate_from_arg_count:Ncnn
\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

```

1295 \cs_new_protected:Npn \cs_generate_from_arg_count:NNnn #1#2#3#4
1296 {
1297   \if_case:w \int_eval:w #3 \int_eval_end:
1298     \cs_generate_from_arg_count_aux:nwn {}

```

```

1299 \or: \cs_generate_from_arg_count_aux:nwn {##1}
1300 \or: \cs_generate_from_arg_count_aux:nwn {##1##2}
1301 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3}
1302 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4}
1303 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5}
1304 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6}
1305 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7}
1306 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7##8}
1307 \or: \cs_generate_from_arg_count_aux:nwn {##1##2##3##4##5##6##7##8##9}
1308 \else:
1309 \cs_generate_from_arg_count_error_msg:Nn #1 {#3}
1310 \use_i:nnn
1311 \fi:
1312 {#2#1}
1313 {#4}
1314 }
1315 \cs_new_protected:Npn
1316 \cs_generate_from_arg_count_aux:nwn #1 #2 \fi: #3
1317 { \fi: #3 #1 }

```

A variant form we need right away, plus one which is used elsewhere but which is most logically created here.

```

1318 \cs_new_protected_nopar:Npn \cs_generate_from_arg_count:cNnn
1319 { \exp_args:Nc \cs_generate_from_arg_count:NNnn }
1320 \cs_new_protected_nopar:Npn \cs_generate_from_arg_count:Ncnn
1321 { \exp_args:NNc \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.

```

1322 \cs_new_protected:Npn \cs_generate_from_arg_count_error_msg:Nn #1#2
1323 {
1324 \msg_kernel_error:nnxx { kernel } { bad-number-of-arguments }
1325 { \token_to_str:N #1 } { \int_eval:n {#2} }
1326 }

```

(End definition for \cs_generate_from_arg_count:NNnn, \cs_generate_from_arg_count:cNnn, and \cs_generate_from_arg_count:Ncnn. These functions are 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

```

```

\cs_set_protected:Npn \cs_set:Nn #1#2
{
\cs_generate_from_arg_count:NNnn #1 \cs_set:Npn
{ \cs_get_arg_count_from_signature:N #1 } {#2}
}

```

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.

```

1327 \cs_set:Npn \cs_tmp:w #1#2#3
1328 {
1329   \cs_set_protected:cpx { cs_ #1 : #2 } ##1##2
1330   {
1331     \exp_not:N \cs_generate_from_arg_count:NNnn ##1
1332     \exp_after:wN \exp_not:N \cs:w cs_#1 : #3 \cs_end:
1333     { \exp_not:N \cs_get_arg_count_from_signature:N ##1 }{##2}
1334   }
1335 }

```

Then we define the 32 variants beginning with N.

```

1336 \cs_tmp:w { set } { Nn } { Npn }
1337 \cs_tmp:w { set } { Nx } { Npx }
1338 \cs_tmp:w { set_nopar } { Nn } { Npn }
1339 \cs_tmp:w { set_nopar } { Nx } { Npx }
1340 \cs_tmp:w { set_protected } { Nn } { Npn }
1341 \cs_tmp:w { set_protected } { Nx } { Npx }
1342 \cs_tmp:w { set_protected_nopar } { Nn } { Npn }
1343 \cs_tmp:w { set_protected_nopar } { Nx } { Npx }
1344 \cs_tmp:w { gset } { Nn } { Npn }
1345 \cs_tmp:w { gset } { Nx } { Npx }
1346 \cs_tmp:w { gset_nopar } { Nn } { Npn }
1347 \cs_tmp:w { gset_nopar } { Nx } { Npx }
1348 \cs_tmp:w { gset_protected } { Nn } { Npn }
1349 \cs_tmp:w { gset_protected } { Nx } { Npx }
1350 \cs_tmp:w { gset_protected_nopar } { Nn } { Npn }
1351 \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
1352 \cs_tmp:w { new } { Nn } { Npn }
1353 \cs_tmp:w { new } { Nx } { Npx }
1354 \cs_tmp:w { new_nopar } { Nn } { Npn }
1355 \cs_tmp:w { new_nopar } { Nx } { Npx }
1356 \cs_tmp:w { new_protected } { Nn } { Npn }
1357 \cs_tmp:w { new_protected } { Nx } { Npx }
1358 \cs_tmp:w { new_protected_nopar } { Nn } { Npn }
1359 \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}
}

```

```

1360 \cs_set:Npn \cs_tmp:w #1#2#3
1361 {
1362   \cs_set_protected:cpx {cs_#1:#2}##1##2{
1363     \exp_not:N\cs_generate_from_arg_count:cNnn {##1}
1364     \exp_after:wN \exp_not:N \cs:w cs_#1:#3 \cs_end:
1365     { \exp_not:N \cs_get_arg_count_from_signature:c {##1} } {##2}
1366   }
1367 }

```

\cs_set:cn The 32 c variants.

\cs_set:cn	1368	\cs_tmp:w { set }	{ cn } { Npn }
\cs_set:cx	1369	\cs_tmp:w { set }	{ cx } { Npx }
\cs_set_nopar:cn	1370	\cs_tmp:w { set_nopar }	{ cn } { Npn }
\cs_set_nopar:cx	1371	\cs_tmp:w { set_nopar }	{ cx } { Npx }
\cs_set_protected:cn	1372	\cs_tmp:w { set_protected }	{ cn } { Npn }
\cs_set_protected:cx	1373	\cs_tmp:w { set_protected }	{ cx } { Npx }
\cs_set_protected_nopar:cn	1374	\cs_tmp:w { set_protected_nopar }	{ cn } { Npn }
\cs_set_protected_nopar:cx	1375	\cs_tmp:w { set_protected_nopar }	{ cx } { Npx }
\cs_gset:cn	1376	\cs_tmp:w { gset }	{ cn } { Npn }
\cs_gset:cx	1377	\cs_tmp:w { gset }	{ cx } { Npx }
\cs_gset_nopar:cn	1378	\cs_tmp:w { gset_nopar }	{ cn } { Npn }
\cs_gset_nopar:cx	1379	\cs_tmp:w { gset_nopar }	{ cx } { Npx }
\cs_gset_protected:cn	1380	\cs_tmp:w { gset_protected }	{ cn } { Npn }
\cs_gset_protected:cx	1381	\cs_tmp:w { gset_protected }	{ cx } { Npx }
\cs_gset_protected_nopar:cn	1382	\cs_tmp:w { gset_protected_nopar }	{ cn } { Npn }
\cs_gset_protected_nopar:cx	1383	\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			
\cs_new_nopar:cn	1384	\cs_tmp:w { new }	{ cn } { Npn }
\cs_new_nopar:cx	1385	\cs_tmp:w { new }	{ cx } { Npx }
\cs_new_protected:cn	1386	\cs_tmp:w { new_nopar }	{ cn } { Npn }
\cs_new_protected:cx	1387	\cs_tmp:w { new_nopar }	{ cx } { Npx }
\cs_new_protected_nopar:cn	1388	\cs_tmp:w { new_protected }	{ cn } { Npn }
\cs_new_protected_nopar:cx	1389	\cs_tmp:w { new_protected }	{ cx } { Npx }
\cs_new_protected_nopar:cx	1390	\cs_tmp:w { new_protected_nopar }	{ cn } { Npn }
	1391	\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	1392	\prg_new_conditional:Npnn \cs_if_eq:NN #1#2 { p , T , F , TF }
\cs_if_eq:Nc	1393	{
\cs_if_eq:cc	1394	\if_meaning:w #1#2
	1395	\prg_return_true: \else: \prg_return_false: \fi:
	1396	}
	1397	\cs_new_nopar:Npn \cs_if_eq_p:cn { \exp_args:Nc \cs_if_eq_p:NN }
	1398	\cs_new_nopar:Npn \cs_if_eq:cNTF { \exp_args:Nc \cs_if_eq:NNTF }

```

1399 \cs_new_nopar:Npn \cs_if_eq:cNT { \exp_args:Nc \cs_if_eq:NNT }
1400 \cs_new_nopar:Npn \cs_if_eq:cNF { \exp_args:Nc \cs_if_eq:NNF }
1401 \cs_new_nopar:Npn \cs_if_eq_p:Nc { \exp_args:NNc \cs_if_eq_p:NN }
1402 \cs_new_nopar:Npn \cs_if_eq:NcTF { \exp_args:NNc \cs_if_eq:NNTF }
1403 \cs_new_nopar:Npn \cs_if_eq:NcT { \exp_args:NNc \cs_if_eq:NNT }
1404 \cs_new_nopar:Npn \cs_if_eq:NcF { \exp_args:NNc \cs_if_eq:NNF }
1405 \cs_new_nopar:Npn \cs_if_eq_p:cc { \exp_args:Ncc \cs_if_eq_p:NN }
1406 \cs_new_nopar:Npn \cs_if_eq:ccTF { \exp_args:Ncc \cs_if_eq:NNTF }
1407 \cs_new_nopar:Npn \cs_if_eq:ccT { \exp_args:Ncc \cs_if_eq:NNT }
1408 \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

```

```

1409 \cs_new:Npn \kernel_register_show:N #1
1410 {
1411   \cs_if_exist:NTF #1
1412   { \tex_showthe:D \use:n #1 }
1413   {
1414     \msg_kernel_error:nnx { kernel } { variable-not-defined }
1415     { \token_to_str:N #1 }
1416   }
1417 }
1418 \cs_new_nopar:Npn \kernel_register_show:c
1419 { \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 hard-coded for speed.

`\luatex_if_engine:`

`\pdftex_if_engine:`

```

1420 \cs_new_eq:NN \luatex_if_engine:T \use_none:n
1421 \cs_new_eq:NN \luatex_if_engine:F \use:n
1422 \cs_new_eq:NN \luatex_if_engine:TF \use_ii:nn
1423 \cs_new_eq:NN \pdftex_if_engine:T \use:n
1424 \cs_new_eq:NN \pdftex_if_engine:F \use_none:n
1425 \cs_new_eq:NN \pdftex_if_engine:TF \use_i:nn
1426 \cs_new_eq:NN \xetex_if_engine:T \use_none:n
1427 \cs_new_eq:NN \xetex_if_engine:F \use:n
1428 \cs_new_eq:NN \xetex_if_engine:TF \use_ii:nn
1429 \cs_new_eq:NN \luatex_if_engine_p: \c_false_bool
1430 \cs_new_eq:NN \pdftex_if_engine_p: \c_true_bool
1431 \cs_new_eq:NN \xetex_if_engine_p: \c_false_bool
1432 \cs_if_exist:NT \xetex_XeTeXversion:D
1433 {
1434   \cs_set_eq:NN \pdftex_if_engine:T \use_none:n
1435   \cs_set_eq:NN \pdftex_if_engine:F \use:n

```

```

1436 \cs_set_eq:NN \pdftex_if_engine:TF \use_ii:nn
1437 \cs_set_eq:NN \xetex_if_engine:T \use:n
1438 \cs_set_eq:NN \xetex_if_engine:F \use_none:n
1439 \cs_set_eq:NN \xetex_if_engine:TF \use_i:nn
1440 \cs_set_eq:NN \pdftex_if_engine_p: \c_false_bool
1441 \cs_set_eq:NN \xetex_if_engine_p: \c_true_bool
1442 }
1443 \cs_if_exist:NT \luatex_directlua:D
1444 {
1445 \cs_set_eq:NN \luatex_if_engine:T \use:n
1446 \cs_set_eq:NN \luatex_if_engine:F \use_none:n
1447 \cs_set_eq:NN \luatex_if_engine:TF \use_i:nn
1448 \cs_set_eq:NN \pdftex_if_engine:T \use_none:n
1449 \cs_set_eq:NN \pdftex_if_engine:F \use:n
1450 \cs_set_eq:NN \pdftex_if_engine:TF \use_ii:nn
1451 \cs_set_eq:NN \luatex_if_engine_p: \c_true_bool
1452 \cs_set_eq:NN \pdftex_if_engine_p: \c_false_bool
1453 }

```

(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!

```

1454 \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.

`\str_if_eq:xx`

```

1455 \prg_new_conditional:Npnn \str_if_eq:nn #1#2 { p , T , F , TF }
1456 {
1457 \if_int_compare:w \pdftex_strcmp:D { \exp_not:n {#1} } { \exp_not:n {#2} }
1458 = \c_zero
1459 \prg_return_true: \else: \prg_return_false: \fi:
1460 }
1461 \prg_new_conditional:Npnn \str_if_eq:xx #1#2 { p , T , F , TF }
1462 {
1463 \if_int_compare:w \pdftex_strcmp:D {#1} {#2} = \c_zero
1464 \prg_return_true: \else: \prg_return_false: \fi:
1465 }

```

(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).

```
1466 \cs_new_eq:NN \prg_break_point:n \use:n
1467 \cs_new:Npn \prg_map_break: #1 \prg_break_point:n #2 { #2 }
1468 \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.

```
1469 (*deprecated)
1470 \cs_new_eq:NN \cs_gnew_nopar:Npn \cs_new_nopar:Npn
1471 \cs_new_eq:NN \cs_gnew:Npn \cs_new:Npn
1472 \cs_new_eq:NN \cs_gnew_protected_nopar:Npn \cs_new_protected_nopar:Npn
1473 \cs_new_eq:NN \cs_gnew_protected:Npn \cs_new_protected:Npn
1474 \cs_new_eq:NN \cs_gnew_nopar:Npx \cs_new_nopar:Npx
1475 \cs_new_eq:NN \cs_gnew:Npx \cs_new:Npx
1476 \cs_new_eq:NN \cs_gnew_protected_nopar:Npx \cs_new_protected_nopar:Npx
1477 \cs_new_eq:NN \cs_gnew_protected:Npx \cs_new_protected:Npx
1478 \cs_new_eq:NN \cs_gnew_nopar:cpn \cs_new_nopar:cpn
1479 \cs_new_eq:NN \cs_gnew:cpn \cs_new:cpn
1480 \cs_new_eq:NN \cs_gnew_protected_nopar:cpn \cs_new_protected_nopar:cpn
1481 \cs_new_eq:NN \cs_gnew_protected:cpn \cs_new_protected:cpn
1482 \cs_new_eq:NN \cs_gnew_nopar:cpx \cs_new_nopar:cpx
1483 \cs_new_eq:NN \cs_gnew:cpx \cs_new:cpx
1484 \cs_new_eq:NN \cs_gnew_protected_nopar:cpx \cs_new_protected_nopar:cpx
1485 \cs_new_eq:NN \cs_gnew_protected:cpx \cs_new_protected:cpx
1486 /deprecated)

1487 (*deprecated)
1488 \cs_new_eq:NN \cs_gnew_eq:NN \cs_new_eq:NN
1489 \cs_new_eq:NN \cs_gnew_eq:cN \cs_new_eq:cN
1490 \cs_new_eq:NN \cs_gnew_eq:Nc \cs_new_eq:Nc
1491 \cs_new_eq:NN \cs_gnew_eq:cc \cs_new_eq:cc
1492 /deprecated)

1493 (*deprecated)
1494 \cs_new_eq:NN \cs_gundefine:N \cs_undefine:N
1495 \cs_new_eq:NN \cs_gundefine:c \cs_undefine:c
1496 /deprecated)

1497 (*deprecated)
1498 \cs_new_eq:NN \group_execute_after:N \group_insert_after:N
1499 /deprecated)
```

Deprecated 2011-09-06, for removal by 2011-12-31.

`\c_pdfTeX_is_engine_bool` Predicates are better
`\c_luatex_is_engine_bool` 1500 \langle *deprecated \rangle
`\c_xetex_is_engine_bool` 1501 \backslash cs_new_eq:NN \backslash c_luatex_is_engine_bool \backslash luatex_if_engine_p:
1502 \backslash cs_new_eq:NN \backslash c_pdfTeX_is_engine_bool \backslash pdfTeX_if_engine_p:
1503 \backslash cs_new_eq:NN \backslash c_xetex_is_engine_bool \backslash xetex_if_engine_p:
1504 \langle /deprecated \rangle
(End definition for \backslash c_pdfTeX_is_engine_bool, \backslash c_luatex_is_engine_bool, and \backslash c_xetex_is_engine_bool.
These functions are documented on page ??.)

\backslash use_i_after_fi:nw These functions return the first argument after ending the conditional. This is rather
 \backslash use_i_after_else:nw specialized, and we want to de-emphasize the use of primitive T_EX conditionals.
 \backslash use_i_after_or:nw 1505 \langle *deprecated \rangle
 \backslash use_i_after_orelse:nw 1506 \backslash cs_set:Npn \backslash use_i_after_fi:nw #1 \backslash fi: { \backslash fi: #1 }
1507 \backslash cs_set:Npn \backslash use_i_after_else:nw #1 \backslash else: #2 \backslash fi: { \backslash fi: #1 }
1508 \backslash cs_set:Npn \backslash use_i_after_or:nw #1 \backslash or: #2 \backslash fi: { \backslash fi: #1 }
1509 \backslash cs_set:Npn \backslash use_i_after_orelse:nw #1#2#3 \backslash fi: { \backslash fi: #1 }
1510 \langle /deprecated \rangle
(End definition for \backslash use_i_after_fi:nw. This function is documented on page ??.)
Deprecated 2011-09-07, for removal by 2011-12-31.

\backslash cs_set_eq:NwN
1511 \langle *deprecated \rangle
1512 \backslash tex_let:D \backslash cs_set_eq:NwN \backslash tex_let:D
1513 \langle /deprecated \rangle
(End definition for \backslash cs_set_eq:NwN. This function is documented on page ??.)
1514 \langle /initex | package \rangle

186 l3expan implementation

1515 \langle *initex | package \rangle
We start by ensuring that the required packages are loaded.
1516 \langle *package \rangle
1517 \backslash ProvidesExplPackage
1518 { \backslash ExplFileName}{ \backslash ExplFileDate}{ \backslash ExplFileVersion}{ \backslash ExplFileDescription}
1519 \backslash package_check_loaded_expl:
1520 \langle /package \rangle
 \backslash exp_after:wN These are defined in l3basics.
 \backslash exp_not:N *(End definition for \backslash exp_after:wN. This function is documented on page 31.)*
 \backslash exp_not:n

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_internal_tl` We need a scratch token list variable. We don't use `tl` methods so that `l3expan` can be loaded earlier.

```
1521 \cs_new_nopar:Npn \l_exp_internal_tl { }
      (End definition for \l_exp_internal_tl. This function is documented on page 32.)
```

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` `#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.

```
1522 \cs_new:Npn \exp_arg_next:nnn #1#2#3 { #2 \::: { #3 {#1} } }
1523 \cs_new:Npn \exp_arg_next: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.

```
1524 \cs_new:Npn \::: #1 {#1}
      (End definition for \:::. This function is documented on page 32.)
```

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

```
1525 \cs_new:Npn \::n #1 \::: #2#3 { #1 \::: { #2 {#3} } }
      (End definition for \::n. This function is documented on page 32.)
```

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

```
1526 \cs_new:Npn \::N #1 \::: #2#3 { #1 \::: {#2#3} }
      (End definition for \::N. This function is documented on page 32.)
```

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.

```
1527 \cs_new:Npn \::c #1 \::: #2#3
1528 { \exp_after:wN \exp_arg_next:nnn \cs:w #3 \cs_end: {#1} {#2} }
(End definition for \::c. This function is documented on page 32.)
```

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

```
1529 \cs_new:Npn \::o #1 \::: #2#3
1530 { \exp_after:wN \exp_arg_next:nnn \exp_after:wN {#3} {#1} {#2} }
(End definition for \::o. This function is documented on page 32.)
```

`\::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.

```
1531 \cs_new:Npn \::f #1 \::: #2#3
1532 {
1533   \exp_after:wN \exp_arg_next:nnn
1534   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1535   {#1} {#2}
1536 }
1537 \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.

```
1538 \cs_new_protected:Npn \::x #1 \::: #2#3
1539 {
1540   \cs_set_nopar:Npx \l_exp_internal_tl { {#3} }
1541   \exp_after:wN \exp_arg_next:nnn \l_exp_internal_tl {#1} {#2}
1542 }
(End definition for \::x. This function is documented on page 32.)
```

`\::v` These functions return the value of a register, i.e., one of `tl`, `clist`, `int`, `skip`, `dim` and `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.

```
1543 \cs_new:Npn \::V #1 \::: #2#3
1544 {
```

```

1545     \exp_after:wN \exp_arg_next:nnn
1546     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1547     {#1} {#2}
1548 }
1549 \cs_new:Npn \::v # 1\::: #2#3
1550 {
1551     \exp_after:wN \exp_arg_next:nnn
1552     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#3} }
1553     {#1} {#2}
1554 }

```

(End definition for \::v. This function is documented on page 32.)

`\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:`.

```

1555 \cs_new:Npn \exp_eval_register:N #1
1556 {
1557     \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.

```

1558     \if_meaning:w \scan_stop: #1
1559     \exp_eval_error_msg:w
1560     \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`.

```

1561     \else:
1562     \exp_after:wN \use_i_ii:nnn
1563     \fi:
1564     \exp_after:wN \c_zero \tex_the:D #1
1565 }
1566 \cs_new:Npn \exp_eval_register:c #1
1567 { \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}

1568 \cs_new:Npn \exp_eval_error_msg:w #1 \tex_the:D #2
1569 {
1570   \fi:
1571   \fi:
1572   \msg_expandable_kernel_error:nnn { kernel } { bad-var } {#2}
1573   \c_zero
1574 }

(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.

<code>\exp_args:No</code> <code>\exp_args:NNo</code> <code>\exp_args:NNNo</code>	<p>Those lovely runs of expansion!</p> <pre> 1575 \cs_new:Npn \exp_args:No #1#2 { \exp_after:wN #1 \exp_after:wN {#2} } 1576 \cs_new:Npn \exp_args:NNo #1#2#3 1577 { \exp_after:wN #1 \exp_after:wN #2 \exp_after:wN {#3} } 1578 \cs_new:Npn \exp_args:NNNo #1#2#3#4 1579 { \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 29.) </pre>
<code>\exp_args:Nc</code>	<p>In <code>l3basics</code></p> <p>(End definition for <code>\exp_args:Nc</code>. This function is documented on page 27.)</p>
<code>\exp_args:cc</code> <code>\exp_args:NNc</code> <code>\exp_args:Ncc</code> <code>\exp_args:Nccc</code>	<p>Here are the functions that turn their argument into csnames but are expandable.</p> <pre> 1580 \cs_new:Npn \exp_args:cc #1#2 1581 { \cs:w #1 \exp_after:wN \cs_end: \cs:w #2 \cs_end: } 1582 \cs_new:Npn \exp_args:NNc #1#2#3 1583 { \exp_after:wN #1 \exp_after:wN #2 \cs:w # 3\cs_end: } 1584 \cs_new:Npn \exp_args:Ncc #1#2#3 1585 { \exp_after:wN #1 \cs:w #2 \exp_after:wN \cs_end: \cs:w #3 \cs_end: } 1586 \cs_new:Npn \exp_args:Nccc #1#2#3#4 1587 { 1588 \exp_after:wN #1 1589 \cs:w #2 \exp_after:wN \cs_end: 1590 \cs:w #3 \exp_after:wN \cs_end: 1591 \cs:w #4 \cs_end: 1592 } (End definition for \exp_args:cc and others. These functions are documented on page ??.) </pre>

```

\exp_args:Nf
\exp_args:NV
\exp_args:Nv
1593 \cs_new:Npn \exp_args:Nf #1#2
1594 { \exp_after:wN #1 \exp_after:wN { \tex_romannumeral:D -'0 #2 } }
1595 \cs_new:Npn \exp_args:Nv #1#2
1596 {
1597   \exp_after:wN #1 \exp_after:wN
1598   { \tex_romannumeral:D \exp_eval_register:c {#2} }
1599 }
1600 \cs_new:Npn \exp_args:NV #1#2
1601 {
1602   \exp_after:wN #1 \exp_after:wN
1603   { \tex_romannumeral:D \exp_eval_register:N #2 }
1604 }

```

(End definition for `\exp_args:Nf`, `\exp_args:NV`, and `\exp_args:Nv`. These functions are documented on page 28.)

`\exp_args:NNV` Some more hand-tuned function with three arguments. If we forced that an `o` argument
`\exp_args:NNv` always has braces, we could implement `\exp_args:Nco` with less tokens and only two
`\exp_args:NNf` arguments.

```

\exp_args:NNV
\exp_args:Ncf
\exp_args:Nco
1605 \cs_new:Npn \exp_args:NNf #1#2#3
1606 {
1607   \exp_after:wN #1
1608   \exp_after:wN #2
1609   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1610 }
1611 \cs_new:Npn \exp_args:NNv #1#2#3
1612 {
1613   \exp_after:wN #1
1614   \exp_after:wN #2
1615   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#3} }
1616 }
1617 \cs_new:Npn \exp_args:NNV #1#2#3
1618 {
1619   \exp_after:wN #1
1620   \exp_after:wN #2
1621   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1622 }
1623 \cs_new:Npn \exp_args:Nco #1#2#3
1624 {
1625   \exp_after:wN #1
1626   \cs:w #2 \exp_after:wN \cs_end:
1627   \exp_after:wN {#3}
1628 }
1629 \cs_new:Npn \exp_args:Ncf #1#2#3
1630 {
1631   \exp_after:wN #1
1632   \cs:w #2 \exp_after:wN \cs_end:
1633   \exp_after:wN { \tex_romannumeral:D -'0 #3 }
1634 }

```

```

1635 \cs_new:Npn \exp_args:NVV #1#2#3
1636 {
1637   \exp_after:wN #1
1638   \exp_after:wN { \tex_romannumeral:D \exp_after:wN
1639     \exp_eval_register:N \exp_after:wN #2 \exp_after:wN }
1640   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #3 }
1641 }

```

(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 1642 \cs_new:Npn \exp_args:NNNV #1#2#3#4
\exp_args:NcNo 1643 {
\exp_args:NNNV 1644   \exp_after:wN #1
1645   \exp_after:wN #2
1646   \exp_after:wN #3
1647   \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #4 }
1648 }
1649 \cs_new:Npn \exp_args:NcNc #1#2#3#4
1650 {
1651   \exp_after:wN #1
1652   \cs:w #2 \exp_after:wN \cs_end:
1653   \exp_after:wN #3
1654   \cs:w #4 \cs_end:
1655 }
1656 \cs_new:Npn \exp_args:NcNo #1#2#3#4
1657 {
1658   \exp_after:wN #1
1659   \cs:w #2 \exp_after:wN \cs_end:
1660   \exp_after:wN #3
1661   \exp_after:wN {#4}
1662 }
1663 \cs_new:Npn \exp_args:Ncco #1#2#3#4
1664 {
1665   \exp_after:wN #1
1666   \cs:w #2 \exp_after:wN \cs_end:
1667   \cs:w #3 \exp_after:wN \cs_end:
1668   \exp_after:wN {#4}
1669 }

```

(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`

```

1670 \cs_new_protected_nopar:Npn \exp_args:Nx { \::x \::: }

```

(End definition for `\exp_args:Nx`. This function is documented on page 28.)

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

```

\exp_args:Nfo 1671 \cs_new_nopar:Npn \exp_args:Nnc { \::n \::c \::: }
\exp_args:Nff 1672 \cs_new_nopar:Npn \exp_args:Nfo { \::f \::o \::: }
\exp_args:Nnf 1673 \cs_new_nopar:Npn \exp_args:Nff { \::f \::f \::: }
\exp_args:Nno 1674 \cs_new_nopar:Npn \exp_args:Nnf { \::n \::f \::: }
\exp_args:NnV 1675 \cs_new_nopar:Npn \exp_args:Nno { \::n \::o \::: }
\exp_args:Noo 1676 \cs_new_nopar:Npn \exp_args:NnV { \::n \::V \::: }
\exp_args:Nof 1677 \cs_new_nopar:Npn \exp_args:Noo { \::o \::o \::: }
\exp_args:Noc 1678 \cs_new_nopar:Npn \exp_args:Nof { \::o \::f \::: }
\exp_args:Ncx 1679 \cs_new_nopar:Npn \exp_args:Noc { \::o \::c \::: }
\exp_args:NNx 1680 \cs_new_protected_nopar:Npn \exp_args:NNx { \::N \::x \::: }
\exp_args:Ncx 1681 \cs_new_protected_nopar:Npn \exp_args:Ncx { \::c \::x \::: }
\exp_args:Nnx 1682 \cs_new_protected_nopar:Npn \exp_args:Nnx { \::n \::x \::: }
\exp_args:Nox 1683 \cs_new_protected_nopar:Npn \exp_args:Nox { \::o \::x \::: }
\exp_args:Nxo 1684 \cs_new_protected_nopar:Npn \exp_args:Nxo { \::x \::o \::: }
\exp_args:Nxx 1685 \cs_new_protected_nopar:Npn \exp_args:Nxx { \::x \::x \::: }

```

(End definition for `\exp_args:Nnc` and others. These functions are documented on page ??.)

```

\exp_args:NNno
\exp_args:NNoo 1686 \cs_new_nopar:Npn \exp_args:NNno { \::N \::n \::o \::: }
\exp_args:NNnc 1687 \cs_new_nopar:Npn \exp_args:NNoo { \::N \::o \::o \::: }
\exp_args:NNno 1688 \cs_new_nopar:Npn \exp_args:NNnc { \::n \::n \::c \::: }
\exp_args:Nooo 1689 \cs_new_nopar:Npn \exp_args:NNno { \::n \::n \::o \::: }
\exp_args:NNnx 1690 \cs_new_nopar:Npn \exp_args:Nooo { \::o \::o \::o \::: }
\exp_args:NNox 1691 \cs_new_protected_nopar:Npn \exp_args:NNnx { \::N \::n \::x \::: }
\exp_args:Nnnx 1692 \cs_new_protected_nopar:Npn \exp_args:NNox { \::N \::o \::x \::: }
\exp_args:Nnox 1693 \cs_new_protected_nopar:Npn \exp_args:Nnnx { \::n \::n \::x \::: }
\exp_args:Nccx 1694 \cs_new_protected_nopar:Npn \exp_args:Nnox { \::n \::o \::x \::: }
\exp_args:Ncnx 1695 \cs_new_protected_nopar:Npn \exp_args:Nccx { \::c \::c \::x \::: }
\exp_args:Noox 1696 \cs_new_protected_nopar:Npn \exp_args:Ncnx { \::c \::n \::x \::: }
1697 \cs_new_protected_nopar:Npn \exp_args:Noox { \::o \::o \::x \::: }

```

(End definition for `\exp_args:NNno` 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 1698 \cs_new:Npn \exp_arg_last_unbraced:nn #1#2 { #2#1 }
\::V_unbraced 1699 \cs_new:Npn \::f_unbraced \::: #1#2
\::v_unbraced 1700 {
\::x_unbraced 1701 \exp_after:wN \exp_arg_last_unbraced:nn
1702 \exp_after:wN { \tex_romannumeral:D -'0 #2 } {#1}
1703 }
1704 \cs_new:Npn \::o_unbraced \::: #1#2
1705 { \exp_after:wN \exp_arg_last_unbraced:nn \exp_after:wN {#2} {#1} }
1706 \cs_new:Npn \::V_unbraced \::: #1#2
1707 {
1708 \exp_after:wN \exp_arg_last_unbraced:nn

```

```

1709     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:N #2 } {#1}
1710   }
1711   \cs_new:Npn \::v_unbraced \::: #1#2
1712   {
1713     \exp_after:wN \exp_arg_last_unbraced:nn
1714     \exp_after:wN { \tex_romannumeral:D \exp_eval_register:c {#2} } {#1}
1715   }
1716   \cs_new_protected:Npn \::x_unbraced \::: #1#2
1717   {
1718     \cs_set_nopar:Npx \l_exp_internal_tl { \exp_not:n {#1} #2 }
1719     \l_exp_internal_tl
1720   }

```

(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:Nc
\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
1721 \cs_new:Npn \exp_last_unbraced:NV #1#2
1722 { \exp_after:wN #1 \tex_romannumeral:D \exp_eval_register:N #2 }
1723 \cs_new:Npn \exp_last_unbraced:Nv #1#2
1724 { \exp_after:wN #1 \tex_romannumeral:D \exp_eval_register:c {#2} }
1725 \cs_new:Npn \exp_last_unbraced:Nc #1#2 { \exp_after:wN #1 #2 }
1726 \cs_new:Npn \exp_last_unbraced:Nf #1#2
1727 { \exp_after:wN #1 \tex_romannumeral:D -‘0 #2 }
1728 \cs_new:Npn \exp_last_unbraced:NcV #1#2#3
1729 { \exp_after:wN #1 \cs:w #2 \exp_after:wN \cs_end: #3 }
1730 \cs_new:Npn \exp_last_unbraced:NcV #1#2#3
1731 {
1732   \exp_after:wN #1
1733   \cs:w #2 \exp_after:wN \cs_end:
1734   \tex_romannumeral:D \exp_eval_register:N #3
1735 }
1736 \cs_new:Npn \exp_last_unbraced:NNV #1#2#3
1737 {
1738   \exp_after:wN #1
1739   \exp_after:wN #2
1740   \tex_romannumeral:D \exp_eval_register:N #3
1741 }
1742 \cs_new:Npn \exp_last_unbraced:NNo #1#2#3
1743 { \exp_after:wN #1 \exp_after:wN #2 #3 }
1744 \cs_new:Npn \exp_last_unbraced:NNNV #1#2#3#4
1745 {
1746   \exp_after:wN #1
1747   \exp_after:wN #2
1748   \exp_after:wN #3
1749   \tex_romannumeral:D \exp_eval_register:N #4
1750 }
1751 \cs_new:Npn \exp_last_unbraced:NNNo #1#2#3#4
1752 { \exp_after:wN #1 \exp_after:wN #2 \exp_after:wN #3 #4 }
1753 \cs_new_nopar:Npn \exp_last_unbraced:Nno { \::n \::o_unbraced \::: }

```

```

1754 \cs_new_nopar:Npn \exp_last_unbraced:Noo { \::o \::o_unbraced \::: }
1755 \cs_new_nopar:Npn \exp_last_unbraced:Nfo { \::f \::o_unbraced \::: }
1756 \cs_new_protected_nopar:Npn \exp_last_unbraced:Nx { \::x_unbraced \::: }
      (End definition for \exp_last_unbraced:NV. This function is documented on page 30.)

```

`\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.

```

1757 \cs_new:Npn \exp_last_two_unbraced:Noo #1#2#3
1758 { \exp_after:wN \exp_last_two_unbraced_aux:noN \exp_after:wN {#3} {#2} #1 }
1759 \cs_new:Npn \exp_last_two_unbraced_aux:noN #1#2#3
1760 { \exp_after:wN #3 #2 #1 }
      (End definition for \exp_last_two_unbraced:Noo. This function is documented on page 30.)

```

186.5 Preventing expansion

```

\exp_not:o
\exp_not:c
\exp_not:f
\exp_not:V
\exp_not:v
1761 \cs_new:Npn \exp_not:o #1 { \etex_unexpanded:D \exp_after:wN {#1} }
1762 \cs_new:Npn \exp_not:c #1 { \exp_after:wN \exp_not:N \cs:w #1 \cs_end: }
1763 \cs_new:Npn \exp_not:f #1
1764 { \etex_unexpanded:D \exp_after:wN { \tex_romannumeral:D -'0 #1 } }
1765 \cs_new:Npn \exp_not:V #1
1766 {
1767   \etex_unexpanded:D \exp_after:wN
1768   { \tex_romannumeral:D \exp_eval_register:N #1 }
1769 }
1770 \cs_new:Npn \exp_not:v #1
1771 {
1772   \etex_unexpanded:D \exp_after:wN
1773   { \tex_romannumeral:D \exp_eval_register:c {#1} }
1774 }
      (End definition for \exp_not:o. This function is documented on page 31.)

```

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
\cs_generate_variant_aux:NNn

```

Test whether the base function is protected or not and define `\cs_tmp:w` as either `\cs_new_nopar:Npx` or `\cs_new_protected_nopar:Npx`, then used to define all the variants. 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`.

```

1775 \cs_new_protected:Npn \cs_generate_variant:Nn #1
1776 {
1777   \chk_if_exist_cs:N #1
1778   \cs_generate_variant_aux:N #1
1779   \cs_split_function:NN #1 \cs_generate_variant_aux:nnNNn
1780   #1
1781 }

```

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.

```

1782 \cs_new_protected:Npn \cs_generate_variant_aux:nnNNn #1#2#3#4#5
1783 { \cs_generate_variant_aux:Nnnw #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.

```

1784 \cs_new_protected:Npn \cs_generate_variant_aux:Nnnw #1#2#3#4 ,
1785 {
1786   \if:w ? #4
1787   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
1788   \fi:
1789   \exp_args:Nnc \cs_generate_variant_aux:NNn
1790   #1
1791   {
1792     #2 : #4
1793     \exp_after:wN \use_i_delimit_by_q_stop:nw
1794     \use_none:nnnnnnnnn #4
1795     \use_none:nnnnnnnnn
1796     \use_none:nnnnnnnnn
1797     \use_none:nnnnnnnn
1798     \use_none:nnnnnnn
1799     \use_none:nnnnn
1800     \use_none:nnnn
1801     \use_none:nnn
1802     \use_none:nn
1803     \use_none:n
1804     { }
1805     \q_stop
1806     #3
1807   }
1808   {#4}

```

```

1809     \cs_generate_variant_aux:Nnnw #1 {#2} {#3}
1810 }

```

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.

```

1811 \cs_new_protected:Npn \cs_generate_variant_aux:NNn #1 #2 #3
1812 {
1813   \cs_if_free:NTF #2
1814   {
1815     \cs_tmp:w #2 { \exp_not:c { exp_args:N #3 } \exp_not:N #1 }
1816     \cs_generate_internal_variant:n {#3}
1817   }
1818   {
1819     \iow_log:x
1820     {
1821       Variant~\token_to_str:N #2~%
1822       already~defined;~ not~ changing~ it~on~line~%
1823       \tex_the:D \tex_inputlineno:D
1824     }
1825   }
1826 }

```

(End definition for \cs_generate_variant:Nn. This function is documented on page ??.)

```

\cs_generate_variant_aux:N
\cs_generate_variant_aux:w

```

The idea here is to pick up protected parent functions, using the nature of the meaning 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.

```

1827 \group_begin:
1828   \tex_lccode:D '\Z = '\d \scan_stop:
1829   \tex_lccode:D '\? = '\ \ \scan_stop:
1830   \tex_catcode:D '\P = 12 \scan_stop:
1831   \tex_catcode:D '\R = 12 \scan_stop:
1832   \tex_catcode:D '\O = 12 \scan_stop:
1833   \tex_catcode:D '\T = 12 \scan_stop:
1834   \tex_catcode:D '\E = 12 \scan_stop:
1835   \tex_catcode:D '\C = 12 \scan_stop:
1836   \tex_catcode:D '\Z = 12 \scan_stop:
1837 \tex_lowercase:D
1838 {
1839   \group_end:
1840   \cs_new_protected:Npn \cs_generate_variant_aux:N #1
1841   {
1842     \exp_after:wN \cs_generate_variant_aux:w
1843     \token_to_meaning:N #1
1844     \q_mark \cs_new_protected_nopar:Npx
1845     ? PROTECTEZ
1846     \q_mark \cs_new_nopar:Npx
1847     \q_stop

```

```

1848     }
1849     \cs_new_protected:Npn \cs_generate_variant_aux:w
1850       #1 ? PROTECTEZ #2 \q_mark #3 #4 \q_stop
1851     {
1852       \cs_set_eq:NN \cs_tmp:w #3
1853     }
1854   }

```

(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`

```

1855 \cs_new_protected:Npn \cs_generate_internal_variant:n #1
1856 {
1857   \cs_if_free:cT { exp_args:N #1 }
1858   {
1859     \cs_new:cpx { exp_args:N #1 }
1860     { \cs_generate_internal_variant_aux:N #1 : }
1861   }
1862 }

```

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.

```

1863 \cs_new:Npn \cs_generate_internal_variant_aux:N #1
1864 {
1865   \exp_not:c { :: #1 }
1866   \if_meaning:w : #1
1867     \exp_after:wN \use_none:n
1868   \fi:
1869   \cs_generate_internal_variant_aux:N
1870 }

```

(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.

```

1871 \cs_generate_variant:Nn \str_if_eq_p:nn { V , o }
1872 \cs_generate_variant:Nn \str_if_eq_p:nn { nV , no , VV }
1873 \cs_generate_variant:Nn \str_if_eq:nnT { V , o }
1874 \cs_generate_variant:Nn \str_if_eq:nnT { nV , no , VV }
1875 \cs_generate_variant:Nn \str_if_eq:nnF { V , o }
1876 \cs_generate_variant:Nn \str_if_eq:nnF { nV , no , VV }
1877 \cs_generate_variant:Nn \str_if_eq:nnTF { V , o }
1878 \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 ??.)

```

1879 </initex | package>

```

187 l3prg implementation

The following test files are used for this code: *m3prg001.lvt, m3prg002.lvt, m3prg003.lvt.*

```

1880 <*initex | package>

1881 <*package>
1882 \ProvidesExplPackage
1883   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
1884 \package_check_loaded_expl:
1885 </package>

```

187.1 Primitive conditionals

`\if_bool:N` Those two primitive TeX conditionals are synonyms. They should not be used outside the kernel code.

```

1886 \tex_let:D \if_bool:N          \tex_ifodd:D
1887 \tex_let:D \if_predicate:w     \tex_ifodd:D

```

(End definition for `\if_bool:N`. This function is documented on page 42.)

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.

```

1888 \cs_new_protected:Npn \bool_new:N #1 { \cs_new_eq:NN #1 \c_false_bool }
1889 \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.

```

1890 \cs_new_protected:Npn \bool_set_true:N #1
1891   { \cs_set_eq:NN #1 \c_true_bool }
1892 \cs_new_protected:Npn \bool_set_false:N #1
1893   { \cs_set_eq:NN #1 \c_false_bool }
1894 \cs_new_protected:Npn \bool_gset_true:N #1
1895   { \cs_gset_eq:NN #1 \c_true_bool }
1896 \cs_new_protected:Npn \bool_gset_false:N #1
1897   { \cs_gset_eq:NN #1 \c_false_bool }
1898 \cs_generate_variant:Nn \bool_set_true:N { c }
1899 \cs_generate_variant:Nn \bool_set_false:N { c }
1900 \cs_generate_variant:Nn \bool_gset_true:N { c }
1901 \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 1902 \cs_new_eq:NN \bool_set_eq:NN \cs_set_eq:NN
\bool_set_eq:Nc 1903 \cs_new_eq:NN \bool_set_eq:Nc \cs_set_eq:Nc
\bool_set_eq:cc 1904 \cs_new_eq:NN \bool_set_eq:cN \cs_set_eq:cN
\bool_gset_eq:NN 1905 \cs_new_eq:NN \bool_set_eq:cc \cs_set_eq:cc
\bool_gset_eq:cN 1906 \cs_new_eq:NN \bool_gset_eq:NN \cs_gset_eq:NN
\bool_gset_eq:Nc 1907 \cs_new_eq:NN \bool_gset_eq:Nc \cs_gset_eq:Nc
\bool_gset_eq:cN 1908 \cs_new_eq:NN \bool_gset_eq:cN \cs_gset_eq:cN
\bool_gset_eq:cc 1909 \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 1910 \cs_new_protected:Npn \bool_set:Nn #1#2
\bool_gset:cn 1911 { \tex_chardef:D #1 = \bool_if_p:n {#2} }
1912 \cs_new_protected:Npn \bool_gset:Nn #1#2
1913 { \tex_global:D \tex_chardef:D #1 = \bool_if_p:n {#2} }
1914 \cs_generate_variant:Nn \bool_set:Nn { c }
1915 \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.

```

1916 \prg_new_conditional:Npnn \bool_if:N #1 { p , T , F , TF }
1917 {
1918   \if_meaning:w \c_true_bool #1
1919   \prg_return_true:
1920   \else:
1921   \prg_return_false:
1922   \fi:
1923 }
1924 \cs_generate_variant:Nn \bool_if_p:N { c }
1925 \cs_generate_variant:Nn \bool_if:NT { c }
1926 \cs_generate_variant:Nn \bool_if:NF { c }
1927 \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`

```

1928 \bool_new:N \l_tmpa_bool
1929 \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.

```

1930 \prg_new_conditional:Npnn \bool_if:n #1 { T , F , TF }
1931 {
1932   \if_predicate:w \bool_if_p:n {#1}
1933   \prg_return_true:
1934   \else:
1935     \prg_return_false:
1936   \fi:
1937 }
1938 \cs_new:Npn \bool_if_p:n #1
1939 {
1940   \group_align_safe_begin:
1941   \bool_get_next:N ( #1 ) S
1942 }

```

The GetNext operation. We make it a switch: If not a ! or (, we assume it is a predicate.

```

1943 \cs_new:Npn \bool_get_next:N #1
1944 {
1945   \use:c
1946   {
1947     bool_
1948     \if_meaning:w !#1 ! \else: \if_meaning:w (#1 ( \else: p \fi: \fi:
1949     :w
1950   }
1951   #1
1952 }

```

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.

```

1953 \cs_new:Npn \bool_get_not_next:N #1
1954 {
1955   \use:c
1956   {
1957     bool_not_
1958     \if_meaning:w !#1 ! \else: \if_meaning:w (#1 ( \else: p \fi: \fi:
1959     :w
1960   }
1961   #1
1962 }

```

We need these later on to nullify the unity operation !!.

```

1963 \cs_new:Npn \bool_get_next:NN #1#2 { \bool_get_next:N #2 }
1964 \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.

```

1965 \cs_new:cpn { bool_!:w } #1#2
1966 {
1967   \if_meaning:w ( #2
1968     \exp_after:wN \bool_Not:w

```

```

1969     \else:
1970         \if_meaning:w ! #2
1971         \exp_after:wN \exp_after:wN \exp_after:wN \bool_get_next:NN
1972     \else:
1973         \exp_after:wN \exp_after:wN \exp_after:wN \bool_Not:N
1974     \fi:
1975 \fi:
1976 #2
1977 }

```

Variant called when already inside a Not. Essentially the opposite of the above.

```

1978 \cs_new:cpn { bool_not_!:w } #1#2
1979 {
1980     \if_meaning:w ( #2
1981     \exp_after:wN \bool_not_Not:w
1982 \else:
1983     \if_meaning:w ! #2
1984     \exp_after:wN \exp_after:wN \exp_after:wN \bool_get_not_next:NN
1985 \else:
1986     \exp_after:wN \exp_after:wN \exp_after:wN \bool_not_Not:N
1987 \fi:
1988 \fi:
1989 #2
1990 }

```

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.

```

1991 \cs_new:Npn \bool_Not:w { \exp_after:wN \int_value:w \bool_get_not_next:N }
1992 \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.

```

1993 \cs_new:Npn \bool_Not:N #1
1994 {
1995     \exp_after:wN \bool_p:w
1996     \if_meaning:w #1 \c_true_bool
1997     \c_false_bool
1998 \else:
1999     \c_true_bool
2000 \fi:
2001 }
2002 \cs_new:Npn \bool_not_Not:N #1
2003 {
2004     \exp_after:wN \bool_p:w
2005     \if_meaning:w #1 \c_true_bool
2006     \c_true_bool
2007 \else:
2008     \c_false_bool
2009 \fi:
2010 }

```

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 `!(...)`.

```
2011 \cs_new:cpn { bool_( :w } #1
2012 { \exp_after:wN \bool_cleanup:N \int_value:w \bool_get_next:N }
2013 \cs_new:cpn { bool_not_( :w } #1
2014 { \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.

```
2015 \cs_new:cpn { bool_p:w } { \exp_after:wN \bool_cleanup:N \int_value:w }
2016 \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.

```
2017 \cs_new:Npn \bool_cleanup:N #1
2018 {
2019   \exp_after:wN \bool_choose:NN \exp_after:wN #1
2020   \int_to_roman:w - '\q
2021 }
2022 \cs_new:Npn \bool_not_cleanup:N #1
2023 {
2024   \exp_after:wN \bool_not_choose:NN \exp_after:wN #1
2025   \int_to_roman:w - '\q
2026 }
```

Branching the six way switch. Reversals should be reasonably straightforward.

```
2027 \cs_new:Npn \bool_choose:NN #1#2 { \use:c { bool_ #2 _ #1 :w } }
2028 \cs_new:Npn \bool_not_choose:NN #1#2 { \use:c { bool_not_ #2 _ #1 :w } }
```

Continues scanning. Must remove the second `&` or `|`.

```
2029 \cs_new_nopar:cpn { bool_&_1:w } & { \bool_get_next:N }
2030 \cs_new_nopar:cpn { bool_|_0:w } | { \bool_get_next:N }
2031 \cs_new_nopar:cpn { bool_not_&_0:w } & { \bool_get_next:N }
2032 \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.

```
2033 \cs_new_nopar:cpn { bool_)_0:w } { \c_false_bool }
2034 \cs_new_nopar:cpn { bool_)_1:w } { \c_true_bool }
2035 \cs_new_nopar:cpn { bool_not_)_0:w } { \c_true_bool }
2036 \cs_new_nopar:cpn { bool_not_)_1:w } { \c_false_bool }
2037 \cs_new_nopar:cpn { bool_S_0:w } { \group_align_safe_end: \c_false_bool }
2038 \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.

```
2039 \cs_new_nopar:cpn { bool_&_0:w } & { \bool_eval_skip_to_end:Nw \c_false_bool }
2040 \cs_new_nopar:cpn { bool_|_1:w } | { \bool_eval_skip_to_end:Nw \c_true_bool }
2041 \cs_new_nopar:cpn { bool_not_&_1:w } &
2042 { \bool_eval_skip_to_end:Nw \c_false_bool }
```

```

2043 \cs_new_nopar:cpn { bool_not_|_0:w } |
2044 { \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.

```

2045 %% (
2046 \cs_new:Npn \bool_eval_skip_to_end:Nw #1#2 )
2047 {
2048   \bool_eval_skip_to_end_aux:Nw #1#2 ( % )
2049   \q_no_value \q_stop
2050   {#2}
2051 }

```

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

```

2052 \cs_new:Npn \bool_eval_skip_to_end_aux:Nw #1#2 ( #3#4 \q_stop #5 % )
2053 {
2054   \quark_if_no_value:NTF #3
2055   {#1}
2056   { \bool_eval_skip_to_end_aux_ii:Nw #1 #5 }
2057 }

```

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!

```

2058 \cs_new:Npn \bool_eval_skip_to_end_aux_ii:Nw #1#2 ( #3 )
2059 { % (
2060   \bool_eval_skip_to_end:Nw #1#3 )
2061 }

```

`\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.

```

2062 \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.

```

2063 \cs_new:Npn \bool_xor_p:nn #1#2
2064 {
2065   \int_compare:nNnTF { \bool_if_p:n {#1} } = { \bool_if_p:n {#2} }
2066   \c_false_bool
2067   \c_true_bool
2068 }

```

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
2069 \cs_new:Npn \bool_while_do:Nn #1#2
2070 { \bool_if:NT #1 { #2 \bool_while_do:Nn #1 {#2} } }
2071 \cs_new:Npn \bool_until_do:Nn #1#2
2072 { \bool_if:NF #1 { #2 \bool_until_do:Nn #1 {#2} } }
2073 \cs_generate_variant:Nn \bool_while_do:Nn { c }
2074 \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
2075 \cs_new:Npn \bool_do_while:Nn #1#2
2076 { #2 \bool_if:NT #1 { \bool_do_while:Nn #1 {#2} } }
2077 \cs_new:Npn \bool_do_until:Nn #1#2
2078 { #2 \bool_if:NF #1 { \bool_do_until:Nn #1 {#2} } }

```

```

2079 \cs_generate_variant:Nn \bool_do_while:Nn { c }
2080 \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
\bool_until_do:nn
\bool_do_until:nn
2081 \cs_new:Npn \bool_while_do:nn #1#2
2082 {
2083   \bool_if:nT {#1}
2084   {
2085     #2
2086     \bool_while_do:nn {#1} {#2}
2087   }
2088 }
2089 \cs_new:Npn \bool_do_while:nn #1#2
2090 {
2091   #2
2092   \bool_if:nT {#1} { \bool_do_while:nn {#1} {#2} }
2093 }
2094 \cs_new:Npn \bool_until_do:nn #1#2
2095 {
2096   \bool_if:nF {#1}
2097   {
2098     #2
2099     \bool_until_do:nn {#1} {#2}
2100   }
2101 }
2102 \cs_new:Npn \bool_do_until:nn #1#2
2103 {
2104   #2
2105   \bool_if:nF {#1} { \bool_do_until:nn {#1} {#2} }
2106 }

```

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.

```

2107 \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`.

```

2108 \cs_new:Npn \prg_case_int:nnn #1
2109 {

```

```

2110 \tex_romannumeral:D
2111 \exp_args:Nf \prg_case_int_aux:nnn { \int_eval:n {#1} }
2112 }
2113 \cs_new:Npn \prg_case_int_aux:nnn #1 #2 #3
2114 { \prg_case_int_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop }
2115 \cs_new:Npn \prg_case_int_aux:nw #1#2#3
2116 {
2117   \int_compare:nNnTF {#1} = {#2}
2118   { \prg_case_end:nw {#3} }
2119   { \prg_case_int_aux:nw {#1} }
2120 }

```

\prg_case_dim:nnn The dimension function is the same, just a change of calculation method.

```

\prg_case_dim_aux:nnn 2121 \cs_new:Npn \prg_case_dim:nnn #1
\prg_case_dim_aux:nw 2122 {
2123   \tex_romannumeral:D
2124   \exp_args:Nf \prg_case_dim_aux:nnn { \dim_eval:n {#1} }
2125 }
2126 \cs_new:Npn \prg_case_dim_aux:nnn #1 #2 #3
2127 { \prg_case_dim_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop }
2128 \cs_new:Npn \prg_case_dim_aux:nw #1#2#3
2129 {
2130   \dim_compare:nNnTF {#1} = {#2}
2131   { \prg_case_end:nw {#3} }
2132   { \prg_case_dim_aux:nw {#1} }
2133 }

```

\prg_case_str:nnn No calculations for strings, otherwise no surprises.

```

\prg_case_str:onn 2134 \cs_new:Npn \prg_case_str:nnn #1#2#3
\prg_case_str:xxn 2135 {
\prg_case_str_aux:nw 2136   \tex_romannumeral:D
\prg_case_str_x_aux:nw 2137   \prg_case_str_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop
2138 }
2139 \cs_new:Npn \prg_case_str_aux:nw #1#2#3
2140 {
2141   \str_if_eq:nnTF {#1} {#2}
2142   { \prg_case_end:nw {#3} }
2143   { \prg_case_str_aux:nw {#1} }
2144 }
2145 \cs_generate_variant:Nn \prg_case_str:nnn { o }
2146 \cs_new:Npn \prg_case_str:xxn #1#2#3
2147 {
2148   \tex_romannumeral:D
2149   \prg_case_str_x_aux:nw {#1} #2 {#1} {#3} \q_recursion_stop
2150 }
2151 \cs_new:Npn \prg_case_str_x_aux:nw #1#2#3
2152 {
2153   \str_if_eq:xxTF {#1} {#2}
2154   { \prg_case_end:nw {#3} }

```



```

2155     { \prg_case_str_x_aux:nw {#1} }
2156   }

```

\prg_case_tl:Nnn Similar again, but this time with some variants.
 \prg_case_tl:cnn
 \prg_case_tl_aux:Nw

```

2157 \cs_new:Npn \prg_case_tl:Nnn #1#2#3
2158 {
2159   \tex_romannumeral:D
2160   \prg_case_tl_aux:Nw #1 #2 #1 {#3} \q_recursion_stop
2161 }
2162 \cs_new:Npn \prg_case_tl_aux:Nw #1#2#3
2163 {
2164   \tl_if_eq:NNTF #1 #2
2165   { \prg_case_end:nw {#3} }
2166   { \prg_case_tl_aux:Nw #1 }
2167 }
2168 \cs_generate_variant:Nn \prg_case_tl:Nnn { c }

```

187.7 Producing n copies

\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

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.

```

2169 \cs_new:Npn \prg_replicate:nn #1
2170 {
2171   \int_to_roman:w
2172   \exp_after:wN \prg_replicate_first_aux:N
2173   \int_value:w \int_eval:w #1 \int_eval_end:
2174   \cs_end:
2175 }
2176 \cs_new:Npn \prg_replicate_aux:N #1

```

```

2177 { \cs:w prg_replicate_#1 :n \prg_replicate_aux:N }
2178 \cs_new:Npn \prg_replicate_first_aux:N #1
2179 { \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.

```

2180 \cs_new:Npn \prg_replicate_ :n #1 { \cs_end: }
2181 \cs_new:cpn { prg_replicate_0:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} }
2182 \cs_new:cpn { prg_replicate_1:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} #1 }
2183 \cs_new:cpn { prg_replicate_2:n } #1 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} #1#1 }
2184 \cs_new:cpn { prg_replicate_3:n } #1
2185 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} #1#1#1 }
2186 \cs_new:cpn { prg_replicate_4:n } #1
2187 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} #1#1#1#1 }
2188 \cs_new:cpn { prg_replicate_5:n } #1
2189 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} #1#1#1#1#1 }
2190 \cs_new:cpn { prg_replicate_6:n } #1
2191 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} #1#1#1#1#1#1 }
2192 \cs_new:cpn { prg_replicate_7:n } #1
2193 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} #1#1#1#1#1#1#1 }
2194 \cs_new:cpn { prg_replicate_8:n } #1
2195 { \cs_end: {#1#1#1#1#1#1#1#1#1#1} #1#1#1#1#1#1#1#1 }
2196 \cs_new:cpn { prg_replicate_9:n } #1
2197 { \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...

```

2198 \cs_new:cpn { prg_replicate_first_--:n } #1
2199 { \c_zero \msg_expandable_kernel_error:nn { prg } { replicate-neg } }
2200 \cs_new:cpn { prg_replicate_first_0:n } #1 { \c_zero }
2201 \cs_new:cpn { prg_replicate_first_1:n } #1 { \c_zero #1 }
2202 \cs_new:cpn { prg_replicate_first_2:n } #1 { \c_zero #1#1 }
2203 \cs_new:cpn { prg_replicate_first_3:n } #1 { \c_zero #1#1#1 }
2204 \cs_new:cpn { prg_replicate_first_4:n } #1 { \c_zero #1#1#1#1 }
2205 \cs_new:cpn { prg_replicate_first_5:n } #1 { \c_zero #1#1#1#1#1 }
2206 \cs_new:cpn { prg_replicate_first_6:n } #1 { \c_zero #1#1#1#1#1#1 }
2207 \cs_new:cpn { prg_replicate_first_7:n } #1 { \c_zero #1#1#1#1#1#1#1 }
2208 \cs_new:cpn { prg_replicate_first_8:n } #1 { \c_zero #1#1#1#1#1#1#1#1 }
2209 \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).

```

2210 \cs_new:Npn \prg_stepwise_function:nnnN #1#2#3#4
2211 {
2212   \prg_stepwise_aux:nnnN {#1} {#2} {#3} #4
2213   \prg_break_point:n { }
2214 }
2215 \cs_new:Npn \prg_stepwise_aux:nnnN #1#2#3#4
2216 {

```

```

2217 \int_compare:nNnTF {#2} > \c_zero
2218 { \exp_args:Nnf \prg_stepwise_aux:NnnnN > }
2219 {
2220   \int_compare:nNnTF {#2} = \c_zero
2221   {
2222     \msg_expandable_kernel_error:nnn { prg } { zero-step } {#4}
2223     \prg_map_break:
2224   }
2225   { \exp_args:Nnf \prg_stepwise_aux:NnnnN < }
2226 }
2227 { \int_eval:n {#1} } {#2} {#3} #4
2228 }
2229 \cs_new:Npn \prg_stepwise_aux:NnnnN #1#2#3#4#5
2230 {
2231   \int_compare:nNnF {#2} #1 {#4}
2232   {
2233     #5 {#2}
2234     \exp_args:Nnf \prg_stepwise_aux:NnnnN
2235     #1 { \int_eval:n { #2 + #3 } } {#3} {#4} #5
2236   }
2237 }

```

(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.

```

2238 \cs_new_protected:Npn \prg_stepwise_inline:nnnn
2239 {
2240   \exp_args:Nnc \prg_stepwise_aux:NNnnnn
2241   \cs_gset_nopar:Npn
2242   { g_prg_stepwise_ \int_use:N \g_prg_map_int :n }
2243 }
2244 \cs_new_protected:Npn \prg_stepwise_variable:nnnN #1#2#3#4#5
2245 {
2246   \exp_args:Nnc \prg_stepwise_aux:NNnnnn
2247   \cs_gset_nopar:Npx
2248   { g_prg_stepwise_ \int_use:N \g_prg_map_int :n }
2249   {#1}{#2}{#3}
2250   {
2251     \tl_set:Nn \exp_not:N #4 {##1}
2252     \exp_not:n {#5}
2253   }
2254 }
2255 \cs_new_protected:Npn \prg_stepwise_aux:NNnnnn #1#2#3#4#5#6
2256 {
2257   #1 #2 ##1 {#6}
2258   \int_gincr:N \g_prg_map_int
2259   \prg_stepwise_aux:nnnN {#3} {#4} {#5} #2
2260   \prg_break_point:n { \int_gdecr:N \g_prg_map_int }

```

```

2261 }
      (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.

```

2262 \prg_new_conditional:Npnn \mode_if_vertical: { p , T , F , TF }
2263 { \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.

```

2264 \prg_new_conditional:Npnn \mode_if_horizontal: { p , T , F , TF }
2265 { \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.

```

2266 \prg_new_conditional:Npnn \mode_if_inner: { p , T , F , TF }
2267 { \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.

```

2268 \prg_new_conditional:Npnn \mode_if_math: { p , T , F , TF }
2269 { \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.

```

2270 \cs_new_nopar:Npn \group_align_safe_begin:

```

```

2271 { \if_int_compare:w \if_false: { \fi: ' } = \c_zero \fi: }
2272 \cs_new_nopar:Npn \group_align_safe_end:
2273 { \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.

```

2274 \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.

```

2275 \group_begin:
2276 \tex_lccode:D '\& = '\g \scan_stop:
2277 \tex_catcode:D '\& = \c_twelve
2278 \tl_to_lowercase:n
2279 {
2280   \group_end:
2281   \cs_new:Npn \prg_variable_get_scope:N #1
2282   {

```

⁴Unless we enforce an extra pass with an appropriate value of `\pretolerance`.

```

2283     \exp_after:wN \exp_after:wN
2284     \exp_after:wN \prg_variable_get_scope_aux:w
2285     \cs_to_str:N #1 \exp_stop_f: \q_stop
2286   }
2287   \cs_new:Npn \prg_variable_get_scope_aux:w #1#2 \q_stop
2288   { \token_if_eq_meaning:NNT & #1 { g } }
2289 }
2290 \group_begin:
2291   \tex_lccode:D '\& = '\_ \scan_stop:
2292   \tex_catcode:D '\& = \c_twelve
2293   \tl_to_lowercase:n
2294   {
2295     \group_end:
2296     \cs_new:Npn \prg_variable_get_type:N #1
2297     {
2298       \exp_after:wN \prg_variable_get_type_aux:w
2299       \token_to_str:N #1 & a \q_stop
2300     }
2301     \cs_new:Npn \prg_variable_get_type_aux:w #1 & #2#3 \q_stop
2302     {
2303       \token_if_eq_meaning:NNTF a #2
2304       {#1}
2305       { \prg_variable_get_type_aux:w #2#3 \q_stop }
2306     }
2307   }

```

(End definition for \prg_variable_get_scope:N. This function is documented on page ??.)

\g_prg_map_int A nesting counter for mapping.

```

2308 \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 Deprecated functions

These were deprecated on 2012-02-08, and will be removed entirely by 2012-05-31.

\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.

```
2309 \cs_new_protected:Npn \prg_define_quicksort:nnn #1#2#3 {
2310   \cs_set:cpx{#1_quicksort:n}##1{
2311     \exp_not:c{#1_quicksort_start_partition:w} ##1
2312     \exp_not:n{#2\q_nil#3\q_stop}
2313   }
2314   \cs_set:cpx{#1_quicksort_braced:n}##1{
2315     \exp_not:c{#1_quicksort_start_partition_braced:n} ##1
2316     \exp_not:N\q_nil\exp_not:N\q_stop
2317   }
2318   \cs_set:cpx {#1_quicksort_start_partition:w} #2 ##1 #3{
2319     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
2320     \exp_not:c{#1_quicksort_do_partition_i:nnnw} {##1}{-}{-}
2321   }
2322   \cs_set:cpx {#1_quicksort_start_partition_braced:n} ##1 {
2323     \exp_not:N \quark_if_nil:nT {##1}\exp_not:N \use_none_delimit_by_q_stop:w
2324     \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn} {##1}{-}{-}
2325   }
```

Now for doing the partitions.

```
2326 \cs_set:cpx {#1_quicksort_do_partition_i:nnnw} ##1##2##3 #2 ##4 #3 {
2327   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnw}
2328   {
2329     \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
2330     \exp_not:c{#1_quicksort_partition_greater_ii:nnnn}
2331     \exp_not:c{#1_quicksort_partition_less_ii:nnnn}
2332   }
2333   {##1}{##2}{##3}{##4}
2334 }
2335 \cs_set:cpx {#1_quicksort_do_partition_i_braced:nnnn} ##1##2##3##4 {
2336   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnw}
2337   {
2338     \exp_not:c{#1_quicksort_compare:nnTF}{##1}{##4}
2339     \exp_not:c{#1_quicksort_partition_greater_ii_braced:nnnn}
2340     \exp_not:c{#1_quicksort_partition_less_ii_braced:nnnn}
2341   }
2342   {##1}{##2}{##3}{##4}
2343 }
2344 \cs_set:cpx {#1_quicksort_do_partition_ii:nnnw} ##1##2##3 #2 ##4 #3 {
2345   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnw}
```

```

2346 {
2347   \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
2348   \exp_not:c{#1_quicksort_partition_less_i:nnnn}
2349   \exp_not:c{#1_quicksort_partition_greater_i:nnnn}
2350 }
2351 {##1}{##2}{##3}{##4}
2352 }
2353 \cs_set:cpx {#1_quicksort_do_partition_ii_braced:nnnn} ##1##2##3##4 {
2354   \exp_not:N \quark_if_nil:nTF {##4} \exp_not:c {#1_do_quicksort_braced:nnnnw}
2355   {
2356     \exp_not:c{#1_quicksort_compare:nnTF}{##4}{##1}
2357     \exp_not:c{#1_quicksort_partition_less_i_braced:nnnn}
2358     \exp_not:c{#1_quicksort_partition_greater_i_braced:nnnn}
2359   }
2360   {##1}{##2}{##3}{##4}
2361 }

```

This part of the code handles the two branches in each sorting. Again we will also have to do it braced.

```

2362 \cs_set:cpx {#1_quicksort_partition_less_i:nnnn} ##1##2##3##4{
2363   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##2}{##4}{##3}}
2364 \cs_set:cpx {#1_quicksort_partition_less_ii:nnnn} ##1##2##3##4{
2365   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##3}{##4}}
2366 \cs_set:cpx {#1_quicksort_partition_greater_i:nnnn} ##1##2##3##4{
2367   \exp_not:c{#1_quicksort_do_partition_i:nnnw}{##1}{##4}{##2}{##3}}
2368 \cs_set:cpx {#1_quicksort_partition_greater_ii:nnnn} ##1##2##3##4{
2369   \exp_not:c{#1_quicksort_do_partition_ii:nnnw}{##1}{##2}{##4}{##3}}
2370 \cs_set:cpx {#1_quicksort_partition_less_i_braced:nnnn} ##1##2##3##4{
2371   \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn}{##1}{##2}{##4}{##3}}
2372 \cs_set:cpx {#1_quicksort_partition_less_ii_braced:nnnn} ##1##2##3##4{
2373   \exp_not:c{#1_quicksort_do_partition_ii_braced:nnnn}{##1}{##2}{##3}{##4}}
2374 \cs_set:cpx {#1_quicksort_partition_greater_i_braced:nnnn} ##1##2##3##4{
2375   \exp_not:c{#1_quicksort_do_partition_i_braced:nnnn}{##1}{##4}{##2}{##3}}
2376 \cs_set:cpx {#1_quicksort_partition_greater_ii_braced:nnnn} ##1##2##3##4{
2377   \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.

```

2378 \cs_set:cpx {#1_do_quicksort_braced:nnnnw} ##1##2##3##4\q_stop {
2379   \exp_not:c{#1_quicksort_braced:n}{##2}
2380   \exp_not:c{#1_quicksort_function:n}{##1}
2381   \exp_not:c{#1_quicksort_braced:n}{##3}
2382 }
2383 }

```

(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.

```

2384 \prg_define_quicksort:nnn {prg}{-}{-}

```

(End definition for \prg_quicksort:n. This function is documented on page ??.)


```

\prg_quicksort_function:n
\prg_quicksort_compare:nnTF
2385 \cs_set:Npn \prg_quicksort_function:n {\ERROR}
2386 \cs_set:Npn \prg_quicksort_compare:nnTF {\ERROR}
      (End definition for \prg_quicksort_function:n. This function is documented on page ??.)
      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
2387 <*deprecated>
2388 \cs_new_protected:Npn \prg_new_map_functions:Nn #1#2 { \deprecated }
2389 \cs_new_protected:Npn \prg_set_map_functions:Nn #1#2 { \deprecated }
2390 </deprecated>
      (End definition for \prg_new_map_functions:Nn. This function is documented on page ??.)
2391 </initex | package>

```

188 l3quark implementation

The following test files are used for this code: *m3quark001.lvt*.

```

2392 <*initex | package>
2393 <*package>
2394 \ProvidesExplPackage
2395   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
2396 \package_check_loaded_expl:
2397 </package>

```

188.1 Quarks

`\quark_new:N` Allocate a new quark.

```

2398 \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 and `\q_no_value` marks an empty argument.

```

\q_no_value
\q_stop
2399 \quark_new:N \q_nil
2400 \quark_new:N \q_mark
2401 \quark_new:N \q_no_value
2402 \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 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.

```

2403 \quark_new:N \q_recursion_tail
2404 \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.

```

2405 \cs_new:Npn \quark_if_recursion_tail_stop:N #1
2406 {
2407   \if_meaning:w \q_recursion_tail #1
2408   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
2409   \fi:
2410 }
2411 \cs_new:Npn \quark_if_recursion_tail_stop_do:Nn #1
2412 {
2413   \if_meaning:w \q_recursion_tail #1
2414   \exp_after:wN \use_i_delimit_by_q_recursion_stop:nw
2415   \else:
2416   \exp_after:wN \use_none:n
2417   \fi:
2418 }

```

(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.

```

2419 \cs_new:Npn \quark_if_recursion_tail_stop:n #1
2420 {
2421   \if_int_compare:w \pdfTeX_strcmp:D
2422   { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2423   \exp_after:wN \use_none_delimit_by_q_recursion_stop:w
2424   \fi:
2425 }
2426 \cs_new:Npn \quark_if_recursion_tail_stop_do:nn #1
2427 {
2428   \if_int_compare:w \pdfTeX_strcmp:D
2429   { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2430   \exp_after:wN \use_i_delimit_by_q_recursion_stop:nw
2431   \else:
2432   \exp_after:wN \use_none:n
2433   \fi:
2434 }
2435 \cs_generate_variant:Nn \quark_if_recursion_tail_stop:n { o }
2436 \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` Analogs of the `\quark_if_recursion_tail_stop...` functions. Break the mapping
`\quark_if_recursion_tail_break:n` using `\prg_map_break:`.

```

2437 \cs_new:Npn \quark_if_recursion_tail_break:N #1
2438 {
2439   \if_meaning:w \q_recursion_tail #1
2440   \exp_after:wN \prg_map_break:
2441   \fi:
2442 }
2443 \cs_new:Npn \quark_if_recursion_tail_break:n #1
2444 {
2445   \if_int_compare:w \pdfTeX_strcmp:D
2446   { \exp_not:N \q_recursion_tail } { \exp_not:n {#1} } = \c_zero
2447   \exp_after:wN \prg_map_break:
2448   \fi:
2449 }

```

(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

```

2450 \prg_new_conditional:Nnn \quark_if_nil:N { p, T, F, TF }
2451 {
2452   \if_meaning:w \q_nil #1
2453   \prg_return_true:
2454   \else:
2455     \prg_return_false:
2456   \fi:
2457 }
2458 \prg_new_conditional:Nnn \quark_if_no_value:N { p, T, F, TF }
2459 {
2460   \if_meaning:w \q_no_value #1
2461   \prg_return_true:
2462   \else:
2463     \prg_return_false:
2464   \fi:
2465 }

```

```

2466 \cs_generate_variant:Nn \quark_if_no_value_p:N { c }
2467 \cs_generate_variant:Nn \quark_if_no_value:NT { c }
2468 \cs_generate_variant:Nn \quark_if_no_value:NF { c }
2469 \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
\quark_if_nil:o
\quark_if_no_value:n
2470 \prg_new_conditional:Nnn \quark_if_nil:n { p, T, F, TF }
2471 {
2472   \if_int_compare:w \pdfTeX_strcmp:D
2473   { \exp_not:N \q_nil } { \exp_not:n {#1} } = \c_zero

```

⁵It may still loop in special circumstances however!

```

2474     \prg_return_true:
2475   \else:
2476     \prg_return_false:
2477   \fi:
2478 }
2479 \prg_new_conditional:Nnn \quark_if_no_value:n { p, T , F , TF }
2480 {
2481   \if_int_compare:w \pdfTeX_strcmp:D
2482     { \exp_not:N \q_no_value } { \exp_not:n {#1} } = \c_zero
2483     \prg_return_true:
2484   \else:
2485     \prg_return_false:
2486   \fi:
2487 }
2488 \cs_generate_variant:Nn \quark_if_nil_p:n { V , o }
2489 \cs_generate_variant:Nn \quark_if_nil:nTF { V , o }
2490 \cs_generate_variant:Nn \quark_if_nil:nT { V , o }
2491 \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.

\q_tl_act_stop

```

2492 \quark_new:N \q_tl_act_mark
2493 \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 97.)

188.2 Scan marks

\g_scan_marks_tl The list of all scan marks currently declared.

```

2494 \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.

```

2495 \cs_new_protected:Npn \scan_new:N #1
2496 {
2497   \tl_if_in:NnTF \g_scan_marks_tl { #1 }
2498   {
2499     \msg_kernel_error:nnx { scan } { already-defined }
2500     { \token_to_str:N #1 }
2501   }
2502   {
2503     \tl_gput_right:Nn \g_scan_marks_tl {#1}
2504     \cs_new_eq:NN #1 \scan_stop:
2505   }
2506 }

```

(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.

```
2507 \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`.

```
2508 \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.)
2509 </initex | package>
```

189 l3token implementation

```
2510 <*initex | package>
2511 <*package>
2512 \ProvidesExplPackage
2513   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
2514 \package_check_loaded_expl:
2515 </package>
```

189.1 Character tokens

`\char_set_catcode:nn` Category code changes.

```
\char_value_catcode:n
\char_show_value_catcode:n
2516 \cs_new_protected:Npn \char_set_catcode:nn #1#2
2517   { \tex_catcode:D #1 = \int_eval:w #2 \int_eval_end: }
2518 \cs_new:Npn \char_value_catcode:n #1
2519   { \tex_the:D \tex_catcode:D \int_eval:w #1 \int_eval_end: }
2520 \cs_new_protected:Npn \char_show_value_catcode:n #1
2521   { \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
2522 \cs_new_protected:Npn \char_set_catcode_escape:N #1
2523   { \char_set_catcode:nn { '#1 } \c_zero }
2524 \cs_new_protected:Npn \char_set_catcode_group_begin:N #1
2525   { \char_set_catcode:nn { '#1 } \c_one }
2526 \cs_new_protected:Npn \char_set_catcode_group_end:N #1
2527   { \char_set_catcode:nn { '#1 } \c_two }
2528 \cs_new_protected:Npn \char_set_catcode_math_toggle:N #1
2529   { \char_set_catcode:nn { '#1 } \c_three }
2530 \cs_new_protected:Npn \char_set_catcode_alignment:N #1
2531   { \char_set_catcode:nn { '#1 } \c_four }
2532 \cs_new_protected:Npn \char_set_catcode_end_line:N #1
2533   { \char_set_catcode:nn { '#1 } \c_five }
2534 \cs_new_protected:Npn \char_set_catcode_parameter:N #1
2535   { \char_set_catcode:nn { '#1 } \c_six }
2536 \cs_new_protected:Npn \char_set_catcode_math_superscript:N #1
2537   { \char_set_catcode:nn { '#1 } \c_seven }
2538 \cs_new_protected:Npn \char_set_catcode_math_subscript:N #1
```

```

2539 { \char_set_catcode:nn { '#1 } \c_eight }
2540 \cs_new_protected:Npn \char_set_catcode_ignore:N #1
2541 { \char_set_catcode:nn { '#1 } \c_nine }
2542 \cs_new_protected:Npn \char_set_catcode_space:N #1
2543 { \char_set_catcode:nn { '#1 } \c_ten }
2544 \cs_new_protected:Npn \char_set_catcode_letter:N #1
2545 { \char_set_catcode:nn { '#1 } \c_eleven }
2546 \cs_new_protected:Npn \char_set_catcode_other:N #1
2547 { \char_set_catcode:nn { '#1 } \c_twelve }
2548 \cs_new_protected:Npn \char_set_catcode_active:N #1
2549 { \char_set_catcode:nn { '#1 } \c_thirteen }
2550 \cs_new_protected:Npn \char_set_catcode_comment:N #1
2551 { \char_set_catcode:nn { '#1 } \c_fourteen }
2552 \cs_new_protected:Npn \char_set_catcode_invalid:N #1
2553 { \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 2554 \cs_new_protected:Npn \char_set_catcode_escape:n #1
    \char_set_catcode_group_end:n 2555 { \char_set_catcode:nn {#1} \c_zero }
  \char_set_catcode_math_toggle:n 2556 \cs_new_protected:Npn \char_set_catcode_group_begin:n #1
    \char_set_catcode_alignment:n 2557 { \char_set_catcode:nn {#1} \c_one }
\char_set_catcode_end_line:n 2558 \cs_new_protected:Npn \char_set_catcode_group_end:n #1
  \char_set_catcode_parameter:n 2559 { \char_set_catcode:nn {#1} \c_two }
  \char_set_catcode_math_superscript:n 2560 \cs_new_protected:Npn \char_set_catcode_math_toggle:n #1
    \char_set_catcode_math_subscript:n 2561 { \char_set_catcode:nn {#1} \c_three }
  \char_set_catcode_ignore:n 2562 \cs_new_protected:Npn \char_set_catcode_alignment:n #1
    \char_set_catcode_space:n 2563 { \char_set_catcode:nn {#1} \c_four }
  \char_set_catcode_letter:n 2564 \cs_new_protected:Npn \char_set_catcode_end_line:n #1
    \char_set_catcode_other:n 2565 { \char_set_catcode:nn {#1} \c_five }
  \char_set_catcode_active:n 2566 \cs_new_protected:Npn \char_set_catcode_parameter:n #1
    \char_set_catcode_comment:n 2567 { \char_set_catcode:nn {#1} \c_six }
  \char_set_catcode_invalid:n 2568 \cs_new_protected:Npn \char_set_catcode_math_superscript:n #1
    \char_set_catcode_math_subscript:n 2569 { \char_set_catcode:nn {#1} \c_seven }
    \char_set_catcode_math_subscript:n 2570 \cs_new_protected:Npn \char_set_catcode_math_subscript:n #1
    { \char_set_catcode:nn {#1} \c_eight }
    \char_set_catcode_ignore:n 2571 { \char_set_catcode:nn {#1} \c_eight }
    \char_set_catcode_ignore:n 2572 \cs_new_protected:Npn \char_set_catcode_ignore:n #1
    { \char_set_catcode:nn {#1} \c_nine }
    \char_set_catcode_space:n 2573 \cs_new_protected:Npn \char_set_catcode_ignore:n #1
    { \char_set_catcode:nn {#1} \c_nine }
    \char_set_catcode_space:n 2574 \cs_new_protected:Npn \char_set_catcode_space:n #1
    { \char_set_catcode:nn {#1} \c_ten }
    \char_set_catcode_letter:n 2575 { \char_set_catcode:nn {#1} \c_ten }
    \char_set_catcode_letter:n 2576 \cs_new_protected:Npn \char_set_catcode_letter:n #1
    { \char_set_catcode:nn {#1} \c_eleven }
    \char_set_catcode_other:n 2577 { \char_set_catcode:nn {#1} \c_eleven }
    \char_set_catcode_other:n 2578 \cs_new_protected:Npn \char_set_catcode_other:n #1
    { \char_set_catcode:nn {#1} \c_twelve }
    \char_set_catcode_active:n 2579 { \char_set_catcode:nn {#1} \c_twelve }
    \char_set_catcode_active:n 2580 \cs_new_protected:Npn \char_set_catcode_active:n #1
    { \char_set_catcode:nn {#1} \c_thirteen }
    \char_set_catcode_comment:n 2581 { \char_set_catcode:nn {#1} \c_thirteen }
    \char_set_catcode_comment:n 2582 \cs_new_protected:Npn \char_set_catcode_comment:n #1
    { \char_set_catcode:nn {#1} \c_fourteen }
    \char_set_catcode_invalid:n 2583 { \char_set_catcode:nn {#1} \c_fourteen }
    \char_set_catcode_invalid:n 2584 \cs_new_protected:Npn \char_set_catcode_invalid:n #1

```

```
2585 { \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_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

```

Pretty repetitive, but necessary!

```

2586 \cs_new_protected:Npn \char_set_mathcode:nn #1#2
2587 { \tex_mathcode:D #1 = \int_eval:w #2 \int_eval_end: }
2588 \cs_new:Npn \char_value_mathcode:n #1
2589 { \tex_the:D \tex_mathcode:D \int_eval:w #1\int_eval_end: }
2590 \cs_new_protected:Npn \char_show_value_mathcode:n #1
2591 { \tex_showthe:D \tex_mathcode:D \int_eval:w #1 \int_eval_end: }
2592 \cs_new_protected:Npn \char_set_lccode:nn #1#2
2593 { \tex_lccode:D #1 = \int_eval:w #2 \int_eval_end: }
2594 \cs_new:Npn \char_value_lccode:n #1
2595 { \tex_the:D \tex_lccode:D \int_eval:w #1\int_eval_end: }
2596 \cs_new_protected:Npn \char_show_value_lccode:n #1
2597 { \tex_showthe:D \tex_lccode:D \int_eval:w #1 \int_eval_end: }
2598 \cs_new_protected:Npn \char_set_uccode:nn #1#2
2599 { \tex_uccode:D #1 = \int_eval:w #2 \int_eval_end: }
2600 \cs_new:Npn \char_value_uccode:n #1
2601 { \tex_the:D \tex_uccode:D \int_eval:w #1\int_eval_end: }
2602 \cs_new_protected:Npn \char_show_value_uccode:n #1
2603 { \tex_showthe:D \tex_uccode:D \int_eval:w #1 \int_eval_end: }
2604 \cs_new_protected:Npn \char_set_sfcode:nn #1#2
2605 { \tex_sfcode:D #1 = \int_eval:w #2 \int_eval_end: }
2606 \cs_new:Npn \char_value_sfcode:n #1
2607 { \tex_the:D \tex_sfcode:D \int_eval:w #1\int_eval_end: }
2608 \cs_new_protected:Npn \char_show_value_sfcode:n #1
2609 { \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.

```
2610 \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
\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

```

We define these useful tokens. We have to do it by hand with the brace tokens for obvious reasons.

```

2611 \cs_new_eq:NN \c_group_begin_token {
2612 \cs_new_eq:NN \c_group_end_token }
2613 \group_begin:
2614 \char_set_catcode_math_toggle:N \*
2615 \token_new:Nn \c_math_toggle_token { * }
2616 \char_set_catcode_alignment:N \*
2617 \token_new:Nn \c_alignment_token { * }
2618 \token_new:Nn \c_parameter_token { # }
2619 \token_new:Nn \c_math_superscript_token { ^ }
2620 \char_set_catcode_math_subscript:N \*

```

```

2621 \token_new:Nn \c_math_subscript_token { * }
2622 \token_new:Nn \c_space_token { ~ }
2623 \token_new:Nn \c_catcode_letter_token { a }
2624 \token_new:Nn \c_catcode_other_token { 1 }
2625 \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!

```

2626 \group_begin:
2627 \char_set_catcode_active:N \*
2628 \tl_const:Nn \c_catcode_active_tl { \exp_not:N * }
2629 \group_end:

```

(End definition for `\c_catcode_active_tl`. This function is documented on page 51.)

`\l_char_active_seq` Two sequences for dealing with special characters. The first is characters which may be active, and contains the active characters themselves to allow easy redefinition. The second longer list is for “special” characters more generally, and these are escaped so that for example bulk code assignments can be carried out. In both cases, the order is by ASCII character code (as is done in for example `\ExplSyntaxOn`). The only complication is dealing with `_`, which has to be done using `\tl_to_lowercase:n`

```

2630 \seq_new:N \l_char_active_seq
2631 \use:n
2632 {
2633   \group_begin:
2634   \char_set_catcode_active:N \"
2635   \char_set_catcode_active:N \$
2636   \char_set_catcode_active:N &
2637   \char_set_catcode_active:N ^
2638   \char_set_catcode_active:N _
2639   \char_set_catcode_active:N ~
2640   \use:nn
2641   {
2642     \group_end:
2643     \seq_set_from_clist:Nn \l_char_active_seq
2644   }
2645 }
2646 { { " , $ , & , ^ , _ , ~ } }
2647 \seq_new:N \l_char_special_seq
2648 \seq_set_from_clist:Nn \l_char_special_seq
2649 { \ , \" , \# , \$ , \% , & , \\ , \^ , \_ , \{ , \} , \~ }

```

(End definition for `\l_char_active_seq` and `\l_char_special_seq`. These functions are 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.


```

2650 \prg_new_conditional:Npnn \token_if_group_begin:N #1 { p , T , F , TF }
2651 {
2652   \if_catcode:w \exp_not:N #1 \c_group_begin_token
2653   \prg_return_true: \else: \prg_return_false: \fi:
2654 }

```

(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.

```

2655 \prg_new_conditional:Npnn \token_if_group_end:N #1 { p , T , F , TF }
2656 {
2657   \if_catcode:w \exp_not:N #1 \c_group_end_token
2658   \prg_return_true: \else: \prg_return_false: \fi:
2659 }

```

(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.

```

2660 \prg_new_conditional:Npnn \token_if_math_toggle:N #1 { p , T , F , TF }
2661 {
2662   \if_catcode:w \exp_not:N #1 \c_math_toggle_token
2663   \prg_return_true: \else: \prg_return_false: \fi:
2664 }

```

(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.

```

2665 \prg_new_conditional:Npnn \token_if_alignment:N #1 { p , T , F , TF }
2666 {
2667   \if_catcode:w \exp_not:N #1 \c_alignment_token
2668   \prg_return_true: \else: \prg_return_false: \fi:
2669 }

```

(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.

```

2670 \group_begin:
2671 \cs_set_eq:NN \c_parameter_token \scan_stop:
2672 \prg_new_conditional:Npnn \token_if_parameter:N #1 { p , T , F , TF }
2673 {
2674   \if_catcode:w \exp_not:N #1 \c_parameter_token
2675   \prg_return_true: \else: \prg_return_false: \fi:
2676 }
2677 \group_end:

```

(End definition for \token_if_parameter:N. This function is documented on page 53.)

`\token_if_math_superscript:N` Check if token is a math superscript token. We use the constant `\c_superscript_token` for this.

```

2678 \prg_new_conditional:Npnn \token_if_math_superscript:N #1 { p , T , F , TF }
2679 {
2680   \if_catcode:w \exp_not:N #1 \c_math_superscript_token
2681   \prg_return_true: \else: \prg_return_false: \fi:
2682 }

```

(End definition for \token_if_math_superscript:N. This function is documented on page 53.)

`\token_if_math_subscript:N` Check if token is a math subscript token. We use the constant `\c_subscript_token` for this.

```

2683 \prg_new_conditional:Npnn \token_if_math_subscript:N #1 { p , T , F , TF }
2684 {
2685   \if_catcode:w \exp_not:N #1 \c_math_subscript_token
2686   \prg_return_true: \else: \prg_return_false: \fi:
2687 }

```

(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.

```

2688 \prg_new_conditional:Npnn \token_if_space:N #1 { p , T , F , TF }
2689 {
2690   \if_catcode:w \exp_not:N #1 \c_space_token
2691   \prg_return_true: \else: \prg_return_false: \fi:
2692 }

```

(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.

```

2693 \prg_new_conditional:Npnn \token_if_letter:N #1 { p , T , F , TF }
2694 {
2695   \if_catcode:w \exp_not:N #1 \c_catcode_letter_token
2696   \prg_return_true: \else: \prg_return_false: \fi:
2697 }

```

(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.

```

2698 \prg_new_conditional:Npnn \token_if_other:N #1 { p , T , F , TF }
2699 {
2700   \if_catcode:w \exp_not:N #1 \c_catcode_other_token
2701   \prg_return_true: \else: \prg_return_false: \fi:
2702 }

```

(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.

```

2703 \prg_new_conditional:Npnn \token_if_active:N #1 { p , T , F , TF }
2704 {

```

```

2705 \if_catcode:w \exp_not:N #1 \c_catcode_active_tl
2706 \prg_return_true: \else: \prg_return_false: \fi:
2707 }

```

(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.

```

2708 \prg_new_conditional:Npnn \token_if_eq_meaning:NN #1#2 { p , T , F , TF }
2709 {
2710   \if_meaning:w #1 #2
2711   \prg_return_true: \else: \prg_return_false: \fi:
2712 }

```

(End definition for `\token_if_eq_meaning:NN`. This function is documented on page 54.)

`\token_if_eq_catcode:NN` Check if the tokens #1 and #2 have same category code.

```

2713 \prg_new_conditional:Npnn \token_if_eq_catcode:NN #1#2 { p , T , F , TF }
2714 {
2715   \if_catcode:w \exp_not:N #1 \exp_not:N #2
2716   \prg_return_true: \else: \prg_return_false: \fi:
2717 }

```

(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.

```

2718 \prg_new_conditional:Npnn \token_if_eq_charcode:NN #1#2 { p , T , F , TF }
2719 {
2720   \if_charcode:w \exp_not:N #1 \exp_not:N #2
2721   \prg_return_true: \else: \prg_return_false: \fi:
2722 }

```

(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.

```

2723 \group_begin:
2724 \char_set_catcode_other:N \M
2725 \char_set_catcode_other:N \A

```

```

2726 \char_set_lccode:nn { '\; } { '\: }
2727 \char_set_lccode:nn { '\T } { '\T }
2728 \char_set_lccode:nn { '\F } { '\F }
2729 \tl_to_lowercase:n
2730 {
2731   \group_end:
2732   \prg_new_conditional:Npnn \token_if_macro:N #1 { p , T , F , TF }
2733   {
2734     \exp_after:wN \token_if_macro_p_aux:w
2735     \token_to_meaning:N #1 MA; \q_stop
2736   }
2737   \cs_new:Npn \token_if_macro_p_aux:w #1 MA #2 ; #3 \q_stop
2738   {
2739     \if_int_compare:w \pdfstrcmp:D { #2 } { cro } = \c_zero
2740     \prg_return_true:
2741   }
2742   \else:
2743     \prg_return_false:
2744   \fi:
2745 }

```

(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.

```

2746 \prg_new_conditional:Npnn \token_if_cs:N #1 { p , T , F , TF }
2747 {
2748   \if_catcode:w \exp_not:N #1 \scan_stop:
2749   \prg_return_true: \else: \prg_return_false: \fi:
2750 }

```

(End definition for `\token_if_cs:N`. This function is documented on page 54.)

`\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.

```

2751 \prg_new_conditional:Npnn \token_if_expandable:N #1 { p , T , F , TF }
2752 {
2753   \cs_if_exist:NTF #1
2754   {
2755     \exp_after:wN \if_meaning:w \exp_not:N #1 #1
2756     \prg_return_false: \else: \prg_return_true: \fi:
2757   }
2758   { \prg_return_false: }
2759 }

```

(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_mathchardef:N
\token_if_dim_register:N
\token_if_int_register:N
\token_if_skip_register:N
\token_if_toks_register:N
\token_if_long_macro:N
\token_if_protected_macro:N
\token_if_protected_long_macro:N
\token_if_chardef_aux:w
\token_if_dim_register_aux:w
\token_if_int_register_aux:w
\token_if_skip_register_aux:w
\token_if_toks_register_aux:w
\token_if_protected_macro_aux:w

```

```

2760 \group_begin:
2761 \char_set_lccode:nn { 'T } { 'T }
2762 \char_set_lccode:nn { 'F } { 'F }
2763 \char_set_lccode:nn { 'X } { 'n }
2764 \char_set_lccode:nn { 'Y } { 't }
2765 \char_set_lccode:nn { 'Z } { 'd }
2766 \tl_map_inline:nn { A C E G H I K L M O P R S U X Y Z R " }
2767 { \char_set_catcode:nn { '#1 } \c_twelve }

```

We convert the token list to lower case and restore the catcode and lowercase code changes.

```

2768 \tl_to_lowercase:n
2769 {
2770 \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>`. Grab until the first occurrence of `char"`, and compare what precedes with `\` or `\math`. In fact, the escape character may not be a backslash, so we compare with the result of converting some other control sequence to a string, namely `\char` or `\mathchar` (the auxiliary adds the `char` back).

```

2771 \prg_new_conditional:Npnn \token_if_chardef:N #1 { p , T , F , TF }
2772 {
2773 \str_if_eq_return:xx
2774 {
2775 \exp_after:wN \token_if_chardef_aux:w
2776 \token_to_meaning:N #1 CHAR" \q_stop
2777 }
2778 { \token_to_str:N \char }
2779 }
2780 \prg_new_conditional:Npnn \token_if_mathchardef:N #1 { p , T , F , TF }
2781 {
2782 \str_if_eq_return:xx
2783 {
2784 \exp_after:wN \token_if_chardef_aux:w
2785 \token_to_meaning:N #1 CHAR" \q_stop
2786 }
2787 { \token_to_str:N \mathchar }
2788 }
2789 \cs_new:Npn \token_if_chardef_aux:w #1 CHAR" #2 \q_stop { #1 CHAR }

```

Dim registers are a little more difficult since their `\meaning` has the form `\dimen<number>`, and we must take care of the two primitives `\dimen` and `\dimendef`.

```

2790 \prg_new_conditional:Npnn \token_if_dim_register:N #1 { p , T , F , TF }
2791 {
2792 \if_meaning:w \tex_dimen:D #1
2793 \prg_return_false:
2794 \else:
2795 \if_meaning:w \tex_dimendef:D #1
2796 \prg_return_false:

```

```

2797         \else:
2798         \str_if_eq_return:xx
2799         {
2800             \exp_after:wN \token_if_dim_register_aux:w
2801             \token_to_meaning:N #1 ZIMEX \q_stop
2802         }
2803         { \token_to_str:N \ }
2804     \fi:
2805 \fi:
2806 }
2807 \cs_new:Npn \token_if_dim_register_aux:w #1 ZIMEX #2 \q_stop { #1 ~ }

```

Integer registers are one step harder since constants are implemented differently from variables, and we also have to take care of the primitives `\count` and `\countdef`.

```

2808 \prg_new_conditional:Npnn \token_if_int_register:N #1 { p , T , F , TF }
2809 {
2810     % \token_if_chardef:NTF #1 { \prg_return_true: }
2811     % {
2812     %     \token_if_mathchardef:NTF #1 { \prg_return_true: }
2813     %     {
2814     \if_meaning:w \tex_count:D #1
2815     \prg_return_false:
2816     \else:
2817     \if_meaning:w \tex_countdef:D #1
2818     \prg_return_false:
2819     \else:
2820     \str_if_eq_return:xx
2821     {
2822         \exp_after:wN \token_if_int_register_aux:w
2823         \token_to_meaning:N #1 COUXY \q_stop
2824     }
2825     { \token_to_str:N \ }
2826     \fi:
2827 \fi:
2828 %     }
2829 % }
2830 }
2831 \cs_new:Npn \token_if_int_register_aux:w #1 COUXY #2 \q_stop { #1 ~ }

```

Skip registers are done the same way as the dimension registers.

```

2832 \prg_new_conditional:Npnn \token_if_skip_register:N #1 { p , T , F , TF }
2833 {
2834     \if_meaning:w \tex_skip:D #1
2835     \prg_return_false:
2836     \else:
2837     \if_meaning:w \tex_skipdef:D #1
2838     \prg_return_false:
2839     \else:
2840     \str_if_eq_return:xx
2841     {

```

```

2842         \exp_after:wN \token_if_skip_register_aux:w
2843         \token_to_meaning:N #1 SKIP \q_stop
2844     }
2845     { \token_to_str:N \ }
2846     \fi:
2847     \fi:
2848 }
2849 \cs_new:Npn \token_if_skip_register_aux:w #1 SKIP #2 \q_stop { #1 ~ }

```

Toks registers.

```

2850 \prg_new_conditional:Npnn \token_if_toks_register:N #1 { p , T , F , TF }
2851 {
2852     \if_meaning:w \tex_toks:D #1
2853     \prg_return_false:
2854     \else:
2855         \if_meaning:w \tex_toksdef:D #1
2856         \prg_return_false:
2857         \else:
2858             \str_if_eq_return:xx
2859             {
2860                 \exp_after:wN \token_if_toks_register_aux:w
2861                 \token_to_meaning:N #1 YOKS \q_stop
2862             }
2863             { \token_to_str:N \ }
2864             \fi:
2865             \fi:
2866         }
2867     \cs_new:Npn \token_if_toks_register_aux:w #1 YOKS #2 \q_stop { #1 ~ }

```

Protected macros.

```

2868 \prg_new_conditional:Npnn \token_if_protected_macro:N #1
2869 { p , T , F , TF }
2870 {
2871     \str_if_eq_return:xx
2872     {
2873         \exp_after:wN \token_if_protected_macro_aux:w
2874         \token_to_meaning:N #1 PROYECYEEZ~MACRO \q_stop
2875     }
2876     { \token_to_str:N \ }
2877 }
2878 \cs_new:Npn \token_if_protected_macro_aux:w
2879 #1 PROYECYEEZ~MACRO #2 \q_stop { #1 ~ }

```

Long macros and protected long macros share an auxiliary.

```

2880 \prg_new_conditional:Npnn \token_if_long_macro:N #1 { p , T , F , TF }
2881 {
2882     \str_if_eq_return:xx
2883     {
2884         \exp_after:wN \token_if_long_macro_aux:w
2885         \token_to_meaning:N #1 LOXG~MACRO \q_stop
2886     }

```

```

2887         { \token_to_str:N \ }
2888     }
2889 \prg_new_conditional:Npnn \token_if_protected_long_macro:N #1
2890 { p , T , F , TF }
2891 {
2892     \str_if_eq_return:xx
2893     {
2894         \exp_after:wN \token_if_long_macro_aux:w
2895         \token_to_meaning:N #1 LOXG-MACRO \q_stop
2896     }
2897     { \token_to_str:N \protected \token_to_str:N \ }
2898 }
2899 \cs_new:Npn \token_if_long_macro_aux:w #1 LOXG-MACRO #2 \q_stop { #1 ~ }

```

Finally the `\tl_to_lowercase:n` ends!

```

2900 }

```

(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.

```

2901 \tex_chardef:D \c_token_A_int = 'A ~ %
2902 \group_begin:
2903 \char_set_catcode_other:N \;
2904 \char_set_lccode:nn { '\; } { '\: }
2905 \char_set_lccode:nn { '\T } { '\T }
2906 \char_set_lccode:nn { '\F } { '\F }

```



```

2907 \tl_to_lowercase:n {
2908   \group_end:
2909   \prg_new_conditional:Npnn \token_if_primitive:N #1 { p , T , F , TF }
2910   {
2911     \token_if_macro:NTF #1
2912     \prg_return_false:
2913     {
2914       \exp_after:wN \token_if_primitive_aux:NNw
2915       \token_to_meaning:N #1 ; ; ; \q_stop #1
2916     }
2917   }
2918   \cs_new:Npn \token_if_primitive_aux:NNw #1#2 #3 ; #4 \q_stop
2919   {
2920     \tl_if_empty:oTF { \token_if_primitive_aux_space:w #3 ~ }
2921     { \token_if_primitive_aux_loop:N #3 ; \q_stop }
2922     { \token_if_primitive_aux_nullfont:N }
2923   }
2924 }
2925 \cs_new:Npn \token_if_primitive_aux_space:w #1 ~ { }
2926 \cs_new:Npn \token_if_primitive_aux_nullfont:N #1
2927 {
2928   \if_meaning:w \tex_nullfont:D #1
2929   \prg_return_true:
2930   \else:
2931   \prg_return_false:
2932   \fi:
2933 }
2934 \cs_new:Npn \token_if_primitive_aux_loop:N #1
2935 {
2936   \if_num:w '#1 < \c_token_A_int %
2937   \exp_after:wN \token_if_primitive_auxii:Nw
2938   \exp_after:wN #1
2939   \else:
2940   \exp_after:wN \token_if_primitive_aux_loop:N
2941   \fi:
2942 }
2943 \cs_new:Npn \token_if_primitive_auxii:Nw #1 #2 \q_stop
2944 {
2945   \if:w : #1
2946   \exp_after:wN \token_if_primitive_aux_undefined:N
2947   \else:
2948   \prg_return_false:
2949   \exp_after:wN \use_none:n
2950   \fi:
2951 }
2952 \cs_new:Npn \token_if_primitive_aux_undefined:N #1
2953 {
2954   \if_cs_exist:N #1
2955   \prg_return_true:
2956   \else:

```

```

2957     \prg_return_false:
2958     \fi:
2959 }

```

(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
2960 \cs_new_eq:NN \l_peek_token ?
2961 \cs_new_eq:NN \g_peek_token ?

```

(End definition for `\l_peek_token`. This function is documented on page 56.)

`\l_peek_search_token` The token to search for as an implicit token: cf. `\l_peek_search_tl`.

```

2962 \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`.

```

2963 \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
\peek_false:w
\peek_tmp:w
2964 \cs_new_nopar:Npn \peek_true:w { }
2965 \cs_new_nopar:Npn \peek_true_aux:w { }
2966 \cs_new_nopar:Npn \peek_false:w { }
2967 \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
2968 \cs_new_protected_nopar:Npn \peek_after:Nw
2969 { \tex_futurelet:D \l_peek_token }
2970 \cs_new_protected_nopar:Npn \peek_gafter:Nw
2971 { \tex_global:D \tex_futurelet:D \g_peek_token }

```

(End definition for `\peek_after:Nw`. This function is documented on page 56.)

`\peek_true_remove:w` A function to remove the next token and then regain control.

```

2972 \cs_new_protected:Npn \peek_true_remove:w
2973 {
2974   \group_align_safe_end:
2975   \tex_afterassignment:D \peek_true_aux:w
2976   \cs_set_eq:NN \peek_tmp:w
2977 }

```

(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.

```

2978 \cs_new_protected:Npn \peek_token_generic:NNTF #1#2#3#4
2979 {
2980   \cs_set_eq:NN \l_peek_search_token #2
2981   \tl_set:Nn \l_peek_search_tl {#2}
2982   \cs_set_nopar:Npx \peek_true:w
2983   {
2984     \exp_not:N \group_align_safe_end:
2985     \exp_not:n {#3}
2986   }
2987   \cs_set_nopar:Npx \peek_false:w
2988   {
2989     \exp_not:N \group_align_safe_end:
2990     \exp_not:n {#4}
2991   }
2992   \group_align_safe_begin:
2993   \peek_after:Nw #1
2994 }
2995 \cs_new_protected:Npn \peek_token_generic:NNT #1#2#3
2996 { \peek_token_generic:NNTF #1 #2 {#3} { } }
2997 \cs_new_protected:Npn \peek_token_generic:NNF #1#2#3
2998 { \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.

```

2999 \cs_new_protected:Npn \peek_token_remove_generic:NNTF #1#2#3#4
3000 {
3001   \cs_set_eq:NN \l_peek_search_token #2
3002   \tl_set:Nn \l_peek_search_tl {#2}
3003   \cs_set_eq:NN \peek_true:w \peek_true_remove:w
3004   \cs_set_nopar:Npx \peek_true_aux:w { \exp_not:n {#3} }
3005   \cs_set_nopar:Npx \peek_false:w
3006   {
3007     \exp_not:N \group_align_safe_end:
3008     \exp_not:n {#4}
3009   }
3010   \group_align_safe_begin:
3011   \peek_after:Nw #1

```

```

3012 }
3013 \cs_new_protected:Npn \peek_token_remove_generic:NNT #1#2#3
3014 { \peek_token_remove_generic:NNTF #1 #2 {#3} { } }
3015 \cs_new_protected:Npn \peek_token_remove_generic:NNF #1#2#3
3016 { \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:
3017 \cs_new_nopar:Npn \peek_execute_branches_catcode:
3018 {
3019   \if_catcode:w
3020     \exp_not:N \l_peek_token \exp_not:N \l_peek_search_token
3021     \exp_after:wN \peek_true:w
3022   \else:
3023     \exp_after:wN \peek_false:w
3024   \fi:
3025 }
3026 \cs_new_nopar:Npn \peek_execute_branches_meaning:
3027 {
3028   \if_meaning:w \l_peek_token \l_peek_search_token
3029     \exp_after:wN \peek_true:w
3030   \else:
3031     \exp_after:wN \peek_false:w
3032   \fi:
3033 }

```

(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 T_EX grabbing brace group
\peek_execute_branches_charcode:NN 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.

```

3034 \cs_new_nopar:Npn \peek_execute_branches_charcode:
3035 {
3036   \bool_if:nTF
3037   {
3038     \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token
3039     || \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token
3040     || \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
3041   }
3042   { \peek_false:w }
3043   {
3044     \exp_after:wN \peek_execute_branches_charcode_aux:NN
3045     \l_peek_search_tl
3046   }
3047 }
3048 \cs_new:Npn \peek_execute_branches_charcode_aux:NN #1#2
3049 {
3050   \if:w \exp_not:N #1 \exp_not:N #2
3051     \exp_after:wN \peek_true:w

```

```

3052     \else:
3053         \exp_after:wN \peek_false:w
3054     \fi:
3055     #2
3056 }

```

(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
\peek_ignore_spaces_execute_branches_aux: things other than spaces.

```

3057 \cs_new_protected_nopar:Npn \peek_ignore_spaces_execute_branches:
3058 {
3059     \token_if_eq_meaning:NNTF \l_peek_token \c_space_token
3060     {
3061         \tex_afterassignment:D \peek_ignore_spaces_execute_branches_aux:
3062         \cs_set_eq:NN \peek_tmp:w
3063     }
3064     { \peek_execute_branches: }
3065 }
3066 \cs_new_protected_nopar:Npn \peek_ignore_spaces_execute_branches_aux:
3067 { \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_new_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.

```

3068 \group_begin:
3069 \cs_set:Npn \peek_def:nnnn #1#2#3#4
3070 {
3071     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { TF }
3072     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { T }
3073     \peek_def_aux:nnnnn {#1} {#2} {#3} {#4} { F }
3074 }
3075 \cs_set:Npn \peek_def_aux:nnnnn #1#2#3#4#5
3076 {
3077     \cs_gset_nopar:cpx { #1 #5 }
3078     {
3079         \tl_if_empty:nF {#2}
3080         { \exp_not:n { \cs_set_eq:NN \peek_execute_branches: #2 } }
3081         \exp_not:c { #3 #5 }
3082         \exp_not:n {#4}
3083     }
3084 }

```

(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 3085 \peek_def:nnnn { peek_catcode:N }
\peek_catcode_remove:N         3086 { }
\peek_catcode_remove_ignore_spaces:N 3087 { peek_token_generic:NN }

```

```

3088     { \peek_execute_branches_catcode: }
3089 \peek_def:nnnn { peek_catcode_ignore_spaces:N }
3090     { \peek_execute_branches_catcode: }
3091     { peek_token_generic:NN }
3092     { \peek_ignore_spaces_execute_branches: }
3093 \peek_def:nnnn { peek_catcode_remove:N }
3094     { }
3095     { peek_token_remove_generic:NN }
3096     { \peek_execute_branches_catcode: }
3097 \peek_def:nnnn { peek_catcode_remove_ignore_spaces:N }
3098     { \peek_execute_branches_catcode: }
3099     { peek_token_remove_generic:NN }
3100     { \peek_ignore_spaces_execute_branches: }

```

(End definition for \peek_catcode:N and others. These functions are documented on page 57.)

\peek_charcode:N Then for character codes.

```

\peek_charcode_ignore_spaces:N 3101 \peek_def:nnnn { peek_charcode:N }
\peek_charcode_remove:N         3102 { }
\peek_charcode_remove_ignore_spaces:N 3103 { peek_token_generic:NN }
3104 { \peek_execute_branches_charcode: }
3105 \peek_def:nnnn { peek_charcode_ignore_spaces:N }
3106 { \peek_execute_branches_charcode: }
3107 { peek_token_generic:NN }
3108 { \peek_ignore_spaces_execute_branches: }
3109 \peek_def:nnnn { peek_charcode_remove:N }
3110 { }
3111 { peek_token_remove_generic:NN }
3112 { \peek_execute_branches_charcode: }
3113 \peek_def:nnnn { peek_charcode_remove_ignore_spaces:N }
3114 { \peek_execute_branches_charcode: }
3115 { peek_token_remove_generic:NN }
3116 { \peek_ignore_spaces_execute_branches: }

```

(End definition for \peek_charcode:N and others. These functions are documented on page 57.)

\peek_meaning:N Finally for meaning, with the group closed to remove the temporary definition functions.

```

\peek_meaning_ignore_spaces:N 3117 \peek_def:nnnn { peek_meaning:N }
\peek_meaning_remove:N        3118 { }
\peek_meaning_remove_ignore_spaces:N 3119 { peek_token_generic:NN }
3120 { \peek_execute_branches_meaning: }
3121 \peek_def:nnnn { peek_meaning_ignore_spaces:N }
3122 { \peek_execute_branches_meaning: }
3123 { peek_token_generic:NN }
3124 { \peek_ignore_spaces_execute_branches: }
3125 \peek_def:nnnn { peek_meaning_remove:N }
3126 { }
3127 { peek_token_remove_generic:NN }
3128 { \peek_execute_branches_meaning: }
3129 \peek_def:nnnn { peek_meaning_remove_ignore_spaces:N }
3130 { \peek_execute_branches_meaning: }

```

```

3131 { peek_token_remove_generic:NN }
3132 { \peek_ignore_spaces_execute_branches: }
3133 \group_end:
      (End definition for \peek_meaning:N and others. These functions are documented on page 58.)

```

189.5 Decomposing a macro definition

`\token_get_prefix_spec:N` We sometimes want to test if a control sequence can be expanded to reveal a hidden value.
`\token_get_arg_spec:N` However, we cannot just expand the macro blindly as it may have arguments and none
`\token_get_replacement_spec:N` might be present. Therefore we define these functions to pick either the prefix(es), the
`\token_get_prefix_arg_replacement_aux:wN` 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.

```

3134 \exp_args:Nno \use:nn
3135 { \cs_new:Npn \token_get_prefix_arg_replacement_aux:wN #1 }
3136 { \tl_to_str:n { macro : } #2 -> #3 \q_stop #4 }
3137 { #4 {#1} {#2} {#3} }
3138 \cs_new:Npn \token_get_prefix_spec:N #1
3139 {
3140   \token_if_macro:NTF #1
3141   {
3142     \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3143     \token_to_meaning:N #1 \q_stop \use_i:nnn
3144   }
3145   { \scan_stop: }
3146 }
3147 \cs_new:Npn \token_get_arg_spec:N #1
3148 {
3149   \token_if_macro:NTF #1
3150   {
3151     \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3152     \token_to_meaning:N #1 \q_stop \use_ii:nnn
3153   }
3154   { \scan_stop: }
3155 }
3156 \cs_new:Npn \token_get_replacement_spec:N #1
3157 {
3158   \token_if_macro:NTF #1
3159   {
3160     \exp_after:wN \token_get_prefix_arg_replacement_aux:wN
3161     \token_to_meaning:N #1 \q_stop \use_iii:nnn
3162   }
3163   { \scan_stop: }
3164 }

```

(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
3165 \group_begin:
3166 \char_set_catcode_active:N \^^@
3167 \cs_set:Npn \char_tmp:NN #1#2
3168 {
3169   \cs_new:Npn #1 ##1
3170   {
3171     \char_set_catcode_active:n { '##1 }
3172     \group_begin:
3173     \char_set_lccode:nn { '\^^@ } { '##1 }
3174     \tl_to_lowercase:n { \group_end: #2 ^^@ }
3175   }
3176 }
3177 \char_tmp:NN \char_set_active:Npn \cs_set:Npn
3178 \char_tmp:NN \char_set_active:Npx \cs_set:Npx
3179 \char_tmp:NN \char_gset_active:Npn \cs_gset:Npn
3180 \char_tmp:NN \char_gset_active:Npx \cs_gset:Npx
3181 \char_tmp:NN \char_set_active_eq:NN \cs_set_eq:NN
3182 \char_tmp:NN \char_gset_active_eq:NN \cs_gset_eq:NN
3183 \group_end:

```

(End definition for `\char_set_active:Npn` and `\char_set_active:Npx`. These functions are documented on page 60.)

`\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.

```

3184 \cs_new_protected_nopar:Npn \peek_execute_branches_N_type:
3185 {
3186   \bool_if:nTF
3187   {
3188     \token_if_eq_catcode_p:NN \l_peek_token \c_group_begin_token ||
3189     \token_if_eq_catcode_p:NN \l_peek_token \c_group_end_token ||
3190     \token_if_eq_meaning_p:NN \l_peek_token \c_space_token
3191   }
3192   { \peek_false:w }
3193   { \peek_true:w }
3194 }
3195 \cs_new_protected_nopar:Npn \peek_N_type:TF
3196 { \peek_token_generic:NNTF \peek_execute_branches_N_type: \scan_stop: }
3197 \cs_new_protected_nopar:Npn \peek_N_type:T
3198 { \peek_token_generic:NNT \peek_execute_branches_N_type: \scan_stop: }
3199 \cs_new_protected_nopar:Npn \peek_N_type:F
3200 { \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.

```

\char_set_catcode:w Primitives renamed.
\char_set_mathcode:w 3201 \*deprecated)
\char_set_lccode:w 3202 \cs_new_eq:NN \char_set_catcode:w \tex_catcode:D
\char_set_uccode:w 3203 \cs_new_eq:NN \char_set_mathcode:w \tex_mathcode:D
\char_set_sfcode:w 3204 \cs_new_eq:NN \char_set_lccode:w \tex_lccode:D
3205 \cs_new_eq:NN \char_set_uccode:w \tex_uccode:D
3206 \cs_new_eq:NN \char_set_sfcode:w \tex_sfcode:D
3207 \end{definition}
(End definition for \char_set_catcode:w. This function is documented on page ??.)

\char_value_catcode:w More w functions we should not have.
\char_show_value_catcode:w 3208 \*deprecated)
\char_value_mathcode:w 3209 \cs_new_nopar:Npn \char_value_catcode:w { \tex_the:D \char_set_catcode:w }
\char_show_value_mathcode:w 3210 \cs_new_nopar:Npn \char_show_value_catcode:w
\char_value_lccode:w 3211 { \tex_showthe:D \char_set_catcode:w }
\char_show_value_lccode:w 3212 \cs_new_nopar:Npn \char_value_mathcode:w { \tex_the:D \char_set_mathcode:w }
\char_value_uccode:w 3213 \cs_new_nopar:Npn \char_show_value_mathcode:w
\char_show_value_uccode:w 3214 { \tex_showthe:D \char_set_mathcode:w }
\char_value_sfcode:w 3215 \cs_new_nopar:Npn \char_value_lccode:w { \tex_the:D \char_set_lccode:w }
\char_show_value_sfcode:w 3216 \cs_new_nopar:Npn \char_show_value_lccode:w
3217 { \tex_showthe:D \char_set_lccode:w }
3218 \cs_new_nopar:Npn \char_value_uccode:w { \tex_the:D \char_set_uccode:w }
3219 \cs_new_nopar:Npn \char_show_value_uccode:w
3220 { \tex_showthe:D \char_set_uccode:w }
3221 \cs_new_nopar:Npn \char_value_sfcode:w { \tex_the:D \char_set_sfcode:w }
3222 \cs_new_nopar:Npn \char_show_value_sfcode:w
3223 { \tex_showthe:D \char_set_sfcode:w }
3224 \end{definition}
(End definition for \char_value_catcode:w. This function is documented on page ??.)

\peek_after:NN The second argument here must be w.
\peek_gafter:NN 3225 \*deprecated)
3226 \cs_new_eq:NN \peek_after:NN \peek_after:Nw
3227 \cs_new_eq:NN \peek_gafter:NN \peek_gafter:Nw
3228 \end{definition}
(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 3229 \*deprecated)
\c_letter_token 3230 \cs_new_eq:NN \c_alignment_tab_token \c_alignment_token
\c_other_char_token 3231 \cs_new_eq:NN \c_math_shift_token \c_math_toggle_token
3232 \cs_new_eq:NN \c_letter_token \c_catcode_letter_token
3233 \cs_new_eq:NN \c_other_char_token \c_catcode_other_token
3234 \end{definition}

```

(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!

```
3235 \*deprecated)
3236 \cs_new_eq:NN \c_active_char_token \c_catcode_active_tl
3237 \*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	3238 *deprecated)	
\char_make_group_end:N	3239 \cs_new_eq:NN \char_make_escape:N	\char_set_catcode_escape:N
\char_make_math_toggle:N	3240 \cs_new_eq:NN \char_make_begin_group:N	\char_set_catcode_group_begin:N
\char_make_alignment:N	3241 \cs_new_eq:NN \char_make_end_group:N	\char_set_catcode_group_end:N
\char_make_end_line:N	3242 \cs_new_eq:NN \char_make_math_shift:N	\char_set_catcode_math_toggle:N
\char_make_parameter:N	3243 \cs_new_eq:NN \char_make_alignment_tab:N	\char_set_catcode_alignment:N
\char_make_math_superscript:N	3244 \cs_new_eq:NN \char_make_end_line:N	\char_set_catcode_end_line:N
\char_make_math_subscript:N	3245 \cs_new_eq:NN \char_make_parameter:N	\char_set_catcode_parameter:N
\char_make_ignore:N	3246 \cs_new_eq:NN \char_make_math_superscript:N	
\char_make_space:N	3247 \char_set_catcode_math_superscript:N	
\char_make_letter:N	3248 \cs_new_eq:NN \char_make_math_subscript:N	
\char_make_other:N	3249 \char_set_catcode_math_subscript:N	
\char_make_active:N	3250 \cs_new_eq:NN \char_make_ignore:N	\char_set_catcode_ignore:N
\char_make_comment:N	3251 \cs_new_eq:NN \char_make_space:N	\char_set_catcode_space:N
\char_make_invalid:N	3252 \cs_new_eq:NN \char_make_letter:N	\char_set_catcode_letter:N
\char_make_escape:n	3253 \cs_new_eq:NN \char_make_other:N	\char_set_catcode_other:N
\char_make_group_begin:n	3254 \cs_new_eq:NN \char_make_active:N	\char_set_catcode_active:N
\char_make_group_end:n	3255 \cs_new_eq:NN \char_make_comment:N	\char_set_catcode_comment:N
\char_make_math_toggle:n	3256 \cs_new_eq:NN \char_make_invalid:N	\char_set_catcode_invalid:N
\char_make_alignment:n	3257 \cs_new_eq:NN \char_make_escape:n	\char_set_catcode_escape:n
\char_make_end_line:n	3258 \cs_new_eq:NN \char_make_begin_group:n	\char_set_catcode_group_begin:n
\char_make_parameter:n	3259 \cs_new_eq:NN \char_make_end_group:n	\char_set_catcode_group_end:n
\char_make_math_superscript:n	3260 \cs_new_eq:NN \char_make_math_shift:n	\char_set_catcode_math_toggle:n
\char_make_math_subscript:n	3261 \cs_new_eq:NN \char_make_alignment_tab:n	\char_set_catcode_alignment:n
\char_make_ignore:n	3262 \cs_new_eq:NN \char_make_end_line:n	\char_set_catcode_end_line:n
\char_make_space:n	3263 \cs_new_eq:NN \char_make_parameter:n	\char_set_catcode_parameter:n
\char_make_letter:n	3264 \cs_new_eq:NN \char_make_math_superscript:n	
\char_make_other:n	3265 \char_set_catcode_math_superscript:n	
\char_make_active:n	3266 \cs_new_eq:NN \char_make_math_subscript:n	
\char_make_comment:n	3267 \char_set_catcode_math_subscript:n	
\char_make_invalid:n	3268 \cs_new_eq:NN \char_make_ignore:n	\char_set_catcode_ignore:n
	3269 \cs_new_eq:NN \char_make_space:n	\char_set_catcode_space:n
	3270 \cs_new_eq:NN \char_make_letter:n	\char_set_catcode_letter:n
	3271 \cs_new_eq:NN \char_make_other:n	\char_set_catcode_other:n
	3272 \cs_new_eq:NN \char_make_active:n	\char_set_catcode_active:n
	3273 \cs_new_eq:NN \char_make_comment:n	\char_set_catcode_comment:n
	3274 \cs_new_eq:NN \char_make_invalid:n	\char_set_catcode_invalid:n
	3275 *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
3276 (*deprecated)
3277 \cs_new_eq:NN \token_if_alignment_tab_p:N \token_if_alignment_p:N
3278 \cs_new_eq:NN \token_if_alignment_tab:NT \token_if_alignment:NT
3279 \cs_new_eq:NN \token_if_alignment_tab:NF \token_if_alignment:NF
3280 \cs_new_eq:NN \token_if_alignment_tab:NTF \token_if_alignment:NTF
3281 \cs_new_eq:NN \token_if_math_shift_p:N \token_if_math_toggle_p:N
3282 \cs_new_eq:NN \token_if_math_shift:NT \token_if_math_toggle:NT
3283 \cs_new_eq:NN \token_if_math_shift:NF \token_if_math_toggle:NF
3284 \cs_new_eq:NN \token_if_math_shift:NTF \token_if_math_toggle:NTF
3285 \cs_new_eq:NN \token_if_other_char_p:N \token_if_other_p:N
3286 \cs_new_eq:NN \token_if_other_char:NT \token_if_other:NT
3287 \cs_new_eq:NN \token_if_other_char:NF \token_if_other:NF
3288 \cs_new_eq:NN \token_if_other_char:NTF \token_if_other:NTF
3289 \cs_new_eq:NN \token_if_active_char_p:N \token_if_active_p:N
3290 \cs_new_eq:NN \token_if_active_char:NT \token_if_active:NT
3291 \cs_new_eq:NN \token_if_active_char:NF \token_if_active:NF
3292 \cs_new_eq:NN \token_if_active_char:NTF \token_if_active:NTF
3293 
      (End definition for \token_if_alignment_tab:N. This function is documented on page ??.)
3294 

```

190 l3int implementation

```

3295 (*initex | package)

      The following test files are used for this code: m3int001,m3int002,m3int03.
3296 (*package)
3297 \ProvidesExplPackage
3298   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
3299 \package_check_loaded_expl:
3300 

```

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 bootstrapping, which is therefore corrected to the “real” version here.

```

3307 <*initex>
3308 \cs_set:Npn \int_eval:n #1 { \int_value:w \int_eval:w #1 \int_eval_end: }
3309 </initex>
3310 <*package>
3311 \cs_new:Npn \int_eval:n #1 { \int_value:w \int_eval:w #1 \int_eval_end: }
3312 </package>

```

(End definition for `\int_eval:n`. This function is documented on page 61.)

`\int_max:nn` Functions for min, max, and absolute value.

```

\int_min:nn 3313 \cs_new:Npn \int_abs:n #1
\int_abs:n 3314 {
3315     \int_value:w
3316     \if_int_compare:w \int_eval:w #1 < \c_zero
3317     -
3318     \fi:
3319     \int_eval:w #1 \int_eval_end:
3320 }
3321 \cs_new:Npn \int_max:nn #1#2
3322 {
3323     \int_value:w \int_eval:w
3324     \if_int_compare:w
3325     \int_eval:w #1 > \int_eval:w #2 \int_eval_end:
3326     #1
3327     \else:
3328     #2
3329     \fi:
3330     \int_eval_end:
3331 }
3332 \cs_new:Npn \int_min:nn #1#2
3333 {
3334     \int_value:w \int_eval:w
3335     \if_int_compare:w
3336     \int_eval:w #1 < \int_eval:w #2 \int_eval_end:
3337     #1
3338     \else:
3339     #2
3340     \fi:
3341     \int_eval_end:
3342 }

```

(End definition for `\int_max:nn`. This function is documented on page 61.)

`\int_div_truncate:nn` As `\int_eval:w` rounds the result of a division we also provide a version that truncates
`\int_div_round:nn` the result. This version is thanks to Heiko Oberdiek: getting things right in all cases is
`\int_mod:nn` not so easy.

```

3343 \cs_new:Npn \int_div_truncate:nn #1#2
3344 {
3345   \int_value:w \int_eval:w
3346   \if_int_compare:w \int_eval:w #1 = \c_zero
3347     0
3348   \else:
3349     ( #1 % )
3350   \if_int_compare:w \int_eval:w #1 < \c_zero
3351     \if_int_compare:w \int_eval:w #2 < \c_zero
3352       - ( #2 + % )
3353     \else:
3354       + ( #2 - % )
3355   \fi:
3356   \else:
3357     \if_int_compare:w \int_eval:w #2 < \c_zero
3358       + ( #2 + % )
3359     \else:
3360       - ( #2 - % )
3361   \fi:
3362   \fi: % ( (
3363     1 ) / 2 )
3364   \fi:
3365   / ( #2 )
3366   \int_eval_end:
3367 }

```

For the sake of completeness:

```

3368 \cs_new:Npn \int_div_round:nn #1#2 { \int_eval:n { ( #1 ) / ( #2 ) } }

```

Finally there's the modulus operation.

```

3369 \cs_new:Npn \int_mod:nn #1#2
3370 {
3371   \int_value:w \int_eval:w
3372   #1 - \int_div_truncate:nn {#1} {#2} * ( #2 )
3373   \int_eval_end:
3374 }

```

(End definition for `\int_div_truncate:nn`. This function is documented on page 62.)

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
3375 <*package>
3376 \cs_new_protected:Npn \int_new:N #1
3377 {
3378   \chk_if_free_cs:N #1
3379   \newcount #1
3380 }
3381 </package>
3382 \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
\c_max_const_int
3383 \cs_new_protected:Npn \int_const:Nn #1#2
3384 {
3385   \int_compare:nNnTF {#2} > \c_minus_one
3386   {
3387     \int_compare:nNnTF {#2} > \c_max_const_int
3388     {
3389       \int_new:N #1
3390       \int_gset:Nn #1 {#2}
3391     }
3392     {
3393       \chk_if_free_cs:N #1
3394       \tex_global:D \int_constdef:Nw #1 =
3395       \int_eval:w #2 \int_eval_end:
3396     }
3397   }
3398   {
3399     \int_new:N #1
3400     \int_gset:Nn #1 {#2}
3401   }
3402 }
3403 \cs_generate_variant:Nn \int_const:Nn { c }
3404 \pdfTeX_if_engine:TF
3405 {
3406   \cs_new_eq:NN \int_constdef:Nw \tex_mathchardef:D
3407   \tex_mathchardef:D \c_max_const_int 32 767 ~
3408 }
3409 {
3410   \cs_new_eq:NN \int_constdef:Nw \tex_chardef:D
3411   \tex_chardef:D \c_max_const_int 1 114 111 ~
3412 }

```

(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
3413 \cs_new_protected:Npn \int_zero:N #1 { #1 = \c_zero }
\int_gzero:N
3414 \cs_new_protected:Npn \int_gzero:N #1 { \tex_global:D #1 = \c_zero }
\int_gzero:c
3415 \cs_generate_variant:Nn \int_zero:N { c }
3416 \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
3417 \cs_new_protected:Npn \int_zero_new:N #1
\int_gzero_new:N
3418 { \cs_if_exist:NTF #1 { \int_zero:N #1 } { \int_new:N #1 } }
\int_gzero_new:c
3419 \cs_new_protected:Npn \int_gzero_new:N #1
3420 { \cs_if_exist:NTF #1 { \int_gzero:N #1 } { \int_new:N #1 } }
3421 \cs_generate_variant:Nn \int_zero_new:N { c }
3422 \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	3423 \cs_new_protected:Npn \int_set_eq:NN #1#2 { #1 = #2 }
\int_set_eq:Nc	3424 \cs_generate_variant:Nn \int_set_eq:NN { c }
\int_set_eq:cc	3425 \cs_generate_variant:Nn \int_set_eq:NN { Nc , cc }
\int_gset_eq:NN	3426 \cs_new_protected:Npn \int_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\int_gset_eq:cN	3427 \cs_generate_variant:Nn \int_gset_eq:NN { c }
\int_gset_eq:Nc	3428 \cs_generate_variant:Nn \int_gset_eq:NN { Nc , cc }
\int_gset_eq:cc	<i>(End definition for \int_set_eq:NN and others. These functions are documented on page ??.)</i>

190.3 Setting and incrementing integers

\int_add:Nn	Adding and subtracting to and from a counter ...
\int_add:cn	3429 \cs_new_protected:Npn \int_add:Nn #1#2
\int_gadd:Nn	3430 { \tex_advance:D #1 by \int_eval:w #2 \int_eval_end: }
\int_gadd:cn	3431 \cs_new_protected:Npn \int_sub:Nn #1#2
\int_sub:Nn	3432 { \tex_advance:D #1 by - \int_eval:w #2 \int_eval_end: }
\int_sub:cn	3433 \cs_new_protected_nopar:Npn \int_gadd:Nn
\int_gsub:Nn	3434 { \tex_global:D \int_add:Nn }
\int_gsub:cn	3435 \cs_new_protected_nopar:Npn \int_gsub:Nn
	3436 { \tex_global:D \int_sub:Nn }
	3437 \cs_generate_variant:Nn \int_add:Nn { c }
	3438 \cs_generate_variant:Nn \int_gadd:Nn { c }
	3439 \cs_generate_variant:Nn \int_sub:Nn { c }
	3440 \cs_generate_variant:Nn \int_gsub:Nn { c }
	<i>(End definition for \int_add:Nn and \int_add:cn. These functions are documented on page ??.)</i>

\int_incr:N	Incrementing and decrementing of integer registers is done with the following functions.
\int_incr:c	3441 \cs_new_protected:Npn \int_incr:N #1
\int_gincr:N	3442 { \tex_advance:D #1 \c_one }
\int_gincr:c	3443 \cs_new_protected:Npn \int_decr:N #1
\int_decr:N	3444 { \tex_advance:D #1 \c_minus_one }
\int_decr:c	3445 \cs_new_protected_nopar:Npn \int_gincr:N
\int_gdecr:N	3446 { \tex_global:D \int_incr:N }
\int_gdecr:c	3447 \cs_new_protected_nopar:Npn \int_gdecr:N
	3448 { \tex_global:D \int_decr:N }
	3449 \cs_generate_variant:Nn \int_incr:N { c }
	3450 \cs_generate_variant:Nn \int_decr:N { c }
	3451 \cs_generate_variant:Nn \int_gincr:N { c }
	3452 \cs_generate_variant:Nn \int_gdecr:N { c }
	<i>(End definition for \int_incr:N and \int_incr:c. These functions are documented on page ??.)</i>

\int_set:Nn	As integers are register-based T _E X will issue an error if they are not defined. Thus there
\int_set:cn	is no need for the checking code seen with token list variables.
\int_gset:Nn	3453 \cs_new_protected:Npn \int_set:Nn #1#2
\int_gset:cn	3454 { #1 ~ \int_eval:w #2\int_eval_end: }
	3455 \cs_new_protected_nopar:Npn \int_gset:Nn { \tex_global:D \int_set:Nn }

```

3456 \cs_generate_variant:Nn \int_set:Nn { c }
3457 \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 3458 \cs_new_eq:NN \int_use:N \tex_the:D
3459 \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 T_EX evaluate this left hand side of the (in)equality.

```

3460 \prg_new_conditional:Npnn \int_compare:n #1 { p , T , F , TF }
3461 { \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.

```

3462 \cs_new:Npn \int_compare_aux:nw #1#2 \q_stop
3463 {
3464   \exp_after:wN \int_compare_aux:Nw
3465   \int_to_roman:w
3466   \if:w #1 -
3467   \else:
3468     -
3469   \fi:
3470   #1#2 \q_mark #1#2 \q_stop
3471 }

```

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.

```

3472 \cs_new:Npn \int_compare_aux:Nw #1#2#3 \q_mark
3473 { \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:

```

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 }

```

So is the one using == we just have to use == in the parameter text.

```

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_true:
3486   \else:
3487   \prg_return_false:
3488   \fi:
3489 }

```

Not equal is just about reversing the truth value.

```

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 }

```

Less than and greater than are also straight forward.

```

3498 \cs_new:cpn { int_compare_<:w } #1 < #2 \q_stop
3499 {
3500   \if_int_compare:w #1 < \int_eval:w #2 \int_eval_end:
3501   \prg_return_true:
3502   \else:
3503   \prg_return_false:
3504   \fi:
3505 }
3506 \cs_new:cpn { int_compare_>:w } #1 > #2 \q_stop
3507 {
3508   \if_int_compare:w #1 > \int_eval:w #2 \int_eval_end:
3509   \prg_return_true:
3510   \else:
3511   \prg_return_false:
3512   \fi:
3513 }

```

The less than or equal operation is just the opposite of the greater than operation. *Vice versa* for less than or equal.

```

3514 \cs_new:cpn { int_compare_<=:w } #1 <= #2 \q_stop
3515 {
3516   \if_int_compare:w #1 > \int_eval:w #2 \int_eval_end:
3517   \prg_return_false:
3518   \else:
3519   \prg_return_true:
3520   \fi:
3521 }
3522 \cs_new:cpn { int_compare_>=:w } #1 >= #2 \q_stop
3523 {
3524   \if_int_compare:w #1 < \int_eval:w #2 \int_eval_end:
3525   \prg_return_false:
3526   \else:
3527   \prg_return_true:
3528   \fi:
3529 }

```

(End definition for \int_compare:n. This function is documented on page ??.)

\int_compare:nNn More efficient but less natural in typing.

```

3530 \prg_new_conditional:Npnn \int_compare:nNn #1#2#3 { p , T , F , TF}
3531 {
3532   \if_int_compare:w \int_eval:w #1 #2 \int_eval:w #3 \int_eval_end:
3533   \prg_return_true:
3534   \else:
3535   \prg_return_false:
3536   \fi:
3537 }

```

(End definition for \int_compare:nNn. This function is documented on page 64.)

\int_if_odd:n A predicate function.

```

\int_if_even:n 3538 \prg_new_conditional:Npnn \int_if_odd:n #1 { p , T , F , TF}
3539 {
3540   \if_int_odd:w \int_eval:w #1 \int_eval_end:
3541   \prg_return_true:
3542   \else:
3543   \prg_return_false:
3544   \fi:
3545 }
3546 \prg_new_conditional:Npnn \int_if_even:n #1 { p , T , F , TF}
3547 {
3548   \if_int_odd:w \int_eval:w #1 \int_eval_end:
3549   \prg_return_false:
3550   \else:
3551   \prg_return_true:
3552   \fi:
3553 }

```

(End definition for \int_if_odd:n. This function is documented on page 64.)

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_until_do:nn
\int_do_while:nn
\int_do_until:nn
3554 \cs_new:Npn \int_while_do:nn #1#2
3555 {
3556   \int_compare:nT {#1}
3557   {
3558     #2
3559     \int_while_do:nn {#1} {#2}
3560   }
3561 }
3562 \cs_new:Npn \int_until_do:nn #1#2
3563 {
3564   \int_compare:nF {#1}
3565   {
3566     #2
3567     \int_until_do:nn {#1} {#2}
3568   }
3569 }
3570 \cs_new:Npn \int_do_while:nn #1#2
3571 {
3572   #2
3573   \int_compare:nT {#1}
3574   { \int_do_while:nn {#1} {#2} }
3575 }
3576 \cs_new:Npn \int_do_until:nn #1#2
3577 {
3578   #2
3579   \int_compare:nF {#1}
3580   { \int_do_until:nn {#1} {#2} }
3581 }
```

(End definition for \int_while_do:nn. This function is documented on page 65.)

`\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
3582 \cs_new:Npn \int_while_do:nNnn #1#2#3#4
3583 {
3584   \int_compare:nNnT {#1} #2 {#3}
3585   {
3586     #4
3587     \int_while_do:nNnn {#1} #2 {#3} {#4}
3588   }
3589 }
3590 \cs_new:Npn \int_until_do:nNnn #1#2#3#4
3591 {
3592   \int_compare:nNnF {#1} #2 {#3}
3593   {
3594     #4
3595     \int_until_do:nNnn {#1} #2 {#3} {#4}
```

```

3596     }
3597   }
3598   \cs_new:Npn \int_do_while:nNnn #1#2#3#4
3599   {
3600     #4
3601     \int_compare:nNnT {#1} #2 {#3}
3602     { \int_do_while:nNnn {#1} #2 {#3} {#4} }
3603   }
3604   \cs_new:Npn \int_do_until:nNnn #1#2#3#4
3605   {
3606     #4
3607     \int_compare:nNnF {#1} #2 {#3}
3608     { \int_do_until:nNnn {#1} #2 {#3} {#4} }
3609   }

```

(End definition for `\int_while_do:nNnn`. This function is documented on page 65.)

190.7 Formatting integers

`\int_to_arabic:n` Nothing exciting here.

```

3610 \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 66.)

`\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.

```

3611 \cs_new:Npn \int_to_symbols:nnn #1#2#3
3612 {
3613   \int_compare:nNnTF {#1} > {#2}
3614   {
3615     \exp_args:NNo \exp_args:No \int_to_symbols_aux:nnnn
3616     {
3617       \prg_case_int:nnn
3618       { 1 + \int_mod:nn { #1 - 1 } {#2} }
3619       {#3} { }
3620     }
3621     {#1} {#2} {#3}
3622   }
3623   { \prg_case_int:nnn {#1} {#3} { } }
3624 }
3625 \cs_new:Npn \int_to_symbols_aux:nnnn #1#2#3#4
3626 {
3627   \exp_args:Nf \int_to_symbols:nnn
3628   { \int_div_truncate:nn { #2 - 1 } {#3} } {#3} {#4}
3629   #1
3630 }

```

(End definition for `\int_to_symbols:nnn`. This function is documented on page 67.)

`\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.

```

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 }
3663 \cs_new:Npn \int_to_Alph:n #1
3664 {
3665   \int_to_symbols:nnn {#1} { 26 }
3666   {
3667     { 1 } { A }
3668     { 2 } { B }
3669     { 3 } { C }
3670     { 4 } { D }
3671     { 5 } { E }
3672     { 6 } { F }
3673     { 7 } { G }
3674     { 8 } { H }
3675     { 9 } { I }
3676     { 10 } { J }

```

```

3677         { 11 } { K }
3678         { 12 } { L }
3679         { 13 } { M }
3680         { 14 } { N }
3681         { 15 } { O }
3682         { 16 } { P }
3683         { 17 } { Q }
3684         { 18 } { R }
3685         { 19 } { S }
3686         { 20 } { T }
3687         { 21 } { U }
3688         { 22 } { V }
3689         { 23 } { W }
3690         { 24 } { X }
3691         { 25 } { Y }
3692         { 26 } { Z }
3693     }
3694 }

```

(End definition for `\int_to_alph:n` and `\int_to_Alph:n`. These functions are documented on page 66.)

```

\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

```

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 `\c_empty_tl`, and feed the absolute value to the next auxiliary function.

```

3695 \cs_new:Npn \int_to_base:nn #1
3696 { \exp_args:Nf \int_to_base_aux_i:nn { \int_eval:n {#1} } }
3697 \cs_new:Npn \int_to_base_aux_i:nn #1#2
3698 {
3699   \int_compare:nNnTF {#1} < \c_zero
3700   { \exp_args:No \int_to_base_aux_ii:nnN { \use_none:n #1 } {#2} - }
3701   { \int_to_base_aux_ii:nnN {#1} {#2} \c_empty_tl }
3702 }

```

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.

```

3703 \cs_new:Npn \int_to_base_aux_ii:nnN #1#2#3
3704 {
3705   \int_compare:nNnTF {#1} < {#2}
3706   { \exp_last_unbraced:Nf #3 { \int_to_letter:n {#1} } }
3707   {
3708     \exp_args:Nf \int_to_base_aux_iii:nnnN
3709     { \int_to_letter:n { \int_mod:nn {#1} {#2} } }
3710     {#1}
3711     {#2}
3712     #3

```

```

3713     }
3714 }
3715 \cs_new:Npn \int_to_base_aux_iii:nnnN #1#2#3#4
3716 {
3717   \exp_args:Nf \int_to_base_aux_ii:nnN
3718     { \int_div_truncate:nn {#2} {#3} }
3719     {#3}
3720     #4
3721     #1
3722 }

```

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:`.

```

3723 \cs_new:Npn \int_to_letter:n #1
3724 {
3725   \exp_after:wN \exp_after:wN
3726   \if_case:w \int_eval:w #1 - \c_ten \int_eval_end:
3727     A
3728   \or: B
3729   \or: C
3730   \or: D
3731   \or: E
3732   \or: F
3733   \or: G
3734   \or: H
3735   \or: I
3736   \or: J
3737   \or: K
3738   \or: L
3739   \or: M
3740   \or: N
3741   \or: O
3742   \or: P
3743   \or: Q
3744   \or: R
3745   \or: S
3746   \or: T
3747   \or: U
3748   \or: V
3749   \or: W
3750   \or: X
3751   \or: Y
3752   \or: Z
3753   \else: \int_value:w \int_eval:w #1 \exp_after:wN \int_eval_end:
3754   \fi:
3755 }

```

(End definition for `\int_to_base:nn`. This function is documented on page 70.)

```

\int_to_binary:n    Wrappers around the generic function.
\int_to_hexadecimal:n 3756 \cs_new:Npn \int_to_binary:n #1
\int_to_octal:n     3757 { \int_to_base:nn {#1} { 2 } }
                    3758 \cs_new:Npn \int_to_hexadecimal:n #1
                    3759 { \int_to_base:nn {#1} { 16 } }
                    3760 \cs_new:Npn \int_to_octal:n #1
                    3761 { \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 67.)

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    3762 \cs_new:Npn \int_to_roman:n #1
\int_to_Roman:n    3763 {
\int_to_roman_aux:N 3764   \exp_after:wN \int_to_roman_aux:N
\int_to_roman_l:w   3765   \int_to_roman:w \int_eval:n {#1} Q
\int_to_roman_c:w   3766 }
\int_to_roman_d:w   3767 \cs_new:Npn \int_to_roman_aux:N #1
\int_to_roman_m:w   3768 {
\int_to_roman_Q:w   3769   \use:c { int_to_roman_ #1 :w }
\int_to_Roman_i:w   3770   \int_to_roman_aux:N
\int_to_Roman_v:w   3771 }
\int_to_Roman_x:w   3772 \cs_new:Npn \int_to_Roman:n #1
\int_to_Roman_l:w   3773 {
\int_to_Roman_c:w   3774   \exp_after:wN \int_to_Roman_aux:N
\int_to_Roman_d:w   3775   \int_to_roman:w \int_eval:n {#1} Q
\int_to_Roman_m:w   3776 }
\int_to_Roman_Q:w   3777 \cs_new:Npn \int_to_Roman_aux:N #1
\int_to_Roman_Q:w   3778 {
                    3779   \use:c { int_to_Roman_ #1 :w }
                    3780   \int_to_Roman_aux:N
                    3781 }
                    3782 \cs_new_nopar:Npn \int_to_roman_i:w { i }
                    3783 \cs_new_nopar:Npn \int_to_roman_v:w { v }
                    3784 \cs_new_nopar:Npn \int_to_roman_x:w { x }
                    3785 \cs_new_nopar:Npn \int_to_roman_l:w { l }
                    3786 \cs_new_nopar:Npn \int_to_roman_c:w { c }
                    3787 \cs_new_nopar:Npn \int_to_roman_d:w { d }
                    3788 \cs_new_nopar:Npn \int_to_roman_m:w { m }
                    3789 \cs_new_nopar:Npn \int_to_roman_Q:w #1 { }
                    3790 \cs_new_nopar:Npn \int_to_Roman_i:w { I }
                    3791 \cs_new_nopar:Npn \int_to_Roman_v:w { V }
                    3792 \cs_new_nopar:Npn \int_to_Roman_x:w { X }
                    3793 \cs_new_nopar:Npn \int_to_Roman_l:w { L }
                    3794 \cs_new_nopar:Npn \int_to_Roman_c:w { C }

```



```

3795 \cs_new_nopar:Npn \int_to_Roman_d:w { D }
3796 \cs_new_nopar:Npn \int_to_Roman_m:w { M }
3797 \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

```

```

3798 \cs_new:Npn \int_get_sign:n #1
3799 {
3800   \int_get_sign_and_digits_aux:nNNN {#1}
3801   \c_true_bool \c_true_bool \c_false_bool
3802 }
3803 \cs_new:Npn \int_get_digits:n #1
3804 {
3805   \int_get_sign_and_digits_aux:nNNN {#1}
3806   \c_true_bool \c_false_bool \c_true_bool
3807 }

```

The auxiliary loops through, finding sign tokens and removing them. The sign itself is carried through as a flag.

```

3808 \cs_new:Npn \int_get_sign_and_digits_aux:nNNN #1#2#3#4
3809 {
3810   \exp_args:Nf \tl_if_head_eq_charcode:nNTF {#1} -
3811   {
3812     \bool_if:NTF #2
3813     {
3814       \int_get_sign_and_digits_aux:oNNN
3815       { \use_none:n #1 } \c_false_bool #3#4
3816     }
3817     {
3818       \int_get_sign_and_digits_aux:oNNN
3819       { \use_none:n #1 } \c_true_bool #3#4
3820     }
3821   }
3822   {
3823     \exp_args:Nf \tl_if_head_eq_charcode:nNTF {#1} +
3824     { \int_get_sign_and_digits_aux:oNNN { \use_none:n #1 } #2#3#4 }
3825     {
3826       \bool_if:NT #3 { \bool_if:NF #2 - }
3827       \bool_if:NT #4 {#1}
3828     }
3829   }
3830 }
3831 \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

```

3832 \cs_new:Npn \int_from_alph:n #1
3833 {
3834   \int_eval:n
3835   {
3836     \int_get_sign:n {#1}
3837     \exp_args:Nf \int_from_alph_aux:n { \int_get_digits:n {#1} }
3838   }
3839 }
3840 \cs_new:Npn \int_from_alph_aux:n #1
3841 { \int_from_alph_aux:nN { 0 } #1 \q_nil }
3842 \cs_new:Npn \int_from_alph_aux:nN #1#2
3843 {
3844   \quark_if_nil:NTF #2
3845   {#1}
3846   {
3847     \exp_args:Nf \int_from_alph_aux:nN
3848     { \int_eval:n { #1 * 26 + \int_from_alph_aux:N #2 } }
3849   }
3850 }
3851 \cs_new:Npn \int_from_alph_aux:N #1
3852 { \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

```

3853 \cs_new:Npn \int_from_base:nn #1#2
3854 {
3855   \int_eval:n
3856   {
3857     \int_get_sign:n {#1}
3858     \exp_args:Nf \int_from_base_aux:nn
3859     { \int_get_digits:n {#1} } {#2}
3860   }
3861 }
3862 \cs_new:Npn \int_from_base_aux:nn #1#2
3863 { \int_from_base_aux:nnN { 0 } { #2 } #1 \q_nil }
3864 \cs_new:Npn \int_from_base_aux:nnN #1#2#3
3865 {
3866   \quark_if_nil:NTF #3
3867   {#1}
3868   {
3869     \exp_args:Nf \int_from_base_aux:nnN
3870     { \int_eval:n { #1 * #2 + \int_from_base_aux:N #3 } }
3871     {#2}

```

```

3872     }
3873 }

```

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.

```

3874 \cs_new:Npn \int_from_base_aux:N #1
3875 {
3876   \int_compare:nNnTF { '#1 } < { 58 }
3877   {#1}
3878   {
3879     \int_eval:n
3880     { '#1 - \int_compare:nNnTF { '#1 } < { 91 } { 55 } { 87 } }
3881   }
3882 }

```

(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.

```

3883 \cs_new:Npn \int_from_binary:n #1
3884 { \int_from_base:nn {#1} \c_two }
3885 \cs_new:Npn \int_from_hexadecimal:n #1
3886 { \int_from_base:nn {#1} \c_sixteen }
3887 \cs_new:Npn \int_from_octal:n #1
3888 { \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 68.)

```

\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.

```

3889 \int_const:cn { c_int_from_roman_i_int } { 1 }
3890 \int_const:cn { c_int_from_roman_v_int } { 5 }
3891 \int_const:cn { c_int_from_roman_x_int } { 10 }
3892 \int_const:cn { c_int_from_roman_l_int } { 50 }
3893 \int_const:cn { c_int_from_roman_c_int } { 100 }
3894 \int_const:cn { c_int_from_roman_d_int } { 500 }
3895 \int_const:cn { c_int_from_roman_m_int } { 1000 }
3896 \int_const:cn { c_int_from_roman_I_int } { 1 }
3897 \int_const:cn { c_int_from_roman_V_int } { 5 }
3898 \int_const:cn { c_int_from_roman_X_int } { 10 }
3899 \int_const:cn { c_int_from_roman_L_int } { 50 }
3900 \int_const:cn { c_int_from_roman_C_int } { 100 }
3901 \int_const:cn { c_int_from_roman_D_int } { 500 }
3902 \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.

```

3903 \cs_new:Npn \int_from_roman:n #1
3904 {
3905   \tl_if_blank:nF {#1}

```

```

3906     {
3907         \exp_after:wN \int_from_roman_end:w
3908         \int_value:w \int_eval:w
3909         \int_from_roman_aux:NN #1 Q \q_stop
3910     }
3911 }
3912 \cs_new:Npn \int_from_roman_aux:NN #1#2
3913 {
3914     \str_if_eq:nnTF {#1} { Q }
3915     {#1#2}
3916     {
3917         \str_if_eq:nnTF {#2} { Q }
3918         {
3919             \cs_if_exist:cF { c_int_from_roman_ #1 _int }
3920             { \int_from_roman_clean_up:w }
3921             +
3922             \use:c { c_int_from_roman_ #1 _int }
3923             #2
3924         }
3925         {
3926             \cs_if_exist:cF { c_int_from_roman_ #1 _int }
3927             { \int_from_roman_clean_up:w }
3928             \cs_if_exist:cF { c_int_from_roman_ #2 _int }
3929             { \int_from_roman_clean_up:w }
3930             \int_compare:nNnTF
3931             { \use:c { c_int_from_roman_ #1 _int } }
3932             <
3933             { \use:c { c_int_from_roman_ #2 _int } }
3934             {
3935                 + \use:c { c_int_from_roman_ #2 _int }
3936                 - \use:c { c_int_from_roman_ #1 _int }
3937                 \int_from_roman_aux:NN
3938             }
3939             {
3940                 + \use:c { c_int_from_roman_ #1 _int }
3941                 \int_from_roman_aux:NN #2
3942             }
3943         }
3944     }
3945 }
3946 \cs_new:Npn \int_from_roman_end:w #1 Q #2 \q_stop
3947 { \tl_if_empty:nTF {#2} {#1} {#2} }
3948 \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
3949 \cs_new_eq:NN \int_show:N \kernel_register_show:N

```

```

3950 \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

3951 \cs_new_protected:Npn \int_show:n #1
3952 { \tex_showthe:D \int_eval:w #1 \int_eval_end: }
      (End definition for \int_show:n. This function is documented on page 69.)

```

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 69.)

\c_zero Again, one in l3basics for obvious reasons.
          (End definition for \c_zero. This function is documented on page 69.)

\c_six Once again, in l3basics.
\c_seven (End definition for \c_six and \c_seven. These functions are documented on page 69.)
\c_twelve
\c_one
\c_sixteen Low-number values not previously defined.
\c_two
3953 \int_const:Nn \c_one { 1 }
\c_three 3954 \int_const:Nn \c_two { 2 }
\c_four 3955 \int_const:Nn \c_three { 3 }
\c_five 3956 \int_const:Nn \c_four { 4 }
\c_eight 3957 \int_const:Nn \c_five { 5 }
\c_nine 3958 \int_const:Nn \c_eight { 8 }
\c_ten 3959 \int_const:Nn \c_nine { 9 }
\c_eleven 3960 \int_const:Nn \c_ten { 10 }
\c_thirteen 3961 \int_const:Nn \c_eleven { 11 }
\c_fourteen 3962 \int_const:Nn \c_thirteen { 13 }
\c_fifteen 3963 \int_const:Nn \c_fourteen { 14 }
3964 \int_const:Nn \c_fifteen { 15 }
          (End definition for \c_one and others. These functions are documented on page 69.)

\c_thirty_two One middling value.
3965 \int_const:Nn \c_thirty_two { 32 }
          (End definition for \c_thirty_two. This function is documented on page 69.)

\c_two_hundred_fifty_five Two classic mid-range integer constants.
\c_two_hundred_fifty_six 3966 \int_const:Nn \c_two_hundred_fifty_five { 255 }
3967 \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 69.)

\c_one_hundred Simple runs of powers of ten.
\c_one_thousand 3968 \int_const:Nn \c_one_hundred { 100 }
\c_ten_thousand 3969 \int_const:Nn \c_one_thousand { 1000 }
3970 \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 69.)

`\c_max_int` The largest number allowed is $2^{31} - 1$
 3971 `\int_const:Nn \c_max_int { 2 147 483 647 }`
 (End definition for `\c_max_int`. This function is documented on page 69.)

190.11 Scratch integers

We provide three local and two global scratch counters, maybe we need more or less.
`\l_tmpa_int` 3972 `\int_new:N \l_tmpa_int`
`\l_tmpb_int` 3973 `\int_new:N \l_tmpb_int`
`\l_tmpc_int` 3974 `\int_new:N \l_tmpc_int`
`\g_tmpa_int` 3975 `\int_new:N \g_tmpa_int`
`\g_tmpb_int` 3976 `\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 70.)

190.12 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

`\int_convert_from_base_ten:nn` Some simple renames.
`\int_convert_to_symbols:nnn` 3977 `{*deprecated}`
`\int_convert_to_base_ten:nn` 3978 `\cs_new_eq:NN \int_convert_from_base_ten:nn \int_to_base:nn`
 3979 `\cs_new_eq:NN \int_convert_to_symbols:nnn \int_to_symbols:nnn`
 3980 `\cs_new_eq:NN \int_convert_to_base_ten:nn \int_from_base:nn`
 3981 `{/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 L^AT_EX 2_ε.
`\int_to_symbol_math:n` 3982 `{*deprecated}`
`\int_to_symbol_text:n` 3983 `\cs_new_nopar:Npn \int_to_symbol:n`
 3984 `{`
 3985 `\scan_align_safe_stop:`
 3986 `\mode_if_math:TF`
 3987 `{ \int_to_symbol_math:n }`
 3988 `{ \int_to_symbol_text:n }`
 3989 `}`
 3990 `\cs_new:Npn \int_to_symbol_math:n #1`
 3991 `{`
 3992 `\int_to_symbols:nnn {#1} { 9 }`
 3993 `{`
 3994 `{ 1 } { }` `*` `}`
 3995 `{ 2 } { }` `\dagger` `}`
 3996 `{ 3 } { }` `\ddagger` `}`
 3997 `{ 4 } { }` `\mathsection` `}`
 3998 `{ 5 } { }` `\mathparagraph` `}`

```

3999      { 6 } { \/ }
4000      { 7 } { ** }
4001      { 8 } { \dagger \dagger }
4002      { 9 } { \ddagger \ddagger }
4003    }
4004  }
4005  \cs_new:Npn \int_to_symbol_text:n #1
4006  {
4007    \int_to_symbols:nnn {#1} { 9 }
4008    {
4009      { 1 } { \textasteriskcentered }
4010      { 2 } { \textdagger }
4011      { 3 } { \textdaggerdbl }
4012      { 4 } { \textsection }
4013      { 5 } { \textparagraph }
4014      { 6 } { \textbardbl }
4015      { 7 } { \textasteriskcentered \textasteriskcentered }
4016      { 8 } { \textdagger \textdagger }
4017      { 9 } { \textdaggerdbl \textdaggerdbl }
4018    }
4019  }
4020  \</deprecated>
      (End definition for \int_to_symbol:n. This function is documented on page ??.)
4021  \</initex | package>

```

191 l3skip implementation

```

4022  \<*initex | package>
4023  \<*package>
4024  \ProvidesExplPackage
4025    {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
4026  \package_check_loaded_expl:
4027  \</package>

```

191.1 Length primitives renamed

```

\if_dim:w Primitives renamed.
\dim_eval:w 4028 \cs_new_eq:NN \if_dim:w \tex_ifdim:D
\dim_eval_end: 4029 \cs_new_eq:NN \dim_eval:w \etex_dimexpr:D
4030 \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 4031 \<*package>
4032 \cs_new_protected:Npn \dim_new:N #1
4033 {

```

```

4034 \chk_if_free_cs:N #1
4035 \newdimen #1
4036 }
4037 \</package>
4038 \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 4039 \cs_new_protected:Npn \dim_zero:N #1 { #1 \c_zero_dim }
\dim_gzero:N 4040 \cs_new_protected:Npn \dim_gzero:N { \tex_global:D \dim_zero:N }
\dim_gzero:c 4041 \cs_generate_variant:Nn \dim_zero:N { c }
4042 \cs_generate_variant:Nn \dim_gzero:N { c }

```

(End definition for \dim_zero:N and \dim_zero:c. These functions are documented on page ??.)

\dim_zero_new:N Create a register if needed, otherwise clear it.

```

\dim_zero_new:c 4043 \cs_new_protected:Npn \dim_zero_new:N #1
\dim_gzero_new:N 4044 { \cs_if_exist:NTF #1 { \dim_zero:N #1 } { \dim_new:N #1 } }
\dim_gzero_new:c 4045 \cs_new_protected:Npn \dim_gzero_new:N #1
4046 { \cs_if_exist:NTF #1 { \dim_gzero:N #1 } { \dim_new:N #1 } }
4047 \cs_generate_variant:Nn \dim_zero_new:N { c }
4048 \cs_generate_variant:Nn \dim_gzero_new:N { c }

```

(End definition for \dim_zero_new:N and others. These functions are documented on page ??.)

191.3 Setting dim variables

\dim_set:Nn Setting dimensions is easy enough.

```

\dim_set:cn 4049 \cs_new_protected:Npn \dim_set:Nn #1#2
\dim_gset:Nn 4050 { #1 ~ \dim_eval:w #2 \dim_eval_end: }
\dim_gset:cn 4051 \cs_new_protected:Npn \dim_gset:Nn { \tex_global:D \dim_set:Nn }
4052 \cs_generate_variant:Nn \dim_set:Nn { c }
4053 \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 4054 \cs_new_protected:Npn \dim_set_eq:NN #1#2 { #1 = #2 }
\dim_set_eq:Nc 4055 \cs_generate_variant:Nn \dim_set_eq:NN { c }
\dim_set_eq:cc 4056 \cs_generate_variant:Nn \dim_set_eq:NN { Nc , cc }
\dim_gset_eq:NN 4057 \cs_new_protected:Npn \dim_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\dim_gset_eq:cN 4058 \cs_generate_variant:Nn \dim_gset_eq:NN { c }
\dim_gset_eq:Nc 4059 \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 only applies to dimensions as skips are not ordered.

```

\dim_set_min:Nn 4060 \cs_new_protected_nopar:Npn \dim_set_max:Nn
\dim_set_min:cn 4061 { \dim_set_max_aux:NNNn < \dim_set:Nn }
\dim_gset_max:Nn 4062 \cs_new_protected_nopar:Npn \dim_gset_max:Nn
\dim_gset_max:cn 4063 { \dim_set_max_aux:NNNn < \dim_gset:Nn }
\dim_gset_min:Nn
\dim_gset_min:cn

```

\dim_set_max_aux:NNNn


```

4064 \cs_new_protected_nopar:Npn \dim_set_min:Nn
4065   { \dim_set_max_aux:NNNn > \dim_set:Nn }
4066 \cs_new_protected_nopar:Npn \dim_gset_min:Nn
4067   { \dim_set_max_aux:NNNn > \dim_gset:Nn }
4068 \cs_new_protected:Npn \dim_set_max_aux:NNNn #1#2#3#4
4069   { \dim_compare:nNnT {#3} #1 {#4} { #2 #3 {#4} } }
4070 \cs_generate_variant:Nn \dim_set_max:Nn { c }
4071 \cs_generate_variant:Nn \dim_gset_max:Nn { c }
4072 \cs_generate_variant:Nn \dim_set_min:Nn { c }
4073 \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
\dim_gadd:Nn 4074 \cs_new_protected:Npn \dim_add:Nn #1#2
\dim_gadd:cn 4075   { \tex_advance:D #1 by \dim_eval:w #2 \dim_eval_end: }
\dim_sub:Nn 4076 \cs_new_protected:Npn \dim_gadd:Nn { \tex_global:D \dim_add:Nn }
\dim_sub:cn 4077 \cs_generate_variant:Nn \dim_add:Nn { c }
\dim_gsub:Nn 4078 \cs_generate_variant:Nn \dim_gadd:Nn { c }
\dim_gsub:cn 4079 \cs_new_protected:Npn \dim_sub:Nn #1#2
4080   { \tex_advance:D #1 by - \dim_eval:w #2 \dim_eval_end: }
4081 \cs_new_protected:Npn \dim_gsub:Nn { \tex_global:D \dim_sub:Nn }
4082 \cs_generate_variant:Nn \dim_sub:Nn { c }
4083 \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.

```

4084 \cs_new:Npn \dim_abs:n #1
4085   {
4086     \dim_use:N
4087       \dim_eval:w
4088         \if_dim:w \dim_eval:w #1 < \c_zero_dim
4089         -
4090         \fi:
4091       \dim_eval:w #1 \dim_eval_end:
4092     \dim_eval_end:
4093   }

```

(End definition for `\dim_abs:n`. This function is documented on page 73.)

`\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.

```

4094 \cs_new:Npn \dim_ratio:nn #1#2
4095   { \dim_ratio_aux:n {#1} / \dim_ratio_aux:n {#2} }
4096 \cs_new:Npn \dim_ratio_aux:n #1
4097   { \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
4098 \prg_new_conditional:Npnn \dim_compare:nNn #1#2#3 { p , T , F , TF }
4099 {
4100   \if_dim:w \dim_eval:w #1 #2 \dim_eval:w #3 \dim_eval_end:
4101   \prg_return_true: \else: \prg_return_false: \fi:
4102 }
```

(End definition for `\dim_compare_p:nNn`. This function is documented on page 74.)

```
\dim_compare:n
\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
```

[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 }
```

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.

```
4103 \prg_new_conditional:Npnn \dim_compare:n #1 { p , T , F , TF }
4104 {
4105   \exp_after:wN \dim_compare_aux:wNN \dim_use:N \dim_eval:w #1
4106   \dim_eval_end:
4107   \prg_return_true:
4108   \else:
4109   \prg_return_false:
4110   \fi:
4111 }
```

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.

```
4112 \exp_args:Nno \use:nn
4113 { \cs_new:Npn \dim_compare_aux:wNN #1 }
4114 { \tl_to_str:n { pt } }
```

```

4115 #2 #3
4116 {
4117   \use:c
4118   {
4119     dim_compare_ #2
4120     \if_meaning:w = #3 = \fi:
4121     :nw
4122   }
4123   { #1 pt } #3
4124 }

```

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.

```

4125 \cs_new:cpn { dim_compare_<:nw } #1 { \if_dim:w #1 < \dim_eval:w }
4126 \cs_new:cpn { dim_compare_=:nw } #1 { \if_dim:w #1 = \dim_eval:w }
4127 \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 `=`.

```

4128 \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.

```

4129 \cs_new:cpn {dim_compare_<=:nw} #1#2 {\reverse_if:N \if_dim:w #1 > \dim_eval:w}
4130 \cs_new:cpn {dim_compare_!=:nw} #1#2 {\reverse_if:N \if_dim:w #1 = \dim_eval:w}
4131 \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
4132 \cs_set:Npn \dim_while_do:nn #1#2
4133 {
4134   \dim_compare:nT {#1}
4135   {
4136     #2
4137     \dim_while_do:nn {#1} {#2}
4138   }
4139 }
4140 \cs_set:Npn \dim_until_do:nn #1#2
4141 {

```

```

4142     \dim_compare:nF {#1}
4143     {
4144         #2
4145         \dim_until_do:nn {#1} {#2}
4146     }
4147 }
4148 \cs_set:Npn \dim_do_while:nn #1#2
4149 {
4150     #2
4151     \dim_compare:nT {#1}
4152     { \dim_do_while:nn {#1} {#2} }
4153 }
4154 \cs_set:Npn \dim_do_until:nn #1#2
4155 {
4156     #2
4157     \dim_compare:nF {#1}
4158     { \dim_do_until:nn {#1} {#2} }
4159 }

```

(End definition for \dim_while_do:nn. This function is documented on page 75.)

\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
4160 \cs_set:Npn \dim_while_do:nNnn #1#2#3#4
4161 {
4162     \dim_compare:nNnT {#1} #2 {#3}
4163     {
4164         #4
4165         \dim_while_do:nNnn {#1} #2 {#3} {#4}
4166     }
4167 }
4168 \cs_set:Npn \dim_until_do:nNnn #1#2#3#4
4169 {
4170     \dim_compare:nNnF {#1} #2 {#3}
4171     {
4172         #4
4173         \dim_until_do:nNnn {#1} #2 {#3} {#4}
4174     }
4175 }
4176 \cs_set:Npn \dim_do_while:nNnn #1#2#3#4
4177 {
4178     #4
4179     \dim_compare:nNnT {#1} #2 {#3}
4180     { \dim_do_while:nNnn {#1} #2 {#3} {#4} }
4181 }
4182 \cs_set:Npn \dim_do_until:nNnn #1#2#3#4
4183 {
4184     #4
4185     \dim_compare:nNnF {#1} #2 {#3}
4186     { \dim_do_until:nNnn {#1} #2 {#3} {#4} }
4187 }

```

(End definition for `\dim_while_do:nNnn`. This function is documented on page 75.)

191.7 Using dim expressions and variables

`\dim_eval:n` Evaluating a dimension expression expandably.

```
4188 \cs_new:Npn \dim_eval:n #1
4189 { \dim_use:N \dim_eval:w #1 \dim_eval_end: }
```

(End definition for `\dim_eval:n`. This function is documented on page 76.)

`\dim_strip_bp:n`

```
4190 \cs_new:Npn \dim_strip_bp:n #1
4191 { \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 83.)

`\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
4192 \cs_new:Npn \dim_strip_pt:n #1
4193 {
4194   \exp_after:wN
4195   \dim_strip_pt:w \dim_use:N \dim_eval:w #1 \dim_eval_end: \q_stop
4196 }
4197 \use:x
4198 {
4199   \cs_new:Npn \exp_not:N \dim_strip_pt:w
4200     ##1 . ##2 \tl_to_str:n { pt } ##3 \exp_not:N \q_stop
4201   {
4202     ##1
4203     \exp_not:N \int_compare:nNnT {##2} > \c_zero
4204     { . ##2 }
4205   }
4206 }
```

(End definition for `\dim_strip_pt:n`. This function is documented on page ??.)

`\dim_use:N` Accessing a $\langle dim \rangle$.

```
\dim_use:c
4207 \cs_new_eq:NN \dim_use:N \tex_the:D
4208 \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
4209 \cs_new_eq:NN \dim_show:N \kernel_register_show:N
4210 \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.

```
4211 \cs_new_protected:Npn \dim_show:n #1
4212 { \tex_showthe:D \dim_eval:w #1 \dim_eval_end: }
      (End definition for \dim_show:n. This function is documented on page 76.)
```

191.9 Constant dimensions

`\c_zero_dim` The source for these depends on whether we are in package mode.

```
\c_max_dim 4213 <*initex>
4214 \dim_new:N \c_zero_dim
4215 \dim_new:N \c_max_dim
4216 \dim_set:Nn \c_max_dim { 16383.99999 pt }
4217 </initex>
4218 <*package>
4219 \cs_new_eq:NN \c_zero_dim \z@
4220 \cs_new_eq:NN \c_max_dim \maxdimen
4221 </package>
      (End definition for \c_zero_dim. This function is documented on page 76.)
```

191.10 Scratch dimensions

`\l_tmpa_dim` We provide three local and two global scratch registers, maybe we need more or less.

```
\l_tmpb_dim 4222 \dim_new:N \l_tmpa_dim
\l_tmpc_dim 4223 \dim_new:N \l_tmpb_dim
\g_tmpa_dim 4224 \dim_new:N \l_tmpc_dim
\g_tmpb_dim 4225 \dim_new:N \g_tmpa_dim
4226 \dim_new:N \g_tmpb_dim
      (End definition for \l_tmpa_dim, \l_tmpb_dim, and \l_tmpc_dim. These functions are documented
on page 77.)
```

191.11 Creating and initialising skip variables

`\skip_new:N` Allocation of a new internal registers.

```
\skip_new:c 4227 <*package>
4228 \cs_new_protected:Npn \skip_new:N #1
4229 {
4230   \chk_if_free_cs:N #1
4231   \newskip #1
4232 }
4233 </package>
4234 \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 4235 \cs_new_protected:Npn \skip_zero:N #1 { #1 \c_zero_skip }
\skip_gzero:N 4236 \cs_new_protected:Npn \skip_gzero:N { \tex_global:D \skip_zero:N }
\skip_gzero:c 4237 \cs_generate_variant:Nn \skip_zero:N { c }
4238 \cs_generate_variant:Nn \skip_gzero:N { c }
```

(End definition for `\skip_zero:N` and `\skip_zero:c`. These functions are documented on page ??.)

```

\skip_zero_new:N Create a register if needed, otherwise clear it.
\skip_zero_new:c 4239 \cs_new_protected:Npn \skip_zero_new:N #1
\skip_gzero_new:N 4240 { \cs_if_exist:NTF #1 { \skip_zero:N #1 } { \skip_new:N #1 } }
\skip_gzero_new:c 4241 \cs_new_protected:Npn \skip_gzero_new:N #1
4242 { \cs_if_exist:NTF #1 { \skip_gzero:N #1 } { \skip_new:N #1 } }
4243 \cs_generate_variant:Nn \skip_zero_new:N { c }
4244 \cs_generate_variant:Nn \skip_gzero_new:N { c }

```

(End definition for `\skip_zero_new:N` and others. These functions are documented on page ??.)

191.12 Setting skip variables

`\skip_set:Nn` Much the same as for dimensions.

```

\skip_set:cn 4245 \cs_new_protected:Npn \skip_set:Nn #1#2
\skip_gset:Nn 4246 { #1 ~ \etex_glueexpr:D #2 \scan_stop: }
\skip_gset:cn 4247 \cs_new_protected:Npn \skip_gset:Nn { \tex_global:D \skip_set:Nn }
4248 \cs_generate_variant:Nn \skip_set:Nn { c }
4249 \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 4250 \cs_new_protected:Npn \skip_set_eq:NN #1#2 { #1 = #2 }
\skip_set_eq:Nc 4251 \cs_generate_variant:Nn \skip_set_eq:NN { c }
\skip_set_eq:cc 4252 \cs_generate_variant:Nn \skip_set_eq:NN { Nc , cc }
\skip_gset_eq:NN 4253 \cs_new_protected:Npn \skip_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\skip_gset_eq:cN 4254 \cs_generate_variant:Nn \skip_gset_eq:NN { c }
\skip_gset_eq:Nc 4255 \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 4256 \cs_new_protected:Npn \skip_add:Nn #1#2
\skip_gadd:Nn 4257 { \tex_advance:D #1 by \etex_glueexpr:D #2 \scan_stop: }
\skip_gadd:cn 4258 \cs_new_protected:Npn \skip_gadd:Nn { \tex_global:D \skip_add:Nn }
\skip_sub:Nn 4259 \cs_generate_variant:Nn \skip_add:Nn { c }
\skip_sub:cn 4260 \cs_generate_variant:Nn \skip_gadd:Nn { c }
\skip_gsub:Nn 4261 \cs_new_protected:Npn \skip_sub:Nn #1#2
\skip_gsub:cn 4262 { \tex_advance:D #1 by - \etex_glueexpr:D #2 \scan_stop: }
4263 \cs_new_protected:Npn \skip_gsub:Nn { \tex_global:D \skip_sub:Nn }
4264 \cs_generate_variant:Nn \skip_sub:Nn { c }
4265 \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:nn` Comparing skips means doing two expansions to make strings, and then testing them. As a result, only equality is tested.

```

4266 \prg_new_conditional:Npnn \skip_if_eq:nn #1#2 { p , T , F , TF }
4267 {
4268   \if_int_compare:w
4269     \pdfTeX_strcmp:D { \skip_eval:n { #1 } } { \skip_eval:n { #2 } }
4270     = \c_zero
4271     \prg_return_true:
4272   \else:
4273     \prg_return_false:
4274   \fi:
4275 }
```

(End definition for \skip_if_eq:nn. This function is documented on page 78.)

`\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.

```

4276 \prg_new_conditional:Npnn \skip_if_infinite_glue:n #1 { p , T , F , TF }
4277 {
4278   \bool_if:nTF
4279   {
4280     \int_compare_p:nNn { \etex_gluestretchorder:D #1 } > \c_zero ||
4281     \int_compare_p:nNn { \etex_glueshrinkorder:D #1 } > \c_zero
4282   }
4283   { \prg_return_true: }
4284   { \prg_return_false: }
4285 }
```

(End definition for \skip_if_infinite_glue:n. This function is documented on page 78.)

191.14 Using skip expressions and variables

`\skip_eval:n` Evaluating a skip expression expandably.

```

4286 \cs_new:Npn \skip_eval:n #1
4287 { \skip_use:N \etex_glueexpr:D #1 \scan_stop: }
(End definition for \skip_eval:n. This function is documented on page 78.)
```

`\skip_use:N` Accessing a $\langle skip \rangle$.

```

\skip_use:c
4288 \cs_new_eq:NN \skip_use:N \tex_the:D
4289 \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 4290 \cs_new_eq:NN \skip_horizontal:N \tex_hskip:D
\skip_horizontal:n 4291 \cs_new:Npn \skip_horizontal:n #1
    \skip_vertical:N 4292 { \skip_horizontal:N \etex_glueexpr:D #1 \scan_stop: }
\skip_vertical:c 4293 \cs_new_eq:NN \skip_vertical:N \tex_vskip:D
\skip_vertical:n 4294 \cs_new:Npn \skip_vertical:n #1
    4295 { \skip_vertical:N \etex_glueexpr:D #1 \scan_stop: }
    4296 \cs_generate_variant:Nn \skip_horizontal:N { c }
    4297 \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 4298 \cs_new_eq:NN \skip_show:N \kernel_register_show:N
    4299 \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.
    4300 \cs_new_protected:Npn \skip_show:n #1
    4301 { \tex_showthe:D \etex_glueexpr:D #1 \scan_stop: }
    (End definition for \skip_show:n. This function is documented on page 79.)

```

191.17 Constant skips

`\c_zero_skip` Skips with no rubber component are just dimensions

```

\c_max_skip 4302 \cs_new_eq:NN \c_zero_skip \c_zero_dim
    4303 \cs_new_eq:NN \c_max_skip \c_max_dim
    (End definition for \c_zero_skip. This function is documented on page 79.)

```

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 4304 \skip_new:N \l_tmpa_skip
\l_tmpc_skip 4305 \skip_new:N \l_tmpb_skip
\g_tmpa_skip 4306 \skip_new:N \l_tmpc_skip
\g_tmpb_skip 4307 \skip_new:N \g_tmpa_skip
    4308 \skip_new:N \g_tmpb_skip
    (End definition for \l_tmpa_skip, \l_tmpb_skip, and \l_tmpc_skip. These functions are docu-
    mented on page 79.)

```

191.19 Creating and initialising muskip variables

`\muskip_new:N` And then we add muskips.

```
\muskip_new:c 4309 <*package>
               4310 \cs_new_protected:Npn \muskip_new:N #1
               4311 {
               4312     \chk_if_free_cs:N #1
               4313     \newmuskip #1
               4314 }
               4315 </package>
               4316 \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 4317 \cs_new_protected:Npn \muskip_zero:N #1
\muskip_gzero:N 4318 { #1 \c_zero_muskip }
\muskip_gzero:c 4319 \cs_new_protected:Npn \muskip_gzero:N { \tex_global:D \muskip_zero:N }
               4320 \cs_generate_variant:Nn \muskip_zero:N { c }
               4321 \cs_generate_variant:Nn \muskip_gzero:N { c }
               (End definition for \muskip_zero:N and \muskip_zero:c. These functions are documented on
               page ??.)
```

`\muskip_zero_new:N` Create a register if needed, otherwise clear it.

```
\muskip_zero_new:c 4322 \cs_new_protected:Npn \muskip_zero_new:N #1
\muskip_gzero_new:N 4323 { \cs_if_exist:NTF #1 { \muskip_zero:N #1 } { \muskip_new:N #1 } }
\muskip_gzero_new:c 4324 \cs_new_protected:Npn \muskip_gzero_new:N #1
               4325 { \cs_if_exist:NTF #1 { \muskip_gzero:N #1 } { \muskip_new:N #1 } }
               4326 \cs_generate_variant:Nn \muskip_zero_new:N { c }
               4327 \cs_generate_variant:Nn \muskip_gzero_new:N { c }
               (End definition for \muskip_zero_new:N and others. These functions are documented on page ??.)
```

191.20 Setting muskip variables

`\muskip_set:Nn` This should be pretty familiar.

```
\muskip_set:cn 4328 \cs_new_protected:Npn \muskip_set:Nn #1#2
\muskip_gset:Nn 4329 { #1 ~ \etex_muexpr:D #2 \scan_stop: }
\muskip_gset:cn 4330 \cs_new_protected:Npn \muskip_gset:Nn { \tex_global:D \muskip_set:Nn }
               4331 \cs_generate_variant:Nn \muskip_set:Nn { c }
               4332 \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 4333 \cs_new_protected:Npn \muskip_set_eq:NN #1#2 { #1 = #2 }
\muskip_set_eq:Nc 4334 \cs_generate_variant:Nn \muskip_set_eq:NN { c }
\muskip_set_eq:cc 4335 \cs_generate_variant:Nn \muskip_set_eq:NN { Nc , cc }
\muskip_gset_eq:NN 4336 \cs_new_protected:Npn \muskip_gset_eq:NN #1#2 { \tex_global:D #1 = #2 }
\muskip_gset_eq:cN 4337 \cs_generate_variant:Nn \muskip_gset_eq:NN { c }
\muskip_gset_eq:Nc 4338 \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`
`\muskip_gadd:Nn` 4339 `\cs_new_protected:Npn \muskip_add:Nn #1#2`
`\muskip_gadd:cn` 4340 `{ \tex_advance:D #1 by \etex_muexpr:D #2 \scan_stop: }`
`\muskip_sub:Nn` 4341 `\cs_new_protected:Npn \muskip_gadd:Nn { \tex_global:D \muskip_add:Nn }`
`\muskip_sub:cn` 4342 `\cs_generate_variant:Nn \muskip_add:Nn { c }`
`\muskip_gsub:Nn` 4343 `\cs_generate_variant:Nn \muskip_gadd:Nn { c }`
`\muskip_gsub:cn` 4344 `\cs_new_protected:Npn \muskip_sub:Nn #1#2`
4345 `{ \tex_advance:D #1 by - \etex_muexpr:D #2 \scan_stop: }`
4346 `\cs_new_protected:Npn \muskip_gsub:Nn { \tex_global:D \muskip_sub:Nn }`
4347 `\cs_generate_variant:Nn \muskip_sub:Nn { c }`
4348 `\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.

4349 `\cs_new:Npn \muskip_eval:n #1`
4350 `{ \muskip_use:N \etex_muexpr:D #1 \scan_stop: }`
(End definition for `\muskip_eval:n`. This function is documented on page 81.)

`\muskip_use:N` Accessing a $\langle muskip \rangle$.

`\muskip_use:c` 4351 `\cs_new_eq:NN \muskip_use:N \tex_the:D`
4352 `\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` 4353 `\cs_new_eq:NN \muskip_show:N \kernel_register_show:N`
4354 `\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.

4355 `\cs_new_protected:Npn \muskip_show:n #1`
4356 `{ \tex_showthe:D \etex_muexpr:D #1 \scan_stop: }`
(End definition for `\muskip_show:n`. This function is documented on page 81.)

191.23 Experimental skip functions

`\skip_split_finite_else_action:nnNN` This macro is useful when performing error checking in certain circumstances. If the `<skip>` register holds finite glue it sets #3 and #4 to the stretch and shrink component, resp. If it holds infinite glue set #3 and #4 to zero and issue the special action #2 which is probably an error message. Assignments are local.

```

4357 \cs_new:Npn \skip_split_finite_else_action:nnNN #1#2#3#4
4358 {
4359   \skip_if_infinite_glue:nTF {#1}
4360   {
4361     #3 = \c_zero_skip
4362     #4 = \c_zero_skip
4363     #2
4364   }
4365   {
4366     #3 = \etex_gluestretch:D #1 \scan_stop:
4367     #4 = \etex_glueshrink:D #1 \scan_stop:
4368   }
4369 }
```

(End definition for `\skip_split_finite_else_action:nnNN`. This function is documented on page 82.)

```

4370 </initex | package>
```

192 l3tl implementation

```

4371 <*initex | package>
4372 <*package>
4373 \ProvidesExplPackage
4374   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
4375 \package_check_loaded_expl:
4376 </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.

`\tl_new:c`

```

4377 \cs_new_protected:Npn \tl_new:N #1
4378 {
4379   \chk_if_free_cs:N #1
4380   \cs_gset_eq:NN #1 \c_empty_tl
4381 }
4382 \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 4383 \cs_new_protected:Npn \tl_const:Nn #1#2
\tl_const:cn 4384 {
\tl_const:cx 4385   \chk_if_free_cs:N #1
               4386   \cs_gset_nopar:Npx #1 { \exp_not:n {#2} }
               4387 }
               4388 \cs_new_protected:Npn \tl_const:Nx #1#2
               4389 {
               4390   \chk_if_free_cs:N #1
               4391   \cs_gset_nopar:Npx #1 {#2}
               4392 }
               4393 \cs_generate_variant:Nn \tl_const:Nn { c }
               4394 \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.

```

               4395 \tl_const:Nn \c_empty_tl { }
               (End definition for \c_empty_tl. This function is documented on page 95.)

```

`\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 4396 \cs_new_protected:Npn \tl_clear:N #1
\tl_gclear:N 4397 { \tl_set_eq:NN #1 \c_empty_tl }
\tl_gclear:c 4398 \cs_new_protected:Npn \tl_gclear:N #1
               4399 { \tl_gset_eq:NN #1 \c_empty_tl }
               4400 \cs_generate_variant:Nn \tl_clear:N { c }
               4401 \cs_generate_variant:Nn \tl_gclear:N { c }
               (End definition for \tl_clear:N and \tl_gclear:N. 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 4402 \cs_new_protected:Npn \tl_clear_new:N #1
\tl_gclear_new:N 4403 { \cs_if_exist:NTF #1 { \tl_clear:N #1 } { \tl_new:N #1 } }
\tl_gclear_new:c 4404 \cs_new_protected:Npn \tl_gclear_new:N #1
               4405 { \cs_if_exist:NTF #1 { \tl_gclear:N #1 } { \tl_new:N #1 } }
               4406 \cs_generate_variant:Nn \tl_clear_new:N { c }
               4407 \cs_generate_variant:Nn \tl_gclear_new:N { c }
               (End definition for \tl_clear_new:N and \tl_gclear_new:N. These functions are documented on
               page ??.)

```

`\tl_set_eq:NN` For setting token list variables equal to each other.

```

\tl_set_eq:Nc 4408 \cs_new_eq:NN \tl_set_eq:NN \cs_set_eq:NN
\tl_set_eq:cN 4409 \cs_new_eq:NN \tl_set_eq:cN \cs_set_eq:cN
\tl_set_eq:cc 4410 \cs_new_eq:NN \tl_set_eq:Nc \cs_set_eq:Nc
\tl_gset_eq:NN 4411 \cs_new_eq:NN \tl_set_eq:cc \cs_set_eq:cc
\tl_gset_eq:Nc 4412 \cs_new_eq:NN \tl_gset_eq:NN \cs_gset_eq:NN
\tl_gset_eq:cN 4413 \cs_new_eq:NN \tl_gset_eq:cN \cs_gset_eq:cN
\tl_gset_eq:Nc 4414 \cs_new_eq:NN \tl_gset_eq:Nc \cs_gset_eq:Nc
\tl_gset_eq:cc 4415 \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

By using `\exp_not:n` token list variables can contain # tokens, which makes the token list registers provided by `TEX` more or less redundant. The `\tl_set:No` version is done “by hand” as it is used quite a lot.

```

\tl_set:Nn 4416 \cs_new_protected:Npn \tl_set:Nn #1#2
\tl_set:NV 4417 { \cs_set_nopar:Npx #1 { \exp_not:n {#2} } }
\tl_set:Nv
\tl_set:No
\tl_set:Nf 4418 \cs_new_protected:Npn \tl_set:No #1#2
\tl_set:Nx 4419 { \cs_set_nopar:Npx #1 { \exp_not:o {#2} } }
\tl_set:cn 4420 \cs_new_protected:Npn \tl_set:Nx #1#2
\tl_set:NV 4421 { \cs_set_nopar:Npx #1 {#2} }
\tl_set:Nv
\tl_set:co 4422 \cs_new_protected:Npn \tl_gset:Nn #1#2
\tl_set:cf 4423 { \cs_gset_nopar:Npx #1 { \exp_not:n {#2} } }
\tl_set:cx 4424 \cs_new_protected:Npn \tl_gset:No #1#2
\tl_gset:Nn 4425 { \cs_gset_nopar:Npx #1 { \exp_not:o {#2} } }
\tl_gset:NV 4426 \cs_new_protected:Npn \tl_gset:Nx #1#2
\tl_gset:Nv 4427 { \cs_gset_nopar:Npx #1 {#2} }
\tl_gset:No 4428 \cs_generate_variant:Nn \tl_set:Nn { NV , Nv , Nf }
\tl_gset:Nf 4429 \cs_generate_variant:Nn \tl_set:Nx { c }
\tl_gset:Nx 4430 \cs_generate_variant:Nn \tl_set:Nn { c , co , cV , cv , cf }
\tl_gset:cn 4431 \cs_generate_variant:Nn \tl_gset:Nn { NV , Nv , Nf }
\tl_gset:NV 4432 \cs_generate_variant:Nn \tl_gset:Nx { c }
\tl_gset:Nv 4433 \cs_generate_variant:Nn \tl_gset:Nn { c , co , cV , cv , cf }
\tl_gset:No
\tl_gset:Nf
\tl_gset:Nx
\tl_gset:cn
\tl_gset:NV
\tl_gset:Nv

```

(End definition for \tl_set:Nn and others. These functions are documented on page ??.)

Adding to the left is done directly to gain a little performance.

```

\tl_put_left:Nn 4434 \cs_new_protected:Npn \tl_put_left:Nn #1#2
\tl_put_left:NV 4435 { \cs_set_nopar:Npx #1 { \exp_not:n {#2} \exp_not:o #1 } }
\tl_put_left:Nv
\tl_put_left:No 4436 \cs_new_protected:Npn \tl_put_left:NV #1#2
\tl_put_left:Nx 4437 { \cs_set_nopar:Npx #1 { \exp_not:V #2 \exp_not:o #1 } }
\tl_put_left:cn 4438 \cs_new_protected:Npn \tl_put_left:No #1#2
\tl_put_left:cV 4439 { \cs_set_nopar:Npx #1 { \exp_not:o {#2} \exp_not:o #1 } }
\tl_put_left:co 4440 \cs_new_protected:Npn \tl_put_left:Nx #1#2
\tl_put_left:cx 4441 { \cs_set_nopar:Npx #1 { #2 \exp_not:o #1 } }
\tl_gput_left:Nn 4442 \cs_new_protected:Npn \tl_gput_left:Nn #1#2
\tl_gput_left:NV 4443 { \cs_gset_nopar:Npx #1 { \exp_not:n {#2} \exp_not:o #1 } }
\tl_gput_left:No 4444 \cs_new_protected:Npn \tl_gput_left:NV #1#2
\tl_gput_left:Nx 4445 { \cs_gset_nopar:Npx #1 { \exp_not:V #2 \exp_not:o #1 } }
\tl_gput_left:cn 4446 \cs_new_protected:Npn \tl_gput_left:No #1#2
\tl_gput_left:cV 4447 { \cs_gset_nopar:Npx #1 { \exp_not:o {#2} \exp_not:o #1 } }
\tl_gput_left:co 4448 \cs_new_protected:Npn \tl_gput_left:Nx #1#2
\tl_gput_left:cx 4449 { \cs_gset_nopar:Npx #1 { #2 \exp_not:o {#1} } }
4450 \cs_generate_variant:Nn \tl_put_left:Nn { c }
4451 \cs_generate_variant:Nn \tl_put_left:NV { c }
4452 \cs_generate_variant:Nn \tl_put_left:No { c }
4453 \cs_generate_variant:Nn \tl_put_left:Nx { c }
4454 \cs_generate_variant:Nn \tl_gput_left:Nn { c }
4455 \cs_generate_variant:Nn \tl_gput_left:NV { c }
4456 \cs_generate_variant:Nn \tl_gput_left:No { c }
4457 \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 4458 \cs_new_protected:Npn \tl_put_right:Nn #1#2
\tl_put_right:No 4459 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:n {#2} } }
\tl_put_right:Nx 4460 \cs_new_protected:Npn \tl_put_right:NV #1#2
\tl_put_right:cn 4461 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:V #2 } }
\tl_put_right:cV 4462 \cs_new_protected:Npn \tl_put_right:No #1#2
\tl_put_right:co 4463 { \cs_set_nopar:Npx #1 { \exp_not:o #1 \exp_not:o {#2} } }
\tl_put_right:cx 4464 \cs_new_protected:Npn \tl_put_right:Nx #1#2
\tl_gput_right:Nn 4465 { \cs_set_nopar:Npx #1 { \exp_not:o #1 #2 } }
\tl_gput_right:NV 4466 \cs_new_protected:Npn \tl_gput_right:Nn #1#2
\tl_gput_right:No 4467 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:n {#2} } }
\tl_gput_right:Nx 4468 \cs_new_protected:Npn \tl_gput_right:NV #1#2
\tl_gput_right:cn 4469 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:V #2 } }
\tl_gput_right:cV 4470 \cs_new_protected:Npn \tl_gput_right:No #1#2
\tl_gput_right:co 4471 { \cs_gset_nopar:Npx #1 { \exp_not:o #1 \exp_not:o {#2} } }
\tl_gput_right:cx 4472 \cs_new_protected:Npn \tl_gput_right:Nx #1#2
4473 { \cs_gset_nopar:Npx #1 { \exp_not:o {#1} #2 } }
4474 \cs_generate_variant:Nn \tl_put_right:Nn { c }
4475 \cs_generate_variant:Nn \tl_put_right:NV { c }
4476 \cs_generate_variant:Nn \tl_put_right:No { c }
4477 \cs_generate_variant:Nn \tl_put_right:Nx { c }
4478 \cs_generate_variant:Nn \tl_gput_right:Nn { c }
4479 \cs_generate_variant:Nn \tl_gput_right:NV { c }
4480 \cs_generate_variant:Nn \tl_gput_right:No { c }
4481 \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.

```

4482 \group_begin:
4483 \tex_lccode:D '\A = '\@ \scan_stop:
4484 \tex_lccode:D '\B = '\@ \scan_stop:
4485 \tex_catcode:D '\A = 8 \scan_stop:
4486 \tex_catcode:D '\B = 3 \scan_stop:
4487 \tex_lowercase:D
4488 {
4489   \group_end:
4490   \tl_const:Nn \c_tl_rescan_marker_tl { A B }
4491 }

```

(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 TeX error occurs:

! File ended while scanning definition of ...

```

\tl_set_rescan:Nno
\tl_set_rescan:Nnx
\tl_set_rescan:cn
\tl_set_rescan:cno
\tl_set_rescan:cnx
\tl_gset_rescan:Nnn
\tl_gset_rescan:Nno
\tl_gset_rescan:Nnx
\tl_gset_rescan:cn
\tl_gset_rescan:cno
\tl_gset_rescan:cnx
\tl_rescan:nn
\tl_set_rescan_aux:NNnn

```

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”.

```

4492 \cs_new_protected_nopar:Npn \tl_set_rescan:Nnn
4493 { \tl_set_rescan_aux:NNnn \tl_set:Nn }
4494 \cs_new_protected_nopar:Npn \tl_gset_rescan:Nnn
4495 { \tl_set_rescan_aux:NNnn \tl_gset:Nn }
4496 \cs_new_protected_nopar:Npn \tl_rescan:nn
4497 { \tl_set_rescan_aux:NNnn \prg_do_nothing: \use:n }
4498 \cs_new_protected:Npn \tl_set_rescan_aux:NNnn #1#2#3#4
4499 {
4500   \group_begin:
4501     \exp_args:No \etex_everyeof:D { \c_tl_rescan_marker_tl \exp_not:N }
4502     \tex_endlinechar:D \c_minus_one
4503     \tex_newlinechar:D \c_minus_one
4504     #3
4505     \use:x
4506     {
4507       \group_end:
4508       #1 \exp_not:N #2
4509       {
4510         \exp_after:wN \tl_rescan_aux:w
4511         \exp_after:wN \prg_do_nothing:
4512         \etex_scantokens:D {#4}
4513       }
4514     }
4515   }
4516   \use:x
4517   {
4518     \cs_new:Npn \exp_not:N \tl_rescan_aux:w ##1
4519       \c_tl_rescan_marker_tl
4520       { \exp_not:N \exp_not:o { ##1 } }
4521   }
4522   \cs_generate_variant:Nn \tl_set_rescan:Nnn { Nno , Nnx }
4523   \cs_generate_variant:Nn \tl_set_rescan:Nnn { c , cno , cnx }
4524   \cs_generate_variant:Nn \tl_gset_rescan:Nnn { Nno , Nnx }
4525   \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 4526 \cs_new_eq:NN \tl_to_lowercase:n \tex_lowercase:D
4527 \cs_new_eq:NN \tl_to_uppercase:n \tex_uppercase:D

```


(End definition for `\tl_to_lowercase:n`. This function is documented on page 87.)

192.5 Modifying token list variables

`\tl_replace_all:Nnn` All of the replace functions are based on `\tl_replace_aux:NNNnn`, whose arguments are:
`\tl_replace_all:cnn` $\langle function \rangle$, $\langle \text{tl_}(g)\text{set:Nx} \rangle$, $\langle \text{tl var} \rangle$, $\langle search\ tokens \rangle$, $\langle replacement\ tokens \rangle$.
`\tl_greplace_all:Nnn` 4528 `\cs_new_protected_nopar:Npn \tl_replace_once:Nnn`
`\tl_greplace_all:cnn` 4529 `{ \tl_replace_aux:NNNnn \tl_replace_once_aux: \tl_set:Nx }`
`\tl_replace_once:Nnn` 4530 `\cs_new_protected_nopar:Npn \tl_greplace_once:Nnn`
`\tl_replace_once:cnn` 4531 `{ \tl_replace_aux:NNNnn \tl_replace_once_aux: \tl_gset:Nx }`
`\tl_greplace_once:Nnn` 4532 `\cs_new_protected_nopar:Npn \tl_replace_all:Nnn`
`\tl_greplace_once:cnn` 4533 `{ \tl_replace_aux:NNNnn \tl_replace_all_aux: \tl_set:Nx }`
`\tl_replace_aux:NNNnn` 4534 `\cs_new_protected_nopar:Npn \tl_greplace_all:Nnn`
`\tl_replace_aux:ii:w` 4535 `{ \tl_replace_aux:NNNnn \tl_replace_all_aux: \tl_gset:Nx }`
`\tl_replace_all_aux:` 4536 `\cs_generate_variant:Nn \tl_replace_once:Nnn { c }`
`\tl_replace_once_aux:` 4537 `\cs_generate_variant:Nn \tl_greplace_once:Nnn { c }`
`\tl_replace_once_aux_end:w` 4538 `\cs_generate_variant:Nn \tl_replace_all:Nnn { c }`
4539 `\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 $\langle search\ tokens \rangle$ by $\langle replacement\ tokens \rangle$. To avoid runaway arguments, we expand something like `\tl_tmp:w \langle token list \rangle \q_mark \langle search\ tokens \rangle \q_stop`, repeating until the end. How do we detect that we have reached the last occurrence of $\langle search\ tokens \rangle$? 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 $\langle replacement\ tokens \rangle$, 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.

```
4540 \cs_new_protected:Npn \tl_replace_aux:NNNnn #1#2#3#4#5
4541 {
4542   \tl_if_empty:nTF {#4}
4543   {
4544     \msg_kernel_error:nxx { tl } { empty-search-pattern }
4545     { \tl_to_str:n {#5} }
4546   }
4547   {
4548     \group_align_safe_begin:
4549     \cs_set:Npx \tl_tmp:w ##1##2 #4
4550     {
4551       ##2
4552       \exp_not:N \q_mark
4553       \exp_not:N \use_none_delimit_by_q_stop:w
4554       \exp_not:n { \exp_not:n {#5} }
```

```

4555         ##1
4556     }
4557     \group_align_safe_end:
4558     #2 #3
4559     {
4560         \exp_after:wN #1
4561         #3 \q_mark #4 \q_stop
4562     }
4563 }
4564 }
4565 \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.

```

4566 \cs_new:Npn \tl_replace_all_aux:
4567 {
4568     \exp_after:wN \tl_replace_aux_ii:w
4569     \tl_tmp:w \tl_replace_all_aux: \prg_do_nothing:
4570 }
4571 \cs_new_nopar:Npn \tl_replace_once_aux:
4572 {
4573     \exp_after:wN \tl_replace_aux_ii:w
4574     \tl_tmp:w { \tl_replace_once_aux_end:w \prg_do_nothing: } \prg_do_nothing:
4575 }
4576 \cs_new:Npn \tl_replace_once_aux_end:w #1 \q_mark #2 \q_stop
4577 { \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 4578 \cs_new_protected:Npn \tl_remove_once:Nn #1#2
\tl_gremove_once:Nn 4579 { \tl_replace_once:Nnn #1 {#2} { } }
\tl_gremove_once:cn 4580 \cs_new_protected:Npn \tl_gremove_once:Nn #1#2
4581 { \tl_greplace_once:Nnn #1 {#2} { } }
4582 \cs_generate_variant:Nn \tl_remove_once:Nn { c }
4583 \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 4584 \cs_new_protected:Npn \tl_remove_all:Nn #1#2
\tl_gremove_all:Nn 4585 { \tl_replace_all:Nnn #1 {#2} { } }
\tl_gremove_all:cn 4586 \cs_new_protected:Npn \tl_gremove_all:Nn #1#2
4587 { \tl_greplace_all:Nnn #1 {#2} { } }

```

```

4588 \cs_generate_variant:Nn \tl_remove_all:Nn { c }
4589 \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.

```

4590 \prg_new_conditional:Npnn \tl_if_blank:n #1 { p , T , F , TF }
4591 { \tl_if_empty_return:o { \use_none:n #1 ? } }
4592 \cs_generate_variant:Nn \tl_if_blank_p:n { V }
4593 \cs_generate_variant:Nn \tl_if_blank:nT { V }
4594 \cs_generate_variant:Nn \tl_if_blank:nF { V }
4595 \cs_generate_variant:Nn \tl_if_blank:nTF { V }
4596 \cs_generate_variant:Nn \tl_if_blank_p:n { o }
4597 \cs_generate_variant:Nn \tl_if_blank:nT { o }
4598 \cs_generate_variant:Nn \tl_if_blank:nF { o }
4599 \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`

```

4600 \prg_new_conditional:Npnn \tl_if_empty:N #1 { p , T , F , TF }
4601 {
4602   \if_meaning:w #1 \c_empty_tl
4603     \prg_return_true:
4604   \else:
4605     \prg_return_false:
4606   \fi:
4607 }
4608 \cs_generate_variant:Nn \tl_if_empty_p:N { c }
4609 \cs_generate_variant:Nn \tl_if_empty:NT { c }
4610 \cs_generate_variant:Nn \tl_if_empty:NF { c }
4611 \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.

```

4612 \prg_new_conditional:Npnn \tl_if_empty:n #1 { p , TF , T , F }
4613 {
4614   \exp_after:wN \if_meaning:w \exp_after:wN \q_nil \tl_to_str:n {#1} \q_nil
4615   \prg_return_true:
4616   \else:
4617     \prg_return_false:
4618   \fi:
4619 }
4620 \cs_generate_variant:Nn \tl_if_empty_p:n { V }
4621 \cs_generate_variant:Nn \tl_if_empty:nTF { V }
4622 \cs_generate_variant:Nn \tl_if_empty:nT { V }
4623 \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.

```

4624 \cs_new:Npn \tl_if_empty_return:o #1
4625 {
4626   \exp_after:wN \if_meaning:w \exp_after:wN \q_nil
4627   \tl_to_str:n \exp_after:wN {#1} \q_nil
4628   \prg_return_true:
4629   \else:
4630     \prg_return_false:
4631   \fi:
4632 }
4633 \prg_new_conditional:Npnn \tl_if_empty:o #1 { p , TF , T , F }
4634 { \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.

```

\tl_if_eq:Nc
\tl_if_eq:cN
\tl_if_eq:cc
4635 \prg_new_conditional:Npnn \tl_if_eq:NN #1#2 { p , T , F , TF }
4636 {
4637   \if_meaning:w #1 #2
4638   \prg_return_true:
4639   \else:
4640     \prg_return_false:
4641   \fi:
4642 }
4643 \cs_generate_variant:Nn \tl_if_eq_p:NN { Nc , c , cc }
4644 \cs_generate_variant:Nn \tl_if_eq:NNTF { Nc , c , cc }

```

```

4645 \cs_generate_variant:Nn \tl_if_eq:NNT { Nc , c , cc }
4646 \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_internal_a_tl 4647 \prg_new_protected_conditional:Npnn \tl_if_eq:nn #1#2 { T , F , TF }
\l_tl_internal_b_tl 4648 {
4649   \group_begin:
4650     \tl_set:Nn \l_tl_internal_a_tl {#1}
4651     \tl_set:Nn \l_tl_internal_b_tl {#2}
4652     \if_meaning:w \l_tl_internal_a_tl \l_tl_internal_b_tl
4653     \group_end:
4654     \prg_return_true:
4655   \else:
4656     \group_end:
4657     \prg_return_false:
4658   \fi:
4659 }
4660 \tl_new:N \l_tl_internal_a_tl
4661 \tl_new:N \l_tl_internal_b_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)`.

```

4662 \cs_new_protected_nopar:Npn \tl_if_in:NnT { \exp_args:No \tl_if_in:nnT }
4663 \cs_new_protected_nopar:Npn \tl_if_in:NnF { \exp_args:No \tl_if_in:nnF }
4664 \cs_new_protected_nopar:Npn \tl_if_in:NnTF { \exp_args:No \tl_if_in:nnTF }
4665 \cs_generate_variant:Nn \tl_if_in:NnT { c }
4666 \cs_generate_variant:Nn \tl_if_in:NnF { c }
4667 \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 T_EX limitations on what can delimit a parameter, hence we are safe. Using two brace groups makes the test work also for empty arguments.

```

4668 \prg_new_protected_conditional:Npnn \tl_if_in:nn #1#2 { T , F , TF }
4669 {
4670   \cs_set:Npn \tl_tmp:w ##1 #2 { }
4671   \tl_if_empty:oTF { \tl_tmp:w #1 {} } {} #2 }
4672   { \prg_return_false: } { \prg_return_true: }
4673 }

```

```

4674 \cs_generate_variant:Nn \tl_if_in:nnT { V , o , no }
4675 \cs_generate_variant:Nn \tl_if_in:nnF { V , o , no }
4676 \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_aux:Nn
4677 \cs_new:Npn \tl_map_function:nN #1#2
4678 {
4679   \tl_map_function_aux:Nn #2 #1
4680   \q_recursion_tail
4681   \prg_break_point:n { }
4682 }
4683 \cs_new_nopar:Npn \tl_map_function:NN
4684 { \exp_args:No \tl_map_function:nN }
4685 \cs_new:Npn \tl_map_function_aux:Nn #1#2
4686 {
4687   \quark_if_recursion_tail_break:n {#2}
4688   #1 {#2} \tl_map_function_aux:Nn #1
4689 }
4690 \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.

```

4691 \cs_new_protected:Npn \tl_map_inline:nn #1#2
4692 {
4693   \int_gincr:N \g_prg_map_int
4694   \cs_gset:cpn { tl_map_inline_ \int_use:N \g_prg_map_int :n }
4695   ##1 {#2}
4696   \exp_args:Nc \tl_map_function_aux:Nn
4697   { tl_map_inline_ \int_use:N \g_prg_map_int :n }
4698   #1 \q_recursion_tail
4699   \prg_break_point:n { \int_gdecr:N \g_prg_map_int }
4700 }
4701 \cs_new_protected:Npn \tl_map_inline:Nn
4702 { \exp_args:No \tl_map_inline:nn }
4703 \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 <token list> <temp> <action> assigns <temp> to each element and executes <action>.

```

\tl_map_variable:Nnn
\tl_map_variable:cNn
4704 \cs_new_protected:Npn \tl_map_variable:nNn #1#2#3
4705 {

```

```

4706 \tl_map_variable_aux:Nnn #2 {#3} #1
4707 \q_recursion_tail
4708 \prg_break_point:n { }
4709 }
4710 \cs_new_protected_nopar:Npn \tl_map_variable:NNn
4711 { \exp_args:No \tl_map_variable:nNn }
4712 \cs_new_protected:Npn \tl_map_variable_aux:Nnn #1#2#3
4713 {
4714 \tl_set:Nn #1 {#3}
4715 \quark_if_recursion_tail_break:N #1
4716 \use:n {#2}
4717 \tl_map_variable_aux:Nnn #1 {#2}
4718 }
4719 \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 4720 \cs_new_eq:NN \tl_map_break: \prg_map_break:
4721 \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.

```

4722 \cs_new_eq:NN \tl_to_str:n \etex_detokenize:D

```

(End definition for \tl_to_str:n. This function is documented on page 90.)

`\tl_to_str:N` These functions return the replacement text of a token list as a string.

```

\tl_to_str:c 4723 \cs_new:Npn \tl_to_str:N #1 { \etex_detokenize:D \exp_after:wN {#1} }
4724 \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.

```

4725 \cs_new:Npn \tl_use:N #1
4726 {
4727 \cs_if_exist:NTF #1 {#1}
4728 { \msg_expandable_kernel_error:nnn { kernel } { bad-var } {#1} }
4729 }
4730 \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 the list are read as a single element. Spaces are ignored. `\tl_length_aux:n` grabs the element and replaces it by +1. The 0 to ensure it works on an empty list.

```

\tl_length:N      4731 \cs_new:Npn \tl_length:n #1
\tl_length:c      4732 {
\tl_length_aux:n  4733   \int_eval:n
                  4734   { 0 \tl_map_function:nN {#1} \tl_length_aux:n }
                  4735 }
\tl_length:N      4736 \cs_new:Npn \tl_length:N #1
\tl_length:c      4737 {
                  4738   \int_eval:n
                  4739   { 0 \tl_map_function:NN #1 \tl_length_aux:n }
                  4740 }
\tl_length_aux:n  4741 \cs_new:Npn \tl_length_aux:n #1 { + \c_one }
                  4742 \cs_generate_variant:Nn \tl_length:n { V , o }
                  4743 \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_stop`.

```

\tl_reverse_items_aux:nwNwn 4744 \cs_new:Npn \tl_reverse_items:n #1
\tl_reverse_items_aux:wn     4745 {
                              4746   \tl_reverse_items_aux:nwNwn #1 ?
                              4747   \q_mark \tl_reverse_items_aux:nwNwn
                              4748   \q_mark \tl_reverse_items_aux:wn
                              4749   \q_stop { }
                              4750 }
                              4751 \cs_new:Npn \tl_reverse_items_aux:nwNwn #1 #2 \q_mark #3 #4 \q_stop #5
                              4752 {
                              4753   #3 #2
                              4754   \q_mark \tl_reverse_items_aux:nwNwn
                              4755   \q_mark \tl_reverse_items_aux:wn
                              4756   \q_stop { {#1} #5 }
                              4757 }
                              4758 \cs_new:Npn \tl_reverse_items_aux:wn #1 \q_stop #2
                              4759 { \exp_not:o { \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, and to get spaces at odd places in the definitions, we nest those in `\tl_tmp:w`, which then receives a single space as its argument: #1 is `_`. Removing leading spaces is done with `\tl_trim_spaces_aux_i:w`, which loops until `\q_mark_` matches the end of the token list: then ##1 is the token list and ##3 is `\tl_trim_spaces_aux_ii:w`. This hands the relevant tokens to the loop `\tl_trim_spaces_aux_iii:w`, responsible for trimming trailing spaces. The end is reached when `_` `\q_nil` matches the one present in the definition of `\tl_trim_spaces:n`. Then `\tl_trim_spaces_aux_iv:w` puts the token list

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.

```

4760 \cs_set:Npn \tl_tmp:w #1
4761 {
4762   \cs_new:Npn \tl_trim_spaces:n ##1
4763   {
4764     \etex_unexpanded:D
4765     \tl_trim_spaces_aux_i:w
4766     \q_mark
4767     ##1
4768     \q_nil
4769     \q_mark #1 { }
4770     \q_mark \tl_trim_spaces_aux_ii:w
4771     \tl_trim_spaces_aux_iii:w
4772     #1 \q_nil
4773     \tl_trim_spaces_aux_iv:w
4774     \q_stop
4775   }
4776   \cs_new:Npn \tl_trim_spaces_aux_i:w ##1 \q_mark #1 ##2 \q_mark ##3
4777   {
4778     ##3
4779     \tl_trim_spaces_aux_i:w
4780     \q_mark
4781     ##2
4782     \q_mark #1 {##1}
4783   }
4784   \cs_new:Npn \tl_trim_spaces_aux_ii:w ##1 \q_mark \q_mark ##2
4785   {
4786     \tl_trim_spaces_aux_iii:w
4787     ##2
4788   }
4789   \cs_new:Npn \tl_trim_spaces_aux_iii:w ##1 #1 \q_nil ##2
4790   {
4791     ##2
4792     ##1 \q_nil
4793     \tl_trim_spaces_aux_iii:w
4794   }
4795   \cs_new:Npn \tl_trim_spaces_aux_iv:w ##1 \q_nil ##2 \q_stop
4796   { \exp_after:wN { \use_none:n ##1 } }
4797 }
4798 \tl_tmp:w { ~ }
4799 \cs_new_protected:Npn \tl_trim_spaces:N #1
4800 { \tl_set:Nx #1 { \exp_after:wN \tl_trim_spaces:n \exp_after:wN {#1} } }
4801 \cs_new_protected:Npn \tl_gtrim_spaces:N #1
4802 { \tl_gset:Nx #1 { \exp_after:wN \tl_trim_spaces:n \exp_after:wN {#1} } }
4803 \cs_generate_variant:Nn \tl_trim_spaces:N { c }
4804 \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:V` `\tl_head:n` and `\tl_tail:n` ensure that a blank argument gives an empty result. The result is returned within the `\unexpanded` primitive.

```

\tl_head:f 4805 \cs_new:Npn \tl_head:w #1#2 \q_stop {#1}
\tl_head:w 4806 \cs_new:Npn \tl_tail:w #1#2 \q_stop {#2}
\tl_head:n 4807 \cs_new:Npn \tl_head:n #1
\tl_tail:n 4808 { \etex_unexpanded:D \exp_after:wN { \tl_head:w #1 { } \q_stop } }
\tl_tail:V 4809 \cs_new:Npn \tl_tail:n #1
\tl_tail:v 4810 { \etex_unexpanded:D \tl_tail_aux:w #1 \q_mark { } \q_mark \q_stop }
\tl_tail:f 4811 \cs_new:Npn \tl_tail_aux:w #1 #2 \q_mark #3 \q_stop { {#2} }
\tl_tail:w 4812 \cs_generate_variant:Nn \tl_head:n { V , v , f }
4813 \cs_generate_variant:Nn \tl_tail:n { V , v , f }
```

(End definition for `\tl_head:n` and others. These functions are documented on page 93.)

`\str_head:n` After `\tl_to_str:n`, we have a list of character tokens, all with category code 12, except the space, which has category code 10. Directly using `\tl_head:w` would thus lose leading spaces. Instead, we take an argument delimited by an explicit space, and then only use `\str_head_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.

```

4814 \cs_new:Npn \str_head:n #1
4815 {
4816   \exp_after:wN \str_head_aux:w
4817   \tl_to_str:n {#1}
4818   { { } } ~ \q_stop
4819 }
4820 \cs_new:Npn \str_head_aux:w #1 ~ %
4821 { \tl_head:w #1 { ~ } }
4822 \cs_new:Npn \str_tail:n #1
4823 {
4824   \exp_after:wN \str_tail_aux:w
4825   \reverse_if:N \if_charcode:w
4826   \scan_stop: \tl_to_str:n {#1} X X \q_stop
4827 }
4828 \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 (begin-group token), or when it is an explicit space, with category code 10 and character code 32.
`\tl_if_head_eq_charcode:nN`
`\tl_if_head_eq_charcode:fN`
`\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).

```

4829 \prg_new_conditional:Npnn \tl_if_head_eq_charcode:nN #1#2 { p , T , F , TF }
4830 {
4831   \if_charcode:w
4832     \exp_not:N #2
4833     \tl_if_head_N_type:nTF { #1 ? }
4834     { \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop }
4835     { \str_head:n {#1} }
4836   \prg_return_true:
4837   \else:
4838     \prg_return_false:
4839   \fi:
4840 }
4841 \cs_generate_variant:Nn \tl_if_head_eq_charcode_p:nN { f }
4842 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNTF { f }
4843 \cs_generate_variant:Nn \tl_if_head_eq_charcode:nNT { f }
4844 \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`.

```

4845 \prg_new_conditional:Npnn \tl_if_head_eq_catcode:nN #1 #2 { p , T , F , TF }
4846 {
4847   \if_catcode:w
4848     \exp_not:N #2
4849     \tl_if_head_N_type:nTF { #1 ? }
4850     { \exp_after:wN \exp_not:N \tl_head:w #1 \q_nil \q_stop }
4851     {
4852       \tl_if_head_group:nTF {#1}
4853       { \c_group_begin_token }
4854       { \c_space_token }
4855     }
4856   \prg_return_true:
4857   \else:

```

```

4858     \prg_return_false:
4859     \fi:
4860 }

```

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.

```

4861 \prg_new_conditional:Npnn \tl_if_head_eq_meaning:nN #1#2 { p , T , F , TF }
4862 {
4863   \tl_if_head_N_type:nTF { #1 ? }
4864   { \tl_if_head_eq_meaning_aux_normal:nN }
4865   { \tl_if_head_eq_meaning_aux_special:nN }
4866   {#1} #2
4867 }
4868 \cs_new:Npn \tl_if_head_eq_meaning_aux_normal:nN #1 #2
4869 {
4870   \exp_after:wN \if_meaning:w \tl_head:w #1 \q_nil \q_stop #2
4871   \prg_return_true:
4872   \else:
4873     \prg_return_false:
4874   \fi:
4875 }
4876 \cs_new:Npn \tl_if_head_eq_meaning_aux_special:nN #1 #2
4877 {
4878   \if_charcode:w \str_head:n {#1} \exp_not:N #2
4879   \exp_after:wN \use:n
4880   \else:
4881     \prg_return_false:
4882     \exp_after:wN \use_none:n
4883   \fi:
4884   {
4885     \if_catcode:w \exp_not:N #2
4886     \tl_if_head_group:nTF {#1}
4887     { \c_group_begin_token }
4888     { \c_space_token }
4889     \prg_return_true:
4890     \else:
4891       \prg_return_false:
4892     \fi:
4893   }
4894 }

```

(End definition for `\tl_if_head_eq_meaning:nN`. This function is documented on page 93.)

`\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”.

```

4895 \prg_new_conditional:Npnn \tl_if_head_N_type:n #1 { p , T , F , TF }
4896 {
4897   \str_if_eq_return:xx
4898   { \exp_not:o { \use:n #1 { } } }
4899   { \exp_not:n { #1 { } } }
4900 }

```

(End definition for `\tl_if_head_N_type:n`. This function is documented on page 94.)

`\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

```

4901 \prg_new_conditional:Npnn \tl_if_head_group:n #1 { p , T , F , TF }
4902 {
4903   \if_catcode:w *
4904   \exp_after:wN \use_none:n
4905   \exp_after:wN {
4906     \exp_after:wN {
4907       \token_to_str:N #1 ?
4908     }
4909   }
4910   *
4911   \prg_return_false:
4912   \else:
4913     \prg_return_true:
4914   \fi:
4915 }

```

(End definition for `\tl_if_head_group:n`. This function is documented on page 94.)

`\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.

```

4916 \prg_new_conditional:Npnn \tl_if_head_space:n #1 { p , T , F , TF }
4917 {
4918   \tex_romannumeral:D \if_false: { \fi:
4919     \tl_if_head_space_aux:w ? #1 ? ~ }
4920 }
4921 \cs_new:Npn \tl_if_head_space_aux:w #1 ~
4922 {
4923   \tl_if_empty:oTF { \use_none:n #1 }
4924   { \exp_after:wN \c_zero \exp_after:wN \prg_return_true: }
4925   { \exp_after:wN \c_zero \exp_after:wN \prg_return_false: }
4926   \exp_after:wN \use_none:n \exp_after:wN { \if_false: } \fi:
4927 }

```

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.

(End definition for `\tl_if_head_space:n`. This function is documented on page ??.)

192.11 Specialist functions for kernel use

`\tl_to_str_active_safe:Nx` For converting a token list to a string where active characters are treated as strings from the start. This is needed by the file-loading modules.

```

4928 \cs_new_protected:Npn \tl_to_str_active_safe:Nx #1#2
4929 {
4930   \group_begin:
4931     \seq_map_inline:Nn \l_char_active_seq
4932       { \cs_set_nopar:Npx ##1 { \token_to_str:N ##1 } }
4933   \use:x
4934   {
4935     \group_end:
4936     \tl_set:Nn \exp_not:N #1 {#2}
4937   }
4938   \tl_set:Nx #1 { \tl_to_str:N #1 }
4939 }

```

(End definition for `\tl_to_str_active_safe:Nx`. This function is documented on page 97.)

192.12 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`

```

4940 \cs_new_protected:Npn \tl_show:N #1 { \cs_show:N #1 }
4941 \cs_generate_variant:Nn \tl_show:N { c }

```

(End definition for `\tl_show:N` and `\tl_show:c`. These functions are documented on page ??.)

`\tl_show:n` For literal token lists, life is easy.

```

4942 \cs_new_eq:NN \tl_show:n \etex_showtokens:D

```

(End definition for `\tl_show:n`. This function is documented on page 95.)

192.13 Constant token lists

`\c_job_name_tl` Inherited from the \LaTeX 3 name for the primitive: this needs to actually contain the text of the job name rather than the name of the primitive, of course. \LaTeX does not quote file names containing spaces, whereas $\text{pdf}\text{\TeX}$ and $\text{X}\text{\TeX}$ do. So there may be a correction to make in the \LaTeX case.

```

4943 \*initex
4944 \tex_everyjob:D \exp_after:wN
4945 {
4946   \tex_the:D \tex_everyjob:D
4947   \luatex_if_engine:T
4948   {
4949     \lua_now:x
4950     { dofile ( assert ( kpse.find_file ("lua $\text{\LaTeX}$ quotejobname.lua" ) ) ) }
4951   }

```

```

4952 }
4953 </initex>
4954 \tl_const:Nx \c_job_name_tl { \tex_jobname:D }
      (End definition for \c_job_name_tl. This function is documented on page 95.)

```

`\c_space_tl` A space as a token list (as opposed to as a character).

```

4955 \tl_const:Nn \c_space_tl { ~ }
      (End definition for \c_space_tl. This function is documented on page 95.)

```

192.14 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.

```

4956 \tl_new:N \g_tmpa_tl
4957 \tl_new:N \g_tmpb_tl
      (End definition for \g_tmpa_tl and \g_tmpb_tl. These functions are documented on page 95.)

```

`\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.

```

4958 \tl_new:N \l_tmpa_tl
4959 \tl_new:N \l_tmpb_tl
      (End definition for \l_tmpa_tl and \l_tmpb_tl. These functions are documented on page 95.)

```

192.15 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.

```

4960 \cs_new:Npn \str_if_eq_return:xx #1 #2
4961 {
4962   \if_int_compare:w \pdfTeX_strcmp:D {#1} {#2} = \c_zero
4963     \prg_return_true:
4964   \else:
4965     \prg_return_false:
4966   \fi:
4967 }
      (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`.

```

4968 \cs_new:Npn \tl_if_single_p:N { \exp_args:No \tl_if_single_p:n }
4969 \cs_new:Npn \tl_if_single:NT { \exp_args:No \tl_if_single:nT }
4970 \cs_new:Npn \tl_if_single:NF { \exp_args:No \tl_if_single:nF }
4971 \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 88.)

```

`\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.

```
4972 \prg_new_conditional:Npnn \tl_if_single:n #1 { p , T , F , TF }
4973 { \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 88.)
```

`\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.

```
4974 \prg_new_conditional:Npnn \tl_if_single_token:n #1 { p , T , F , TF }
4975 {
4976   \tl_if_head_N_type:nTF {#1}
4977   { \str_if_eq_return:xx { \exp_not:o { \use_none:n #1 } } { } }
4978   { \str_if_eq_return:xx { \exp_not:n {#1} } { ~ } }
4979 }
(End definition for \tl_if_single_token:n. This function is documented on page 88.)
```

`\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 97.)

`\tl_act:NNNnn` To help control the expansion, `\tl_act:NNNnn` starts with `\romannumeral` and ends by producing `\c_zero` once the result has been obtained. Then loop over tokens, groups, and spaces in #5. The marker `\q_tl_act_mark` is used both to avoid losing outer braces and to detect the end of the token list more easily. The result is stored as an argument for the dummy function `\tl_act_result:n`.

```

\tl_act_aux:NNNnn
\tl_act_output:n
\tl_act_reverse_output:n
\tl_act_group_recurse:Nnn
\tl_act_loop:w
\tl_act_normal:NwnNNN
\tl_act_group:nwnNNN
\tl_act_space:wwnNNN
\tl_act_end:w
4980 \cs_new:Npn \tl_act:NNNnn { \tex_romannumeral:D \tl_act_aux:NNNnn }
4981 \cs_new:Npn \tl_act_aux:NNNnn #1 #2 #3 #4 #5
4982 {
4983   \group_align_safe_begin:
4984   \tl_act_loop:w #5 \q_tl_act_mark \q_tl_act_stop
4985   {#4} #1 #2 #3
4986   \tl_act_result:n { }
4987 }
```

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.

```

4988 \cs_new:Npn \tl_act_loop:w #1 \q_tl_act_stop
4989 {
4990   \tl_if_head_N_type:nTF {#1}
4991   { \tl_act_normal:Nwnnnn }
4992   {
4993     \tl_if_head_group:nTF {#1}
4994     { \tl_act_group:wnnnnn }
4995     { \tl_act_space:wnnnnn }
4996   }
4997   #1 \q_tl_act_stop
4998 }
4999 \cs_new:Npn \tl_act_normal:Nwnnnn #1 #2 \q_tl_act_stop #3#4
5000 {
5001   \if_meaning:w \q_tl_act_mark #1
5002   \exp_after:wN \tl_act_end:wn
5003   \fi:
5004   #4 {#3} #1
5005   \tl_act_loop:w #2 \q_tl_act_stop
5006   {#3} #4
5007 }
5008 \cs_new:Npn \tl_act_end:wn #1 \tl_act_result:n #2
5009 { \group_align_safe_end: \c_zero #2 }
5010 \cs_new:Npn \tl_act_group:wnnnnn #1 #2 \q_tl_act_stop #3#4#5
5011 {
5012   #5 {#3} {#1}
5013   \tl_act_loop:w #2 \q_tl_act_stop
5014   {#3} #4 #5
5015 }
5016 \exp_last_unbraced:NNo
5017 \cs_new:Npn \tl_act_space:wnnnnn \c_space_tl #1 \q_tl_act_stop #2#3#4#5
5018 {
5019   #5 {#2}
5020   \tl_act_loop:w #1 \q_tl_act_stop
5021   {#2} #3 #4 #5
5022 }

```

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.

```

5023 \cs_new:Npn \tl_act_output:n #1 #2 \tl_act_result:n #3
5024 { #2 \tl_act_result:n { #3 #1 } }
5025 \cs_new:Npn \tl_act_reverse_output:n #1 #2 \tl_act_result:n #3
5026 { #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.

```

5027 \cs_new:Npn \tl_act_group_recurse:Nnn #1#2#3
5028 {

```

```

5029 \exp_args:Nf #1
5030 { \exp_after:wN \exp_after:wN \exp_after:wN { #2 {#3} } }
5031 }

```

(End definition for `\tl_act:NNNnn` and `\tl_act_aux:NNNnn`. These functions are documented on page ??.)

`\tl_reverse_tokens: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.

```

5032 \cs_new:Npn \tl_reverse_tokens:n #1
5033 {
5034   \etex_unexpanded:D \exp_after:wN
5035   {
5036     \tex_romannumeral:D
5037     \tl_act_aux:NNNnn
5038     \tl_act_reverse_normal:nN
5039     \tl_act_reverse_group:nn
5040     \tl_act_reverse_space:n
5041     { }
5042     {#1}
5043   }
5044 }
5045 \cs_new:Npn \tl_act_reverse_space:n #1
5046 { \tl_act_reverse_output:n {~} }
5047 \cs_new:Npn \tl_act_reverse_normal:nN #1 #2
5048 { \tl_act_reverse_output:n {#2} }
5049 \cs_new:Npn \tl_act_reverse_group:nn #1
5050 {
5051   \tl_act_group_recurse:Nnn
5052   \tl_act_reverse_output:n
5053   { \tl_reverse_tokens:n }
5054 }

```

(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 5055 \cs_new:Npn \tl_reverse:n #1
\tl_reverse_group_preserve:nn 5056 {
5057   \etex_unexpanded:D \exp_after:wN
5058   {
5059     \tex_romannumeral:D
5060     \tl_act_aux:NNNnn
5061     \tl_act_reverse_normal:nN
5062     \tl_act_reverse_group_preserve:nn
5063     \tl_act_reverse_space:n

```

```

5064         { }
5065         {#1}
5066     }
5067 }
5068 \cs_new:Npn \tl_act_reverse_group_preserve:nn #1 #2
5069 { \tl_act_reverse_output:n { {#2} } }
5070 \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 5071 \cs_new_protected:Npn \tl_reverse:N #1
\tl_greverse:N 5072 { \tl_set:Nx #1 { \exp_args:No \tl_reverse:n { #1 } } }
\tl_greverse:c 5073 \cs_new_protected:Npn \tl_greverse:N #1
5074 { \tl_gset:Nx #1 { \exp_args:No \tl_reverse:n { #1 } } }
5075 \cs_generate_variant:Nn \tl_reverse:N { c }
5076 \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 the *left*, into the integer expression, and the sum is ended by the `\c_zero` inserted by `\tl_act_end:wn`. Somewhat a hack.

```

\tl_act_length_normal:nN 5077 \cs_new:Npn \tl_length_tokens:n #1
\tl_act_length_group:nn 5078 {
\tl_act_length_space:n 5079     \int_eval:n
5080     {
5081         \tl_act_aux:NNNnn
5082         \tl_act_length_normal:nN
5083         \tl_act_length_group:nn
5084         \tl_act_length_space:n
5085         { }
5086         {#1}
5087     }
5088 }
5089 \cs_new:Npn \tl_act_length_normal:nN #1 #2 { 1 + }
5090 \cs_new:Npn \tl_act_length_space:n #1 { 1 + }
5091 \cs_new:Npn \tl_act_length_group:nn #1 #2
5092 { 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.

```

\c_tl_act_lowercase_tl 5093 \tl_const:Nn \c_tl_act_uppercase_tl
5094 {
5095     aA bB cC dD eE fF gG hH iI jJ kK lL mM
5096     nN oO pP qQ rR sS tT uU vV wW xX yY zZ
5097 }
5098 \tl_const:Nn \c_tl_act_lowercase_tl
5099 {

```

```

5100      Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm
5101      Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz
5102    }

```

(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.

```

5103 \cs_new:Npn \tl_expandable_uppercase:n #1
5104 {
5105   \etex_unexpanded:D \exp_after:wN
5106   {
5107     \tex_romannumeral:D
5108     \tl_act_case_aux:nn { \c_tl_act_uppercase_tl } {#1}
5109   }
5110 }
5111 \cs_new:Npn \tl_expandable_lowercase:n #1
5112 {
5113   \etex_unexpanded:D \exp_after:wN
5114   {
5115     \tex_romannumeral:D
5116     \tl_act_case_aux:nn { \c_tl_act_lowercase_tl } {#1}
5117   }
5118 }
5119 \cs_new:Npn \tl_act_case_aux:nn
5120 {
5121   \tl_act_aux:NNNnn
5122   \tl_act_case_normal:nN
5123   \tl_act_case_group:nn
5124   \tl_act_case_space:n
5125 }
5126 \cs_new:Npn \tl_act_case_space:n #1 { \tl_act_output:n {~} }
5127 \cs_new:Npn \tl_act_case_normal:nN #1 #2
5128 {
5129   \exp_args:Nf \tl_act_output:n
5130   {
5131     \exp_args:NNo \prg_case_str:nnn #2 {#1}
5132     { \exp_stop_f: #2 }
5133   }
5134 }
5135 \cs_new:Npn \tl_act_case_group:nn #1 #2
5136 {
5137   \exp_after:wN \tl_act_output:n \exp_after:wN

```

```

5138     { \exp_after:wN { \tex_romannumeral:D \tl_act_case_aux:nn {#1} {#2} } }
5139   }

```

(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 the correct item. If the resulting offset is too large, then **\quark_if_recursion_tail_stop:n** terminates the loop, and returns nothing at all.

```

\tl_item_aux:nn 5140 \cs_new:Npn \tl_item:nn #1#2
5141   {
5142     \exp_args:Nf \tl_item_aux:nn
5143     {
5144       \int_eval:n
5145       {
5146         \int_compare:nNnT {#2} < \c_zero
5147         { \tl_length:n {#1} + }
5148         #2
5149       }
5150     }
5151     #1
5152     \q_recursion_tail
5153     \prg_break_point:n { }
5154   }
5155 \cs_new:Npn \tl_item_aux:nn #1#2
5156   {
5157     \quark_if_recursion_tail_break:n {#2}
5158     \int_compare:nNnTF {#1} = \c_zero
5159     { \tl_map_break:n { \exp_not:n {#2} } }
5160     { \exp_args:Nf \tl_item_aux:nn { \int_eval:n { #1 - 1 } } }
5161   }
5162 \cs_new_nopar:Npn \tl_item:Nn { \exp_args:No \tl_item:nn }
5163 \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.

```

5164 \prg_new_conditional:Npnn \tl_if_empty:x #1 { p , T , F , TF }
5165 { \str_if_eq_return:xx { } {#1} }

```

(End definition for \tl_if_empty:x. This function is documented on page ??.)

192.16 Deprecated functions

\tl_new:Nn Use either **\tl_const:Nn** or **\tl_new:N**.

\tl_new:cn 5166 *<deprecated>*

\tl_new:Nx 5167 **\cs_new_protected:Npn \tl_new:Nn #1#2**

```

5168   {

```

```

5169 \tl_new:N #1
5170 \tl_gset:Nn #1 {#2}
5171 }
5172 \cs_generate_variant:Nn \tl_new:Nn { c }
5173 \cs_generate_variant:Nn \tl_new:Nn { Nx }
5174 </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

```

5175 <*deprecated>
5176 \cs_new_protected_nopar:Npn \tl_gset:Nc
5177 { \tex_global:D \tl_set:Nc }
5178 \cs_new_protected:Npn \tl_set:Nc #1#2
5179 { \tl_set:No #1 { \cs:w #2 \cs_end: } }
5180 </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

```

5181 <*deprecated>
5182 \cs_new_eq:NN \tl_replace_in:Nnn \tl_replace_once:Nnn
5183 \cs_new_eq:NN \tl_replace_in:cnn \tl_replace_once:cnn
5184 \cs_new_eq:NN \tl_greplace_in:Nnn \tl_greplace_once:Nnn
5185 \cs_new_eq:NN \tl_greplace_in:cnn \tl_greplace_once:cnn
5186 \cs_new_eq:NN \tl_replace_all_in:Nnn \tl_replace_all:Nnn
5187 \cs_new_eq:NN \tl_replace_all_in:cnn \tl_replace_all:cnn
5188 \cs_new_eq:NN \tl_greplace_all_in:Nnn \tl_greplace_all:Nnn
5189 \cs_new_eq:NN \tl_greplace_all_in:cnn \tl_greplace_all:cnn
5190 </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

```

5191 <*deprecated>
5192 \cs_new_eq:NN \tl_remove_in:Nn \tl_remove_once:Nn
5193 \cs_new_eq:NN \tl_remove_in:cn \tl_remove_once:cn
5194 \cs_new_eq:NN \tl_gremove_in:Nn \tl_gremove_once:Nn
5195 \cs_new_eq:NN \tl_gremove_in:cn \tl_gremove_once:cn
5196 \cs_new_eq:NN \tl_remove_all_in:Nn \tl_remove_all:Nn
5197 \cs_new_eq:NN \tl_remove_all_in:cn \tl_remove_all:cn
5198 \cs_new_eq:NN \tl_gremove_all_in:Nn \tl_gremove_all:Nn
5199 \cs_new_eq:NN \tl_gremove_all_in:cn \tl_gremove_all:cn
5200 </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

```

5201 <*deprecated>
5202 \cs_new_eq:NN \tl_elt_count:n \tl_length:n

```

```

5203 \cs_new_eq:NN \tl_elt_count:V \tl_length:V
5204 \cs_new_eq:NN \tl_elt_count:o \tl_length:o
5205 \cs_new_eq:NN \tl_elt_count:N \tl_length:N
5206 \cs_new_eq:NN \tl_elt_count:c \tl_length:c
5207 </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 5208 <*deprecated>
\tl_head_iii:n 5209 \cs_new_eq:NN \tl_head_i:n \tl_head:n
\tl_head_iii:f 5210 \cs_new_eq:NN \tl_head_i:w \tl_head:w
\tl_head_iii:w 5211 \cs_new:Npn \tl_head_iii:n #1 { \tl_head_iii:w #1 \q_stop }
5212 \cs_generate_variant:Nn \tl_head_iii:n { f }
5213 \cs_new:Npn \tl_head_iii:w #1#2#3#4 \q_stop {#1#2#3}
5214 </deprecated>
(End definition for \tl_head_i:n. This function is documented on page ??.)
5215 </initex | package>

```

193 l3seq implementation

The following test files are used for this code: `m3seq002`, `m3seq003`.

```

5216 <*initex | package>
5217 <*package>
5218 \ProvidesExplPackage
5219 {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
5220 \package_check_loaded_expl:
5221 </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.

```

5222 \cs_new:Npn \seq_item:n
5223 {
5224   \msg_expandable_kernel_error:nn { seq } { misused }
5225   \use_none:n
5226 }
(End definition for \seq_item:n. This function is documented on page 105.)

```

`\l_seq_internal_a_tl` Scratch space for various internal uses.
`\l_seq_internal_b_tl` 5227 `\tl_new:N \l_seq_internal_a_tl`
5228 `\tl_new:N \l_seq_internal_b_tl`
(End definition for \l_seq_internal_a_tl and \l_seq_internal_b_tl. These functions are documented on page ??.)

193.1 Allocation and initialisation

`\seq_new:N` Internally, sequences are just token lists.
`\seq_new:c` 5229 `\cs_new_eq:NN \seq_new:N \tl_new:N`
5230 `\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` 5231 `\cs_new_eq:NN \seq_clear:N \tl_clear:N`
`\seq_gclear:N` 5232 `\cs_new_eq:NN \seq_clear:c \tl_clear:c`
`\seq_gclear:c` 5233 `\cs_new_eq:NN \seq_gclear:N \tl_gclear:N`
5234 `\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` 5235 `\cs_new_eq:NN \seq_clear_new:N \tl_clear_new:N`
`\seq_gclear_new:N` 5236 `\cs_new_eq:NN \seq_clear_new:c \tl_clear_new:c`
`\seq_gclear_new:c` 5237 `\cs_new_eq:NN \seq_gclear_new:N \tl_gclear_new:N`
5238 `\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` 5239 `\cs_new_eq:NN \seq_set_eq:NN \tl_set_eq:NN`
`\seq_set_eq:Nc` 5240 `\cs_new_eq:NN \seq_set_eq:Nc \tl_set_eq:Nc`
`\seq_set_eq:cc` 5241 `\cs_new_eq:NN \seq_set_eq:cN \tl_set_eq:cN`
`\seq_gset_eq:NN` 5242 `\cs_new_eq:NN \seq_set_eq:cc \tl_set_eq:cc`
`\seq_gset_eq:cN` 5243 `\cs_new_eq:NN \seq_gset_eq:NN \tl_gset_eq:NN`
`\seq_gset_eq:Nc` 5244 `\cs_new_eq:NN \seq_gset_eq:Nc \tl_gset_eq:Nc`
`\seq_gset_eq:cN` 5245 `\cs_new_eq:NN \seq_gset_eq:cN \tl_gset_eq:cN`
`\seq_gset_eq:cc` 5246 `\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 is
`\seq_set_split_aux:Nnn` enclosed within braces. After `\tl_replace_all:Nnn`, the token list `\l_seq_internal_a_tl`
`\seq_set_split_aux_i:w` is a repetition of the pattern `\seq_set_split_aux_i:w \prg_do_nothing: <item`
`\seq_set_split_aux_ii:w` *with spaces>* `\seq_set_split_aux_end:.` Then, x-expansion causes `\seq_set_split_aux_i:w`
`\seq_set_split_aux_end:` `aux_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.

```

5247 \cs_new_protected_nopar:Npn \seq_set_split:Nnn
5248 { \seq_set_split_aux:NNnn \tl_set:Nx }
5249 \cs_new_protected_nopar:Npn \seq_gset_split:Nnn
5250 { \seq_set_split_aux:NNnn \tl_gset:Nx }
5251 \cs_new_protected:Npn \seq_set_split_aux:NNnn #1 #2 #3 #4
5252 {
5253   \tl_if_empty:nTF {#3}
5254   { #1 #2 { \tl_map_function:nN {#4} \seq_wrap_item:n } }
5255   {
5256     \tl_set:Nn \l_seq_internal_a_tl
5257     {
5258       \seq_set_split_aux_i:w \prg_do_nothing:
5259       #4
5260       \seq_set_split_aux_end:
5261     }
5262     \tl_replace_all:Nnn \l_seq_internal_a_tl { #3 }
5263     {
5264       \seq_set_split_aux_end:
5265       \seq_set_split_aux_i:w \prg_do_nothing:
5266     }
5267     \tl_set:Nx \l_seq_internal_a_tl { \l_seq_internal_a_tl }
5268     #1 #2 { \l_seq_internal_a_tl }
5269   }
5270 }
5271 \cs_new:Npn \seq_set_split_aux_i:w #1 \seq_set_split_aux_end:
5272 {
5273   \exp_not:N \seq_set_split_aux_ii:w
5274   \exp_args:No \tl_trim_spaces:n {#1}
5275   \exp_not:N \seq_set_split_aux_end:
5276 }
5277 \cs_new:Npn \seq_set_split_aux_ii:w #1 \seq_set_split_aux_end:
5278 { \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
\seq_gconcat:NNN
\seq_gconcat:ccc
5279 \cs_new_protected:Npn \seq_concat:NNN #1#2#3
5280 { \tl_set:Nx #1 { \exp_not:o {#2} \exp_not:o {#3} } }
5281 \cs_new_protected:Npn \seq_gconcat:NNN #1#2#3
5282 { \tl_gset:Nx #1 { \exp_not:o {#2} \exp_not:o {#3} } }
5283 \cs_generate_variant:Nn \seq_concat:NNN { ccc }
5284 \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
\seq_put_left:NV
\seq_put_left:Nv
\seq_put_left:No
\seq_put_left:Nx
\seq_put_left:cn
\seq_put_left:cV
\seq_put_left:cv
\seq_put_left:co
\seq_put_left:cx

```

The code here is just a wrapper for adding to token lists.

```

5285 \cs_new_protected:Npn \seq_put_left:Nn #1#2
5286 { \tl_put_left:Nn #1 { \seq_item:n {#2} } }
5287 \cs_new_protected:Npn \seq_put_right:Nn #1#2
5288 { \tl_put_right:Nn #1 { \seq_item:n {#2} } }
5289 \cs_generate_variant:Nn \seq_put_left:Nn { NV , Nv , No , Nx }
5290 \cs_generate_variant:Nn \seq_put_left:Nn { c , cV , cv , co , cx }
5291 \cs_generate_variant:Nn \seq_put_right:Nn { NV , Nv , No , Nx }
5292 \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 ??.)

```

\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

```

The same for global addition.

```

5293 \cs_new_protected:Npn \seq_gput_left:Nn #1#2
5294 { \tl_gput_left:Nn #1 { \seq_item:n {#2} } }
5295 \cs_new_protected:Npn \seq_gput_right:Nn #1#2
5296 { \tl_gput_right:Nn #1 { \seq_item:n {#2} } }
5297 \cs_generate_variant:Nn \seq_gput_left:Nn { NV , Nv , No , Nx }
5298 \cs_generate_variant:Nn \seq_gput_left:Nn { c , cV , cv , co , cx }
5299 \cs_generate_variant:Nn \seq_gput_right:Nn { NV , Nv , No , Nx }
5300 \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.

```

5301 \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.

```

5302 \seq_new:N \l_seq_internal_remove_seq

```

(End definition for `\l_seq_internal_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

```

```

5303 \cs_new_protected:Npn \seq_remove_duplicates:N
5304 { \seq_remove_duplicates_aux:NN \seq_set_eq:NN }
5305 \cs_new_protected:Npn \seq_gremove_duplicates:N
5306 { \seq_remove_duplicates_aux:NN \seq_gset_eq:NN }
5307 \cs_new_protected:Npn \seq_remove_duplicates_aux:NN #1#2
5308 {
5309   \seq_clear:N \l_seq_internal_remove_seq
5310   \seq_map_inline:Nn #2
5311   {
5312     \seq_if_in:NnF \l_seq_internal_remove_seq {##1}
5313     { \seq_put_right:Nn \l_seq_internal_remove_seq {##1} }
5314   }
5315   #1 #2 \l_seq_internal_remove_seq

```

```

5316 }
5317 \cs_generate_variant:Nn \seq_remove_duplicates:N { c }
5318 \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 to an intermediate sequence. The approach taken is therefore similar to that in \seq_pop_right_aux_ii:NNN, using a “flexible” x-type expansion to do most of the work. As \tl_if_eq:nnT is not expandable, a two-part strategy is needed. First, the x-type 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.

```

5319 \cs_new_protected:Npn \seq_remove_all:Nn
5320 { \seq_remove_all_aux:NNn \tl_set:Nx }
5321 \cs_new_protected:Npn \seq_gremove_all:Nn
5322 { \seq_remove_all_aux:NNn \tl_gset:Nx }
5323 \cs_new_protected:Npn \seq_remove_all_aux:NNn #1#2#3
5324 {
5325   \seq_push_item_def:n
5326   {
5327     \str_if_eq:nnT {##1} {#3}
5328     {
5329       \if_false: { \fi: }
5330       \tl_set:Nn \l_seq_internal_b_tl {##1}
5331       #1 #2
5332       { \if_false: } \fi:
5333       \exp_not:o {#2}
5334       \tl_if_eq:NNT \l_seq_internal_a_tl \l_seq_internal_b_tl
5335       { \use_none:nn }
5336     }
5337     \seq_wrap_item:n {##1}
5338   }
5339   \tl_set:Nn \l_seq_internal_a_tl {#3}
5340   #1 #2 {#2}
5341   \seq_pop_item_def:
5342 }
5343 \cs_generate_variant:Nn \seq_remove_all:Nn { c }
5344 \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

```

5345 \prg_new_eq_conditional:NNn \seq_if_empty:N \tl_if_empty:N
5346 { p , T , F , TF }
5347 \prg_new_eq_conditional:NNn \seq_if_empty:c \tl_if_empty:c
5348 { 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 5349 \prg_new_protected_conditional:Npnn \seq_if_in:Nn #1#2
\seq_if_in:cv 5350 { T , F , TF }
\seq_if_in:co 5351 {
\seq_if_in:cx 5352   \group_begin:
\seq_if_in_aux: 5353   \tl_set:Nn \l_seq_internal_a_tl {#2}
5354   \cs_set_protected:Npn \seq_item:n ##1
5355   {
5356     \tl_set:Nn \l_seq_internal_b_tl {##1}
5357     \if_meaning:w \l_seq_internal_a_tl \l_seq_internal_b_tl
5358     \exp_after:wN \seq_if_in_aux:
5359     \fi:
5360   }
5361   #1
5362   \seq_break:n { \prg_return_false: }
5363   \prg_break_point:n { \group_end: }
5364 }
5365 \cs_new_nopar:Npn \seq_if_in_aux: { \seq_break:n { \prg_return_true: } }
5366 \cs_generate_variant:Nn \seq_if_in:NnT { NV , Nv , No , Nx }
5367 \cs_generate_variant:Nn \seq_if_in:NnT { c , cV , cv , co , cx }
5368 \cs_generate_variant:Nn \seq_if_in:NnF { NV , Nv , No , Nx }
5369 \cs_generate_variant:Nn \seq_if_in:NnF { c , cV , cv , co , cx }
5370 \cs_generate_variant:Nn \seq_if_in:NnTF { NV , Nv , No , Nx }
5371 \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 5372 \cs_new_protected:Npn \seq_get_left:NN #1#2
5373 {
5374   \seq_if_empty_err_break:N #1
5375   \exp_after:wN \seq_get_left_aux:NnwN #1 \q_stop #2
5376   \prg_break_point:n { }
5377 }

```

```

5378 \cs_new_protected:Npn \seq_get_left_aux:NnwN \seq_item:n #1#2 \q_stop #3
5379 { \tl_set:Nn #3 {#1} }
5380 \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
5381 \cs_new_protected_nopar:Npn \seq_pop_left:NN
5382 { \seq_pop_left_aux:NNN \tl_set:Nn }
5383 \cs_new_protected_nopar:Npn \seq_gpop_left:NN
5384 { \seq_pop_left_aux:NNN \tl_gset:Nn }
5385 \cs_new_protected:Npn \seq_pop_left_aux:NNN #1#2#3
5386 {
5387   \seq_if_empty_err_break:N #2
5388   \exp_after:wN \seq_pop_left_aux:NnwNNN #2 \q_stop #1#2#3
5389   \prg_break_point:n { }
5390 }
5391 \cs_new_protected:Npn \seq_pop_left_aux:NnwNNN \seq_item:n #1#2 \q_stop #3#4#5
5392 {
5393   #3 #4 {#2}
5394   \tl_set:Nn #5 {#1}
5395 }
5396 \cs_generate_variant:Nn \seq_pop_left:NN { c }
5397 \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.

```

```

5398 \cs_new_protected:Npn \seq_get_right:NN #1#2
5399 {
5400   \seq_if_empty_err_break:N #1
5401   \seq_get_right_aux:NN #1#2
5402   \prg_break_point:n { }
5403 }
5404 \cs_new_protected:Npn \seq_get_right_aux:NN #1#2
5405 {
5406   \seq_push_item_def:n { }
5407   \exp_after:wN \exp_after:wN \exp_after:wN \seq_get_right_loop:nn
5408   \exp_after:wN \use_none:n #1
5409   { \tl_set:Nn #2 }
5410   { }
5411   {

```

```

5412         \seq_pop_item_def:
5413         \seq_break:
5414     }
5415 }
5416 \cs_new:Npn \seq_get_right_loop:nn #1#2
5417 {
5418     #2 {#1}
5419     \seq_get_right_loop:nn
5420 }
5421 \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 The approach to popping from the right is a bit more involved, but does use some of the
\seq_pop_right:cN same ideas as getting from the right. What is needed is a “flexible length” way to set a to-
\seq_gpop_right:NN ken list variable. This is supplied by the { \if_false:} \fi: ... \if_false: { \fi: }
\seq_gpop_right:cN construct. Using an x-type expansion and a “non-expanding” definition for \seq_item:n,
\seq_pop_right_aux:NNN the left-most $n - 1$ entries in a sequence of n items will be stored back in the sequence.
\seq_pop_right_aux_ii:NNN That needs a loop of unknown length, hence using the strange \if_false: way of in-
cluding 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.

```

5422 \cs_new_protected_nopar:Npn \seq_pop_right:NN
5423 { \seq_pop_right_aux:NNN \tl_set:Nx }
5424 \cs_new_protected_nopar:Npn \seq_gpop_right:NN
5425 { \seq_pop_right_aux:NNN \tl_gset:Nx }
5426 \cs_new_protected:Npn \seq_pop_right_aux:NNN #1#2#3
5427 {
5428     \seq_if_empty_err_break:N #2
5429     \seq_pop_right_aux_ii:NNN #1 #2 #3
5430     \prg_break_point:n { }
5431 }
5432 \cs_new_protected:Npn \seq_pop_right_aux_ii:NNN #1#2#3
5433 {
5434     \seq_push_item_def:n { \seq_wrap_item:n {##1} }
5435     #1 #2 { \if_false: } \fi:
5436     \exp_after:wN \exp_after:wN \exp_after:wN \seq_get_right_loop:nn
5437     \exp_after:wN \use_none:n #2
5438     {
5439         \if_false: { \fi: }
5440         \tl_set:Nn #3
5441     }
5442     { }
5443     {
5444         \seq_pop_item_def:
5445         \seq_break:
5446     }
5447 }
5448 \cs_generate_variant:Nn \seq_pop_right:NN { c }

```

```

5449 \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:n
\seq_break:n
5450 \cs_new_eq:NN \seq_break: \prg_map_break:
5451 \cs_new_eq:NN \seq_break:n \prg_map_break:n
5452 \cs_new_eq:NN \seq_map_break: \prg_map_break:
5453 \cs_new_eq:NN \seq_map_break:n \prg_map_break:n
      (End definition for \seq_map_break:. This function is documented on page 106.)

```

\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.

```

5454 \cs_new_protected:Npn \seq_if_empty_err_break:N #1
5455 {
5456   \if_meaning:w #1 \c_empty_tl
5457     \msg_kernel_error:nxx { seq } { empty-sequence } { \token_to_str:N #1 }
5458     \exp_after:wN \seq_break:
5459   \fi:
5460 }
      (End definition for \seq_if_empty_err_break:N. This function is documented on page 105.)

```

\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.

```

5461 \cs_new:Npn \seq_map_function:NN #1#2
5462 {
5463   \exp_after:wN \seq_map_function_aux:NNn \exp_after:wN #2 #1
5464   { ? \seq_map_break: } { }
5465   \prg_break_point:n { }
5466 }
5467 \cs_new:Npn \seq_map_function_aux:NNn #1#2#3
5468 {
5469   \use_none:n #2
5470   #1 {#3}
5471   \seq_map_function_aux:NNn #1
5472 }
5473 \cs_generate_variant:Nn \seq_map_function:NN { c }
      (End definition for \seq_map_function:NN and \seq_map_function:cN. These functions are docu-
mented on page ??.)

```

`\seq_push_item_def:n` The definition of `\seq_item:n` needs to be saved and restored at various points within
`\seq_push_item_def:x` the mapping and manipulation code. That is handled here: as always, this approach uses
`\seq_push_item_def_aux:` global assignments.

```

\seq_pop_item_def:
5474 \cs_new_protected:Npn \seq_push_item_def:n
5475 {
5476   \seq_push_item_def_aux:
5477   \cs_gset:Npn \seq_item:n ##1
5478 }
5479 \cs_new_protected:Npn \seq_push_item_def:x
5480 {
5481   \seq_push_item_def_aux:
5482   \cs_gset:Npx \seq_item:n ##1
5483 }
5484 \cs_new_protected:Npn \seq_push_item_def_aux:
5485 {
5486   \cs_gset_eq:cN { seq_item_ \int_use:N \g_prg_map_int :n }
5487   \seq_item:n
5488   \int_gincr:N \g_prg_map_int
5489 }
5490 \cs_new_protected_nopar:Npn \seq_pop_item_def:
5491 {
5492   \int_gdecr:N \g_prg_map_int
5493   \cs_gset_eq:Nc \seq_item:n
5494   { seq_item_ \int_use:N \g_prg_map_int :n }
5495 }

```

(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`.

```

5496 \cs_new_protected:Npn \seq_map_inline:Nn #1#2
5497 {
5498   \seq_push_item_def:n {#2}
5499   #1
5500   \prg_break_point:n { \seq_pop_item_def: }
5501 }
5502 \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
`\seq_map_variable:Ncn` for the code set up so that the number of # tokens required is as expected.

```

\seq_map_variable:cNn
\seq_map_variable:ccn
5503 \cs_new_protected:Npn \seq_map_variable:NNn #1#2#3
5504 {
5505   \seq_push_item_def:x
5506   {
5507     \tl_set:Nn \exp_not:N #2 {##1}
5508     \exp_not:n {#3}
5509   }

```



```

5510     #1
5511     \prg_break_point:n { \seq_pop_item_def: }
5512   }
5513   \cs_generate_variant:Nn \seq_map_variable:NNn { Nc }
5514   \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.

```

\seq_push:NV 5515 \cs_new_eq:NN \seq_push:Nn \seq_put_left:Nn
\seq_push:Nv 5516 \cs_new_eq:NN \seq_push:NV \seq_put_left:Nv
\seq_push:No 5517 \cs_new_eq:NN \seq_push:Nv \seq_put_left:Nv
\seq_push:Nx 5518 \cs_new_eq:NN \seq_push:No \seq_put_left:No
\seq_push:cn 5519 \cs_new_eq:NN \seq_push:Nx \seq_put_left:Nx
\seq_push:cV 5520 \cs_new_eq:NN \seq_push:cn \seq_put_left:cn
\seq_push:cV 5521 \cs_new_eq:NN \seq_push:cV \seq_put_left:cV
\seq_push:cv 5522 \cs_new_eq:NN \seq_push:cv \seq_put_left:cv
\seq_push:co 5523 \cs_new_eq:NN \seq_push:co \seq_put_left:co
\seq_push:cx 5524 \cs_new_eq:NN \seq_push:cx \seq_put_left:cx
\seq_gpush:Nn 5525 \cs_new_eq:NN \seq_gpush:Nn \seq_gput_left:Nn
\seq_gpush:NV 5526 \cs_new_eq:NN \seq_gpush:NV \seq_gput_left:Nv
\seq_gpush:Nv 5527 \cs_new_eq:NN \seq_gpush:Nv \seq_gput_left:Nv
\seq_gpush:No 5528 \cs_new_eq:NN \seq_gpush:No \seq_gput_left:No
\seq_gpush:Nx 5529 \cs_new_eq:NN \seq_gpush:Nx \seq_gput_left:Nx
\seq_gpush:cn 5530 \cs_new_eq:NN \seq_gpush:cn \seq_gput_left:cn
\seq_gpush:cV 5531 \cs_new_eq:NN \seq_gpush:cV \seq_gput_left:cV
\seq_gpush:cv 5532 \cs_new_eq:NN \seq_gpush:cv \seq_gput_left:cv
\seq_gpush:co 5533 \cs_new_eq:NN \seq_gpush:co \seq_gput_left:co
\seq_gpush:cx 5534 \cs_new_eq:NN \seq_gpush:cx \seq_gput_left:cx

```

(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.

```

\seq_pop:NN 5535 \cs_new_eq:NN \seq_get:NN \seq_get_left:NN
\seq_pop:cN 5536 \cs_new_eq:NN \seq_get:cN \seq_get_left:cN
\seq_gpop:NN 5537 \cs_new_eq:NN \seq_pop:NN \seq_pop_left:NN
\seq_gpop:cN 5538 \cs_new_eq:NN \seq_pop:cN \seq_pop_left:cN
5539 \cs_new_eq:NN \seq_gpop:NN \seq_gpop_left:NN
5540 \cs_new_eq:NN \seq_gpop:cN \seq_gpop_left:cN

```

(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`.
`\seq_show:c`

```

5541 \cs_new_protected:Npn \seq_show:N #1
5542 {
5543   \msg_aux_show:Nnx
5544   #1
5545   { seq }
5546   { \seq_map_function:NN #1 \msg_aux_show:n }
5547 }
5548 \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**.

```

5549 \cs_new:Npn \seq_if_empty_break_return_false:N #1
5550 {
5551   \if_meaning:w #1 \c_empty_tl
5552   \prg_return_false:
5553   \exp_after:wN \seq_break:
5554   \fi:
5555 }

```

(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`
`\seq_get_right:NN`
`\seq_get_right:cN`

```

5556 \prg_new_protected_conditional:Npnn \seq_get_left:NN #1 #2 { T , F , TF }
5557 {
5558   \seq_if_empty_break_return_false:N #1
5559   \exp_after:wN \seq_get_left_aux:Nw #1 \q_stop #2
5560   \prg_return_true:
5561   \seq_break:
5562   \prg_break_point:n { }
5563 }
5564 \prg_new_protected_conditional:Npnn \seq_get_right:NN #1#2 { T , F , TF }
5565 {
5566   \seq_if_empty_break_return_false:N #1
5567   \seq_get_right_aux:NN #1#2
5568   \prg_return_true: \seq_break:
5569   \prg_break_point:n { }
5570 }
5571 \cs_generate_variant:Nn \seq_get_left:NNT { c }
5572 \cs_generate_variant:Nn \seq_get_left:NNF { c }
5573 \cs_generate_variant:Nn \seq_get_left:NNTF { c }
5574 \cs_generate_variant:Nn \seq_get_right:NNT { c }
5575 \cs_generate_variant:Nn \seq_get_right:NNF { c }
5576 \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 5577 \prg_new_protected_conditional:Npnn \seq_pop_left:NN #1#2 { T , F , TF }
\seq_gpop_left:NN 5578 {
\seq_gpop_left:cN 5579   \seq_if_empty_break_return_false:N #1
\seq_pop_right:NN 5580   \exp_after:wN \seq_pop_left_aux:NnwNNN #1 \q_stop \tl_set:Nn #1#2
\seq_pop_right:cN 5581   \prg_return_true: \seq_break:
\seq_gpop_right:NN 5582   \prg_break_point:n { }
\seq_gpop_right:cN 5583 }
5584 \prg_new_protected_conditional:Npnn \seq_gpop_left:NN #1#2 { T , F , TF }
5585 {
5586   \seq_if_empty_break_return_false:N #1
5587   \exp_after:wN \seq_pop_left_aux:NnwNNN #1 \q_stop \tl_gset:Nn #1#2
5588   \prg_return_true: \seq_break:
5589   \prg_break_point:n { }
5590 }
5591 \prg_new_protected_conditional:Npnn \seq_pop_right:NN #1#2 { T , F , TF }
5592 {
5593   \seq_if_empty_break_return_false:N #1
5594   \seq_pop_right_aux_ii:NNN \tl_set:Nx #1 #2
5595   \prg_return_true: \seq_break:
5596   \prg_break_point:n { }
5597 }
5598 \prg_new_protected_conditional:Npnn \seq_gpop_right:NN #1#2 { T , F , TF }
5599 {
5600   \seq_if_empty_break_return_false:N #1
5601   \seq_pop_right_aux_ii:NNN \tl_gset:Nx #1 #2
5602   \prg_return_true: \seq_break:
5603   \prg_break_point:n { }
5604 }
5605 \cs_generate_variant:Nn \seq_pop_left:NNT { c }
5606 \cs_generate_variant:Nn \seq_pop_left:NNF { c }
5607 \cs_generate_variant:Nn \seq_pop_left:NNTF { c }
5608 \cs_generate_variant:Nn \seq_gpop_left:NNT { c }
5609 \cs_generate_variant:Nn \seq_gpop_left:NNF { c }
5610 \cs_generate_variant:Nn \seq_gpop_left:NNTF { c }
5611 \cs_generate_variant:Nn \seq_pop_right:NNT { c }
5612 \cs_generate_variant:Nn \seq_pop_right:NNF { c }
5613 \cs_generate_variant:Nn \seq_pop_right:NNTF { c }
5614 \cs_generate_variant:Nn \seq_gpop_right:NNT { c }
5615 \cs_generate_variant:Nn \seq_gpop_right:NNF { c }
5616 \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
`\seq_length:c` functions: turn each entry into a +1 then use integer evaluation to actually do the math-
`\seq_length_aux:n` ematics.

```

5617 \cs_new:Npn \seq_length:N #1
5618 {
5619   \int_eval:n
5620   {
5621     0
5622     \seq_map_function:NN #1 \seq_length_aux:n
5623   }
5624 }
5625 \cs_new:Npn \seq_length_aux:n #1 { +1 }
5626 \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`

```

5627 \cs_new:Npn \seq_item:Nn #1#2
5628 {
5629   \exp_last_unbraced:Nfo \seq_item_aux:nnn
5630   {
5631     \int_eval:n
5632     {
5633       \int_compare:nNnT {#2} < \c_zero
5634       { \seq_length:N #1 + }
5635       #2
5636     }
5637   }
5638   #1
5639   { ? \seq_break: }
5640   { }
5641   \prg_break_point:n { }
5642 }
5643 \cs_new:Npn \seq_item_aux:nnn #1#2#3
5644 {
5645   \use_none:n #2
5646   \int_compare:nNnTF {#1} = \c_zero
5647   { \seq_break:n { \exp_not:n {#3} } }
5648   { \exp_args:Nf \seq_item_aux:nnn { \int_eval:n { #1 - 1 } } }
5649 }
5650 \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 5651 \cs_new:Npn \seq_use:N #1 { \seq_map_function:NN #1 \use:n }
5652 \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
\seq_mapthread_function:NcN
\seq_mapthread_function:cNN
\seq_mapthread_function:ccN
  \seq_mapthread_function_aux:NN
\seq_mapthread_function_aux:Nnnwnn

```

The idea here is to first expand both of the sequences, adding the usual `{ ? \seq_break: } { }` to the end of each one. This is most conveniently done in two steps using an auxiliary function. The mapping then throws away the first token of #2 and #5, which for items in the sequences will both be `\seq_item:n`. The function to be mapped will then be applied to the two entries. When the code hits the end of one of the sequences, the break 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.

```

5653 \cs_new:Npn \seq_mapthread_function:NNN #1#2#3
5654 {
5655   \exp_after:wN \seq_mapthread_function_aux:NN
5656   \exp_after:wN #3
5657   \exp_after:wN #1
5658   #2
5659   { ? \seq_break: } { }
5660   \prg_break_point:n { }
5661 }
5662 \cs_new:Npn \seq_mapthread_function_aux:NN #1#2
5663 {
5664   \exp_after:wN \seq_mapthread_function_aux:Nnnwnn
5665   \exp_after:wN #1
5666   #2
5667   { ? \seq_break: } { }
5668   \q_stop
5669 }
5670 \cs_new:Npn \seq_mapthread_function_aux:Nnnwnn #1#2#3#4 \q_stop #5#6
5671 {
5672   \use_none:n #2
5673   \use_none:n #5
5674   #1 {#3} {#6}
5675   \seq_mapthread_function_aux:Nnnwnn #1 #4 \q_stop
5676 }
5677 \cs_generate_variant:Nn \seq_mapthread_function:NNN { Nc }
5678 \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
\seq_set_from_clist:cN
\seq_set_from_clist:Nc
\seq_set_from_clist:cc
\seq_set_from_clist:Nn
\seq_set_from_clist:cn
\seq_gset_from_clist:NN
\seq_gset_from_clist:cN
\seq_gset_from_clist:Nc
\seq_gset_from_clist:cc
\seq_gset_from_clist:Nn
\seq_gset_from_clist:cn

```

Setting a sequence from a comma-separated list is done using a simple mapping.

```

5679 \cs_new_protected:Npn \seq_set_from_clist:NN #1#2
5680 {
5681   \tl_set:Nx #1
5682   { \clist_map_function:NN #2 \seq_wrap_item:n }
5683 }
5684 \cs_new_protected:Npn \seq_set_from_clist:Nn #1#2
5685 {
5686   \tl_set:Nx #1
5687   { \clist_map_function:nN {#2} \seq_wrap_item:n }
5688 }
5689 \cs_new_protected:Npn \seq_gset_from_clist:NN #1#2
5690 {

```

```

5691 \tl_gset:Nx #1
5692 { \clist_map_function:NN #2 \seq_wrap_item:n }
5693 }
5694 \cs_new_protected:Npn \seq_gset_from_clist:Nn #1#2
5695 {
5696 \tl_gset:Nx #1
5697 { \clist_map_function:nN {#2} \seq_wrap_item:n }
5698 }
5699 \cs_generate_variant:Nn \seq_set_from_clist:NN { Nc }
5700 \cs_generate_variant:Nn \seq_set_from_clist:NN { c , cc }
5701 \cs_generate_variant:Nn \seq_set_from_clist:Nn { c }
5702 \cs_generate_variant:Nn \seq_gset_from_clist:NN { Nc }
5703 \cs_generate_variant:Nn \seq_gset_from_clist:NN { c , cc }
5704 \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 `\exp_stop_f:` marker.

```

\seq_reverse:c \exp_stop_f: marker.
\seq_greverse:N
\seq_greverse:c \cs_new_protected:Npn \seq_reverse:N #1
\seq_reverse_aux:NN {
\seq_reverse_aux_item:nwn \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.

```

5705 \cs_new_protected_nopar:Npn \seq_tmp:w { }
5706 \cs_new_protected_nopar:Npn \seq_reverse:N
5707 { \seq_reverse_aux:NN \tl_set:Nx }
5708 \cs_new_protected_nopar:Npn \seq_greverse:N
5709 { \seq_reverse_aux:NN \tl_gset:Nx }
5710 \cs_new_protected:Npn \seq_reverse_aux:NN #1 #2

```

```

5711 {
5712   \cs_set_eq:NN \seq_tmp:w \seq_item:n
5713   \cs_set_eq:NN \seq_item:n \seq_reverse_aux_item:nwn
5714   #1 #2 { #2 \exp_not:n { } }
5715   \cs_set_eq:NN \seq_item:n \seq_tmp:w
5716 }
5717 \cs_new:Npn \seq_reverse_aux_item:nwn #1 #2 \exp_not:n #3
5718 {
5719   #2
5720   \exp_not:n { \seq_item:n {#1} #3 }
5721 }
5722 \cs_generate_variant:Nn \seq_reverse:N { c }
5723 \cs_generate_variant:Nn \seq_reverse: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.

```

5724 \cs_new_protected_nopar:Npn \seq_set_filter:NNn
5725   { \seq_set_filter_aux:NNNn \tl_set:Nx }
5726 \cs_new_protected_nopar:Npn \seq_gset_filter:NNn
5727   { \seq_set_filter_aux:NNNn \tl_gset:Nx }
5728 \cs_new_protected:Npn \seq_set_filter_aux:NNNn #1#2#3#4
5729   {
5730     \seq_push_item_def:n { \bool_if:nT {#4} { \seq_wrap_item:n {##1} } }
5731     #1 #2 { #3 \prg_break_point:n { } }
5732     \seq_pop_item_def:
5733   }

```

(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
5734 \cs_new_protected_nopar:Npn \seq_set_map:NNn
5735   { \seq_set_map_aux:NNNn \tl_set:Nx }
5736 \cs_new_protected_nopar:Npn \seq_gset_map:NNn
5737   { \seq_set_map_aux:NNNn \tl_gset:Nx }
5738 \cs_new_protected:Npn \seq_set_map_aux:NNNn #1#2#3#4
5739   {
5740     \seq_push_item_def:n { \exp_not:N \seq_item:n {#4} }
5741     #1 #2 { #3 }
5742     \seq_pop_item_def:
5743   }

```

(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`

```

5744 <*deprecated>
5745 \cs_new_eq:NN \seq_top:NN \seq_get_left:NN
5746 \cs_new_eq:NN \seq_top:cN \seq_get_left:cN
5747 </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`

```

5748 <*deprecated>
5749 \cs_new_eq:NN \seq_display:N \seq_show:N
5750 \cs_new_eq:NN \seq_display:c \seq_show:c
5751 </deprecated>
      (End definition for \seq_display:N and \seq_display:c. These functions are documented on
page ??.)
5752 </initex | package>

```

194 l3clist implementation

The following test files are used for this code: `m3clist002`.

```

5753 <*initex | package>
5754 <*package>
5755 \ProvidesExplPackage
5756   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
5757 \package_check_loaded_expl:
5758 </package>

```

`\l_clist_internal_clist` Scratch space for various internal uses. This comma list variable cannot be declared as such because it comes before `\clist_new:N`

```

5759 \tl_new:N \l_clist_internal_clist
      (End definition for \l_clist_internal_clist. This function is documented on page ??.)

```

`\clist_tmp:w` A temporary function for various purposes.

```

5760 \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` 5761 `\cs_new_eq:NN \clist_new:N \tl_new:N`
 5762 `\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` 5763 `\cs_new_eq:NN \clist_clear:N \tl_clear:N`
`\clist_gclear:N` 5764 `\cs_new_eq:NN \clist_clear:c \tl_clear:c`
`\clist_gclear:c` 5765 `\cs_new_eq:NN \clist_gclear:N \tl_gclear:N`
 5766 `\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` 5767 `\cs_new_eq:NN \clist_clear_new:N \tl_clear_new:N`
`\clist_gclear_new:N` 5768 `\cs_new_eq:NN \clist_clear_new:c \tl_clear_new:c`
`\clist_gclear_new:c` 5769 `\cs_new_eq:NN \clist_gclear_new:N \tl_gclear_new:N`
 5770 `\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` 5771 `\cs_new_eq:NN \clist_set_eq:NN \tl_set_eq:NN`
`\clist_set_eq:Nc` 5772 `\cs_new_eq:NN \clist_set_eq:Nc \tl_set_eq:Nc`
`\clist_set_eq:cc` 5773 `\cs_new_eq:NN \clist_set_eq:cN \tl_set_eq:cN`
`\clist_gset_eq:NN` 5774 `\cs_new_eq:NN \clist_set_eq:cc \tl_set_eq:cc`
`\clist_gset_eq:cN` 5775 `\cs_new_eq:NN \clist_gset_eq:NN \tl_gset_eq:NN`
`\clist_gset_eq:Nc` 5776 `\cs_new_eq:NN \clist_gset_eq:Nc \tl_gset_eq:Nc`
`\clist_gset_eq:cN` 5777 `\cs_new_eq:NN \clist_gset_eq:cN \tl_gset_eq:cN`
`\clist_gset_eq:cc` 5778 `\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
`\clist_concat:ccc` addition of a comma to the output. So a little work to do.

`\clist_gconcat:NNN` 5779 `\cs_new_protected_nopar:Npn \clist_concat:NNN`
`\clist_gconcat:ccc` 5780 `{ \clist_concat_aux:NNNN \tl_set:Nx }`
`\clist_concat_aux:NNNN` 5781 `\cs_new_protected_nopar:Npn \clist_gconcat:NNN`
 5782 `{ \clist_concat_aux:NNNN \tl_gset:Nx }`
 5783 `\cs_new_protected:Npn \clist_concat_aux:NNNN #1#2#3#4`
 5784 `{`
 5785 `#1 #2`
 5786 `{`
 5787 `\exp_not:o #3`
 5788 `\clist_if_empty:NF #3 { \clist_if_empty:NF #4 { , } }`
 5789 `\exp_not:o #4`
 5790 `}`

```

5791 }
5792 \cs_generate_variant:Nn \clist_concat:NNN { ccc }
5793 \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
\clist_trim_spaces_generic_aux:w
\clist_trim_spaces_generic_aux_ii:nn

```

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`.

```

5794 \cs_set:Npn \clist_tmp:w #1
5795 {
5796   \cs_new:Npn \clist_trim_spaces_generic:nw ##1 ##2 ,
5797   {
5798     \tl_trim_spaces_aux_i:w
5799     ##2
5800     \q_nil
5801     \q_mark #1 { }
5802     \q_mark \tl_trim_spaces_aux_ii:w
5803     \tl_trim_spaces_aux_iii:w
5804     #1 \q_nil
5805     \clist_trim_spaces_generic_aux:w
5806     \q_stop
5807     {##1}
5808   }
5809 }
5810 \clist_tmp:w {~}
5811 \cs_new:Npn \clist_trim_spaces_generic_aux:w #1 \q_nil #2 \q_stop
5812 { \exp_args:No \clist_trim_spaces_generic_aux_ii:nn { \use_none:n #1 } }
5813 \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
\clist_trim_spaces_aux:nn

```

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.

```

5814 \cs_new:Npn \clist_trim_spaces:n #1
5815 {
5816   \clist_trim_spaces_generic:nw
5817   { \clist_trim_spaces_aux:nn { } }
5818   \q_mark #1 ,
5819   \q_recursion_tail, \q_recursion_stop
5820 }

```

```

5821 \cs_new:Npn \clist_trim_spaces_aux:nn #1 #2
5822 {
5823   \quark_if_recursion_tail_stop:n {#2}
5824   \tl_if_empty:nTF {#2}
5825   {
5826     \clist_trim_spaces_generic:nw
5827     { \clist_trim_spaces_aux:nn {#1} } \q_mark
5828   }
5829   {
5830     #1 \exp_not:n {#2}
5831     \clist_trim_spaces_generic:nw
5832     { \clist_trim_spaces_aux:nn { , } } \q_mark
5833   }
5834 }

```

(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 5835 \cs_new_protected:Npn \clist_set:Nn #1#2
\clist_set:No 5836 { \tl_set:Nx #1 { \clist_trim_spaces:n {#2} } }
\clist_set:Nx 5837 \cs_new_protected:Npn \clist_gset:Nn #1#2
\clist_set:cn 5838 { \tl_gset:Nx #1 { \clist_trim_spaces:n {#2} } }
\clist_set:cV 5839 \cs_generate_variant:Nn \clist_set:Nn { NV , No , Nx , c , cV , co , cx }
\clist_set:co 5840 \cs_generate_variant:Nn \clist_gset:Nn { NV , No , Nx , c , cV , co , cx }
\clist_set: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.

```

\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

```

```

5841 \cs_new_protected_nopar:Npn \clist_put_left:Nn
5842 { \clist_put_left_aux:NNNn \clist_concat:NNN \clist_set:Nn }
5843 \cs_new_protected_nopar:Npn \clist_gput_left:Nn
5844 { \clist_put_left_aux:NNNn \clist_gconcat:NNN \clist_set:Nn }
5845 \cs_new_protected:Npn \clist_put_left_aux:NNNn #1#2#3#4
5846 {
5847   #2 \l_clist_internal_clist {#4}
5848   #1 #3 \l_clist_internal_clist #3
5849 }
5850 \cs_generate_variant:Nn \clist_put_left:Nn { NV , No , Nx }
5851 \cs_generate_variant:Nn \clist_put_left:Nn { c , cV , co , cx }
5852 \cs_generate_variant:Nn \clist_gput_left:Nn { NV , No , Nx }
5853 \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 ??.)

```

5857 { \clist_put_right_aux:NNNn \clist_gconcat:NNN \clist_gset:Nn }
5858 \cs_new_protected:Npn \clist_put_right_aux:NNNn #1#2#3#4
5859 {
5860   #2 \l_clist_internal_clist {#4}
5861   #1 #3 #3 \l_clist_internal_clist
5862 }
5863 \cs_generate_variant:Nn \clist_put_right:Nn { NV , No , Nx }
5864 \cs_generate_variant:Nn \clist_put_right:Nn { c , cV , co , cx }
5865 \cs_generate_variant:Nn \clist_gput_right:Nn { NV , No , Nx }
5866 \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 5867 \cs_new_protected:Npn \clist_get:NN #1#2
5868 { \exp_after:wN \clist_get_aux:wN #1 , \q_stop #2 }
5869 \cs_new_protected:Npn \clist_get_aux:wN #1 , #2 \q_stop #3
5870 { \tl_set:Nn #3 {#1} }
5871 \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 5872 \cs_new_protected_nopar:Npn \clist_pop:NN
\clist_pop_aux:NNN 5873 { \clist_pop_aux:NNN \tl_set:Nf }
\clist_pop_aux:NwNNN 5874 \cs_new_protected_nopar:Npn \clist_gpop:NN
\clist_pop_aux:wNN 5875 { \clist_pop_aux:NNN \tl_gset:Nf }
5876 \cs_new_protected:Npn \clist_pop_aux:NNN #1#2#3
5877 {
5878   \exp_after:wN \clist_pop_aux:wNNN #2 , \q_nil \q_stop #1#2#3
5879 }
5880 \cs_new_protected:Npn \clist_pop_aux:wNNN #1 , #2 \q_stop #3#4#5
5881 {
5882   \tl_set:Nn #5 {#1}
5883   \quark_if_nil:nTF {#2}
5884     { #3 #4 { } }
5885     { #3 #4 { \clist_pop_aux:w \exp_stop_f: #2 } }
5886 }
5887 \cs_new_protected:Npn \clist_pop_aux:w #1 , \q_nil {#1}
5888 \cs_generate_variant:Nn \clist_pop:NN { c }
5889 \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.

<code>\clist_push:NV</code>	5890	<code>\cs_new_eq:NN \clist_push:Nn \clist_put_left:Nn</code>
<code>\clist_push:No</code>	5891	<code>\cs_new_eq:NN \clist_push:NV \clist_put_left:NV</code>
<code>\clist_push:Nx</code>	5892	<code>\cs_new_eq:NN \clist_push:No \clist_put_left:No</code>
<code>\clist_push:cn</code>	5893	<code>\cs_new_eq:NN \clist_push:Nx \clist_put_left:Nx</code>
<code>\clist_push:cV</code>	5894	<code>\cs_new_eq:NN \clist_push:cn \clist_put_left:cn</code>
<code>\clist_push:co</code>	5895	<code>\cs_new_eq:NN \clist_push:cV \clist_put_left:cV</code>
<code>\clist_push:cx</code>	5896	<code>\cs_new_eq:NN \clist_push:co \clist_put_left:co</code>
<code>\clist_gpush:Nn</code>	5897	<code>\cs_new_eq:NN \clist_gpush:Nn \clist_gput_left:Nn</code>
<code>\clist_gpush:NV</code>	5898	<code>\cs_new_eq:NN \clist_gpush:NV \clist_gput_left:NV</code>
<code>\clist_gpush:No</code>	5899	<code>\cs_new_eq:NN \clist_gpush:No \clist_gput_left:No</code>
<code>\clist_gpush:Nx</code>	5900	<code>\cs_new_eq:NN \clist_gpush:Nx \clist_gput_left:Nx</code>
<code>\clist_gpush:cn</code>	5901	<code>\cs_new_eq:NN \clist_gpush:cn \clist_gput_left:cn</code>
<code>\clist_gpush:cV</code>	5902	<code>\cs_new_eq:NN \clist_gpush:cV \clist_gput_left:cV</code>
<code>\clist_gpush:co</code>	5903	<code>\cs_new_eq:NN \clist_gpush:co \clist_gput_left:co</code>
<code>\clist_gpush:cx</code>	5904	<code>\cs_new_eq:NN \clist_gpush:cx \clist_gput_left:cx</code>
	5905	<code>\cs_new_eq:NN \clist_gpush:cx \clist_gput_left:cx</code>

(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`

5906	<code>\cs_new_eq:NN \clist_use:N \tl_use:N</code>
5907	<code>\cs_new_eq:NN \clist_use:c \tl_use:c</code>

(End definition for \clist_use:N and \clist_use:c. These functions are documented on page ??.)

194.6 Modifying comma lists

`\l_clist_internal_remove_clist` An internal comma list for the removal routines.

5908	<code>\clist_new:N \l_clist_internal_remove_clist</code>
------	--

(End definition for \l_clist_internal_remove_clist. This function is documented on page ??.)

`\clist_remove_duplicates:N` Removing duplicates means making a new list then copying it.

<code>\clist_remove_duplicates:c</code>	5909	<code>\cs_new_protected:Npn \clist_remove_duplicates:N</code>
<code>\clist_gremove_duplicates:N</code>	5910	<code>{ \clist_remove_duplicates_aux:NN \clist_set_eq:NN }</code>
<code>\clist_gremove_duplicates:c</code>	5911	<code>\cs_new_protected:Npn \clist_gremove_duplicates:N</code>
<code>\clist_remove_duplicates_aux:NN</code>	5912	<code>{ \clist_remove_duplicates_aux:NN \clist_gset_eq:NN }</code>
	5913	<code>\cs_new_protected:Npn \clist_remove_duplicates_aux:NN #1#2</code>
	5914	<code>{</code>
	5915	<code>\clist_clear:N \l_clist_internal_remove_clist</code>
	5916	<code>\clist_map_inline:Nn #2</code>
	5917	<code>{</code>
	5918	<code>\clist_if_in:NnF \l_clist_internal_remove_clist {##1}</code>
	5919	<code>{ \clist_put_right:Nn \l_clist_internal_remove_clist {##1} }</code>
	5920	<code>}</code>
	5921	<code>#1 #2 \l_clist_internal_remove_clist</code>
	5922	<code>}</code>

5923 \cs_generate_variant:Nn \clist_remove_duplicates:N { c }

5924 \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 $\langle item \rangle$ that should be removed, surrounded with commas, and call that function followed by the expanded comma list, and another copy of the $\langle item \rangle$. 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 $\langle item \rangle$ 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.

```

5925 \cs_new_protected:Npn \clist_remove_all:Nn
5926   { \clist_remove_all_aux:NNn \tl_set:Nx }
5927 \cs_new_protected:Npn \clist_gremove_all:Nn
5928   { \clist_remove_all_aux:NNn \tl_gset:Nx }
5929 \cs_new_protected:Npn \clist_remove_all_aux:NNn #1#2#3
5930   {
5931     \cs_set:Npn \clist_tmp:w ##1 , #3 ,
5932     {
5933       ##1
5934       , \q_mark , \use_none_delimit_by_q_stop:w ,
5935       \clist_remove_all_aux:
5936     }
5937     #1 #2
5938     {
5939       \exp_after:wN \clist_remove_all_aux:
5940       #2 , \q_mark , #3 , \q_stop
5941     }
5942     \clist_if_empty:NF #2
5943     {
5944       #1 #2
5945       {
5946         \exp_args:No \exp_not:o
5947         { \exp_after:wN \use_none:n #2 }
5948       }
5949     }
5950   }
5951 \cs_new:Npn \clist_remove_all_aux:
5952   { \exp_after:wN \clist_remove_all_aux:w \clist_tmp:w , }
5953 \cs_new:Npn \clist_remove_all_aux:w #1 , \q_mark , #2 , { \exp_not:n {#1} }

```

```

5954 \cs_generate_variant:Nn \clist_remove_all:Nn { c }
5955 \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

\clist_if_empty:N Simple copies from the token list variable material.

```

\clist_if_empty:c 5956 \prg_new_eq_conditional:NNn \clist_if_empty:N \tl_if_empty:N { p , T , F , TF }
5957 \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 5958 \prg_new_eq_conditional:NNn \clist_if_eq:NN \tl_if_eq:NN { p , T , F , TF }
\clist_if_eq:cN 5959 \prg_new_eq_conditional:NNn \clist_if_eq:Nc \tl_if_eq:Nc { p , T , F , TF }
\clist_if_eq:cc 5960 \prg_new_eq_conditional:NNn \clist_if_eq:cN \tl_if_eq:cN { p , T , F , TF }
5961 \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 5962 \prg_new_protected_conditional:Npnn \clist_if_in:Nn #1#2 { T , F , TF }
\clist_if_in:cn 5963 {
\clist_if_in:cV 5964   \exp_args:No \clist_if_in_return:nn #1 {#2}
\clist_if_in:co 5965 }
\clist_if_in:nn 5966 \prg_new_protected_conditional:Npnn \clist_if_in:nn #1#2 { T , F , TF }
\clist_if_in:nV 5967 {
\clist_if_in:no 5968   \clist_set:Nn \l_clist_internal_clist {#1}
5969   \exp_args:No \clist_if_in_return:nn \l_clist_internal_clist {#2}
5970 }
\clist_if_in_return:nn 5971 \cs_new_protected:Npn \clist_if_in_return:nn #1#2
5972 {
5973   \cs_set:Npn \clist_tmp:w ##1 ,#2, { }
5974   \tl_if_empty:oTF
5975     { \clist_tmp:w ,#1, {} {} } ,#2, {
5976     { \prg_return_false: } { \prg_return_true: }
5977   }
5978   \cs_generate_variant:Nn \clist_if_in:NnT { NV , No }
5979   \cs_generate_variant:Nn \clist_if_in:NnT { c , cV , co }
5980   \cs_generate_variant:Nn \clist_if_in:NnF { NV , No }
5981   \cs_generate_variant:Nn \clist_if_in:NnF { c , cV , co }
5982   \cs_generate_variant:Nn \clist_if_in:NnTF { NV , No }
5983   \cs_generate_variant:Nn \clist_if_in:NnTF { c , cV , co }
5984   \cs_generate_variant:Nn \clist_if_in:nnT { nV , no }
5985   \cs_generate_variant:Nn \clist_if_in:nnF { nV , no }
5986   \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.

```

5987 \cs_new:Npn \clist_map_function:NN #1#2
5988 {
5989   \clist_if_empty:NF #1
5990   {
5991     \exp_last_unbraced:NNo \clist_map_function_aux:Nw #2 #1
5992     , \q_recursion_tail ,
5993     \prg_break_point:n { }
5994   }
5995 }
5996 \cs_new:Npn \clist_map_function_aux:Nw #1#2 ,
5997 {
5998   \quark_if_recursion_tail_break:n {#2}
5999   #1 {#2}
6000   \clist_map_function_aux:Nw #1
6001 }
6002 \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 each item. Space trimming is again based on `\clist_trim_spaces_generic:nw`. The 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`.

```

6003 \cs_new:Npn \clist_map_function:nN #1#2
6004 {
6005   \clist_trim_spaces_generic:nw { \clist_map_function_n_aux:Nn #2 }
6006   \q_mark #1, \q_recursion_tail,
6007   \prg_break_point:n { }
6008 }
6009 \cs_new:Npn \clist_map_function_n_aux:Nn #1 #2
6010 {
6011   \quark_if_recursion_tail_break:n {#2}
6012   \tl_if_empty:nF {#2} { \clist_map_aux_unbrace:Nw #1 #2, }
6013   \clist_trim_spaces_generic:nw { \clist_map_function_n_aux:Nn #1 }
6014   \q_mark
6015 }
6016 \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 to avoid any issues with TeX’s groups. We use a different function for each level of 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.

```

6017 \cs_new_protected:Npn \clist_map_inline:Nn #1#2
6018 {
6019   \clist_if_empty:NF #1
6020   {
6021     \int_gincr:N \g_prg_map_int
6022     \cs_gset:cpn { \clist_map_ \int_use:N \g_prg_map_int :n } ##1 {#2}
6023     \exp_last_unbraced:Nco \clist_map_function_aux:Nw
6024     { \clist_map_ \int_use:N \g_prg_map_int :n }
6025     #1 , \q_recursion_tail ,
6026     \prg_break_point:n { \int_gdecr:N \g_prg_map_int }
6027   }
6028 }
6029 \cs_new_protected:Npn \clist_map_inline:nn #1
6030 {
6031   \clist_set:Nn \l_clist_internal_clist {#1}
6032   \clist_map_inline:Nn \l_clist_internal_clist
6033 }
6034 \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.

```

6035 \cs_new_protected:Npn \clist_map_variable:NNn #1#2#3
6036 {
6037   \clist_if_empty:NF #1
6038   {
6039     \exp_args:Nno \use:nn
6040     { \clist_map_variable_aux:Nnw #2 {#3} }
6041     #1
6042     , \q_recursion_tail , \q_recursion_stop
6043     \prg_break_point:n { }
6044   }
6045 }
6046 \cs_new_protected:Npn \clist_map_variable:nNn #1
6047 {
6048   \clist_set:Nn \l_clist_internal_clist {#1}
6049   \clist_map_variable:NNn \l_clist_internal_clist
6050 }
6051 \cs_new_protected:Npn \clist_map_variable_aux:Nnw #1#2#3,
6052 {

```

```

6053 \tl_set:Nn #1 {#3}
6054 \quark_if_recursion_tail_stop:N #1
6055 \use:n {#2}
6056 \clist_map_variable_aux:Nnw #1 {#2}
6057 }
6058 \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 6059 \cs_new_eq:NN \clist_map_break: \prg_map_break:
6060 \cs_new_eq:NN \clist_map_break:n \prg_map_break:n

```

(End definition for \clist_map_break:. This function is documented on page 112.)

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 in a scratch variable, then show that variable, omitting its name from the 4-th argument.

```

\clist_show:c
\clist_show:n

```

```

6061 \cs_new_protected:Npn \clist_show:N #1
6062 {
6063   \msg_aux_show:Nnx
6064   #1
6065   { clist }
6066   { \clist_map_function:NN #1 \msg_aux_show:n }
6067 }
6068 \cs_new_protected:Npn \clist_show:n #1
6069 {
6070   \clist_set:Nn \l_clist_internal_clist {#1}
6071   \msg_aux_show:Nnx
6072   \l_clist_internal_clist
6073   { clist }
6074   { \clist_map_function:NN \l_clist_internal_clist \msg_aux_show:n }
6075 }
6076 \cs_generate_variant:Nn \clist_show:N { c }

```

(End definition for \clist_show:N and \clist_show:c. These functions are documented on page 113.)

194.10 Scratch comma lists

\l_tmpa_clist Temporary comma list variables.

```

\l_tmpb_clist 6077 \clist_new:N \l_tmpa_clist
\g_tmpa_clist 6078 \clist_new:N \l_tmpb_clist
\g_tmpb_clist 6079 \clist_new:N \g_tmpa_clist
6080 \clist_new:N \g_tmpb_clist

```

(End definition for \l_tmpa_clist and \l_tmpb_clist. These functions are documented on page 113.)

194.11 Experimental functions

`\clist_length:N` Counting the items in a comma list 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. In the case of an n-type comma-list, we could of course use `\clist_map_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).

```

6081 \cs_new:Npn \clist_length:N #1
6082 {
6083   \int_eval:n
6084   {
6085     0
6086     \clist_map_function:NN #1 \clist_length_aux:n
6087   }
6088 }
6089 \cs_new:Npn \clist_length_aux:n #1 { +1 }
6090 \cs_new:Npx \clist_length:n #1
6091 {
6092   \exp_not:N \int_eval:n
6093   {
6094     0
6095     \exp_not:N \clist_length_n_aux:w \c_space_tl
6096     #1 \exp_not:n { , \q_recursion_tail , \q_recursion_stop }
6097   }
6098 }
6099 \cs_new:Npx \clist_length_n_aux:w #1 ,
6100 {
6101   \exp_not:n { \exp_args:Nf \quark_if_recursion_tail_stop:n } {#1}
6102   \exp_not:N \tl_if_blank:nF {#1} { + \c_one }
6103   \exp_not:N \clist_length_n_aux:w \c_space_tl
6104 }
6105 \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 the list. If the item number is less than $-\langle length \rangle$ or more than $\langle length \rangle - 1$, the result is empty. If it is negative, but not less than $-\langle length \rangle$, add the $\langle length \rangle$ to the item number before performing the loop. The loop itself is very simple, return the item if the counter reached zero, otherwise, decrease the counter and repeat.

```

6106 \cs_new:Npn \clist_item:Nn #1#2
6107 {
6108   \exp_args:Nfo \clist_item_aux:nnNn
6109   { \clist_length:N #1 }
6110   #1
6111   \clist_item_N_loop:nw
6112   {#2}
6113 }
6114 \cs_new:Npn \clist_item_aux:nnNn #1#2#3#4

```

```

6115 {
6116   \int_compare:nNnTF {#4} < \c_zero
6117   {
6118     \int_compare:nNnTF {#4} < { - #1 }
6119     { \use_none_delimit_by_q_stop:w }
6120     { \exp_args:Nf #3 { \int_eval:n { #4 + #1 } } }
6121   }
6122   {
6123     \int_compare:nNnTF {#4} < {#1}
6124     { #3 {#4} }
6125     { \use_none_delimit_by_q_stop:w }
6126   }
6127   #2, \q_stop
6128 }
6129 \cs_new:Npn \clist_item_N_loop:nw #1 #2,
6130 {
6131   \int_compare:nNnTF {#1} = \c_zero
6132   { \use_i_delimit_by_q_stop:nw { \exp_not:n {#2} } }
6133   { \exp_args:Nf \clist_item_N_loop:nw { \int_eval:n { #1 - 1 } } }
6134 }
6135 \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

```

6136 \cs_new:Npn \clist_item:nn #1#2
6137 {
6138   \exp_args:Nf \clist_item_aux:nnNn
6139   { \clist_length:n {#1} }
6140   {#1}
6141   \clist_item_n_aux:nw
6142   {#2}
6143 }
6144 \cs_new:Npn \clist_item_n_aux:nw #1
6145 { \clist_item_n_loop:nw {#1} \prg_do_nothing: }
6146 \cs_new:Npn \clist_item_n_loop:nw #1 #2,
6147 {
6148   \exp_args:No \tl_if_blank:nTF {#2}
6149   { \clist_item_n_loop:nw {#1} \prg_do_nothing: }
6150   {
6151     \int_compare:nNnTF {#1} = \c_zero
6152     { \exp_args:No \clist_item_n_end:n {#2} }
6153     {
6154       \exp_args:Nf \clist_item_n_loop:nw
6155       { \int_eval:n { #1 - 1 } }
6156       \prg_do_nothing:
6157     }

```

```

6158     }
6159   }
6160   \cs_new:Npn \clist_item_n_end:n #1 #2 \q_stop
6161   {
6162     \exp_after:wN \exp_after:wN \exp_after:wN \clist_item_n_strip:w
6163     \tl_trim_spaces:n {#1} ,
6164   }
6165   \cs_new:Npn \clist_item_n_strip:w #1 , { \exp_not:n {#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.

```

6166 \cs_new_protected:Npn \clist_set_from_seq:NN
6167 { \clist_set_from_seq_aux:NNNN \clist_clear:N \tl_set:Nx }
6168 \cs_new_protected:Npn \clist_gset_from_seq:NN
6169 { \clist_set_from_seq_aux:NNNN \clist_gclear:N \tl_gset:Nx }
6170 \cs_new_protected:Npn \clist_set_from_seq_aux:NNNN #1#2#3#4
6171 {
6172   \seq_if_empty:NTF #4
6173   { #1 #3 }
6174   {
6175     #2 #3
6176     {
6177       \exp_last_unbraced:Nf \use_none:n
6178       { \seq_map_function:NN #4 \clist_wrap_item:n }
6179     }
6180   }
6181 }
6182 \cs_new:Npn \clist_wrap_item:n #1
6183 {
6184   ,
6185   \tl_if_empty:oTF { \clist_set_from_seq_aux:w #1 ~ , #1 ~ }
6186   { \exp_not:n {#1} }
6187   { \exp_not:n { {#1} } }
6188 }
6189 \cs_new:Npn \clist_set_from_seq_aux:w #1 , #2 ~ { }
6190 \cs_generate_variant:Nn \clist_set_from_seq:NN { Nc }
6191 \cs_generate_variant:Nn \clist_set_from_seq:NN { c , cc }
6192 \cs_generate_variant:Nn \clist_gset_from_seq:NN { Nc }
6193 \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.
\clist_const:Nx 6194 \cs_new_protected:Npn \clist_const:Nn #1#2
\clist_const:cx 6195 { \tl_const:Nx #1 { \clist_trim_spaces:n {#2} } }

```

6196 \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 braces. The argument of `\tl_if_empty:oTF` is empty if #1 is ? followed by blank spaces (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.

```

6197 \prg_new_conditional:Npnn \clist_if_empty:n #1 { p , T , F , TF }
6198 {
6199   \clist_if_empty_n_aux:w ? #1
6200   , \q_mark \prg_return_false:
6201   , \q_mark \prg_return_true:
6202   \q_stop
6203 }
6204 \cs_new:Npn \clist_if_empty_n_aux:w #1 ,
6205 {
6206   \tl_if_empty:oTF { \use_none:nn #1 ? }
6207   { \clist_if_empty_n_aux:w ? }
6208   { \clist_if_empty_n_aux:wNw }
6209 }
6210 \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
6211 <*deprecated>
6212 \cs_new_eq:NN \clist_top:NN \clist_get:NN
6213 \cs_new_eq:NN \clist_top:cN \clist_get:cN
6214 </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
6215 <*deprecated>
6216 \cs_new_eq:NN \clist_remove_element:Nn \clist_remove_all:Nn
6217 \cs_new_eq:NN \clist_gremove_element:Nn \clist_gremove_all:Nn
6218 </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`

6219 `*deprecated`

6220 `\cs_new_eq:NN \clist_display:N \clist_show:N`

6221 `\cs_new_eq:NN \clist_display:c \clist_show:c`

6222 `\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 2011-12-31.

`\clist_trim_spaces:N`

`\clist_trim_spaces:c`

`\clist_gtrim_spaces:N`

`\clist_gtrim_spaces:c`

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.

6223 `*deprecated`

6224 `\cs_new_protected:Npn \clist_trim_spaces:N #1 { \clist_set:No #1 {#1} }`

6225 `\cs_new_protected:Npn \clist_gtrim_spaces:N #1 { \clist_gset:No #1 {#1} }`

6226 `\cs_generate_variant:Nn \clist_trim_spaces:N { c }`

6227 `\cs_generate_variant:Nn \clist_gtrim_spaces:N { c }`

6228 `\deprecated`

(End definition for `\clist_trim_spaces:N` and others. These functions are documented on page ??.)

6229 `\initex | package`

195 l3prop implementation

The following test files are used for this code: `m3prop001`.

6230 `*initex | package`

6231 `*package`

6232 `\ProvidesExplPackage`

6233 `{\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}`

6234 `\package_check_loaded_expl:`

6235 `\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.

6236 `\quark_new:N \q_prop`

(End definition for `\q_prop`. This function is documented on page 120.)

`\c_empty_prop` An empty prop contains exactly one `\q_prop`.

6237 `\tl_const:Nn \c_empty_prop { \q_prop }`

(End definition for `\c_empty_prop`. This function is documented on page 120.)

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
`\prop_new:c` need to do things by hand.

```
6238 \cs_new_protected:Npn \prop_new:N #1 { \cs_new_eq:NN #1 \c_empty_prop }
6239 \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 6240 \cs_new_protected:Npn \prop_clear:N #1 { \cs_set_eq:NN #1 \c_empty_prop }
\prop_gclear:N 6241 \cs_new_protected:Npn \prop_clear:c #1 { \cs_set_eq:cN {#1} \c_empty_prop }
\prop_gclear:c 6242 \cs_new_protected:Npn \prop_gclear:N #1 { \cs_gset_eq:NN #1 \c_empty_prop }
6243 \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 6244 \cs_new_protected:Npn \prop_clear_new:N #1
\prop_gclear_new:N 6245 { \cs_if_exist:NTF #1 { \prop_clear:N #1 } { \prop_new:N #1 } }
\prop_gclear_new:c 6246 \cs_generate_variant:Nn \prop_clear_new:N { c }
6247 \cs_new_protected:Npn \prop_gclear_new:N #1
6248 { \cs_if_exist:NTF #1 { \prop_gclear:N #1 } { \prop_new:N #1 } }
6249 \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 6250 \cs_new_eq:NN \prop_set_eq:NN \tl_set_eq:NN
\prop_set_eq:Nc 6251 \cs_new_eq:NN \prop_set_eq:Nc \tl_set_eq:Nc
\prop_set_eq:cc 6252 \cs_new_eq:NN \prop_set_eq:cN \tl_set_eq:cN
\prop_gset_eq:NN 6253 \cs_new_eq:NN \prop_set_eq:cc \tl_set_eq:cc
\prop_gset_eq:cN 6254 \cs_new_eq:NN \prop_gset_eq:NN \tl_gset_eq:NN
\prop_gset_eq:Nc 6255 \cs_new_eq:NN \prop_gset_eq:Nc \tl_gset_eq:Nc
\prop_gset_eq:cN 6256 \cs_new_eq:NN \prop_gset_eq:cN \tl_gset_eq:cN
\prop_gset_eq:cc 6257 \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, $\{\langle extract_1 \rangle\} \{\langle value \rangle\} \{\langle extract_2 \rangle\}$. In order for $\langle extract_1 \rangle \langle extract_2 \rangle$ to be a well-formed property list, $\langle extract_1 \rangle$ has a leading and trailing $\backslash q_prop$, retaining exactly the structure of a property list, while $\langle extract_2 \rangle$ omits the leading $\backslash q_prop$.

If the key is not there, then $\langle extra\ argument \rangle$ is $? \backslash use_ii:nn \{ \}$, and $\backslash prop_split_aux:nnnn ? \backslash u$ removes the three brace groups that just follow. Then $\backslash use_ii:nn$ removes the true branch, leaving the false branch, with no trailing material.

```

6258 \cs_new_protected:Npn \prop_split:NnTF #1#2
6259 { \exp_args:NNo \prop_split_aux:NnTF #1 { \tl_to_str:n {#2} } }
6260 \cs_new_protected:Npn \prop_split_aux:NnTF #1#2
6261 {
6262   \cs_set_protected:Npn \prop_split_aux:w
6263     ##1 \q_prop #2 \q_prop ##2 ##3 ##4 \q_mark ##5 \q_stop
6264     { \prop_split_aux:nnnn ##3 { {##1 \q_prop } {##2} {##4} } }
6265   \exp_after:wN \prop_split_aux:w #1 \q_mark
6266     \q_prop #2 \q_prop { } { ? \use_ii:nn { } } \q_mark \q_stop
6267 }
6268 \cs_new:Npn \prop_split_aux:nnnn #1#2#3#4 { #3 #2 }
6269 \cs_new_protected:Npn \prop_split_aux:w { }

```

(End definition for $\backslash prop_split:NnTF$. This function is documented on page ??.)

$\backslash prop_split:Nnn$ The goal here is to provide a common interface for both true and false branches of $\backslash prop_split:NnTF$. 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 $\backslash q_no_value$, 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 $\backslash q_prop$.

```

6270 \cs_new_protected:Npn \prop_split:Nnn #1#2#3
6271 {
6272   \prop_split:NnTF #1 {#2}
6273   {#3}
6274   { \exp_args:Nno \use:n {#3} {#1} { \q_no_value } { } }
6275 }

```

(End definition for $\backslash prop_split:Nnn$. This function is documented on page 120.)

$\backslash prop_del:Nn$ 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
6276 \cs_new_protected:Npn \prop_del:Nn #1#2
6277 { \prop_split:NnTF #1 {#2} { \prop_del_aux:NNnnn \tl_set:Nn #1 } { } }
6278 \cs_new_protected:Npn \prop_gdel:Nn #1#2
6279 { \prop_split:NnTF #1 {#2} { \prop_del_aux:NNnnn \tl_gset:Nn #1 } { } }
6280 \cs_new_protected:Npn \prop_del_aux:NNnnn #1#2#3#4#5
6281 { #1 #2 { #3 #5 } }
6282 \cs_generate_variant:Nn \prop_del:Nn { NV }
6283 \cs_generate_variant:Nn \prop_del:Nn { c , cV }
6284 \cs_generate_variant:Nn \prop_gdel:Nn { NV }
6285 \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,
 \prop_get:NVN just set the token list variable to the return value, otherwise to \q_no_value.

```

\prop_get:NoN 6286 \cs_new_protected:Npn \prop_get:NnN #1#2#3
\prop_get:cnN 6287 {
\prop_get:cVN 6288   \prop_split:NnTF #1 {#2}
\prop_get:NoN 6289   { \prop_get_aux:Nnnn #3 }
\prop_get_aux:Nnnn 6290   { \tl_set:Nn #3 { \q_no_value } }
6291 }
6292 \cs_new_protected:Npn \prop_get_aux:Nnnn #1#2#3#4
6293 { \tl_set:Nn #1 {#3} }
6294 \cs_generate_variant:Nn \prop_get:NnN { NV , No }
6295 \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 6296 \cs_new_protected:Npn \prop_pop:NnN #1#2#3
\prop_gpop:NnN 6297 {
\prop_gpop:NoN 6298   \prop_split:NnTF #1 {#2}
\prop_gpop:cnN 6299   { \prop_pop_aux:NNNnnn \tl_set:Nn #1 #3 }
\prop_gpop:coN 6300   { \tl_set:Nn #3 { \q_no_value } }
\prop_pop_aux:NNNnnn 6301 }
6302 \cs_new_protected:Npn \prop_gpop:NnN #1#2#3
6303 {
6304   \prop_split:NnTF #1 {#2}
6305   { \prop_pop_aux:NNNnnn \tl_gset:Nn #1 #3 }
6306   { \tl_set:Nn #3 { \q_no_value } }
6307 }
6308 \cs_new_protected:Npn \prop_pop_aux:NNNnnn #1#2#3#4#5#6
6309 {
6310   \tl_set:Nn #3 {#5}
6311   #1 #2 { #4 #6 }
6312 }
6313 \cs_generate_variant:Nn \prop_pop:NnN { No }
6314 \cs_generate_variant:Nn \prop_pop:NnN { c , co }
6315 \cs_generate_variant:Nn \prop_gpop:NnN { No }
6316 \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 6317 \cs_new_protected:Npn \prop_put:Nnn { \prop_put_aux:NNnn \tl_set:Nx }
\prop_put:NVn 6318 \cs_new_protected:Npn \prop_gput:Nnn { \prop_put_aux:NNnn \tl_gset:Nx }
\prop_put:NVV 6319 \cs_new_protected:Npn \prop_put_aux:NNnn #1#2#3#4
\prop_put:Non 6320 {

```

```

6321     \prop_split:Nnn #2 {#3} { \prop_put_aux:NNnnnnn #1 #2 {#3} {#4} }
6322   }
6323   \cs_new_protected:Npn \prop_put_aux:NNnnnnn #1#2#3#4#5#6#7
6324   {
6325     #1 #2
6326     {
6327       \exp_not:n { #5 #7 }
6328       \tl_to_str:n {#3} \exp_not:n { \q_prop {#4} \q_prop }
6329     }
6330   }
6331   \cs_generate_variant:Nn \prop_put:Nnn
6332   { NnV , Nno , Nnx , NV , NVV , No , Noo }
6333   \cs_generate_variant:Nn \prop_put:Nnn
6334   { c , cnV , cno , cnx , cV , cVV , co , coo }
6335   \cs_generate_variant:Nn \prop_gput:Nnn
6336   { NnV , Nno , Nnx , NV , NVV , No , Noo }
6337   \cs_generate_variant:Nn \prop_gput:Nnn
6338   { 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

```

6339   \cs_new_protected_nopar:Npn \prop_put_if_new:Nnn
6340   { \prop_put_if_new_aux:NNnn \tl_put_right:Nx }
6341   \cs_new_protected_nopar:Npn \prop_gput_if_new:Nnn
6342   { \prop_put_if_new_aux:NNnn \tl_gput_right:Nx }
6343   \cs_new_protected:Npn \prop_put_if_new_aux:NNnn #1#2#3#4
6344   {
6345     \prop_split:NnTF #2 {#3}
6346     { \use_none:nnn }
6347     {
6348       #1 #2
6349       { \tl_to_str:n {#3} \exp_not:n { \q_prop {#4} \q_prop } }
6350     }
6351   }
6352   \cs_generate_variant:Nn \prop_put_if_new:Nnn { c }
6353   \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
6354   \prg_new_conditional:Npnn \prop_if_empty:N #1 { p , T , F , TF }
6355   {
6356     \if_meaning:w #1 \c_empty_prop
6357     \prg_return_true:
6358   }

```

```

6359     \prg_return_false:
6360     \fi:
6361   }
6362   \cs_generate_variant:Nn \prop_if_empty_p:N {c}
6363   \cs_generate_variant:Nn \prop_if_empty:NTF {c}
6364   \cs_generate_variant:Nn \prop_if_empty:NT {c}
6365   \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 \prg_new_protected_conditional:Npnn \prop_if_in:Nn #1 #2
\prop_if_in:cV {
\prop_if_in:co   \prop_split:NnTF #1 {#2}
\prop_if_in_aux:nwn {
\prop_if_in_aux:N   \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.

```

6366 \prg_new_conditional:Npnn \prop_if_in:Nn #1#2 { p , T , F , TF }
6367 {
6368   \exp_last_unbraced:Noo \prop_if_in_aux:nwn
6369   { \tl_to_str:n {#2} } #1
6370   \tl_to_str:n {#2} \q_prop { }
6371   \q_recursion_tail
6372   \prg_break_point:n { }
6373 }
6374 \cs_new:Npn \prop_if_in_aux:nwn #1 \q_prop #2 \q_prop #3
6375 {
6376   \str_if_eq:xxTF {#1} {#2}
6377   { \prop_if_in_aux:N }
6378   { \prop_if_in_aux:nwn {#1} }
6379 }
6380 \cs_new:Npn \prop_if_in_aux:N #1
6381 {
6382   \if_meaning:w \q_prop #1

```

```

6383     \prg_return_true:
6384     \else:
6385         \prg_return_false:
6386     \fi:
6387     \prop_map_break:
6388 }
6389 \cs_generate_variant:Nn \prop_if_in_p:Nn { NV , No }
6390 \cs_generate_variant:Nn \prop_if_in_p:Nn { c , cV , co }
6391 \cs_generate_variant:Nn \prop_if_in:NnT { NV , No }
6392 \cs_generate_variant:Nn \prop_if_in:NnT { c , cV , co }
6393 \cs_generate_variant:Nn \prop_if_in:NnF { NV , No }
6394 \cs_generate_variant:Nn \prop_if_in:NnF { c , cV , co }
6395 \cs_generate_variant:Nn \prop_if_in:NnTF { NV , No }
6396 \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.

```

6397 \prg_new_protected_conditional:Npnn \prop_get:NnN #1#2#3 { T , F , TF }
6398 {
6399     \prop_split:NnTF #1 {#2}
6400     { \prop_get_aux_true:Nnnn #3 }
6401     { \prg_return_false: }
6402 }
6403 \cs_new_protected:Npn \prop_get_aux_true:Nnnn #1#2#3#4
6404 {
6405     \tl_set:Nn #1 {#3}
6406     \prg_return_true:
6407 }
6408 \cs_generate_variant:Nn \prop_get:NnNT { NV , No }
6409 \cs_generate_variant:Nn \prop_get:NnNF { NV , No }
6410 \cs_generate_variant:Nn \prop_get:NnNTF { NV , No }
6411 \cs_generate_variant:Nn \prop_get:NnNT { c , cV , co }
6412 \cs_generate_variant:Nn \prop_get:NnNF { c , cV , co }
6413 \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
 \prop_map_function:cc
 \prop_map_function_aux:Nwn

```

6414 \cs_new:Npn \prop_map_function:NN #1#2
6415 {
6416     \exp_last_unbraced:NNo \prop_map_function_aux:Nwn #2
6417     #1 \q_recursion_tail \q_prop { }

```

```

6418     \prg_break_point:n { }
6419   }
6420   \cs_new:Npn \prop_map_function_aux:Nwn #1 \q_prop #2 \q_prop #3
6421   {
6422     \if_meaning:w \q_recursion_tail #2
6423     \exp_after:wN \prop_map_break:
6424     \fi:
6425     #1 {#2} {#3}
6426     \prop_map_function_aux:Nwn #1
6427   }
6428   \cs_generate_variant:Nn \prop_map_function:NN { Nc }
6429   \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 6430 \cs_new_protected:Npn \prop_map_inline:Nn #1#2
6431   {
6432     \int_gincr:N \g_prg_map_int
6433     \cs_gset:cpn { prop_map_inline_ \int_use:N \g_prg_map_int :nn }
6434     ##1##2 {#2}
6435     \exp_last_unbraced:Nco \prop_map_function_aux:Nwn
6436     { prop_map_inline_ \int_use:N \g_prg_map_int :nn }
6437     #1
6438     \q_recursion_tail \q_prop { }
6439     \prg_break_point:n { \int_gdecr:N \g_prg_map_int }
6440   }
6441   \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 6442 \cs_new_eq:NN \prop_map_break: \prg_map_break:
6443 \cs_new_eq:NN \prop_map_break:n \prg_map_break:n

```

(End definition for \prop_map_break:. This function is documented on page 119.)

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.

```

6444 \cs_new_protected:Npn \prop_show:N #1
6445   {
6446     \msg_aux_show:Nnx
6447     #1
6448     { prop }
6449     { \prop_map_function:NN #1 \msg_aux_show:nn }
6450   }
6451   \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

<code>\prop_pop:NnN</code> <code>\prop_pop:cnN</code> <code>\prop_gpop:cnN</code> <code>\prop_gpop:cnN</code> <code>\prop_pop_aux_true:NNNnnn</code>	<p>Popping an item from a property list, keeping track of whether the key was present or not, is implemented as a conditional. If the key was missing, neither the property list, nor the token list are altered. Otherwise, <code>\prg_return_true:</code> is used after the assignments.</p> <pre> 6452 \prg_new_protected_conditional:Npnn \prop_pop:NnN #1#2#3 { T , F , TF } 6453 { 6454 \prop_split:NnTF #1 {#2} 6455 { \prop_pop_aux_true:NNNnnn \tl_set:Nn #1 #3 } 6456 { \prg_return_false: } 6457 } 6458 \prg_new_protected_conditional:Npnn \prop_gpop:NnN #1#2#3 { T , F , TF } 6459 { 6460 \prop_split:NnTF #1 {#2} 6461 { \prop_pop_aux_true:NNNnnn \tl_gset:Nn #1 #3 } 6462 { \prg_return_false: } 6463 } 6464 \cs_new_protected:Npn \prop_pop_aux_true:NNNnnn #1#2#3#4#5#6 6465 { 6466 \tl_set:Nn #3 {#5} 6467 #1 #2 { #4 #6 } 6468 \prg_return_true: 6469 } 6470 \cs_generate_variant:Nn \prop_pop:NnNT { c } 6471 \cs_generate_variant:Nn \prop_pop:NnNF { c } 6472 \cs_generate_variant:Nn \prop_pop:NnNTF { c } 6473 \cs_generate_variant:Nn \prop_gpop:NnNT { c } 6474 \cs_generate_variant:Nn \prop_gpop:NnNF { c } 6475 \cs_generate_variant:Nn \prop_gpop:NnNTF { c } </pre> <p style="text-align: center;"><i>(End definition for <code>\prop_pop:NnN</code> and others. These functions are documented on page ??.)</i></p>
<code>\prop_map_tokens:Nn</code> <code>\prop_map_tokens:cn</code> <code>\prop_map_tokens_aux:nwn</code>	<p>The mapping grabs one key–value pair at a time, and stops when reaching the marker key <code>\q_recursion_tail</code>, which cannot appear in normal keys since those are strings. The odd construction <code>\use:n {#1}</code> allows #1 to contain any token.</p> <pre> 6476 \cs_new:Npn \prop_map_tokens:Nn #1#2 6477 { 6478 \exp_last_unbraced:Nno \prop_map_tokens_aux:nwn {#2} #1 6479 \q_recursion_tail \q_prop { } 6480 \prg_break_point:n { } 6481 } 6482 \cs_new:Npn \prop_map_tokens_aux:nwn #1 \q_prop #2 \q_prop #3 6483 { 6484 \if_meaning:w \q_recursion_tail #2 6485 \exp_after:wN \prop_map_break: 6486 \fi: 6487 \use:n {#1} {#2} {#3} 6488 \prop_map_tokens_aux:nwn {#1} 6489 } 6490 \cs_generate_variant:Nn \prop_map_tokens:Nn { c } </pre>

(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 simple instance of mapping some tokens. Map the function `\prop_get_aux:nnn` which takes as its three arguments the $\langle key \rangle$ that we are looking for, the current $\langle key \rangle$ and the current $\langle value \rangle$. If the $\langle keys \rangle$ match, the $\langle value \rangle$ is returned. If none of the keys match, this expands to nothing.

```

6491 \cs_new:Npn \prop_get:Nn #1#2
6492 {
6493   \exp_last_unbraced:Noo \prop_get_Nn_aux:nwn
6494   { \tl_to_str:n {#2} } #1
6495   \tl_to_str:n {#2} \q_prop { }
6496   \prg_break_point:n { }
6497 }
6498 \cs_new:Npn \prop_get_Nn_aux:nwn #1 \q_prop #2 \q_prop #3
6499 {
6500   \str_if_eq:xxTF {#1} {#2}
6501   { \prg_map_break:n { \exp_not:n {#3} } }
6502   { \prop_get_Nn_aux:nwn {#1} }
6503 }
6504 \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
6505 <deprecated>
6506 \cs_new_eq:NN \prop_display:N \prop_show:N
6507 \cs_new_eq:NN \prop_display:c \prop_show:c
6508 </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:NVN
\prop_gget:cnN
\prop_gget:cVN
\prop_gget_aux:Nnnn
6509 <deprecated>
6510 \cs_new_protected:Npn \prop_gget:NnN #1#2#3
6511 { \prop_split:Nnn #1 {#2} { \prop_gget_aux:Nnnn #3 } }
6512 \cs_new_protected:Npn \prop_gget_aux:Nnnn #1#2#3#4
6513 { \tl_gset:Nn #1 {#3} }
6514 \cs_generate_variant:Nn \prop_gget:NnN { NV }
6515 \cs_generate_variant:Nn \prop_gget:NnN { c , cV }
6516 </deprecated>

```

(End definition for `\prop_gget:NnN` and others. These functions are documented on page ??.)

`\prop_get_gdel:NnN` This name seems very odd.

```

6517 <*deprecated>
6518 \cs_new_eq:NN \prop_get_gdel:NnN \prop_gpop:NnN
6519 </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

```

6520 <*deprecated>
6521 \cs_generate_variant:Nn \prop_if_in:NnT { cc }
6522 \cs_generate_variant:Nn \prop_if_in:NnF { cc }
6523 \cs_generate_variant:Nn \prop_if_in:NnTF { cc }
6524 </deprecated>
      (End definition for \prop_if_in:cc. This function is documented on page ??.)

```

`\prop_gput:ccx` Another one.

```

6525 <*deprecated>
6526 \cs_generate_variant:Nn \prop_gput:Nnn { ccx }
6527 </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 6528 <*deprecated>
\prop_if_eq:cN 6529 \prg_new_eq_conditional:NNn \prop_if_eq:NN \tl_if_eq:NN { p , T , F , TF }
\prop_if_eq:cc 6530 \prg_new_eq_conditional:NNn \prop_if_eq:cN \tl_if_eq:cN { p , T , F , TF }
6531 \prg_new_eq_conditional:NNn \prop_if_eq:Nc \tl_if_eq:Nc { p , T , F , TF }
6532 \prg_new_eq_conditional:NNn \prop_if_eq:cc \tl_if_eq:cc { p , T , F , TF }
6533 </deprecated>
      (End definition for \prop_if_eq:NN and others. These functions are documented on page ??.)
6534 </initex | package>

```

196 l3box implementation

```

6535 <*initex | package>
6536 <*package>
6537 \ProvidesExplPackage
6538   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
6539 \package_check_loaded_expl:
6540 </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 $\langle box \rangle$ register: remember that box 255 is not generally available.

`\box_new:c`

```

6541 <*package>
6542 \cs_new_protected:Npn \box_new:N #1
6543 {
6544   \chk_if_free_cs:N #1
6545   \newbox #1
6546 }
6547 </package>
6548 \cs_generate_variant:Nn \box_new:N { c }

```

`\box_clear:N` Clear a $\langle box \rangle$ register.

`\box_clear:c`

`\box_gclear:N`

`\box_gclear:c`

```

6549 \cs_new_protected:Npn \box_clear:N #1
6550 { \box_set_eq:NN #1 \c_empty_box }
6551 \cs_new_protected:Npn \box_gclear:N #1
6552 { \box_gset_eq:NN #1 \c_empty_box }
6553 \cs_generate_variant:Nn \box_clear:N { c }
6554 \cs_generate_variant:Nn \box_gclear:N { c }

```

`\box_clear_new:N` Clear or new.

`\box_clear_new:c`

`\box_gclear_new:N`

`\box_gclear_new:c`

```

6555 \cs_new_protected:Npn \box_clear_new:N #1
6556 {
6557   \cs_if_exist:NTF #1
6558   { \box_set_eq:NN #1 \c_empty_box }
6559   { \box_new:N #1 }
6560 }
6561 \cs_new_protected:Npn \box_gclear_new:N #1
6562 {
6563   \cs_if_exist:NTF #1
6564   { \box_gset_eq:NN #1 \c_empty_box }
6565   { \box_new:N #1 }
6566 }
6567 \cs_generate_variant:Nn \box_clear_new:N { c }
6568 \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`

`\box_set_eq:Nc`

`\box_set_eq:cc`

`\box_gset_eq:NN`

`\box_gset_eq:cN`

`\box_gset_eq:Nc`

```

6569 \cs_new_protected:Npn \box_set_eq:NN #1#2
6570 { \tex_setbox:D #1 \tex_copy:D #2 }
6571 \cs_new_protected:Npn \box_gset_eq:NN
6572 { \tex_global:D \box_set_eq:NN }
6573 \cs_generate_variant:Nn \box_set_eq:NN { cN , Nc , cc }
6574 \cs_generate_variant:Nn \box_gset_eq:NN { cN , Nc , cc }

```

`\box_set_eq_clear:NN` 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:cN`

`\box_set_eq_clear:Nc`

`\box_set_eq_clear:cc`

`\box_gset_eq_clear:NN`

`\box_gset_eq_clear:cN`

`\box_gset_eq_clear:Nc`

`\box_gset_eq_clear:cc`

```

6575 \cs_new_protected:Npn \box_set_eq_clear:NN #1#2
6576 { \tex_setbox:D #1 \tex_box:D #2 }
6577 \cs_new_protected:Npn \box_gset_eq_clear:NN
6578 { \tex_global:D \box_set_eq_clear:NN }
6579 \cs_generate_variant:Nn \box_set_eq_clear:NN { cN , Nc , cc }
6580 \cs_generate_variant:Nn \box_gset_eq_clear:NN { cN , Nc , cc }

```

196.2 Measuring and setting box dimensions

<code>\box_ht:N</code>	Accessing the height, depth, and width of a $\langle box \rangle$ register.
<code>\box_ht:c</code>	6581 <code>\cs_new_eq:NN \box_ht:N \tex_ht:D</code>
<code>\box_dp:N</code>	6582 <code>\cs_new_eq:NN \box_dp:N \tex_dp:D</code>
<code>\box_dp:c</code>	6583 <code>\cs_new_eq:NN \box_wd:N \tex_wd:D</code>
<code>\box_wd:N</code>	6584 <code>\cs_generate_variant:Nn \box_ht:N { c }</code>
<code>\box_wd:c</code>	6585 <code>\cs_generate_variant:Nn \box_dp:N { c }</code>
	6586 <code>\cs_generate_variant:Nn \box_wd:N { c }</code>
 <code>\box_set_ht:Nn</code>	 Measuring is easy: all primitive work. These primitives are not expandable, so the derived
<code>\box_set_ht:cn</code>	functions are not either.
<code>\box_set_dp:Nn</code>	6587 <code>\cs_new_protected:Npn \box_set_dp:Nn #1#2</code>
<code>\box_set_dp:cn</code>	6588 <code>{ \box_dp:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
<code>\box_set_wd:Nn</code>	6589 <code>\cs_new_protected:Npn \box_set_ht:Nn #1#2</code>
<code>\box_set_wd:cn</code>	6590 <code>{ \box_ht:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
	6591 <code>\cs_new_protected:Npn \box_set_wd:Nn #1#2</code>
	6592 <code>{ \box_wd:N #1 \dim_eval:w #2 \dim_eval_end: }</code>
	6593 <code>\cs_generate_variant:Nn \box_set_ht:Nn { c }</code>
	6594 <code>\cs_generate_variant:Nn \box_set_dp:Nn { c }</code>
	6595 <code>\cs_generate_variant:Nn \box_set_wd:Nn { c }</code>

196.3 Using boxes

<code>\box_use_clear:N</code>	Using a $\langle box \rangle$. These are just T _E X primitives with meaningful names.
<code>\box_use_clear:c</code>	6596 <code>\cs_new_eq:NN \box_use_clear:N \tex_box:D</code>
<code>\box_use:N</code>	6597 <code>\cs_new_eq:NN \box_use:N \tex_copy:D</code>
<code>\box_use:c</code>	6598 <code>\cs_generate_variant:Nn \box_use_clear:N { c }</code>
	6599 <code>\cs_generate_variant:Nn \box_use:N { c }</code>
 <code>\box_move_left:nn</code>	 Move box material in different directions.
<code>\box_move_right:nn</code>	6600 <code>\cs_new_protected:Npn \box_move_left:nn #1#2</code>
<code>\box_move_up:nn</code>	6601 <code>{ \tex_moveleft:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
<code>\box_move_down:nn</code>	6602 <code>\cs_new_protected:Npn \box_move_right:nn #1#2</code>
	6603 <code>{ \tex_moveright:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
	6604 <code>\cs_new_protected:Npn \box_move_up:nn #1#2</code>
	6605 <code>{ \tex_raise:D \dim_eval:w #1 \dim_eval_end: #2 }</code>
	6606 <code>\cs_new_protected:Npn \box_move_down:nn #1#2</code>
	6607 <code>{ \tex_lower:D \dim_eval:w #1 \dim_eval_end: #2 }</code>

196.4 Box conditionals

<code>\if_hbox:N</code>	The primitives for testing if a $\langle box \rangle$ is empty/void or which type of box it is.
<code>\if_vbox:N</code>	6608 <code>\cs_new_eq:NN \if_hbox:N \tex_ifhbox:D</code>
<code>\if_box_empty:N</code>	6609 <code>\cs_new_eq:NN \if_vbox:N \tex_ifvbox:D</code>
	6610 <code>\cs_new_eq:NN \if_box_empty:N \tex_ifvoid:D</code>

```

\box_if_horizontal:N
\box_if_horizontal:c
\box_if_vertical:N
\box_if_vertical:c
6611 \prg_new_conditional:Npnn \box_if_horizontal:N #1 { p , T , F , TF }
6612 { \if_hbox:N #1 \prg_return_true: \else: \prg_return_false: \fi: }
6613 \prg_new_conditional:Npnn \box_if_vertical:N #1 { p , T , F , TF }
6614 { \if_vbox:N #1 \prg_return_true: \else: \prg_return_false: \fi: }
6615 \cs_generate_variant:Nn \box_if_horizontal_p:N { c }
6616 \cs_generate_variant:Nn \box_if_horizontal:NT { c }
6617 \cs_generate_variant:Nn \box_if_horizontal:NF { c }
6618 \cs_generate_variant:Nn \box_if_horizontal:NTF { c }
6619 \cs_generate_variant:Nn \box_if_vertical_p:N { c }
6620 \cs_generate_variant:Nn \box_if_vertical:NT { c }
6621 \cs_generate_variant:Nn \box_if_vertical:NF { c }
6622 \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
6623 \prg_new_conditional:Npnn \box_if_empty:N #1 { p , T , F , TF }
6624 { \if_box_empty:N #1 \prg_return_true: \else: \prg_return_false: \fi: }
6625 \cs_generate_variant:Nn \box_if_empty_p:N { c }
6626 \cs_generate_variant:Nn \box_if_empty:NT { c }
6627 \cs_generate_variant:Nn \box_if_empty:NF { c }
6628 \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
6629 \cs_new_protected:Npn \box_set_to_last:N #1
6630 { \tex_setbox:D #1 \tex_lastbox:D }
6631 \cs_new_protected:Npn \box_gset_to_last:N
6632 { \tex_global:D \box_set_to_last:N }
6633 \cs_generate_variant:Nn \box_set_to_last:N { c }
6634 \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
6635 \*package
6636 \cs_new_eq:NN \c_empty_box \voidb@x
6637 \*package
6638 \*initex
6639 \box_new:N \c_empty_box
6640 \*initex

(End definition for \c_empty_box. This function is documented on page 126.)

```

196.7 Scratch boxes

```

\l_tmpa_box
\l_tmpb_box
6641 <*package>
6642 \cs_new_eq:NN \l_tmpa_box \@tempboxa
6643 </package>
6644 <*initex>
6645 \box_new:N \l_tmpa_box
6646 </initex>
6647 \box_new:N \l_tmpb_box
        (End definition for \l_tmpa_box and \l_tmpb_box. These functions are documented on page 126.)

```

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`

```

6648 \cs_new_protected:Npn \box_show:N #1
6649 {
6650   \cs_if_exist:NTF #1
6651   { \tex_showbox:D \use:n {#1} }
6652   {
6653     \msg_kernel_error:nxx { kernel } { variable-not-defined }
6654     { \token_to_str:N #1 }
6655   }
6656 }
6657 \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
6658 \cs_new_protected:Npn \box_show:Nnn #1#2#3
6659 {
6660   \group_begin:
6661   \int_set:Nn \tex_showboxbreadth:D {#2}
6662   \int_set:Nn \tex_showboxdepth:D {#3}
6663   \int_set_eq:NN \tex_tracingonline:D \c_one
6664   \box_show:N #1
6665   \group_end:
6666 }
6667 \cs_generate_variant:Nn \box_show:Nnn { c }
6668 \cs_new_protected:Npn \box_show_full:N #1
6669 { \box_show:Nnn #1 { \c_max_int } { \c_max_int } }
6670 \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.

```
6671 \cs_new_protected:Npn \hbox:n { \tex_hbox:D \scan_stop: }
```

(End definition for `\hbox:n`. This function is documented on page 126.)

`\hbox_set:Nn`

`\hbox_set:cn`

```
6672 \cs_new_protected:Npn \hbox_set:Nn #1#2 { \tex_setbox:D #1 \tex_hbox:D {#2} }
```

`\hbox_gset:Nn` 6673 `\cs_new_protected:Npn \hbox_gset:Nn { \tex_global:D \hbox_set:Nn }`

`\hbox_gset:cn` 6674 `\cs_generate_variant:Nn \hbox_set:Nn { c }`

```
6675 \cs_generate_variant:Nn \hbox_gset:Nn { c }
```

(End definition for `\hbox_set:Nn` and `\hbox_set:cn`. These functions are documented on page ??.)

`\hbox_set_to_wd:Nnn` Storing material in a horizontal box with a specified width.

`\hbox_set_to_wd:cnn` 6676 `\cs_new_protected:Npn \hbox_set_to_wd:Nnn #1#2#3`

```
6677 { \tex_setbox:D #1 \tex_hbox:D to \dim_eval:w #2 \dim_eval_end: {#3} }
```

`\hbox_gset_to_wd:Nnn` 6678 `\cs_new_protected:Npn \hbox_gset_to_wd:Nnn`

```
6679 { \tex_global:D \hbox_set_to_wd:Nnn }
```

```
6680 \cs_generate_variant:Nn \hbox_set_to_wd:Nnn { c }
```

```
6681 \cs_generate_variant:Nn \hbox_gset_to_wd:Nnn { c }
```

(End definition for `\hbox_set_to_wd:Nnn` and `\hbox_set_to_wd:cnn`. These functions are documented on page ??.)

`\hbox_set:Nw` Storing material in a horizontal box. This type is useful in environment definitions.

`\hbox_set:cw` 6682 `\cs_new_protected:Npn \hbox_set:Nw #1`

```
6683 { \tex_setbox:D #1 \tex_hbox:D \c_group_begin_token }
```

`\hbox_gset:Nw` 6684 `\cs_new_protected:Npn \hbox_gset:Nw`

```
6685 { \tex_global:D \hbox_set:Nw }
```

`\hbox_set_end:` 6686 `\cs_generate_variant:Nn \hbox_set:Nw { c }`

```
6687 \cs_generate_variant:Nn \hbox_gset:Nw { c }
```

```
6688 \cs_new_eq:NN \hbox_set_end: \c_group_end_token
```

```
6689 \cs_new_eq:NN \hbox_gset_end: \c_group_end_token
```

(End definition for `\hbox_set:Nw` and `\hbox_set:cw`. These functions are documented on page ??.)

`\hbox_set_inline_begin:N` Renamed September 2011.

`\hbox_set_inline_begin:c` 6690 `\cs_new_eq:NN \hbox_set_inline_begin:N \hbox_set:Nw`

`\hbox_gset_inline_begin:N` 6691 `\cs_new_eq:NN \hbox_set_inline_begin:c \hbox_set:cw`

`\hbox_gset_inline_begin:c` 6692 `\cs_new_eq:NN \hbox_set_inline_end: \hbox_set_end:`

`\hbox_set_inline_end:` 6693 `\cs_new_eq:NN \hbox_gset_inline_begin:N \hbox_gset:Nw`

`\hbox_gset_inline_end:` 6694 `\cs_new_eq:NN \hbox_gset_inline_begin:c \hbox_gset:cw`

```
6695 \cs_new_eq:NN \hbox_gset_inline_end: \hbox_gset_end:
```

(End definition for `\hbox_set_inline_begin:N` and `\hbox_set_inline_begin:c`. These functions are documented on page ??.)

`\hbox_to_wd:nn` Put a horizontal box directly into the input stream.

`\hbox_to_zero:n`

```

6696 \cs_new_protected:Npn \hbox_to_wd:nn #1#2
6697   { \tex_hbox:D to \dim_eval:w #1 \dim_eval_end: {#2} }
6698 \cs_new_protected:Npn \hbox_to_zero:n #1 { \tex_hbox:D to \c_zero_skip {#1} }
      (End definition for \hbox_to_wd:nn. This function is documented on page 127.)

```

`\hbox_overlap_left:n` Put a zero-sized box with the contents pushed against one side (which makes it stick out

`\hbox_overlap_right:n` on the other) directly into the input stream.

```

6699 \cs_new_protected:Npn \hbox_overlap_left:n #1
6700   { \hbox_to_zero:n { \tex_hss:D #1 } }
6701 \cs_new_protected:Npn \hbox_overlap_right:n #1
6702   { \hbox_to_zero:n { #1 \tex_hss:D } }
      (End definition for \hbox_overlap_left:n. This function is documented on page 127.)

```

`\hbox_unpack:N` Unpacking a box and if requested also clear it.

`\hbox_unpack:c`

`\hbox_unpack_clear:N`

`\hbox_unpack_clear:c`

```

6703 \cs_new_eq:NN \hbox_unpack:N \tex_unhcopy:D
6704 \cs_new_eq:NN \hbox_unpack_clear:N \tex_unhbox:D
6705 \cs_generate_variant:Nn \hbox_unpack:N { c }
6706 \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

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 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.

```

6707 \cs_new_protected:Npn \vbox:n #1 { \tex_vbox:D { #1 \par } }
6708 \cs_new_protected:Npn \vbox_top:n #1 { \tex_vtop:D { #1 \par } }
      (End definition for \vbox:n. This function is documented on page 128.)

```

`\vbox_to_ht:nn` Put a vertical box directly into the input stream.

`\vbox_to_zero:n`

`\vbox_to_ht:nn`

`\vbox_to_zero:n`

```

6709 \cs_new_protected:Npn \vbox_to_ht:nn #1#2
6710   { \tex_vbox:D to \dim_eval:w #1 \dim_eval_end: { #2 \par } }
6711 \cs_new_protected:Npn \vbox_to_zero:n #1
6712   { \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 128.)

```

`\vbox_set:Nn` Storing material in a vertical box with a natural height.

`\vbox_set:cn`

`\vbox_gset:Nn`

`\vbox_gset:cn`

```

6713 \cs_new_protected:Npn \vbox_set:Nn #1#2
6714   { \tex_setbox:D #1 \tex_vbox:D { #2 \par } }
6715 \cs_new_protected:Npn \vbox_gset:Nn { \tex_global:D \vbox_set:Nn }
6716 \cs_generate_variant:Nn \vbox_set:Nn { c }
6717 \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 of the first object in the box.

```

\vbox_set_top:cn
\vbox_gset_top:Nn
\vbox_gset_top:cn
6718 \cs_new_protected:Npn \vbox_set_top:Nn #1#2
6719 { \tex_setbox:D #1 \tex_vtop:D { #2 \par } }
6720 \cs_new_protected:Npn \vbox_gset_top:Nn
6721 { \tex_global:D \vbox_set_top:Nn }
6722 \cs_generate_variant:Nn \vbox_set_top:Nn { c }
6723 \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
\vbox_gset_to_ht:Nnn
\vbox_gset_to_ht:cnn
6724 \cs_new_protected:Npn \vbox_set_to_ht:Nnn #1#2#3
6725 { \tex_setbox:D #1 \tex_vbox:D to \dim_eval:w #2 \dim_eval_end: { #3 \par } }
6726 \cs_new_protected:Npn \vbox_gset_to_ht:Nnn
6727 { \tex_global:D \vbox_set_to_ht:Nnn }
6728 \cs_generate_variant:Nn \vbox_set_to_ht:Nnn { c }
6729 \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
\vbox_gset:Nw
\vbox_gset:cw
\vbox_set_end:
\vbox_gset_end:
6730 \cs_new_protected:Npn \vbox_set:Nw #1
6731 { \tex_setbox:D #1 \tex_vbox:D \c_group_begin_token }
6732 \cs_new_protected:Npn \vbox_gset:Nw
6733 { \tex_global:D \vbox_set:Nw }
6734 \cs_generate_variant:Nn \vbox_set:Nw { c }
6735 \cs_generate_variant:Nn \vbox_gset:Nw { c }
6736 \cs_new_protected:Npn \vbox_set_end:
6737 {
6738   \par
6739   \c_group_end_token
6740 }
6741 \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
\vbox_gset_inline_begin:N
\vbox_gset_inline_begin:c
\vbox_set_inline_end:
\vbox_gset_inline_end:
6742 \cs_new_eq:NN \vbox_set_inline_begin:N \vbox_set:Nw
6743 \cs_new_eq:NN \vbox_set_inline_begin:c \vbox_set:cw
6744 \cs_new_eq:NN \vbox_set_inline_end: \vbox_set_end:
6745 \cs_new_eq:NN \vbox_gset_inline_begin:N \vbox_gset:Nw
6746 \cs_new_eq:NN \vbox_gset_inline_begin:c \vbox_gset:cw
6747 \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` 6748 `\cs_new_eq:NN \vbox_unpack:N \tex_unvcopy:D`

`\vbox_unpack_clear:N` 6749 `\cs_new_eq:NN \vbox_unpack_clear:N \tex_unvbox:D`

`\vbox_unpack_clear:c` 6750 `\cs_generate_variant:Nn \vbox_unpack:N { c }`

6751 `\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.

6752 `\cs_new_protected:Npn \vbox_set_split_to_ht:NNn #1#2#3`

6753 `{ \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 129.)

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.

6754 `\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` 6755 `\fp_new:N \l_box_cos_fp`

6756 `\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` 6757 `\dim_new:N \l_box_top_dim`

`\l_box_left_dim` 6758 `\dim_new:N \l_box_bottom_dim`

`\l_box_right_dim` 6759 `\dim_new:N \l_box_left_dim`

6760 `\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` 6761 `\dim_new:N \l_box_top_new_dim`

`\l_box_left_new_dim` 6762 `\dim_new:N \l_box_bottom_new_dim`

`\l_box_right_new_dim` 6763 `\dim_new:N \l_box_left_new_dim`

6764 `\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_internal_box` Scratch space.

`\l_box_internal_fp` 6765 `\box_new:N \l_box_internal_box`

6766 `\fp_new:N \l_box_internal_fp`

(End definition for `\l_box_internal_box` and `\l_box_internal_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`
`\l_box_x_new_fp`
`\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
`\box_rotate_aux:N` check to avoid doing any real work for the trivial rotation.
`\box_rotate_set_sin_cos:`
`\box_rotate_x:nnN`
`\box_rotate_y:nnN`
`\box_rotate_quadrant_one:`
`\box_rotate_quadrant_two:`
`\box_rotate_quadrant_three:`
`\box_rotate_quadrant_four:`

```

6771 \cs_new_protected:Npn \box_rotate:Nn #1#2
6772 {
6773   \hbox_set:Nn #1
6774   {
6775     \group_begin:
6776     \fp_set:Nn \l_box_angle_fp {#2}
6777     \box_rotate_set_sin_cos:
6778     \fp_compare:NNTF \l_box_sin_fp = \c_zero_fp
6779     {
6780       \fp_compare:NNTF \l_box_cos_fp = \c_one_fp
6781       { \box_use:N #1 }
6782       { \box_rotate_aux:N #1 }
6783     }
6784     { \box_rotate_aux:N #1 }
6785   \group_end:
6786 }
6787 }

```

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.

```

6788 \cs_new_protected:Npn \box_rotate_aux:N #1
6789 {
6790   \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6791   \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6792   \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6793   \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 :

$$\begin{aligned}
 P'_x &= P_x - O_x \\
 P'_y &= P_y - O_y \\
 P''_x &= (P'_x \cos(\alpha)) - (P'_y \sin(\alpha)) \\
 P''_y &= (P'_x \sin(\alpha)) + (P'_y \cos(\alpha)) \\
 P'''_x &= P''_x + O_x + L_x \\
 P'''_y &= P''_y + O_y
 \end{aligned}$$

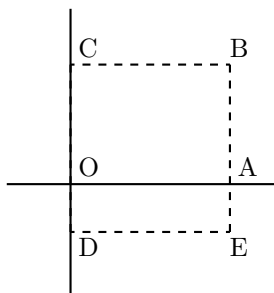


Figure 1: Co-ordinates of a box prior to rotation.

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 $\text{T}_{\text{E}}\text{X}$ 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.)

```

6794     \fp_compare:NNTF \l_box_sin_fp > \c_zero_fp
6795     {
6796         \fp_compare:NNTF \l_box_cos_fp > \c_zero_fp
6797         { \box_rotate_quadrant_one: }
6798         { \box_rotate_quadrant_two: }
6799     }
6800     {
6801         \fp_compare:NNTF \l_box_cos_fp < \c_zero_fp
6802         { \box_rotate_quadrant_three: }
6803         { \box_rotate_quadrant_four: }
6804     }

```

The position of the box edges are now known, but the box at this stage be misplaced relative to the current $\text{T}_{\text{E}}\text{X}$ 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.

```

6805     \hbox_set:Nn \l_box_internal_box { \box_use:N #1 }
6806     \hbox_set:Nn \l_box_internal_box
6807     {
6808         \tex_kern:D -\l_box_left_new_dim
6809         \hbox:n
6810         {
6811             \driver_box_rotate_begin:
6812             \box_use:N \l_box_internal_box
6813             \driver_box_rotate_end:
6814         }
6815     }

```

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.

```

6816     \box_set_ht:Nn \l_box_internal_box { \l_box_top_new_dim }
6817     \box_set_dp:Nn \l_box_internal_box { -\l_box_bottom_new_dim }
6818     \box_set_wd:Nn \l_box_internal_box

```

```

6819     { \l_box_right_new_dim - \l_box_left_new_dim }
6820     \box_use:N \l_box_internal_box
6821 }

```

A simple conversion from degrees to radians followed by calculation of the sine and cosine.

```

6822 \cs_new_protected:Npn \box_rotate_set_sin_cos:
6823 {
6824   \fp_set_eq:NN \l_box_internal_fp \l_box_angle_fp
6825   \fp_div:Nn \l_box_internal_fp { 180 }
6826   \fp_mul:Nn \l_box_internal_fp { \c_pi_fp }
6827   \fp_sin:Nn \l_box_sin_fp { \l_box_internal_fp }
6828   \fp_cos:Nn \l_box_cos_fp { \l_box_internal_fp }
6829 }

```

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.

```

6830 \cs_new_protected:Npn \box_rotate_x:nnN #1#2#3
6831 {
6832   \fp_set_from_dim:Nn \l_box_x_fp {#1}
6833   \fp_set_from_dim:Nn \l_box_y_fp {#2}
6834   \fp_set_eq:NN \l_box_x_new_fp \l_box_x_fp
6835   \fp_set_eq:NN \l_box_internal_fp \l_box_y_fp
6836   \fp_mul:Nn \l_box_x_new_fp { \l_box_cos_fp }
6837   \fp_mul:Nn \l_box_internal_fp { \l_box_sin_fp }
6838   \fp_sub:Nn \l_box_x_new_fp { \l_box_internal_fp }
6839   \dim_set:Nn #3 { \fp_to_dim:N \l_box_x_new_fp }
6840 }
6841 \cs_new_protected:Npn \box_rotate_y:nnN #1#2#3
6842 {
6843   \fp_set_from_dim:Nn \l_box_x_fp {#1}
6844   \fp_set_from_dim:Nn \l_box_y_fp {#2}
6845   \fp_set_eq:NN \l_box_y_new_fp \l_box_y_fp
6846   \fp_set_eq:NN \l_box_internal_fp \l_box_x_fp
6847   \fp_mul:Nn \l_box_y_new_fp { \l_box_cos_fp }
6848   \fp_mul:Nn \l_box_internal_fp { \l_box_sin_fp }
6849   \fp_add:Nn \l_box_y_new_fp { \l_box_internal_fp }
6850   \dim_set:Nn #3 { \fp_to_dim:N \l_box_y_new_fp }
6851 }

```

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.

```

6852 \cs_new_protected:Npn \box_rotate_quadrant_one:
6853 {

```

```

6854 \box_rotate_y:nnN \l_box_right_dim \l_box_top_dim
6855 \l_box_top_new_dim
6856 \box_rotate_y:nnN \l_box_left_dim \l_box_bottom_dim
6857 \l_box_bottom_new_dim
6858 \box_rotate_x:nnN \l_box_left_dim \l_box_top_dim
6859 \l_box_left_new_dim
6860 \box_rotate_x:nnN \l_box_right_dim \l_box_bottom_dim
6861 \l_box_right_new_dim
6862 }
6863 \cs_new_protected:Npn \box_rotate_quadrant_two:
6864 {
6865 \box_rotate_y:nnN \l_box_right_dim \l_box_bottom_dim
6866 \l_box_top_new_dim
6867 \box_rotate_y:nnN \l_box_left_dim \l_box_top_dim
6868 \l_box_bottom_new_dim
6869 \box_rotate_x:nnN \l_box_right_dim \l_box_top_dim
6870 \l_box_left_new_dim
6871 \box_rotate_x:nnN \l_box_left_dim \l_box_bottom_dim
6872 \l_box_right_new_dim
6873 }
6874 \cs_new_protected:Npn \box_rotate_quadrant_three:
6875 {
6876 \box_rotate_y:nnN \l_box_left_dim \l_box_bottom_dim
6877 \l_box_top_new_dim
6878 \box_rotate_y:nnN \l_box_right_dim \l_box_top_dim
6879 \l_box_bottom_new_dim
6880 \box_rotate_x:nnN \l_box_right_dim \l_box_bottom_dim
6881 \l_box_left_new_dim
6882 \box_rotate_x:nnN \l_box_left_dim \l_box_top_dim
6883 \l_box_right_new_dim
6884 }
6885 \cs_new_protected:Npn \box_rotate_quadrant_four:
6886 {
6887 \box_rotate_y:nnN \l_box_left_dim \l_box_top_dim
6888 \l_box_top_new_dim
6889 \box_rotate_y:nnN \l_box_right_dim \l_box_bottom_dim
6890 \l_box_bottom_new_dim
6891 \box_rotate_x:nnN \l_box_left_dim \l_box_bottom_dim
6892 \l_box_left_new_dim
6893 \box_rotate_x:nnN \l_box_right_dim \l_box_top_dim
6894 \l_box_right_new_dim
6895 }

```

(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

```

6896 \fp_new:N \l_box_scale_x_fp
6897 \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`
`\box_resize_aux:Nnn`

```

6898 \cs_new_protected:Npn \box_resize:Nnn #1#2#3
6899 {
6900   \hbox_set:Nn #1
6901   {
6902     \group_begin:
6903     \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6904     \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6905     \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6906     \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.

```

6907     \fp_set_from_dim:Nn \l_box_scale_x_fp {#2}
6908     \fp_set_from_dim:Nn \l_box_internal_fp { \l_box_right_dim }
6909     \fp_div:Nn \l_box_scale_x_fp { \l_box_internal_fp }

```

The y -scaling needs both the height and the depth of the current box.

```

6910     \fp_set_from_dim:Nn \l_box_scale_y_fp {#3}
6911     \fp_set_from_dim:Nn \l_box_internal_fp
6912     { \l_box_top_dim - \l_box_bottom_dim }
6913     \fp_div:Nn \l_box_scale_y_fp { \l_box_internal_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.

```

6914     \fp_compare:NNNTF \l_box_scale_x_fp = \c_one_fp
6915     {
6916       \fp_compare:NNNTF \l_box_scale_y_fp = \c_one_fp
6917       { \box_use:N #1 }
6918       { \box_resize_aux:Nnn #1 {#2} {#3} }
6919     }
6920     { \box_resize_aux:Nnn #1 {#2} {#3} }
6921   \group_end:
6922 }
6923 }
6924 \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.

```

6925 \cs_new_protected:Npn \box_resize_aux:Nnn #1#2#3
6926 {
6927   \dim_compare:nNnTF {#2} > \c_zero_dim
6928   { \dim_set:Nn \l_box_right_new_dim {#2} }
6929   { \dim_set:Nn \l_box_right_new_dim { \c_zero_dim - ( #2 ) } }
6930   \dim_compare:nNnTF {#3} > \c_zero_dim
6931   {
6932     \dim_set:Nn \l_box_top_new_dim

```

```

6933         { \fp_use:N \l_box_scale_y_fp \l_box_top_dim }
6934     \dim_set:Nn \l_box_bottom_new_dim
6935         { \fp_use:N \l_box_scale_y_fp \l_box_bottom_dim }
6936     }
6937     {
6938         \dim_set:Nn \l_box_top_new_dim
6939         { - \fp_use:N \l_box_scale_y_fp \l_box_top_dim }
6940         \dim_set:Nn \l_box_bottom_new_dim
6941         { - \fp_use:N \l_box_scale_y_fp \l_box_bottom_dim }
6942     }
6943     \box_resize_common:N #1
6944 }

```

(End definition for `\box_resize:Nnn` and `\box_resize:cnn`. These functions are documented on page ??.)

`\box_resize_to_ht_plus_dp:Nn` 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 `\box_resize_to_wd:Nn` 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).

```

6945 \cs_new_protected:Npn \box_resize_to_ht_plus_dp:Nn #1#2
6946 {
6947     \hbox_set:Nn #1
6948     {
6949         \group_begin:
6950             \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6951             \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6952             \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6953             \dim_zero:N \l_box_left_dim
6954             \fp_set_from_dim:Nn \l_box_scale_y_fp {#2}
6955             \fp_set_from_dim:Nn \l_box_internal_fp
6956                 { \l_box_top_dim - \l_box_bottom_dim }
6957             \fp_div:Nn \l_box_scale_y_fp { \l_box_internal_fp }
6958             \fp_set_eq:NN \l_box_scale_x_fp \l_box_scale_y_fp
6959             \fp_compare:NNNTF \l_box_scale_y_fp = \c_one_fp
6960                 { \box_use:N #1 }
6961                 { \box_resize_aux:Nnn #1 {#2} {#2} }
6962             \group_end:
6963         }
6964     }
6965     \cs_generate_variant:Nn \box_resize_to_ht_plus_dp:Nn { c }
6966     \cs_new_protected:Npn \box_resize_to_wd:Nn #1#2
6967     {
6968         \hbox_set:Nn #1
6969         {
6970             \group_begin:
6971                 \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6972                 \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6973                 \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6974                 \dim_zero:N \l_box_left_dim

```

```

6975         \fp_set_from_dim:Nn \l_box_scale_x_fp {#2}
6976         \fp_set_from_dim:Nn \l_box_internal_fp { \l_box_right_dim }
6977         \fp_div:Nn \l_box_scale_x_fp { \l_box_internal_fp }
6978         \fp_set_eq:NN \l_box_scale_y_fp \l_box_scale_x_fp
6979         \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6980         { \box_use:N #1 }
6981         { \box_resize_aux:Nnn #1 {#2} {#2} }
6982     \group_end:
6983 }
6984 }
6985 \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 also relatively easy to find, allowing only for the need to keep them positive in all cases.

`\box_scale:cnn` 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.

`\box_scale_aux:Nnn`

```

6986 \cs_new_protected:Npn \box_scale:Nnn #1#2#3
6987 {
6988     \hbox_set:Nn #1
6989     {
6990         \group_begin:
6991         \fp_set:Nn \l_box_scale_x_fp {#2}
6992         \fp_set:Nn \l_box_scale_y_fp {#3}
6993         \dim_set:Nn \l_box_top_dim { \box_ht:N #1 }
6994         \dim_set:Nn \l_box_bottom_dim { -\box_dp:N #1 }
6995         \dim_set:Nn \l_box_right_dim { \box_wd:N #1 }
6996         \dim_zero:N \l_box_left_dim
6997         \fp_compare:NNTF \l_box_scale_x_fp = \c_one_fp
6998         {
6999             \fp_compare:NNTF \l_box_scale_y_fp = \c_one_fp
7000             { \box_use:N #1 }
7001             { \box_scale_aux:Nnn #1 {#2} {#3} }
7002         }
7003         { \box_scale_aux:Nnn #1 {#2} {#3} }
7004     \group_end:
7005 }
7006 }
7007 \cs_generate_variant:Nn \box_scale:Nnn { c }
7008 \cs_new_protected:Npn \box_scale_aux:Nnn #1#2#3
7009 {
7010     \fp_compare:NNTF \l_box_scale_y_fp > \c_zero_fp
7011     {
7012         \dim_set:Nn \l_box_top_new_dim { #3 \l_box_top_dim }
7013         \dim_set:Nn \l_box_bottom_new_dim { #3 \l_box_bottom_dim }
7014     }
7015     {

```



```

7016         \dim_set:Nn \l_box_top_new_dim { -#3 \l_box_bottom_dim }
7017         \dim_set:Nn \l_box_bottom_new_dim { -#3 \l_box_top_dim }
7018     }
7019     \fp_compare:NNTF \l_box_scale_x_fp > \c_zero_fp
7020     { \l_box_right_new_dim #2 \l_box_right_dim }
7021     { \l_box_right_new_dim -#2 \l_box_right_dim }
7022     \box_resize_common:N #1
7023 }

```

(End definition for `\box_scale:Nnn` and `\box_scale:cnn`. These functions are documented on page ??.)

`\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.

```

7024 \cs_new_protected:Npn \box_resize_common:N #1
7025 {
7026     \hbox_set:Nn \l_box_internal_box
7027     {
7028         \driver_box_scale_begin:
7029         \hbox_overlap_right:n { \box_use:N #1 }
7030         \driver_box_scale_end:
7031     }

```

The new height and depth can be applied directly.

```

7032     \box_set_ht:Nn \l_box_internal_box { \l_box_top_new_dim }
7033     \box_set_dp:Nn \l_box_internal_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.

```

7034     \fp_compare:NNTF \l_box_scale_x_fp < \c_zero_fp
7035     {
7036         \hbox_to_wd:nn { \l_box_right_new_dim }
7037         {
7038             \tex_kern:D \l_box_right_new_dim
7039             \box_use:N \l_box_internal_box
7040             \tex_hss:D
7041         }
7042     }
7043     {
7044         \box_set_wd:Nn \l_box_internal_box { \l_box_right_new_dim }
7045         \box_use:N \l_box_internal_box
7046     }
7047 }

```

(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`

```

7048 \cs_new_protected:Npn \box_clip:N #1
7049 { \hbox_set:Nn #1 { \driver_box_use_clip:N #1 } }
7050 \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.

```

7051 \cs_new_protected:Npn \box_trim:Nnnnn #1#2#3#4#5
7052 {
7053   \box_set_wd:Nn #1 { \box_wd:N #1 - \dim_eval:n {#4} - \dim_eval:n {#2} }
7054   \hbox_set:Nn #1
7055   {
7056     \skip_horizontal:n { - \dim_eval:n {#2} }
7057     \box_use:N #1
7058   }

```

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.

```

7059   \dim_compare:nNnTF { \box_dp:N #1 } > {#3}
7060   { \box_set_dp:Nn #1 { \box_dp:N #1 - \dim_eval:n {#3} } }
7061   {
7062     \hbox_set:Nn #1
7063     {
7064       \box_move_down:nn { \dim_eval:n {#3} - \box_dp:N #1 }
7065       { \box_use:N #1 }
7066     }
7067     \box_set_dp:Nn #1 \c_zero_dim
7068   }
7069   \dim_compare:nNnTF { \box_ht:N #1 } > {#5}
7070   { \box_set_ht:Nn #1 { \box_ht:N #1 - \dim_eval:n {#5} } }
7071   {
7072     \hbox_set:Nn #1
7073     {
7074       \box_move_up:nn { \dim_eval:n {#5} - \box_ht:N #1 }
7075       { \box_use:N #1 }
7076     }
7077     \box_set_ht:Nn #1 \c_zero_dim
7078   }
7079 }
7080 \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.

```

7081 \cs_new_protected:Npn \box_viewport:Nnnnn #1#2#3#4#5
7082 {
7083   \box_set_wd:Nn #1 { \dim_eval:n {#4} - \dim_eval:n {#2} }

```

```

7084 \hbox_set:Nn #1
7085 {
7086   \skip_horizontal:n { - \dim_eval:n {#2} }
7087   \box_use:N #1
7088 }
7089 \dim_compare:nNnTF {#3} > \c_zero_dim
7090 {
7091   \hbox_set:Nn #1 { \box_move_down:nn {#3} { \box_use:N #1 } }
7092   \box_set_dp:Nn #1 \c_zero_dim
7093 }
7094 { \box_set_dp:Nn #1 { - \dim_eval:n {#3} } }
7095 \dim_compare:nNnTF {#5} > \c_zero_dim
7096 { \box_set_ht:Nn #1 {#5} }
7097 {
7098   \hbox_set:Nn #1
7099     { \box_move_up:nn { -\dim_eval:n {#5} } { \box_use:N #1 } }
7100   \box_set_ht:Nn #1 \c_zero_dim
7101 }
7102 }
7103 \cs_generate_variant:Nn \box_viewport:Nnnnn { c }

```

(End definition for \box_viewport:Nnnnn and \box_viewport:cnnnn. These functions are documented on page ??.)

196.13 Deprecated functions

\l_last_box Deprecated 2011-11-13, for removal by 2012-02-28.

```

7104 \cs_new_eq:NN \l_last_box \tex_lastbox:D
       (End definition for \l_last_box. This function is documented on page ??.)
7105 </initex | package>

```

197 l3coffins Implementation

```

7106 <*initex | package>
7107 <*package>
7108 \ProvidesExplPackage
7109   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
7110 \package_check_loaded_expl:
7111 </package>

```

197.1 Coffins: data structures and general variables

\l_coffin_internal_box Scratch variables.

```

\l_coffin_internal_dim 7112 \box_new:N \l_coffin_internal_box
\l_coffin_internal_fp   7113 \dim_new:N \l_coffin_internal_dim
\l_coffin_internal_tl   7114 \fp_new:N \l_coffin_internal_fp
                        7115 \tl_new:N \l_coffin_internal_tl
       (End definition for \l_coffin_internal_box. This function is documented on page ??.)

```

<code>\c_coffin_corners_prop</code>	<p>The “corners”; of a coffin define the real content, as opposed to the \TeX bounding box. They all start off in the same place, of course.</p> <pre> 7116 \prop_new:N \c_coffin_corners_prop 7117 \prop_put:Nnn \c_coffin_corners_prop { tl } { { 0 pt } { 0 pt } } 7118 \prop_put:Nnn \c_coffin_corners_prop { tr } { { 0 pt } { 0 pt } } 7119 \prop_put:Nnn \c_coffin_corners_prop { bl } { { 0 pt } { 0 pt } } 7120 \prop_put:Nnn \c_coffin_corners_prop { br } { { 0 pt } { 0 pt } } </pre> <p>(End definition for <code>\c_coffin_corners_prop</code>. This function is documented on page ??.)</p>
<code>\c_coffin_poles_prop</code>	<p>Pole positions are given for horizontal, vertical and reference-point based values.</p> <pre> 7121 \prop_new:N \c_coffin_poles_prop 7122 \tl_set:Nn \l_coffin_internal_tl { { 0 pt } { 0 pt } { 0 pt } { 1000 pt } } 7123 \prop_put:Nno \c_coffin_poles_prop { l } { \l_coffin_internal_tl } 7124 \prop_put:Nno \c_coffin_poles_prop { hc } { \l_coffin_internal_tl } 7125 \prop_put:Nno \c_coffin_poles_prop { r } { \l_coffin_internal_tl } 7126 \tl_set:Nn \l_coffin_internal_tl { { 0 pt } { 0 pt } { 1000 pt } { 0 pt } } 7127 \prop_put:Nno \c_coffin_poles_prop { b } { \l_coffin_internal_tl } 7128 \prop_put:Nno \c_coffin_poles_prop { vc } { \l_coffin_internal_tl } 7129 \prop_put:Nno \c_coffin_poles_prop { t } { \l_coffin_internal_tl } 7130 \prop_put:Nno \c_coffin_poles_prop { B } { \l_coffin_internal_tl } 7131 \prop_put:Nno \c_coffin_poles_prop { H } { \l_coffin_internal_tl } 7132 \prop_put:Nno \c_coffin_poles_prop { T } { \l_coffin_internal_tl } </pre> <p>(End definition for <code>\c_coffin_poles_prop</code>. This function is documented on page ??.)</p>
<code>\l_coffin_calc_a_fp</code> <code>\l_coffin_calc_b_fp</code> <code>\l_coffin_calc_c_fp</code> <code>\l_coffin_calc_d_fp</code> <code>\l_coffin_calc_result_fp</code>	<p>Used for calculations of intersections and in other internal places.</p> <pre> 7133 \fp_new:N \l_coffin_calc_a_fp 7134 \fp_new:N \l_coffin_calc_b_fp 7135 \fp_new:N \l_coffin_calc_c_fp 7136 \fp_new:N \l_coffin_calc_d_fp 7137 \fp_new:N \l_coffin_calc_result_fp </pre> <p>(End definition for <code>\l_coffin_calc_a_fp</code>. This function is documented on page ??.)</p>
<code>\l_coffin_error_bool</code>	<p>For propagating errors so that parts of the code can work around them.</p> <pre> 7138 \bool_new:N \l_coffin_error_bool </pre> <p>(End definition for <code>\l_coffin_error_bool</code>. This function is documented on page ??.)</p>
<code>\l_coffin_offset_x_dim</code> <code>\l_coffin_offset_y_dim</code>	<p>The offset between two sets of coffin handles when typesetting. These values are corrected from those requested in an alignment for the positions of the handles.</p> <pre> 7139 \dim_new:N \l_coffin_offset_x_dim 7140 \dim_new:N \l_coffin_offset_y_dim </pre> <p>(End definition for <code>\l_coffin_offset_x_dim</code>. This function is documented on page ??.)</p>
<code>\l_coffin_pole_a_tl</code> <code>\l_coffin_pole_b_tl</code>	<p>Needed for finding the intersection of two poles.</p> <pre> 7141 \tl_new:N \l_coffin_pole_a_tl 7142 \tl_new:N \l_coffin_pole_b_tl </pre> <p>(End definition for <code>\l_coffin_pole_a_tl</code>. This function is documented on page ??.)</p>

<code>\l_coffin_sin_fp</code> <code>\l_coffin_cos_fp</code>	<p>Used for rotations to get the sine and cosine values.</p> <pre> 7143 \fp_new:N \l_coffin_sin_fp 7144 \fp_new:N \l_coffin_cos_fp (End definition for \l_coffin_sin_fp. This function is documented on page ??.) </pre>
<code>\l_coffin_x_dim</code> <code>\l_coffin_y_dim</code> <code>\l_coffin_x_prime_dim</code> <code>\l_coffin_y_prime_dim</code>	<p>For calculating intersections and so forth.</p> <pre> 7145 \dim_new:N \l_coffin_x_dim 7146 \dim_new:N \l_coffin_y_dim 7147 \dim_new:N \l_coffin_x_prime_dim 7148 \dim_new:N \l_coffin_y_prime_dim (End definition for \l_coffin_x_dim. This function is documented on page ??.) </pre>
<code>\l_coffin_x_fp</code> <code>\l_coffin_y_fp</code> <code>\l_coffin_x_prime_fp</code> <code>\l_coffin_y_prime_fp</code>	<p>Used for calculations where there are clear <i>x</i>- and <i>y</i>-components, for example during vector rotation.</p> <pre> 7149 \fp_new:N \l_coffin_x_fp 7150 \fp_new:N \l_coffin_y_fp 7151 \fp_new:N \l_coffin_x_prime_fp 7152 \fp_new:N \l_coffin_y_prime_fp (End definition for \l_coffin_x_fp. This function is documented on page ??.) </pre>
<code>\l_coffin_Depth_dim</code> <code>\l_coffin_Height_dim</code> <code>\l_coffin_TotalHeight_dim</code> <code>\l_coffin_Width_dim</code>	<p>Dimensions for the various parts of a coffin.</p> <pre> 7153 \dim_new:N \l_coffin_Depth_dim 7154 \dim_new:N \l_coffin_Height_dim 7155 \dim_new:N \l_coffin_TotalHeight_dim 7156 \dim_new:N \l_coffin_Width_dim (End definition for \l_coffin_Depth_dim. This function is documented on page ??.) </pre>
<code>\coffin_saved_Depth:</code> <code>\coffin_saved_Height:</code> <code>\coffin_saved_TotalHeight:</code> <code>\coffin_saved_Width:</code>	<p>Used to save the meaning of <code>\Depth</code>, <code>\Height</code>, <code>\TotalHeight</code> and <code>\Width</code>.</p> <pre> 7157 \cs_new_nopar:Npn \coffin_saved_Depth: { } 7158 \cs_new_nopar:Npn \coffin_saved_Height: { } 7159 \cs_new_nopar:Npn \coffin_saved_TotalHeight: { } 7160 \cs_new_nopar:Npn \coffin_saved_Width: { } (End definition for \coffin_saved_Depth:. This function is documented on page ??.) </pre>

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.

```

7161 \cs_new_protected:Npn \coffin_if_exist:NT #1#2
7162 {
7163   \cs_if_exist:NTF #1
7164   {
7165     \cs_if_exist:CTF { l_coffin_poles_ \int_value:w #1 _prop }
7166     { #2 }

```

```

7167         {
7168             \msg_kernel_error:nnx { coffins } { unknown-coffin }
7169             { \token_to_str:N #1 }
7170         }
7171     }
7172     {
7173         \msg_kernel_error:nnx { coffins } { unknown-coffin }
7174         { \token_to_str:N #1 }
7175     }
7176 }

```

(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
7177 \cs_new_protected:Npn \coffin_clear:N #1
7178 {
7179     \coffin_if_exist:NT #1
7180     {
7181         \box_clear:N #1
7182         \coffin_reset_structure:N #1
7183     }
7184 }
7185 \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.

```

7186 \cs_new_protected:Npn \coffin_new:N #1
7187 {
7188     \box_new:N #1
7189     \prop_clear_new:c { l_coffin_corners_ \int_value:w #1 _prop }
7190     \prop_clear_new:c { l_coffin_poles_ \int_value:w #1 _prop }
7191     \prop_gset_eq:cN { l_coffin_corners_ \int_value:w #1 _prop }
7192     \c_coffin_corners_prop
7193     \prop_gset_eq:cN { l_coffin_poles_ \int_value:w #1 _prop }
7194     \c_coffin_poles_prop
7195 }
7196 \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 update the handle positions.

```

7197 \cs_new_protected:Npn \hcoffin_set:Nn #1#2
7198 {
7199     \coffin_if_exist:NT #1
7200     {

```

```

7201     \hbox_set:Nn #1
7202     {
7203         \color_group_begin:
7204         \color_ensure_current:
7205         #2
7206         \color_group_end:
7207     }
7208     \coffin_reset_structure:N #1
7209     \coffin_update_poles:N #1
7210     \coffin_update_corners:N #1
7211 }
7212 }
7213 \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:cn The default handles and poles are set as for a horizontal coffin, before finding the top baseline using a temporary box.

```

7214 \cs_new_protected:Npn \vcoffin_set:Nnn #1#2#3
7215 {
7216     \coffin_if_exist:NT #1
7217     {
7218         \vbox_set:Nn #1
7219         {
7220             \dim_set:Nn \tex_hsize:D {#2}
7221             \color_group_begin:
7222             \color_ensure_current:
7223             #3
7224             \color_group_end:
7225         }
7226         \coffin_reset_structure:N #1
7227         \coffin_update_poles:N #1
7228         \coffin_update_corners:N #1
7229         \vbox_set_top:Nn \l_coffin_internal_box { \vbox_unpack:N #1 }
7230         \coffin_set_pole:Nnx #1 { T }
7231         {
7232             { 0 pt }
7233             { \dim_eval:n { \box_ht:N #1 - \box_ht:N \l_coffin_internal_box } }
7234             { 1000 pt }
7235             { 0 pt }
7236         }
7237         \box_clear:N \l_coffin_internal_box
7238     }
7239 }
7240 \cs_generate_variant:Nn \vcoffin_set:Nnn { c }

```

(End definition for \vcoffin_set:Nnn and \vcoffin_set:cn. These functions are documented on page ??.)

`\hcoffin_set:Nw` These are the “begin”/“end” versions of the above: watch the grouping!

```

\hcoffin_set:cnw 7241 \cs_new_protected:Npn \hcoffin_set:Nw #1
\hcoffin_set_end: 7242 {
                  7243   \coffin_if_exist:NT #1
                  7244   {
                  7245     \hbox_set:Nw #1 \color_group_begin: \color_ensure_current:
                  7246     \cs_set_protected_nopar:Npn \hcoffin_set_end:
                  7247     {
                  7248       \color_group_end:
                  7249       \hbox_set_end:
                  7250       \coffin_reset_structure:N #1
                  7251       \coffin_update_poles:N #1
                  7252       \coffin_update_corners:N #1
                  7253     }
                  7254   }
                  7255 }
7256 \cs_new_protected_nopar:Npn \hcoffin_set_end: { }
7257 \cs_generate_variant:Nn \hcoffin_set:Nw { c }

```

(End definition for `\hcoffin_set:Nw` and `\hcoffin_set:cnw`. These functions are documented on page ??.)

`\vcoffin_set:Nnw` The same for vertical coffins.

```

\vcoffin_set:cnw 7258 \cs_new_protected:Npn \vcoffin_set:Nnw #1#2
\vcoffin_set_end: 7259 {
                  7260   \coffin_if_exist:NT #1
                  7261   {
                  7262     \vbox_set:Nw #1
                  7263     \dim_set:Nn \tex_hsize:D {#2}
                  7264     \color_group_begin: \color_ensure_current:
                  7265     \cs_set_protected:Npn \vcoffin_set_end:
                  7266     {
                  7267       \color_group_end:
                  7268       \vbox_set_end:
                  7269       \coffin_reset_structure:N #1
                  7270       \coffin_update_poles:N #1
                  7271       \coffin_update_corners:N #1
                  7272       \vbox_set_top:Nn \l_coffin_internal_box { \vbox_unpack:N #1 }
                  7273       \coffin_set_pole:Nnx #1 { T }
                  7274       {
                  7275         { 0 pt }
                  7276         {
                  7277           \dim_eval:n { \box_ht:N #1 - \box_ht:N \l_coffin_internal_box }
                  7278         }
                  7279         { 1000 pt }
                  7280         { 0 pt }
                  7281       }
                  7282       \box_clear:N \l_coffin_internal_box
                  7283     }
                  7284   }

```



```

7285     }
7286     \cs_new_protected_nopar:Npn \vcoffin_set_end: { }
7287     \cs_generate_variant:Nn \vcoffin_set:Nnw { c }

```

(End definition for \vcoffin_set:Nnw and \vcoffin_set:cnw. 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 7288 \cs_new_protected:Npn \coffin_set_eq:NN #1#2
\coffin_set_eq:cN 7289 {
\coffin_set_eq:cc 7290     \coffin_if_exist:NT #1
7291     {
7292         \box_set_eq:NN #1 #2
7293         \coffin_set_eq_structure:NN #1 #2
7294     }
7295 }
7296 \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 7297 \coffin_new:N \c_empty_coffin
\l_coffin_aligned_internal_coffin 7298 \hbox_set:Nn \c_empty_coffin { }
7299 \coffin_new:N \l_coffin_aligned_coffin
7300 \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 7301 \cs_new_eq:NN \coffin_dp:N \box_dp:N
\coffin_ht:N 7302 \cs_new_eq:NN \coffin_dp:c \box_dp:c
\coffin_ht:c 7303 \cs_new_eq:NN \coffin_ht:N \box_ht:N
\coffin_wd:N 7304 \cs_new_eq:NN \coffin_ht:c \box_ht:c
\coffin_wd:c 7305 \cs_new_eq:NN \coffin_wd:N \box_wd:N
7306 \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.

```

7307 \cs_new_protected:Npn \coffin_get_pole:NnN #1#2#3
7308 {
7309     \prop_get:cnNF
7310     { l_coffin_poles_ \int_value:w #1 _prop } {#2} #3
7311     {

```

```

7312         \msg_kernel_error:nxxx { coffins } { unknown-coffin-pole }
7313         {#2} { \token_to_str:N #1 }
7314         \tl_set:Nn #3 { { 0 pt } { 0 pt } { 0 pt } { 0 pt } }
7315     }
7316 }

```

(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.

```

7317 \cs_new_protected:Npn \coffin_reset_structure:N #1
7318 {
7319     \prop_set_eq:cN { l_coffin_corners_ \int_value:w #1 _prop }
7320     \c_coffin_corners_prop
7321     \prop_set_eq:cN { l_coffin_poles_ \int_value:w #1 _prop }
7322     \c_coffin_poles_prop
7323 }

```

(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

```

7324 \cs_new_protected:Npn \coffin_set_eq_structure:NN #1#2
7325 {
7326     \prop_set_eq:cc { l_coffin_corners_ \int_value:w #1 _prop }
7327     { l_coffin_corners_ \int_value:w #2 _prop }
7328     \prop_set_eq:cc { l_coffin_poles_ \int_value:w #1 _prop }
7329     { l_coffin_poles_ \int_value:w #2 _prop }
7330 }
7331 \cs_new_protected:Npn \coffin_gset_eq_structure:NN #1#2
7332 {
7333     \prop_gset_eq:cc { l_coffin_corners_ \int_value:w #1 _prop }
7334     { l_coffin_corners_ \int_value:w #2 _prop }
7335     \prop_gset_eq:cc { l_coffin_poles_ \int_value:w #1 _prop }
7336     { l_coffin_poles_ \int_value:w #2 _prop }
7337 }

```

(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

```

7338 \cs_new_protected:Npn \coffin_set_user_dimensions:N #1
7339 {
7340     \cs_set_eq:NN \coffin_saved_Height: \Height
7341     \cs_set_eq:NN \coffin_saved_Depth: \Depth
7342     \cs_set_eq:NN \coffin_saved_TotalHeight: \TotalHeight
7343     \cs_set_eq:NN \coffin_saved_Width: \Width
7344     \cs_set_eq:NN \Height \l_coffin_Height_dim
7345     \cs_set_eq:NN \Depth \l_coffin_Depth_dim
7346     \cs_set_eq:NN \TotalHeight \l_coffin_TotalHeight_dim
7347     \cs_set_eq:NN \Width \l_coffin_Width_dim
7348     \dim_set:Nn \Height { \box_ht:N #1 }
7349     \dim_set:Nn \Depth { \box_dp:N #1 }
7350     \dim_set:Nn \TotalHeight { \box_ht:N #1 + \box_dp:N #1 }

```

```

7351     \dim_set:Nn \Width      { \box_wd:N #1 }
7352   }
7353   \cs_new_protected_nopar:Npn \coffin_end_user_dimensions:
7354   {
7355     \cs_set_eq:NN \Height     \coffin_saved_Height:
7356     \cs_set_eq:NN \Depth      \coffin_saved_Depth:
7357     \cs_set_eq:NN \TotalHeight \coffin_saved_TotalHeight:
7358     \cs_set_eq:NN \Width      \coffin_saved_Width:
7359   }

```

(End definition for \coffin_set_user_dimensions:N. This function is documented on page ??.)

\coffin_set_horizontal_pole:Nnn 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.

```

\coffin_set_horizontal_pole:cnn
\coffin_set_vertical_pole:Nnn
\coffin_set_vertical_pole:cnn
\coffin_set_pole:Nnn
\coffin_set_pole:Nnx
7360   \cs_new_protected:Npn \coffin_set_horizontal_pole:Nnn #1#2#3
7361   {
7362     \coffin_if_exist:NT #1
7363     {
7364       \coffin_set_user_dimensions:N #1
7365       \coffin_set_pole:Nnx #1 {#2}
7366       {
7367         { 0 pt } { \dim_eval:n {#3} }
7368         { 1000 pt } { 0 pt }
7369       }
7370       \coffin_end_user_dimensions:
7371     }
7372   }
7373   \cs_new_protected:Npn \coffin_set_vertical_pole:Nnn #1#2#3
7374   {
7375     \coffin_if_exist:NT #1
7376     {
7377       \coffin_set_user_dimensions:N #1
7378       \coffin_set_pole:Nnx #1 {#2}
7379       {
7380         { \dim_eval:n {#3} } { 0 pt }
7381         { 0 pt } { 1000 pt }
7382       }
7383       \coffin_end_user_dimensions:
7384     }
7385   }
7386   \cs_new_protected:Npn \coffin_set_pole:Nnn #1#2#3
7387   { \prop_put:cnn { l_coffin_poles_ \int_value:w #1 _prop } {#2} {#3} }
7388   \cs_generate_variant:Nn \coffin_set_horizontal_pole:Nnn { c }
7389   \cs_generate_variant:Nn \coffin_set_vertical_pole:Nnn { c }
7390   \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.

```

7391 \cs_new_protected:Npn \coffin_update_corners:N #1
7392 {
7393   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { tl }
7394   { { 0 pt } { \dim_use:N \box_ht:N #1 } }
7395   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { tr }
7396   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
7397   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { bl }
7398   { { 0 pt } { \dim_eval:n { - \box_dp:N #1 } } }
7399   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } { br }
7400   { { \dim_use:N \box_wd:N #1 } { \dim_eval:n { - \box_dp:N #1 } } }
7401 }

```

(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.

```

7402 \cs_new_protected:Npn \coffin_update_poles:N #1
7403 {
7404   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { hc }
7405   {
7406     { \dim_eval:n { 0.5 \box_wd:N #1 } }
7407     { 0 pt } { 0 pt } { 1000 pt }
7408   }
7409   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { r }
7410   {
7411     { \dim_use:N \box_wd:N #1 }
7412     { 0 pt } { 0 pt } { 1000 pt }
7413   }
7414   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { vc }
7415   {
7416     { 0 pt }
7417     { \dim_eval:n { ( \box_ht:N #1 - \box_dp:N #1 ) / 2 } }
7418     { 1000 pt }
7419     { 0 pt }
7420   }
7421   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { t }
7422   {
7423     { 0 pt }
7424     { \dim_use:N \box_ht:N #1 }
7425     { 1000 pt }
7426     { 0 pt }
7427   }
7428   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } { b }
7429   {
7430     { 0 pt }
7431     { \dim_eval:n { - \box_dp:N #1 } }
7432     { 1000 pt }
7433     { 0 pt }

```

```

7434     }
7435 }
(End definition for \coffin_update_poles:N. This function is documented on page ??.)

```

197.5 Coffins: calculation of pole intersections

\coffin_calculate_intersection:Nnn 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.

```

7436 \cs_new_protected:Npn \coffin_calculate_intersection:Nnn #1#2#3
7437 {
7438   \coffin_get_pole:NnN #1 {#2} \l_coffin_pole_a_tl
7439   \coffin_get_pole:NnN #1 {#3} \l_coffin_pole_b_tl
7440   \bool_set_false:N \l_coffin_error_bool
7441   \exp_last_two_unbraced:Noo
7442     \coffin_calculate_intersection:nnnnnnnn
7443     \l_coffin_pole_a_tl \l_coffin_pole_b_tl
7444   \bool_if:NT \l_coffin_error_bool
7445   {
7446     \msg_kernel_error:nn { coffins } { no-pole-intersection }
7447     \dim_zero:N \l_coffin_x_dim
7448     \dim_zero:N \l_coffin_y_dim
7449   }
7450 }

```

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.

```

7451 \cs_new_protected:Npn \coffin_calculate_intersection:nnnnnnnn
7452   #1#2#3#4#5#6#7#8
7453 {
7454   \dim_compare:nNnTF {#3} = { \c_zero_dim }

```

The case where the first pole is vertical. So the x -component of the intersection will be at a . There is then a test on the second pole: if it is also vertical then there is an error.

```

7455   {
7456     \dim_set:Nn \l_coffin_x_dim {#1}
7457     \dim_compare:nNnTF {#7} = { \c_zero_dim
7458       { \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.

```

7459         {
7460             \dim_compare:nNnTF {#8} = \c_zero_dim
7461             { \dim_set:Nn \l_coffin_y_dim {#6} }
7462             {
7463                 \coffin_calculate_intersection_aux:nnnnnN
7464                 {#1} {#5} {#6} {#7} {#8} \l_coffin_y_dim
7465             }
7466         }
7467     }

```

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.

```

7468     {
7469         \dim_compare:nNnTF {#4} = \c_zero_dim
7470         {
7471             \dim_set:Nn \l_coffin_y_dim {#2}
7472             \dim_compare:nNnTF {#8} = { \c_zero_dim }
7473             { \bool_set_true:N \l_coffin_error_bool }
7474             {
7475                 \dim_compare:nNnTF {#7} = \c_zero_dim
7476                 { \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.

```

7477         {
7478             \coffin_calculate_intersection_aux:nnnnnN
7479             {#2} {#6} {#5} {#8} {#7} \l_coffin_x_dim
7480         }
7481     }
7482 }

```

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.

```

7483     {
7484         \dim_compare:nNnTF {#7} = \c_zero_dim
7485         {
7486             \dim_set:Nn \l_coffin_x_dim {#5}
7487             \coffin_calculate_intersection_aux:nnnnnN
7488             {#5} {#1} {#2} {#3} {#4} \l_coffin_y_dim
7489         }
7490         {
7491             \dim_compare:nNnTF {#8} = \c_zero_dim
7492             {
7493                 \dim_set:Nn \l_coffin_x_dim {#6}
7494                 \coffin_calculate_intersection_aux:nnnnnN
7495                 {#6} {#2} {#1} {#4} {#3} \l_coffin_x_dim
7496             }

```

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.

```

7497         {
7498             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#3}
7499             \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#4}
7500             \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#7}
7501             \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#8}
7502             \fp_div:Nn \l_coffin_calc_b_fp \l_coffin_calc_a_fp
7503             \fp_div:Nn \l_coffin_calc_d_fp \l_coffin_calc_c_fp
7504             \fp_compare:nNnTF
7505                 \l_coffin_calc_b_fp = \l_coffin_calc_d_fp
7506                 { \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.

```

7507         {
7508             \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#6}
7509             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
7510             \fp_sub:Nn \l_coffin_calc_result_fp
7511                 { \l_coffin_calc_a_fp }
7512             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#1}
7513             \fp_mul:Nn \l_coffin_calc_a_fp
7514                 { \l_coffin_calc_b_fp }
7515             \fp_add:Nn \l_coffin_calc_result_fp
7516                 { \l_coffin_calc_a_fp }
7517             \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#5}
7518             \fp_mul:Nn \l_coffin_calc_a_fp
7519                 { \l_coffin_calc_d_fp }
7520             \fp_sub:Nn \l_coffin_calc_result_fp
7521                 { \l_coffin_calc_a_fp }
7522             \fp_sub:Nn \l_coffin_calc_b_fp
7523                 { \l_coffin_calc_d_fp }
7524             \fp_div:Nn \l_coffin_calc_result_fp
7525                 { \l_coffin_calc_b_fp }
7526             \dim_set:Nn \l_coffin_x_dim
7527                 { \fp_to_dim:N \l_coffin_calc_result_fp }
7528             \coffin_calculate_intersection_aux:nnnnnN
7529                 { \l_coffin_x_dim }
7530                 {#5} {#6} {#8} {#7} \l_coffin_y_dim
7531         }
7532     }
7533 }
7534

```

```

7535     }
7536 }

```

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.

```

7537 \cs_new_protected:Npn \coffin_calculate_intersection_aux:nnnnnN
7538   #1#2#3#4#5#6
7539   {
7540     \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#1}
7541     \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
7542     \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#3}
7543     \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#4}
7544     \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#5}
7545     \fp_sub:Nn \l_coffin_calc_result_fp { \l_coffin_calc_a_fp }
7546     \fp_div:Nn \l_coffin_calc_result_fp { \l_coffin_calc_d_fp }
7547     \fp_mul:Nn \l_coffin_calc_result_fp { \l_coffin_calc_c_fp }
7548     \fp_add:Nn \l_coffin_calc_result_fp { \l_coffin_calc_b_fp }
7549     \dim_set:Nn #6 { \fp_to_dim:N \l_coffin_calc_result_fp }
7550   }

```

(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.

```

7551 \cs_new_protected:Npn \coffin_join:NnnNnnnn #1#2#3#4#5#6#7#8
7552   {
7553     \coffin_align:NnnNnnnnN
7554     #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.

```

7555   \hbox_set:Nn \l_coffin_aligned_coffin
7556   {
7557     \dim_compare:nNnT { \l_coffin_offset_x_dim } < \c_zero_dim
7558     { \tex_kern:D -\l_coffin_offset_x_dim }
7559     \hbox_unpack:N \l_coffin_aligned_coffin
7560     \dim_set:Nn \l_coffin_internal_dim
7561     { \l_coffin_offset_x_dim - \box_wd:N #1 + \box_wd:N #4 }
7562     \dim_compare:nNnT \l_coffin_internal_dim < \c_zero_dim
7563     { \tex_kern:D -\l_coffin_internal_dim }
7564   }

```


The coffin structure is reset, and the corners are cleared: only those from the two parent coffins are needed.

```

7565 \coffin_reset_structure:N \l_coffin_aligned_coffin
7566 \prop_clear:c
7567 { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7568 \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.

```

7569 \dim_compare:nNnTF \l_coffin_offset_x_dim < \c_zero_dim
7570 {
7571   \coffin_offset_poles:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
7572   \coffin_offset_poles:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
7573   \coffin_offset_corners:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
7574   \coffin_offset_corners:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
7575 }
7576 {
7577   \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
7578   \coffin_offset_poles:Nnn #4
7579   { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7580   \coffin_offset_corners:Nnn #1 { 0 pt } { 0 pt }
7581   \coffin_offset_corners:Nnn #4
7582   { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7583 }
7584 \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
7585 \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
7586 }
7587 \cs_generate_variant:Nn \coffin_join:NnnNnnnn { c , Nnnc , cnnc }

```

(End definition for \coffin_join:NnnNnnnn and others. These functions are documented on page ??.)

\coffin_attach:NnnNnnnn 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.
\coffin_attach:cnNnnnnn The function used when marking a position is hear also as it is similar but without the structure updates.
\coffin_attach:Nnncnnnn
\coffin_attach:cnncnnnn

```

\coffin_attach_mark:NnnNnnnn 7588 \cs_new_protected:Npn \coffin_attach:NnnNnnnn #1#2#3#4#5#6#7#8
7589 {
7590   \coffin_align:NnnNnnnnN
7591   #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
7592   \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
7593   \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
7594   \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
7595   \coffin_reset_structure:N \l_coffin_aligned_coffin
7596   \prop_set_eq:cc
7597   { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7598   { l_coffin_corners_ \int_value:w #1 _prop }
7599   \coffin_update_poles:N \l_coffin_aligned_coffin
7600   \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }

```

```

7601 \coffin_offset_poles:Nnn #4
7602 { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
7603 \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
7604 \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
7605 }
7606 \cs_new_protected:Npn \coffin_attach_mark:NnnNnnnn #1#2#3#4#5#6#7#8
7607 {
7608   \coffin_align:NnnNnnnnN
7609   #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
7610   \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
7611   \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
7612   \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
7613   \box_set_eq:NN #1 \l_coffin_aligned_coffin
7614 }
7615 \cs_generate_variant:Nn \coffin_attach:NnnNnnnn { c , nnc , 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.

```

7616 \cs_new_protected:Npn \coffin_align:NnnNnnnnN #1#2#3#4#5#6#7#8#9
7617 {
7618   \coffin_calculate_intersection:Nnn #4 {#5} {#6}
7619   \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
7620   \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
7621   \coffin_calculate_intersection:Nnn #1 {#2} {#3}
7622   \dim_set:Nn \l_coffin_offset_x_dim
7623   { \l_coffin_x_dim - \l_coffin_x_prime_dim + #7 }
7624   \dim_set:Nn \l_coffin_offset_y_dim
7625   { \l_coffin_y_dim - \l_coffin_y_prime_dim + #8 }
7626   \hbox_set:Nn \l_coffin_aligned_internal_coffin
7627   {
7628     \box_use:N #1
7629     \tex_kern:D -\box_wd:N #1
7630     \tex_kern:D \l_coffin_offset_x_dim
7631     \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #4 }
7632   }
7633   \coffin_set_eq:NN #9 \l_coffin_aligned_internal_coffin
7634 }

```

(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.

```

7635 \cs_new_protected:Npn \coffin_offset_poles:Nnn #1#2#3
7636 {
7637   \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7638   { \coffin_offset_pole:Nnnnnnn #1 {##1} ##2 {#2} {#3} }
7639 }
7640 \cs_new_protected:Npn \coffin_offset_pole:Nnnnnnn #1#2#3#4#5#6#7#8
7641 {
7642   \dim_set:Nn \l_coffin_x_dim { #3 + #7 }
7643   \dim_set:Nn \l_coffin_y_dim { #4 + #8 }
7644   \tl_if_in:nnTF {#2} { - }
7645   { \tl_set:Nn \l_coffin_internal_tl { {#2} } }
7646   { \tl_set:Nn \l_coffin_internal_tl { { #1 - #2 } } }
7647   \exp_last_unbraced:NNo \coffin_set_pole:Nnx \l_coffin_aligned_coffin
7648   { \l_coffin_internal_tl }
7649   {
7650     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7651     {#5} {#6}
7652   }
7653 }

```

(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:Nnnnnn about naming: every corner can be saved here as order is unimportant.

```

7654 \cs_new_protected:Npn \coffin_offset_corners:Nnn #1#2#3
7655 {
7656   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7657   { \coffin_offset_corner:Nnnnn #1 {##1} ##2 {#2} {#3} }
7658 }
7659 \cs_new_protected:Npn \coffin_offset_corner:Nnnnn #1#2#3#4#5#6
7660 {
7661   \prop_put:cnx
7662   { l_coffin_corners_ \int_value:w \l_coffin_aligned_coffin _prop }
7663   { #1 - #2 }
7664   {
7665     { \dim_eval:n { #3 + #5 } }
7666     { \dim_eval:n { #4 + #6 } }
7667   }
7668 }

```

(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.

```

7669 \cs_new_protected:Npn \coffin_update_vertical_poles:NNN #1#2#3

```

```

7670 {
7671   \coffin_get_pole:NnN #3 { #1 -T } \l_coffin_pole_a_tl
7672   \coffin_get_pole:NnN #3 { #2 -T } \l_coffin_pole_b_tl
7673   \exp_last_two_unbraced:Noo \coffin_update_T:nnnnnnnnN
7674   \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
7675   \coffin_get_pole:NnN #3 { #1 -B } \l_coffin_pole_a_tl
7676   \coffin_get_pole:NnN #3 { #2 -B } \l_coffin_pole_b_tl
7677   \exp_last_two_unbraced:Noo \coffin_update_B:nnnnnnnnN
7678   \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
7679 }
7680 \cs_new_protected:Npn \coffin_update_T:nnnnnnnnN #1#2#3#4#5#6#7#8#9
7681 {
7682   \dim_compare:nNnTF {#2} < {#6}
7683   {
7684     \coffin_set_pole:Nnx #9 { T }
7685     { { 0 pt } {#6} { 1000 pt } { 0 pt } }
7686   }
7687   {
7688     \coffin_set_pole:Nnx #9 { T }
7689     { { 0 pt } {#2} { 1000 pt } { 0 pt } }
7690   }
7691 }
7692 \cs_new_protected:Npn \coffin_update_B:nnnnnnnnN #1#2#3#4#5#6#7#8#9
7693 {
7694   \dim_compare:nNnTF {#2} < {#6}
7695   {
7696     \coffin_set_pole:Nnx #9 { B }
7697     { { 0 pt } {#2} { 1000 pt } { 0 pt } }
7698   }
7699   {
7700     \coffin_set_pole:Nnx #9 { B }
7701     { { 0 pt } {#6} { 1000 pt } { 0 pt } }
7702   }
7703 }

```

(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.

```

7704 \cs_new_protected:Npn \coffin_typeset:Nnnnn #1#2#3#4#5
7705 {
7706   \coffin_align:NnnNnnnnN \c_empty_coffin { H } { 1 }
7707   #1 {#2} {#3} {#4} {#5} \l_coffin_aligned_coffin
7708   \hbox_unpack:N \c_empty_box
7709   \box_use:N \l_coffin_aligned_coffin
7710 }
7711 \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.

```
7712 \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.

```
7713 \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 minimum size of the bounding box after rotation.

`\l_coffin_right_corner_dim`

`\l_coffin_bottom_corner_dim`

`\l_coffin_top_corner_dim`

```
7714 \dim_new:N \l_coffin_left_corner_dim
7715 \dim_new:N \l_coffin_right_corner_dim
7716 \dim_new:N \l_coffin_bottom_corner_dim
7717 \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 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.

`\coffin_rotate:cn`

```
7718 \cs_new_protected:Npn \coffin_rotate:Nn #1#2
7719 {
7720   \fp_set:Nn \l_coffin_internal_fp {#2}
7721   \fp_div:Nn \l_coffin_internal_fp { 180 }
7722   \fp_mul:Nn \l_coffin_internal_fp { \c_pi_fp }
7723   \fp_sin:Nn \l_coffin_sin_fp { \l_coffin_internal_fp }
7724   \fp_cos:Nn \l_coffin_cos_fp { \l_coffin_internal_fp }
```

The corners and poles of the coffin can now be rotated around the origin. This is best achieved using mapping functions.

```
7725 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7726 { \coffin_rotate_corner:Nnnn #1 {##1} ##2 }
7727 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7728 { \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.

```
7729 \coffin_set_bounding:N #1
7730 \prop_map_inline:Nn \l_coffin_bounding_prop
7731 { \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.

```
7732 \coffin_find_corner_maxima:N #1
7733 \coffin_find_bounding_shift:
7734 \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.

```

7735 \hbox_set:Nn #1
7736 {
7737   \tex_kern:D \l_coffin_bounding_shift_dim
7738   \tex_kern:D -\l_coffin_left_corner_dim
7739   \box_move_down:nn { \l_coffin_bottom_corner_dim }
7740   { \box_use:N #1 }
7741 }

```

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.

```

7742 \box_set_ht:Nn #1
7743 { \l_coffin_top_corner_dim - \l_coffin_bottom_corner_dim }
7744 \box_set_dp:Nn #1 { 0 pt }
7745 \box_set_wd:Nn #1
7746 { \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.

```

7747 \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7748 { \coffin_shift_corner:Nnnn #1 {##1} ##2 }
7749 \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7750 { \coffin_shift_pole:Nnnnnn #1 {##1} ##2 }
7751 }
7752 \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.

```

7753 \cs_new_protected:Npn \coffin_set_bounding:N #1
7754 {
7755   \prop_put:Nnx \l_coffin_bounding_prop { tl }
7756   { { 0 pt } { \dim_use:N \box_ht:N #1 } }
7757   \prop_put:Nnx \l_coffin_bounding_prop { tr }
7758   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
7759   \dim_set:Nn \l_coffin_internal_dim { - \box_dp:N #1 }
7760   \prop_put:Nnx \l_coffin_bounding_prop { bl }
7761   { { 0 pt } { \dim_use:N \l_coffin_internal_dim } }
7762   \prop_put:Nnx \l_coffin_bounding_prop { br }
7763   { { \dim_use:N \box_wd:N #1 } { \dim_use:N \l_coffin_internal_dim } }
7764 }

```

(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.

```

7765 \cs_new_protected:Npn \coffin_rotate_bounding:nnn #1#2#3
7766 {
7767   \coffin_rotate_vector:nnNN {#2} {#3} \l_coffin_x_dim \l_coffin_y_dim
7768   \prop_put:Nnx \l_coffin_bounding_prop {#1}
7769   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7770 }
7771 \cs_new_protected:Npn \coffin_rotate_corner:Nnnn #1#2#3#4
7772 {
7773   \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7774   \prop_put:cnx { \l_coffin_corners_ \int_value:w #1 _prop } {#2}
7775   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7776 }

```

(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.

```

7777 \cs_new_protected:Npn \coffin_rotate_pole:Nnnnnn #1#2#3#4#5#6
7778 {
7779   \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7780   \coffin_rotate_vector:nnNN {#5} {#6}
7781   \l_coffin_x_prime_dim \l_coffin_y_prime_dim
7782   \coffin_set_pole:Nnx #1 {#2}
7783   {
7784     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7785     { \dim_use:N \l_coffin_x_prime_dim }
7786     { \dim_use:N \l_coffin_y_prime_dim }
7787   }
7788 }

```

(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.

```

7789 \cs_new_protected:Npn \coffin_rotate_vector:nnNN #1#2#3#4
7790 {
7791   \fp_set_from_dim:Nn \l_coffin_x_fp {#1}
7792   \fp_set_from_dim:Nn \l_coffin_y_fp {#2}
7793   \fp_set_eq:NN \l_coffin_x_prime_fp \l_coffin_x_fp
7794   \fp_set_eq:NN \l_coffin_internal_fp \l_coffin_y_fp
7795   \fp_mul:Nn \l_coffin_x_prime_fp { \l_coffin_cos_fp }
7796   \fp_mul:Nn \l_coffin_internal_fp { \l_coffin_sin_fp }
7797   \fp_sub:Nn \l_coffin_x_prime_fp { \l_coffin_internal_fp }
7798   \fp_set_eq:NN \l_coffin_y_prime_fp \l_coffin_y_fp
7799   \fp_set_eq:NN \l_coffin_internal_fp \l_coffin_x_fp

```

```

7800 \fp_mul:Nn \l_coffin_y_prime_fp { \l_coffin_cos_fp }
7801 \fp_mul:Nn \l_coffin_internal_fp { \l_coffin_sin_fp }
7802 \fp_add:Nn \l_coffin_y_prime_fp { \l_coffin_internal_fp }
7803 \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_x_prime_fp }
7804 \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_y_prime_fp }
7805 }

```

(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.

```

7806 \cs_new_protected:Npn \coffin_find_corner_maxima:N #1
7807 {
7808   \dim_set:Nn \l_coffin_top_corner_dim { -\c_max_dim }
7809   \dim_set:Nn \l_coffin_right_corner_dim { -\c_max_dim }
7810   \dim_set:Nn \l_coffin_bottom_corner_dim { \c_max_dim }
7811   \dim_set:Nn \l_coffin_left_corner_dim { \c_max_dim }
7812   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7813     { \coffin_find_corner_maxima_aux:nn ##2 }
7814 }
7815 \cs_new_protected:Npn \coffin_find_corner_maxima_aux:nn #1#2
7816 {
7817   \dim_set_min:Nn \l_coffin_left_corner_dim {#1}
7818   \dim_set_max:Nn \l_coffin_right_corner_dim {#1}
7819   \dim_set_min:Nn \l_coffin_bottom_corner_dim {#2}
7820   \dim_set_max:Nn \l_coffin_top_corner_dim {#2}
7821 }

```

(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.

```

7822 \cs_new_protected_nopar:Npn \coffin_find_bounding_shift:
7823 {
7824   \dim_set:Nn \l_coffin_bounding_shift_dim { \c_max_dim }
7825   \prop_map_inline:Nn \l_coffin_bounding_prop
7826     { \coffin_find_bounding_shift_aux:nn ##2 }
7827 }
7828 \cs_new_protected:Npn \coffin_find_bounding_shift_aux:nn #1#2
7829 { \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.

```

7830 \cs_new_protected:Npn \coffin_shift_corner:Nnnn #1#2#3#4
7831 {

```



```

7832 \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _ prop } {#2}
7833 {
7834   { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
7835   { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
7836 }
7837 }
7838 \cs_new_protected:Npn \coffin_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 - \l_coffin_left_corner_dim } }
7843     { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
7844     {#5} {#6}
7845   }
7846 }

```

(End definition for \coffin_shift_corner:Nnnnn. 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

```

7847 \fp_new:N \l_coffin_scale_x_fp
7848 \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

```

7849 \dim_new:N \l_coffin_scaled_total_height_dim
7850 \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.

```

7851 \cs_new_protected:Npn \coffin_resize:Nnn #1#2#3
7852 {
7853   \coffin_set_user_dimensions:N #1
7854   \box_resize:Nnn #1 {#2} {#3}
7855   \fp_set_from_dim:Nn \l_coffin_scale_x_fp {#2}
7856   \fp_set_from_dim:Nn \l_coffin_internal_fp { \Width }
7857   \fp_div:Nn \l_coffin_scale_x_fp { \l_coffin_internal_fp }
7858   \fp_set_from_dim:Nn \l_coffin_scale_y_fp {#3}
7859   \fp_set_from_dim:Nn \l_coffin_internal_fp { \TotalHeight }
7860   \fp_div:Nn \l_coffin_scale_y_fp { \l_coffin_internal_fp }
7861   \coffin_resize_common:Nnn #1 {#2} {#3}
7862 }
7863 \cs_generate_variant:Nn \coffin_resize:Nnn { c }

```

(End definition for \coffin_resize:Nnn and \coffin_resize:cnn. 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.

```

7864 \cs_new_protected:Npn \coffin_resize_common:Nnn #1#2#3
7865 {
7866   \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7867   { \coffin_scale_corner:Nnnn #1 {##1} ##2 }
7868   \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7869   { \coffin_scale_pole:Nnnnnn #1 {##1} ##2 }

```

Negative x -scaling values will place the poles in the wrong location: this is corrected here.

```

7870   \fp_compare:NNNT \l_coffin_scale_x_fp < \c_zero_fp
7871   {
7872     \prop_map_inline:cn { l_coffin_corners_ \int_value:w #1 _prop }
7873     { \coffin_x_shift_corner:Nnnn #1 {##1} ##2 }
7874     \prop_map_inline:cn { l_coffin_poles_ \int_value:w #1 _prop }
7875     { \coffin_x_shift_pole:Nnnnnn #1 {##1} ##2 }
7876   }
7877   \coffin_end_user_dimensions:
7878 }

```

(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.

```

7879 \cs_new_protected:Npn \coffin_scale:Nnn #1#2#3
7880 {
7881   \box_scale:Nnn #1 {#2} {#3}
7882   \coffin_set_user_dimensions:N #1
7883   \fp_set:Nn \l_coffin_scale_x_fp {#2}
7884   \fp_set:Nn \l_coffin_scale_y_fp {#3}
7885   \fp_compare:NNNTF \l_coffin_scale_y_fp > \c_zero_fp
7886   { \l_coffin_scaled_total_height_dim #3 \TotalHeight }
7887   { \l_coffin_scaled_total_height_dim -#3 \TotalHeight }
7888   \fp_compare:NNNTF \l_coffin_scale_x_fp > \c_zero_fp
7889   { \l_coffin_scaled_width_dim -#2 \Width }
7890   { \l_coffin_scaled_width_dim #2 \Width }
7891   \coffin_resize_common:Nnn #1
7892   { \l_coffin_scaled_width_dim } { \l_coffin_scaled_total_height_dim }
7893 }
7894 \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 function 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.

```

7895 \cs_new_protected:Npn \coffin_scale_vector:nnNN #1#2#3#4

```

```

7896 {
7897   \fp_set_from_dim:Nn \l_coffin_internal_fp {#1}
7898   \fp_mul:Nn \l_coffin_internal_fp { \l_coffin_scale_x_fp }
7899   \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_internal_fp }
7900   \fp_set_from_dim:Nn \l_coffin_internal_fp {#2}
7901   \fp_mul:Nn \l_coffin_internal_fp { \l_coffin_scale_y_fp }
7902   \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_internal_fp }
7903 }

```

(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
7904 \cs_new_protected:Npn \coffin_scale_corner:Nnnn #1#2#3#4
7905 {
7906   \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7907   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7908   { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
7909 }
7910 \cs_new_protected:Npn \coffin_scale_pole:Nnnnnn #1#2#3#4#5#6
7911 {
7912   \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
7913   \coffin_set_pole:Nnx #1 {#2}
7914   {
7915     { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
7916     {#5} {#6}
7917   }
7918 }

```

(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.

```

7919 \cs_new_protected:Npn \coffin_x_shift_corner:Nnnn #1#2#3#4
7920 {
7921   \prop_put:cnx { l_coffin_corners_ \int_value:w #1 _prop } {#2}
7922   {
7923     { \dim_eval:n { #3 + \box_wd:N #1 } } {#4}
7924   }
7925 }
7926 \cs_new_protected:Npn \coffin_x_shift_pole:Nnnnnn #1#2#3#4#5#6
7927 {
7928   \prop_put:cnx { l_coffin_poles_ \int_value:w #1 _prop } {#2}
7929   {
7930     { \dim_eval:n { #3 + \box_wd:N #1 } } {#4}
7931     {#5} {#6}
7932   }
7933 }

```

(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 7934 \coffin_new:N \l_coffin_display_coffin
\l_coffin_display_pole_coffin 7935 \coffin_new:N \l_coffin_display_coord_coffin
7936 \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.

```

7937 \prop_new:N \l_coffin_display_handles_prop
7938 \prop_put:Nnn \l_coffin_display_handles_prop { tl }
7939 { { b } { r } { -1 } { 1 } }
7940 \prop_put:Nnn \l_coffin_display_handles_prop { thc }
7941 { { b } { hc } { 0 } { 1 } }
7942 \prop_put:Nnn \l_coffin_display_handles_prop { tr }
7943 { { b } { l } { 1 } { 1 } }
7944 \prop_put:Nnn \l_coffin_display_handles_prop { vc1 }
7945 { { vc } { r } { -1 } { 0 } }
7946 \prop_put:Nnn \l_coffin_display_handles_prop { vhc }
7947 { { vc } { hc } { 0 } { 0 } }
7948 \prop_put:Nnn \l_coffin_display_handles_prop { vcr }
7949 { { vc } { l } { 1 } { 0 } }
7950 \prop_put:Nnn \l_coffin_display_handles_prop { bl }
7951 { { t } { r } { -1 } { -1 } }
7952 \prop_put:Nnn \l_coffin_display_handles_prop { bhc }
7953 { { t } { hc } { 0 } { -1 } }
7954 \prop_put:Nnn \l_coffin_display_handles_prop { br }
7955 { { t } { l } { 1 } { -1 } }
7956 \prop_put:Nnn \l_coffin_display_handles_prop { Tl }
7957 { { t } { r } { -1 } { -1 } }
7958 \prop_put:Nnn \l_coffin_display_handles_prop { Thc }
7959 { { t } { hc } { 0 } { -1 } }
7960 \prop_put:Nnn \l_coffin_display_handles_prop { Tr }
7961 { { t } { l } { 1 } { -1 } }
7962 \prop_put:Nnn \l_coffin_display_handles_prop { Hl }
7963 { { vc } { r } { -1 } { 1 } }
7964 \prop_put:Nnn \l_coffin_display_handles_prop { Hhc }
7965 { { vc } { hc } { 0 } { 1 } }
7966 \prop_put:Nnn \l_coffin_display_handles_prop { Hr }
7967 { { vc } { l } { 1 } { 1 } }
7968 \prop_put:Nnn \l_coffin_display_handles_prop { Bl }
7969 { { b } { r } { -1 } { -1 } }
7970 \prop_put:Nnn \l_coffin_display_handles_prop { Bhc }
7971 { { b } { hc } { 0 } { -1 } }
7972 \prop_put:Nnn \l_coffin_display_handles_prop { Br }
7973 { { b } { l } { 1 } { -1 } }

```

(End definition for `\l_coffin_display_handles_prop`. This function is documented on page ??.)

`\l_coffin_display_offset_dim` The standard offset for the label from the handle position when displaying handles.

```

7974 \dim_new:N \l_coffin_display_offset_dim
7975 \dim_set:Nn \l_coffin_display_offset_dim { 2 pt }
      (End definition for \l_coffin_display_offset_dim. This function is documented on page ??.)

```

`\l_coffin_display_x_dim` As the intersections of poles have to be calculated to find which ones to print, there is
`\l_coffin_display_y_dim` a need to avoid repetition. This is done by saving the intersection into two dedicated values.

```

7976 \dim_new:N \l_coffin_display_x_dim
7977 \dim_new:N \l_coffin_display_y_dim
      (End definition for \l_coffin_display_x_dim. This function is documented on page ??.)

```

`\l_coffin_display_poles_prop` A property list for printing poles: various things need to be deleted from this to get a “nice” output.

```

7978 \prop_new:N \l_coffin_display_poles_prop
      (End definition for \l_coffin_display_poles_prop. This function is documented on page ??.)

```

`\l_coffin_display_font_tl` Stores the settings used to print coffin data: this keeps things flexible.

```

7979 \tl_new:N \l_coffin_display_font_tl
7980 <*initex>
7981 \tl_set:Nn \l_coffin_display_font_tl { } % TODO
7982 </initex>
7983 <*package>
7984 \tl_set:Nn \l_coffin_display_font_tl { \sfamily \tiny }
7985 </package>
      (End definition for \l_coffin_display_font_tl. This function is documented on page ??.)

```

`\coffin_mark_handle:Nnnn` Marking a single handle is relatively easy. The standard attachment function is used,
`\coffin_mark_handle:cnnn` meaning that there are two calculations for the location. However, this is likely to be
`\coffin_mark_handle_aux:nnnnNnn` okay given the load expected. Contrast with the more optimised version for showing all handles which comes next.

```

7986 \cs_new_protected:Npn \coffin_mark_handle:Nnnn #1#2#3#4
7987 {
7988   \hcoffin_set:Nn \l_coffin_display_pole_coffin
7989   {
7990     <*initex>
7991     \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TODO
7992     </initex>
7993     <*package>
7994     \color {#4}
7995     \rule { 1 pt } { 1 pt }
7996     </package>
7997   }
7998   \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
7999   \l_coffin_display_pole_coffin { hc } { vc } { 0 pt } { 0 pt }
8000   \hcoffin_set:Nn \l_coffin_display_coord_coffin
8001   {
8002     <*initex>

```

```

8003          % TODO
8004 \end{initex}
8005 \begin{package}
8006     \color{#4}
8007 \end{package}
8008     \l_coffin_display_font_tl
8009     ( \tl_to_str:n { #2 , #3 } )
8010 }
8011 \prop_get:NnN \l_coffin_display_handles_prop
8012 { #2 #3 } \l_coffin_internal_tl
8013 \quark_if_no_value:NTF \l_coffin_internal_tl
8014 {
8015     \prop_get:NnN \l_coffin_display_handles_prop
8016     { #3 #2 } \l_coffin_internal_tl
8017     \quark_if_no_value:NTF \l_coffin_internal_tl
8018     {
8019         \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
8020         \l_coffin_display_coord_coffin { 1 } { vc }
8021         { 1 pt } { 0 pt }
8022     }
8023     {
8024         \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
8025         \l_coffin_internal_tl #1 {#2} {#3}
8026     }
8027 }
8028 {
8029     \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
8030     \l_coffin_internal_tl #1 {#2} {#3}
8031 }
8032 }
8033 \cs_new_protected:Npn \coffin_mark_handle_aux:nnnnNnn #1#2#3#4#5#6#7
8034 {
8035     \coffin_attach_mark:NnnNnnnn #5 {#6} {#7}
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_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 the definitive version for the baseline and bottom. Two loops are then used to find the combinations of handles for all of these poles. This is done such that poles are removed during the loops to avoid duplication.

\coffin_display_handles:cn

\coffin_display_handles_aux:nnnnnn

\coffin_display_handles_aux:nnnn

\coffin_display_attach:Nnnnnn

```

8041 \cs_new_protected:Npn \coffin_display_handles:Nn #1#2
8042 {
8043     \hcoffin_set:Nn \l_coffin_display_pole_coffin
8044     {

```

```

8045 <*initex>
8046     \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TODO
8047 </initex>
8048 <*package>
8049     \color {#2}
8050     \rule { 1 pt } { 1 pt }
8051 </package>
8052 }
8053 \prop_set_eq:Nc \l_coffin_display_poles_prop
8054 { \l_coffin_poles_ \int_value:w #1 _prop }
8055 \coffin_get_pole:NnN #1 { H } \l_coffin_pole_a_tl
8056 \coffin_get_pole:NnN #1 { T } \l_coffin_pole_b_tl
8057 \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
8058 { \prop_del:Nn \l_coffin_display_poles_prop { T } }
8059 \coffin_get_pole:NnN #1 { B } \l_coffin_pole_b_tl
8060 \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
8061 { \prop_del:Nn \l_coffin_display_poles_prop { B } }
8062 \coffin_set_eq:NN \l_coffin_display_coffin #1
8063 \prop_map_inline:Nn \l_coffin_display_poles_prop
8064 {
8065     \prop_del:Nn \l_coffin_display_poles_prop {##1}
8066     \coffin_display_handles_aux:nnnnnn {##1} ##2 {#2}
8067 }
8068 \box_use:N \l_coffin_display_coffin
8069 }

```

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.

```

8070 \cs_new_protected:Npn \coffin_display_handles_aux:nnnnnn #1#2#3#4#5#6
8071 {
8072     \prop_map_inline:Nn \l_coffin_display_poles_prop
8073     {
8074         \bool_set_false:N \l_coffin_error_bool
8075         \coffin_calculate_intersection:nnnnnnnn {#2} {#3} {#4} {#5} ##2
8076         \bool_if:NF \l_coffin_error_bool
8077         {
8078             \dim_set:Nn \l_coffin_display_x_dim { \l_coffin_x_dim }
8079             \dim_set:Nn \l_coffin_display_y_dim { \l_coffin_y_dim }
8080             \coffin_display_attach:Nnnnn
8081             \l_coffin_display_pole_coffin { hc } { vc }
8082             { 0 pt } { 0 pt }
8083             \hcoffin_set:Nn \l_coffin_display_coord_coffin
8084             {
8085 <*initex>
8086             % TODO
8087 </initex>
8088 <*package>
8089             \color {#6}
8090 </package>

```

```

8091         \l_coffin_display_font_tl
8092         ( \tl_to_str:n { #1 , ##1 } )
8093     }
8094     \prop_get:NnN \l_coffin_display_handles_prop
8095     { #1 ##1 } \l_coffin_internal_tl
8096     \quark_if_no_value:NTF \l_coffin_internal_tl
8097     {
8098         \prop_get:NnN \l_coffin_display_handles_prop
8099         { ##1 #1 } \l_coffin_internal_tl
8100         \quark_if_no_value:NTF \l_coffin_internal_tl
8101         {
8102             \coffin_display_attach:Nnnnn
8103             \l_coffin_display_coord_coffin { 1 } { vc }
8104             { 1 pt } { 0 pt }
8105         }
8106         {
8107             \exp_last_unbraced:No
8108             \coffin_display_handles_aux:nnnn
8109             \l_coffin_internal_tl
8110         }
8111     }
8112     {
8113         \exp_last_unbraced:No \coffin_display_handles_aux:nnnn
8114         \l_coffin_internal_tl
8115     }
8116 }
8117 }
8118 }
8119 \cs_new_protected:Npn \coffin_display_handles_aux:nnnn #1#2#3#4
8120 {
8121     \coffin_display_attach:Nnnnn
8122     \l_coffin_display_coord_coffin {#1} {#2}
8123     { #3 \l_coffin_display_offset_dim }
8124     { #4 \l_coffin_display_offset_dim }
8125 }
8126 \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.

```

8127 \cs_new_protected:Npn \coffin_display_attach:Nnnnn #1#2#3#4#5
8128 {
8129     \coffin_calculate_intersection:Nnn #1 {#2} {#3}
8130     \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
8131     \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
8132     \dim_set:Nn \l_coffin_offset_x_dim
8133     { \l_coffin_display_x_dim - \l_coffin_x_prime_dim + #4 }
8134     \dim_set:Nn \l_coffin_offset_y_dim
8135     { \l_coffin_display_y_dim - \l_coffin_y_prime_dim + #5 }
8136     \hbox_set:Nn \l_coffin_aligned_coffin

```



```

8137 {
8138   \box_use:N \l_coffin_display_coffin
8139   \tex_kern:D -\box_wd:N \l_coffin_display_coffin
8140   \tex_kern:D \l_coffin_offset_x_dim
8141   \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #1 }
8142 }
8143 \box_set_ht:Nn \l_coffin_aligned_coffin
8144 { \box_ht:N \l_coffin_display_coffin }
8145 \box_set_dp:Nn \l_coffin_aligned_coffin
8146 { \box_dp:N \l_coffin_display_coffin }
8147 \box_set_wd:Nn \l_coffin_aligned_coffin
8148 { \box_wd:N \l_coffin_display_coffin }
8149 \box_set_eq:NN \l_coffin_display_coffin \l_coffin_aligned_coffin
8150 }

```

(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`

```

8151 \cs_new_protected:Npn \coffin_show_structure:N #1
8152 {
8153   \cs_if_exist:cTF { l_coffin_poles_ \int_value:w #1 _prop }
8154   {
8155     \msg_aux_show:Nnx #1 { coffins }
8156     {
8157       \prop_map_function:cN
8158       { l_coffin_poles_ \int_value:w #1 _prop }
8159       \msg_aux_show_unbraced:nn
8160     }
8161   }
8162   {
8163     \msg_aux_use:nn { LaTeX / coffins } { no-pole }
8164     \msg_aux_show:x { }
8165   }
8166 }
8167 \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

```

8168 \msg_kernel_new:nnnn { coffins } { no-pole-intersection }
8169 { No~intersection~between~coffin~poles. }
8170 {
8171   \c_msg_coding_error_text_tl
8172   LaTeX~was~asked~to~find~the~intersection~between~two~poles,~
8173   but~they~do~not~have~a~unique~meeting~point:~
8174   the~value~(0~pt,~0~pt)~will~be~used.
8175 }

```

```

8176 \msg_kernel_new:nnnn { coffins } { unknown-coffin }
8177 { Unknown~coffin~'#1'. }
8178 { The~coffin~'#1'~was~never~defined. }
8179 \msg_kernel_new:nnnn { coffins } { unknown-coffin-pole }
8180 { Pole~'#1'~unknown~for~coffin~'#2'. }
8181 {
8182   \c_msg_coding_error_text_tl
8183   LaTeX~was~asked~to~find~a~typesetting~pole~for~a~coffin,~
8184   but~either~the~coffin~does~not~exist~or~the~pole~name~is~wrong.
8185 }
8186 \msg_kernel_new:nnn { coffins } { show }
8187 {
8188   Size~of~coffin~\token_to_str:N #1 : \\
8189   > ~ ht~~~\dim_use:N \box_ht:N #1 \\
8190   > ~ dp~~~\dim_use:N \box_dp:N #1 \\
8191   > ~ wd~~~\dim_use:N \box_wd:N #1 \\
8192   Poles~of~coffin~\token_to_str:N #1 :
8193 }
8194 \msg_kernel_new:nnn { coffins } { no-pole }
8195 {
8196   ----No~poles~found---- \\
8197   Is~this~really~a~coffin?
8198 }
8199 </initex | package>

```

198 l3color Implementation

```

8200 <*initex | package>
8201 <*package>
8202 \ProvidesExplPackage
8203   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
8204 \package_check_loaded_expl:
8205 </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.

```

8206 \cs_new_eq:NN \color_group_begin: \group_begin:
8207 \cs_new_protected_nopar:Npn \color_group_end:
8208 {
8209   \tex_par:D
8210   \group_end:
8211 }

```

(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”.

```

8212 <*initex>

```

```

8213 \cs_new_protected_nopar:Npn \color_ensure_current:
8214 { \driver_color_ensure_current: }
8215 \</initex>
8216 \*package>
8217 \cs_new_protected_nopar:Npn \color_ensure_current: { \set@color }
8218 \</package>
      (End definition for \color_ensure_current:. This function is documented on page ??.)
8219 \</initex | package>

```

199 l3io implementation

```

8220 \*initex | package>
8221 \*package>
8222 \ProvidesExplPackage
8223 { \ExplFileName } { \ExplFileDate } { \ExplFileVersion } { \ExplFileDescription }
8224 \package_check_loaded_expl:
8225 \</package>

```

199.1 Primitives

`\if_eof:w` The primitive conditional

```

8226 \cs_new_eq:NN \if_eof:w \tex_ifeof:D
      (End definition for \if_eof:w. This function is documented on page 141.)

```

199.2 Variables and constants

`\c_term_iow` 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.

```

8227 \cs_new_eq:NN \c_term_iow \c_sixteen
      (End definition for \c_term_iow. This function is documented on page 140.)

```

`\c_log_iow` Here we allocate two output streams for writing to the transcript file only (`\c_log_iow`)
`\c_term_iow` and to both the terminal and transcript file (`\c_term_iow`).

```

8228 \cs_new_eq:NN \c_log_iow \c_minus_one
8229 \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 140.)

```

`\c_iow_streams_tl` The list of streams available, by number.

```

\c_iow_streams_tl 8230 \tl_const:Nn \c_iow_streams_tl
8231 {
8232   \c_zero
8233   \c_one
8234   \c_two
8235   \c_three
8236   \c_four
8237   \c_five
8238   \c_six

```

```

8239 \c_seven
8240 \c_eight
8241 \c_nine
8242 \c_ten
8243 \c_eleven
8244 \c_twelve
8245 \c_thirteen
8246 \c_fourteen
8247 \c_fifteen
8248 }
8249 \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.

```

8250 \prop_new:N \g_iow_streams_prop
8251 \prop_new:N \g_ior_streams_prop
8252 <*package>
8253 \prop_put:Nnn \g_iow_streams_prop { 0 } { LaTeX2e~reserved }
8254 \prop_put:Nnn \g_iow_streams_prop { 1 } { LaTeX2e~reserved }
8255 \prop_put:Nnn \g_iow_streams_prop { 2 } { LaTeX2e~reserved }
8256 \prop_put:Nnn \g_ior_streams_prop { 0 } { LaTeX2e~reserved }
8257 </package>
      (End definition for \g_iow_streams_prop and \g_ior_streams_prop. These functions are docu-
mented on page ??.)

```

`\l_ior_internal_tl` For expanding file names.

```

\l_iow_internal_tl
8258 \tl_new:N \l_ior_internal_tl
8259 \tl_new:N \l_iow_internal_tl
      (End definition for \l_ior_internal_tl and \l_iow_internal_tl. 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.

```

8260 \int_new:N \l_iow_stream_int
8261 \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
8262 <*initex>
8263 \alloc_setup_type:nnn { ior } \c_zero \c_sixteen
8264 \cs_new_protected:Npn \ior_raw_new:N #1

```

```

8265 { \alloc_reg:nNN { ior } \tex_chardef:D #1 }
8266 \alloc_setup_type:nnn { iow } \c_zero \c_sixteen
8267 \cs_new_protected:Npn \iow_raw_new:N #1
8268 { \alloc_reg:nNN { iow } \tex_chardef:D #1 }
8269 \end{document}
8270 \end{document}
8271 \cs_set_eq:NN \iow_raw_new:N \newwrite
8272 \cs_set_eq:NN \ior_raw_new:N \newread
8273 \end{document}
8274 \cs_generate_variant:Nn \ior_raw_new:N { c }
8275 \cs_generate_variant:Nn \iow_raw_new:N { c }
      (End definition for \ior_raw_new:N and \iow_raw_new:N. 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 8276 \cs_new_protected:Npn \ior_new:N #1 { \cs_new_eq:NN #1 \c_term_ior }
\iow_new:N 8277 \cs_generate_variant:Nn \ior_new:N { c }
\iow_new:c 8278 \cs_new_protected:Npn \iow_new:N #1 { \cs_new_eq:NN #1 \c_term_iow }
            8279 \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. Before any actual file operations there is a

`\ior_open_unsafe:Nn` precaution against special characters in file names.

```

\iow_open_unsafe:Nn 8280 \cs_new_protected:Npn \ior_open:Nn #1#2
                    8281 {
                    8282   \group_begin:
                    8283     \tl_to_str_active_safe:Nx \l_ior_internal_tl {#2}
                    8284     \tl_if_in:NnT \l_ior_internal_tl { ~ }
                    8285     {
                    8286       \msg_kernel_error:nxx { io } { space-in-file-name }
                    8287       { \l_ior_internal_tl }
                    8288     }
                    8289     \exp_args:NNNo \group_end:
                    8290     \ior_open_unsafe:Nn #1 \l_ior_internal_tl
                    8291   }
                    8292   \cs_generate_variant:Nn \ior_open:Nn { c }
                    8293   \cs_new_protected:Npn \iow_open:Nn #1#2
                    8294   {
                    8295     \group_begin:
                    8296       \tl_to_str_active_safe:Nx \l_iow_internal_tl {#2}
                    8297       \tl_if_in:NnT \l_iow_internal_tl { ~ }
                    8298       {
                    8299         \msg_kernel_error:nxx { io } { space-in-file-name }
                    8300         { \l_iow_internal_tl }
                    8301       }
                    8302       \exp_args:NNNo \group_end:

```

```

8303     \ior_open_unsafe:Nn #1 \l_ior_internal_tl
8304   }
8305   \cs_generate_variant:Nn \ior_open:Nn { c }
8306   \cs_new_protected:Npn \ior_open_unsafe:Nn #1#2
8307   {
8308     \ior_close:N #1
8309     \int_set:Nn \l_ior_stream_int \c_sixteen
8310     \tl_map_function:NN \c_ior_streams_tl \ior_alloc_read:n
8311     \int_compare:nNnTF \l_ior_stream_int = \c_sixteen
8312     { \msg_kernel_fatal:nn { ior } { streams-exhausted } }
8313     {
8314       \ior_stream_alloc:N #1
8315       \prop_gput:NVn \g_ior_streams_prop \l_ior_stream_int {#2}
8316       \tex_openin:D #1#2 \scan_stop:
8317     }
8318   }
8319   \cs_new_protected:Npn \ior_open_unsafe:Nn #1#2
8320   {
8321     \ior_close:N #1
8322     \int_set:Nn \l_ior_stream_int \c_sixteen
8323     \tl_map_function:NN \c_ior_streams_tl \ior_alloc_write:n
8324     \int_compare:nNnTF \l_ior_stream_int = \c_sixteen
8325     { \msg_kernel_fatal:nn { ior } { streams-exhausted } }
8326     {
8327       \ior_stream_alloc:N #1
8328       \prop_gput:NVn \g_ior_streams_prop \l_ior_stream_int {#2}
8329       \tex_immediate:D \tex_openout:D #1#2 \scan_stop:
8330     }
8331   }

```

(End definition for `\ior_open:Nn` and others. These functions are documented on page 141.)

`\ior_alloc_read:n` These functions are used to see if a particular stream is available. The property list
`\ior_alloc_write:n` contains file names for streams in use, so any unused ones are for the taking.

```

8332   \cs_new_protected:Npn \ior_alloc_write:n #1
8333   {
8334     \prop_if_in:NnF \g_ior_streams_prop {#1}
8335     {
8336       \int_set:Nn \l_ior_stream_int {#1}
8337       \tl_map_break:
8338     }
8339   }
8340   \cs_new_protected:Npn \ior_alloc_read:n #1
8341   {
8342     \prop_if_in:NnF \g_ior_streams_prop {#1}
8343     {
8344       \int_set:Nn \l_ior_stream_int {#1}
8345       \tl_map_break:
8346     }
8347   }

```

(End definition for \ior_alloc_read:n. This function is documented on page ??.)

\iow_stream_alloc:N \ior_stream_alloc:N \iow_stream_alloc_aux: \ior_stream_alloc_aux: \g_iow_internal_iow \g_ior_internal_ior	Allocating a raw stream is much easier in IniTeX mode than for the package. For the format, all streams will be allocated by l3io and so there is a simple check to see if a raw stream is actually available. On the other hand, for the package there will be non-managed streams. So if the managed one is not open, a check is made to see if some other managed stream is available before deciding to open a new one. If a new one is needed, we get the number allocated by L ^A T _E X 2 _ε to get “back on track” with allocation.
--	---

```

8348 \iow_new:N \g_iow_internal_iow
8349 \ior_new:N \g_ior_internal_ior
8350 \cs_new_protected:Npn \iow_stream_alloc:N #1
8351 {
8352   \cs_if_exist:cTF { g_iow_ \int_use:N \l_iow_stream_int _iow }
8353   { \cs_gset_eq:Nc #1 { g_iow_ \int_use:N \l_iow_stream_int _iow } }
8354   {
8355     \*package
8356     \iow_stream_alloc_aux:
8357     \int_compare:nNnT \l_iow_stream_int = \c_sixteen
8358     {
8359       \iow_raw_new:N \g_iow_internal_iow
8360       \int_set:Nn \l_iow_stream_int { \g_iow_internal_iow }
8361       \cs_gset_eq:Nc
8362       { g_iow_ \int_use:N \l_iow_stream_int _iow } \g_iow_internal_iow
8363     }
8364   }
8365   \*initex
8366   \iow_raw_new:c { g_iow_ \int_use:N \l_iow_stream_int _iow }
8367   \*initex
8368   \cs_gset_eq:Nc #1 { g_iow_ \int_use:N \l_iow_stream_int _iow }
8369   }
8370 }
8371 \*package
8372 \cs_new_protected_nopar:Npn \iow_stream_alloc_aux:
8373 {
8374   \int_incr:N \l_iow_stream_int
8375   \int_compare:nNnT \l_iow_stream_int < \c_sixteen
8376   {
8377     \cs_if_exist:cTF { g_iow_ \int_use:N \l_iow_stream_int _iow }
8378     {
8379       \prop_if_in:NVT \g_iow_streams_prop \l_iow_stream_int
8380       { \iow_stream_alloc_aux: }
8381     }
8382     { \iow_stream_alloc_aux: }
8383   }
8384 }
8385 \*package
8386 \cs_new_protected:Npn \ior_stream_alloc:N #1
8387 {
8388   \cs_if_exist:cTF { g_ior_ \int_use:N \l_ior_stream_int _ior }

```

```

8389     { \cs_gset_eq:Nc #1 { g_ior_ \int_use:N \l_ior_stream_int _ior } }
8390     {
8391     (*package)
8392     \ior_stream_alloc_aux:
8393     \int_compare:nNnT \l_ior_stream_int = \c_sixteen
8394     {
8395     \ior_raw_new:N \g_ior_internal_ior
8396     \int_set:Nn \l_ior_stream_int { \g_ior_internal_ior }
8397     \cs_gset_eq:cN
8398     { g_ior_ \int_use:N \l_ior_stream_int _ior } \g_ior_internal_ior
8399     }
8400     </package>
8401     (*initex)
8402     \ior_raw_new:c { g_ior_ \int_use:N \l_ior_stream_int _ior }
8403     </initex>
8404     \cs_gset_eq:Nc #1 { g_ior_ \int_use:N \l_ior_stream_int _ior }
8405     }
8406     }
8407     (*package)
8408     \cs_new_protected_nopar:Npn \ior_stream_alloc_aux:
8409     {
8410     \int_incr:N \l_ior_stream_int
8411     \int_compare:nNnT \l_ior_stream_int < \c_sixteen
8412     {
8413     \cs_if_exist:cTF { g_ior_ \int_use:N \l_ior_stream_int _ior }
8414     {
8415     \prop_if_in:NVT \g_ior_streams_prop \l_ior_stream_int
8416     { \ior_stream_alloc_aux: }
8417     }
8418     { \ior_stream_alloc_aux: }
8419     }
8420     }
8421     </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
8422 \cs_new_protected:Npn \ior_close:N #1
8423 {
8424   \cs_if_exist:NT #1
8425   {
8426     \int_compare:nNnF #1 = \c_minus_one
8427     {
8428       \int_compare:nNnF #1 = \c_sixteen
8429       { \tex_closein:D #1 }
8430       \prop_gdel:NV \g_ior_streams_prop #1
8431       \cs_gset_eq:NN #1 \c_term_ior
8432     }

```



```

8433     }
8434   }
8435   \cs_new_protected:Npn \iow_close:N #1
8436   {
8437     \cs_if_exist:NT #1
8438     {
8439       \int_compare:nNnF #1 = \c_minus_one
8440       {
8441         \int_compare:nNnF #1 = \c_sixteen
8442         { \tex_closein:D #1 }
8443         \prop_gdel:NV \g_iow_streams_prop #1
8444         \cs_gset_eq:NN #1 \c_term_iow
8445       }
8446     }
8447   }
8448   \cs_generate_variant:Nn \ior_close:N { c }
8449   \cs_generate_variant:Nn \iow_close:N { c }

```

(End definition for `\iow_close:N` and `\iow_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`.

```

8450   \cs_new_protected_nopar:Npn \ior_list_streams:
8451   { \ior_list_streams_aux:Nn \g_ior_streams_prop { ior } }
8452   \cs_new_protected_nopar:Npn \iow_list_streams:
8453   { \ior_list_streams_aux:Nn \g_iow_streams_prop { iow } }
8454   \cs_new_protected:Npn \ior_list_streams_aux:Nn #1#2
8455   {
8456     \msg_aux_use:nn { LaTeX / #2 }
8457     { \prop_if_empty:NTF #1 { show-no-stream } { show-open-streams } }
8458     \msg_aux_show:x
8459     { \prop_map_function:NN #1 \msg_aux_show_unbraced:nn }
8460   }

```

(End definition for `\ior_list_streams:`. This function is documented on page ??.)

Text for the error messages.

```

8461   \msg_kernel_new:nnnn { iow } { streams-exhausted }
8462   { Output~streams~exhausted }
8463   {
8464     TeX~can~only~open~up~to~16~output~streams~at~one~time.\\
8465     All~16 are currently~in~use,~and~something~wanted~to~open
8466     another~one.
8467   }
8468   \msg_kernel_new:nnnn { ior } { streams-exhausted }
8469   { Input~streams~exhausted }
8470   {
8471     TeX~can~only~open~up~to~16~input~streams~at~one~time.\\
8472     All~16 are currently~in~use,~and~something~wanted~to~open

```

```

8473     another~one.
8474 }

```

199.4 Deferred writing

`\iow_shipout_x:Nn` First the easy part, this is the primitive.

```

\iow_shipout_x:Nx 8475 \cs_new_eq:NN \iow_shipout_x:Nn \tex_write:D
8476 \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 8477 \cs_new_protected:Npn \iow_shipout:Nn #1#2
8478 { \iow_shipout_x:Nn #1 { \exp_not:n {#2} } }
8479 \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.

```

8480 \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.

```

8481 \cs_new_protected:Npn \iow_now:Nn #1#2
8482 { \iow_now:Nx #1 { \exp_not:n {#2} } }

```

(End definition for `\iow_now:Nn`. This function is documented on page 137.)

`\iow_log:n` Writing to the log and the terminal directly are relatively easy.

```

\iow_log:x 8483 \cs_set_protected_nopar:Npn \iow_log:x { \iow_now:Nx \c_log_iow }
\iow_term:n 8484 \cs_new_protected_nopar:Npn \iow_log:n { \iow_now:Nn \c_log_iow }
\iow_term:x 8485 \cs_set_protected_nopar:Npn \iow_term:x { \iow_now:Nx \c_term_iow }
8486 \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 8487 \cs_new_protected:Npn \iow_now_when_avail:Nn #1
8488 { \cs_if_free:NTF #1 { \use_none:n } { \iow_now:Nn #1 } }
8489 \cs_new_protected:Npn \iow_now_when_avail:Nx #1
8490 { \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

```
8491 \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.

```
8492 \cs_new_eq:NN \iow_char:N \cs_to_str:N  
(End definition for \iow_char:N. This function is documented on page 138.)
```

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.

`\l_iow_line_length_int` 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`.

```
8493 \int_new:N \l_iow_line_length_int  
8494 \int_set:Nn \l_iow_line_length_int { 78 }  
(End definition for \l_iow_line_length_int. This function is documented on page 139.)
```

`\l_iow_target_length_int` This stores the target line length: the full length minus any part for a leader at the start of each line.

```
8495 \int_new:N \l_iow_target_length_int  
(End definition for \l_iow_target_length_int. This function is documented on page ??.)
```

`\l_iow_current_line_int` These store the number of characters in the line and word currently being constructed, and the current indentation, respectively.

```
\l_iow_current_indentation_int  
8496 \int_new:N \l_iow_current_line_int  
8497 \int_new:N \l_iow_current_word_int  
8498 \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 ??.)
```

`\l_iow_current_line_tl` These hold the current line of text and current word, and a number of spaces for indentation, respectively.

```
\l_iow_current_word_tl  
\l_iow_current_indentation_tl  
8499 \tl_new:N \l_iow_current_line_tl  
8500 \tl_new:N \l_iow_current_word_tl  
8501 \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 ??.)
```

`\l_iow_wrap_tl` Used for the expansion step before detokenizing.

```
8502 \tl_new:N \l_iow_wrap_tl  
(End definition for \l_iow_wrap_tl. This function is documented on page ??.)
```

`\l_iow_wrapped_tl` The output from wrapping text: fully expanded and with lines which are not overly long.

```
8503 \tl_new:N \l_iow_wrapped_tl
      (End definition for \l_iow_wrapped_tl. This function is documented on page ??.)
```

`\l_iow_line_start_bool` Boolean to avoid adding a space at the beginning of forced newlines.

```
8504 \bool_new:N \l_iow_line_start_bool
      (End definition for \l_iow_line_start_bool. This function is documented on page ??.)
```

`\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.

```
8505 \group_begin:
8506   \char_set_catcode_other:N \*
8507   \char_set_lccode:nn {'\*} {'\ }
8508   \tl_to_lowercase:n { \tl_const:Nn \c_catcode_other_space_tl { * } }
8509 \group_end:
      (End definition for \c_catcode_other_space_tl. This function is documented on page 139.)
```

`\c_iow_wrap_marker_tl` Every special action of the wrapping code is preceded 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-
`\c_iow_wrap_newline_marker_tl` delimited argument to know what operation to perform. The setting of `\escapechar` here
`\c_iow_wrap_indent_marker_tl` is not very important, but makes `\c_iow_wrap_marker_tl` look nicer. Note that `\iow_-`
`\c_iow_wrap_unindent_marker_tl` `wrap_new_marker:n` does not survive the group, but all constants are defined globally.
`\iow_wrap_new_marker:n`

```
8510 \group_begin:
8511   \int_set_eq:NN \tex_escapechar:D \c_minus_one
8512   \tl_const:Nx \c_iow_wrap_marker_tl
8513   { \tl_to_str:n { \^^I \^^O \^^W \^^_ \^^W \^^R \^^A \^^P } }
8514   \cs_set:Npn \iow_wrap_new_marker:n #1
8515   {
8516     \tl_const:cx { c_iow_wrap_ #1 _marker_tl }
8517     {
8518       \c_catcode_other_space_tl
8519       \c_iow_wrap_marker_tl
8520       \c_catcode_other_space_tl
8521       #1
8522       \c_catcode_other_space_tl
8523     }
8524   }
8525   \iow_wrap_new_marker:n { end }
8526   \iow_wrap_new_marker:n { newline }
8527   \iow_wrap_new_marker:n { indent }
8528   \iow_wrap_new_marker:n { unindent }
8529 \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.

```

8530 \cs_new_protected:Npn \iow_indent:n #1 { }
8531 \cs_new:Npx \iow_indent_expandable:n #1
8532 {
8533   \c_iow_wrap_indent_marker_tl
8534   #1
8535   \c_iow_wrap_unindent_marker_tl
8536 }

```

(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.

```

8537 \cs_new_protected:Npn \iow_wrap:xnnnN #1#2#3#4#5
8538 {
8539   \group_begin:
8540   \int_set:Nn \l_iow_target_length_int { \l_iow_line_length_int - ( #3 ) }
8541   \int_zero:N \l_iow_current_indentation_int
8542   \tl_clear:N \l_iow_current_indentation_tl
8543   \int_zero:N \l_iow_current_line_int
8544   \tl_clear:N \l_iow_current_line_tl
8545   \tl_clear:N \l_iow_wrap_tl
8546   \bool_set_true:N \l_iow_line_start_bool
8547   \int_set_eq:NN \tex_escapechar:D \c_minus_one
8548   \cs_set_nopar:Npx \{ { \token_to_str:N \{ }
8549   \cs_set_nopar:Npx \# { \token_to_str:N \# }
8550   \cs_set_nopar:Npx \} { \token_to_str:N \} }
8551   \cs_set_nopar:Npx \% { \token_to_str:N \% }
8552   \cs_set_nopar:Npx \~ { \token_to_str:N \~ }
8553   \int_set:Nn \tex_escapechar:D { 92 }
8554   \cs_set_eq:NN \_ \c_iow_wrap_newline_marker_tl
8555   \cs_set_eq:NN \ \c_catcode_other_space_tl
8556   \cs_set_eq:NN \iow_indent:n \iow_indent_expandable:n
8557   #4
8558   <*initex>
8559   \tl_set:Nx \l_iow_wrap_tl {#1}
8560   </initex>
8561   <*package>
8562   \protected@edef \l_iow_wrap_tl {#1}
8563   </package>
8564   \cs_set:Npn \_ { \iow_newline: #2 }
8565   \use:x
8566   {
8567     \iow_wrap_loop:w
8568     \tl_to_str:N \l_iow_wrap_tl

```

```

8569         \tl_to_str:N \c_iow_wrap_end_marker_tl
8570         \c_space_tl \c_space_tl
8571         \exp_not:N \q_stop
8572     }
8573     \exp_args:NNo \group_end:
8574     #5 \l_iow_wrapped_tl
8575 }

```

(End definition for \iow_wrap:xnnnN. This function is documented on page [139](#).)

\iow_wrap_loop:w The loop grabs one word in the input, and checks whether it is the special marker, or a normal word.

```

8576 \cs_new_protected:Npn \iow_wrap_loop:w #1 ~ %
8577 {
8578     \tl_set:Nn \l_iow_current_word_tl {#1}
8579     \tl_if_eq:NNTF \l_iow_current_word_tl \c_iow_wrap_marker_tl
8580     { \iow_wrap_special:w }
8581     { \iow_wrap_word: }
8582 }

```

(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.

```

8583 \cs_new_protected_nopar:Npn \iow_wrap_word:
8584 {
8585     \int_set:Nn \l_iow_current_word_int
8586     { \str_length_skip_spaces:N \l_iow_current_word_tl }
8587     \int_add:Nn \l_iow_current_line_int { \l_iow_current_word_int }
8588     \int_compare:nNnTF \l_iow_current_line_int < \l_iow_target_length_int
8589     { \iow_wrap_word_fits: }
8590     { \iow_wrap_word_newline: }
8591     \iow_wrap_loop:w
8592 }
8593 \cs_new_protected_nopar:Npn \iow_wrap_word_fits:
8594 {
8595     \bool_if:NNTF \l_iow_line_start_bool
8596     {
8597         \bool_set_false:N \l_iow_line_start_bool
8598         \tl_put_right:Nx \l_iow_current_line_tl
8599         { \l_iow_current_indentation_tl \l_iow_current_word_tl }
8600         \int_add:Nn \l_iow_current_line_int
8601         { \l_iow_current_indentation_int }
8602     }
8603     {
8604         \tl_put_right:Nx \l_iow_current_line_tl
8605         { ~ \l_iow_current_word_tl }
8606         \int_incr:N \l_iow_current_line_int

```

```

8607     }
8608   }
8609   \cs_new_protected_nopar:Npn \iow_wrap_word_newline:
8610   {
8611     \tl_put_right:Nx \l_iow_wrapped_tl
8612     { \l_iow_current_line_tl \\\ }
8613     \int_set:Nn \l_iow_current_line_int
8614     {
8615       \l_iow_current_word_int
8616       + \l_iow_current_indentation_int
8617     }
8618     \tl_set:Nx \l_iow_current_line_tl
8619     { \l_iow_current_indentation_tl \l_iow_current_word_tl }
8620   }

```

(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
`\iow_wrap_indent:w` word is empty. To indent more, add four spaces to the start of the indentation token list.
`\iow_wrap_unindent:w` To reduce indentation, rebuild the indentation token list using `\prg_replicate:nn`. At
`\iow_wrap_end:w` the end, we simply save the last line (without the run-on text), and prevent the loop.

```

8621 \cs_new_protected:Npn \iow_wrap_special:w #1 ~ #2 ~ #3 ~ %
8622 {
8623   \use:c { iow_wrap_#1: }
8624   \str_if_eq:xxTF { #2~#3 } { ~ \c_iow_wrap_marker_tl }
8625   { \iow_wrap_special:w }
8626   { \iow_wrap_loop:w #2 ~ #3 ~ }
8627 }
8628 \cs_new_protected_nopar:Npn \iow_wrap_newline:
8629 {
8630   \tl_put_right:Nx \l_iow_wrapped_tl
8631   { \l_iow_current_line_tl \\\ }
8632   \int_zero:N \l_iow_current_line_int
8633   \tl_clear:N \l_iow_current_line_tl
8634   \bool_set_true:N \l_iow_line_start_bool
8635 }
8636 \cs_new_protected_nopar:Npx \iow_wrap_indent:
8637 {
8638   \int_add:Nn \l_iow_current_indentation_int \c_four
8639   \tl_put_right:Nx \exp_not:N \l_iow_current_indentation_tl
8640   { \c_space_tl \c_space_tl \c_space_tl \c_space_tl }
8641 }
8642 \cs_new_protected_nopar:Npn \iow_wrap_unindent:
8643 {
8644   \int_sub:Nn \l_iow_current_indentation_int \c_four
8645   \tl_set:Nx \l_iow_current_indentation_tl
8646   { \prg_replicate:nn \l_iow_current_indentation_int { ~ } }

```

```

8647 }
8648 \cs_new_protected_nopar:Npn \iow_wrap_end:
8649 {
8650   \tl_put_right:Nx \l_iow_wrapped_tl
8651     { \l_iow_current_line_tl }
8652   \use_none_delimit_by_q_stop:w
8653 }

```

(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.

```

8654 \cs_new_nopar:Npn \str_length_skip_spaces:N
8655 { \exp_args:No \str_length_skip_spaces:n }
8656 \cs_new:Npn \str_length_skip_spaces:n #1
8657 {
8658   \int_value:w \int_eval:w
8659     \exp_after:wN \str_length_loop:NNNNNNNNN \tl_to_str:n {#1}
8660     {X8}{X7}{X6}{X5}{X4}{X3}{X2}{X1}{X0} \q_stop
8661   \int_eval_end:
8662 }
8663 \cs_new:Npn \str_length_loop:NNNNNNNNN #1#2#3#4#5#6#7#8#9
8664 {
8665   \if_catcode:w X #9
8666     \exp_after:wN \use_none_delimit_by_q_stop:w
8667   \else:
8668     9 +
8669     \exp_after:wN \str_length_loop:NNNNNNNNN
8670   \fi:
8671 }

```

(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.

```

8672 \prg_new_conditional:Nnn \ior_if_eof:N { p , T , F , TF }
8673 {
8674   \cs_if_exist:NTF #1
8675   {
8676     \if_int_compare:w #1 = \c_sixteen
8677       \prg_return_true:
8678     \else:
8679       \if_eof:w #1
8680         \prg_return_true:
8681       \else:
8682         \prg_return_false:
8683       \fi:
8684     \fi:

```



```

8685     }
8686     { \prg_return_true: }
8687 }

```

(End definition for \ior_if_eof_p:N. This function is documented on page 140.)

\ior_to:NN And here we read from files.

```

\ior_gto:NN 8688 \cs_new_protected:Npn \ior_to:NN #1#2
8689 { \tex_read:D #1 to #2 }
8690 \cs_new_protected:Npn \ior_gto:NN #1#2
8691 { \tex_global:D \tex_read:D #1 to #2 }

```

(End definition for \ior_to:NN. This function is documented on page 140.)

\ior_str_to:NN Reading as strings is also a primitive wrapper.

```

\ior_str_gto:NN 8692 \cs_new_protected:Npn \ior_str_to:NN #1#2
8693 { \etex_readline:D #1 to #2 }
8694 \cs_new_protected:Npn \ior_str_gto:NN #1#2
8695 { \tex_global:D \etex_readline:D #1 to #2 }

```

(End definition for \ior_str_to:NN. This function is documented on page 140.)

199.9 Experimental functions

\ior_map_inline:nn The five arguments of \ior_str_map_inline_aux:NNNnn are: the read stream used at this level of map nesting; the control sequence which will hold the function to map; either \ior_to:NN or \ior_str_to:NN; the file name; and finally the inline function. We make sure to decrease \g_prg_map_int and close the file after the mapping, even if the mapping exits prematurely because of a \prg_map_break: instruction. The \ior_if_eof:nF tests are a little bit tricky to get right: if the file does not exist, we should do nothing, hence the preliminary test. Then the end-of-file test must be done after reading each line.

```

8696 \cs_new_protected_nopar:Npn \ior_map_inline:nn
8697 { \ior_map_inline_aux:Nnn \ior_to:NN }
8698 \cs_new_protected_nopar:Npn \ior_str_map_inline:nn
8699 { \ior_map_inline_aux:Nnn \ior_str_to:NN }
8700 \cs_new_protected_nopar:Npn \ior_map_inline_aux:Nnn
8701 {
8702   \exp_args:Ncc \ior_map_inline_aux:NNNnn
8703   { g_ior_map_ \int_use:N \g_prg_map_int _ior }
8704   { ior_map_ \int_use:N \g_prg_map_int :n }
8705 }
8706 \cs_new_protected:Npn \ior_map_inline_aux:NNNnn #1#2#3#4#5
8707 {
8708   \cs_if_exist:NF #1 { \ior_new:N #1 }
8709   \cs_set:Npn #2 ##1 {#5}
8710   \ior_open:Nn #1 {#4}
8711   \int_gincr:N \g_prg_map_int
8712   \ior_if_eof:NF #1 { \ior_map_inline_loop:NNN #1#2#3 }
8713   \prg_break_point:n
8714   {

```

```

8715         \int_gdecr:N \g_prg_map_int
8716         \ior_close:N #1
8717     }
8718 }
8719 \cs_new_protected:Npn \ior_map_inline_loop:NNN #1#2#3
8720 {
8721     #3 #1 \l_ior_internal_tl
8722     \ior_if_eof:NF #1
8723     {
8724         \exp_args:No #2 \l_ior_internal_tl
8725         \ior_map_inline_loop:NNN #1#2#3
8726     }
8727 }

```

(End definition for \ior_map_inline:nn. This function is documented on page ??.)

199.10 Messages

```

8728 \msg_kernel_new:nnnn { io } { space-in-file-name }
8729 { Space~in~file~name~'~#1'. }
8730 {
8731     Spaces~are~not~permitted~in~files~loaded~by~LaTeX: \\
8732     Further~errors~may~follow!
8733 }

```

199.11 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, so a bit of a hack is needed.

```

8734 <*/deprecated>
8735 \cs_new_protected:Npn \iow_now_buffer_safe:Nn #1#2
8736 { \iow_wrap:xnnnN { \exp_not:n {#2} } { } \c_zero { } \iow_now:Nn #1 }
8737 \cs_new_protected:Npn \iow_now_buffer_safe:Nx #1#2
8738 { \iow_wrap:xnnnN {#2} { } \c_zero { } \iow_now:Nn #1 }
8739 </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:
8740 <*/deprecated>
8741 \cs_new_eq:NN \ior_open_streams: \ior_list_streams:
8742 \cs_new_eq:NN \iow_open_streams: \iow_list_streams:
8743 </deprecated>

```

(End definition for \ior_open_streams:. This function is documented on page ??.)

```

8744 </initex | package>

```

200 l3msg implementation

```

8745 <*initex | package>
8746 <*package>
8747 \ProvidesExplPackage
8748   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
8749 \package_check_loaded_expl:
8750 </package>

\l_msg_internal_tl A general scratch for the module.

8751 \tl_new:N \l_msg_internal_tl
      (End definition for \l_msg_internal_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

8752 \tl_const:Nn \c_msg_text_prefix_tl { msg~text~>~ }
8753 \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 8754 \cs_new_protected:Npn \msg_new:nnnn #1#2
\msg_gset:nnn   8755 {
\msg_set:nnnn   8756   \cs_if_exist:cT { \c_msg_text_prefix_tl #1 / #2 }
\msg_set:nnn    8757   {
8758     \msg_kernel_error:nnxx { msg } { message-already-defined }
8759     {#1} {#2}
8760   }
8761   \msg_gset:nnnn {#1} {#2}
8762 }
8763 \cs_new_protected:Npn \msg_new:nnn #1#2#3
8764 { \msg_new:nnnn {#1} {#2} {#3} { } }
8765 \cs_new_protected:Npn \msg_set:nnnn #1#2#3#4
8766 {
8767   \cs_set:cpn { \c_msg_text_prefix_tl #1 / #2 }
8768   ##1##2##3##4 {#3}
8769   \cs_set:cpn { \c_msg_more_text_prefix_tl #1 / #2 }
8770   ##1##2##3##4 {#4}
8771 }
8772 \cs_new_protected:Npn \msg_set:nnn #1#2#3
8773 { \msg_set:nnnn {#1} {#2} {#3} { } }
8774 \cs_new_protected:Npn \msg_gset:nnnn #1#2#3#4
8775 {

```

```

8776 \cs_gset:cpn { \c_msg_text_prefix_tl #1 / #2 }
8777   ##1##2##3##4 {#3}
8778 \cs_gset:cpn { \c_msg_more_text_prefix_tl #1 / #2 }
8779   ##1##2##3##4 {#4}
8780 }
8781 \cs_new_protected:Npn \msg_gset:nnn #1#2#3
8782 { \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      8783 \tl_const:Nn \c_msg_coding_error_text_tl
\c_msg_critical_text_tl      8784 {
\c_msg_fatal_text_tl         8785   This-is-a-coding-error.
\c_msg_help_text_tl          8786   \\\
\c_msg_no_info_text_tl       8787 }
\c_msg_on_line_text_tl       8788 \tl_const:Nn \c_msg_continue_text_tl
\c_msg_return_text_tl        8789 { Type-<return>-to-continue }
\c_msg_trouble_text_tl       8790 \tl_const:Nn \c_msg_critical_text_tl
                               8791 { Reading-the-current-file-will-stop }
                               8792 \tl_const:Nn \c_msg_fatal_text_tl
                               8793 { This-is-a-fatal-error:-LaTeX-will-abort }
                               8794 \tl_const:Nn \c_msg_help_text_tl
                               8795 { For-immediate-help-type-H-<return> }
                               8796 \tl_const:Nn \c_msg_no_info_text_tl
                               8797 {
                               8798   LaTeX-does-not-know-anything-more-about-this-error,~sorry.
                               8799   \c_msg_return_text_tl
                               8800 }
                               8801 \tl_const:Nn \c_msg_on_line_text_tl { on-line }
                               8802 \tl_const:Nn \c_msg_return_text_tl
                               8803 {
                               8804   \\\
                               8805   Try~typing~<return>-to-proceed.
                               8806   \\\
                               8807   If~that~doesn't~work,~type~X-<return>-to-quit.
                               8808 }
                               8809 \tl_const:Nn \c_msg_trouble_text_tl
                               8810 {
                               8811   \\\
                               8812   More-errors~will~almost~certainly~follow: \\\
                               8813   the~LaTeX-run~should~be~aborted.
                               8814 }

```

(End definition for \c_msg_coding_error_text_tl and others. These functions are documented on page 144.)

\msg_newline: New lines are printed in the same way as for low-level file writing.
\msg_two_newlines:

```

8815 \cs_new_nopar:Npn \msg_newline: { ^^J }
8816 \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:
8817 \cs_new_nopar:Npn \msg_line_number: { \int_use:N \tex_inputlineno:D }
8818 \cs_set_nopar:Npn \msg_line_context:
8819 {
8820   \c_msg_on_line_text_tl
8821   \c_space_tl
8822   \msg_line_number:
8823 }
      (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.

```

8824 \char_set_catcode_letter:N \.
8825 \tl_new:N
8826 \c_msg_hide_tl.....
8827 \tl_const:Nn \c_msg_hide_tl
8828 { \c_msg_hide_tl..... }
8829 \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.

```

8830 \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
`\msg_interrupt_no_details:xx` create a message which hides all of T_EX’s own information by filling the output up with
`\msg_interrupt_details:xxx` dots. To achieve this, dots have to be letters. The odd `\c_msg_hide_tl<dots>` actually
`\msg_interrupt_text:n` does the hiding: it is the large run of dots in the name that is important here. The
`\msg_interrupt_more_text:n` meaning of `\` is altered so that the explanation text is a simple run whilst the initial
`\msg_interrupt_aux:` error has line-continuation shown.

```

8831 \cs_new_protected:Npn \msg_interrupt:xxx #1#2#3
8832 {
8833   \group_begin:
8834   \tl_if_empty:nTF {#3}
8835     { \msg_interrupt_no_details:xx {#1} {#2} }
8836     { \msg_interrupt_details:xxx {#1} {#2} {#3} }
8837   \msg_interrupt_aux:
8838   \group_end:
8839 }

```

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.

```

8840 \cs_new_protected:Npn \msg_interrupt_no_details:xx #1#2
8841 {
8842   \iow_wrap:xnnnN
8843     { \c_msg_no_info_text_tl }
8844     { |~ } { 2 } { } \msg_interrupt_more_text:n
8845   \iow_wrap:xnnnN { #1 \c_msg_continue_text_tl }
8846     { ! ~ } { 2 } { } \msg_interrupt_text:n
8847 }
8848 \cs_new_protected:Npn \msg_interrupt_details:xxx #1#2#3
8849 {
8850   \iow_wrap:xnnnN
8851     { \c_msg_help_text_tl }
8852     { |~ } { 2 } { } \msg_interrupt_more_text:n
8853   \iow_wrap:xnnnN { #1 \c_msg_help_text_tl }
8854     { ! ~ } { 2 } { } \msg_interrupt_text:n
8855 }
8856 \cs_new_protected:Npn \msg_interrupt_text:n #1
8857 { \tl_set:Nn \l_msg_text_tl {#1} }
8858 \cs_new_protected:Npn \msg_interrupt_more_text:n #1
8859 {
8860   <*initex>
8861     \tl_set:Nx \l_msg_internal_tl
8862   </initex>
8863   <*package>
8864     \protected@edef \l_msg_internal_tl
8865   </package>
8866   {
8867     |,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
8868     #1
8869     \msg_newline:
8870     |.....
8871   }
8872   \tex_errhelp:D \exp_after:wN { \l_msg_internal_tl }
8873 }

```

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.

```

8874 \group_begin: % {
8875   \char_set_lccode:nn {'\} } {'\ }
8876   \char_set_lccode:nn {'\& } {'!\}
8877   \char_set_catcode_active:N \&
8878   \tl_to_lowercase:n

```

```

8879 {
8880   \group_end:
8881   \cs_new_protected:Npn \msg_interrupt_aux:
8882     {
8883       \iow_term:x
8884       {
8885         \iow_newline:
8886         !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
8887         \iow_newline:
8888         !
8889       }
8890       \tl_put_right:No \l_msg_text_tl { \c_msg_hide_tl }
8891       \cs_set_protected_nopar:Npx &
8892       { \tex_errmessage:D { \exp_not:o { \l_msg_text_tl } } }
8893       &
8894     }
8895 }

```

(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.

```

8896 \cs_new_protected:Npn \msg_log:x #1
8897 {
8898   \iow_log:x { ..... }
8899   \iow_wrap:xnnnN { . ~ #1 } { . ~ } { 2 } { }
8900   \iow_log:x
8901   \iow_log:x { ..... }
8902 }
8903 \cs_new_protected:Npn \msg_term:x #1
8904 {
8905   \iow_term:x { ***** }
8906   \iow_wrap:xnnnN { * ~ #1 } { * ~ } { 2 } { }
8907   \iow_term:x
8908   \iow_term:x { ***** }
8909 }

```

(End definition for `\msg_log:x`. This function is documented on page 148.)

200.4 Displaying messages

L^AT_EX is handling error messages and so the T_EX ones are disabled.

```

8910 \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`
`\msg_error_text:n`
`\msg_warning_text:n`
`\msg_info_text:n`

```

8911 \cs_new:Npn \msg_fatal_text:n #1 { Fatal~#1~error }
8912 \cs_new:Npn \msg_critical_text:n #1 { Critical~#1~error }
8913 \cs_new:Npn \msg_error_text:n #1 { #1~error }
8914 \cs_new:Npn \msg_warning_text:n #1 { #1~warning }
8915 \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 145.)

`\msg_see_documentation_text:n` Contextual footer information.

```

8916 \cs_new:Npn \msg_see_documentation_text:n #1
8917 { \\\ \ 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 required.

```

8918 \prop_new:N \l_msg_redirect_classes_prop
8919 \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.

```

8920 \cs_new_protected:Npn \msg_class_set:nn #1#2
8921 {
8922   \prop_clear_new:c { l_msg_redirect_ #1 _prop }
8923   \cs_set_protected:cpn { msg_ #1 :nnxxxx } ##1##2##3##4##5##6
8924   { \msg_use:nnnnxxxx {#1} {#2} {##1} {##2} {##3} {##4} {##5} {##6} }
8925   \cs_set_protected:cpx { msg_ #1 :nnxxx } ##1##2##3##4##5
8926   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} {##4} {##5} { } }
8927   \cs_set_protected:cpx { msg_ #1 :nnxx } ##1##2##3##4
8928   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} {##4} { } { } }
8929   \cs_set_protected:cpx { msg_ #1 :nnx } ##1##2##3
8930   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} {##3} { } { } { } }
8931   \cs_set_protected:cpx { msg_ #1 :nn } ##1##2
8932   { \exp_not:c { msg_ #1 :nnxxxx } {##1} {##2} { } { } { } { } }
8933 }
      (End definition for \msg_class_set:nn. This function is documented on page 145.)

```

`\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
\msg_no_more_text:xxxx
8934 \prg_new_conditional:Npnn \msg_if_more_text:N #1 { p , T , F , TF }
8935 {
8936   \cs_if_eq:NNTF #1 \msg_no_more_text:xxxx
8937   { \prg_return_false: }
8938   { \prg_return_true: }
8939 }
8940 \cs_new:Npn \msg_no_more_text:xxxx #1#2#3#4 { }
8941 \cs_generate_variant:Nn \msg_if_more_text_p:N { c }
8942 \cs_generate_variant:Nn \msg_if_more_text:NT { c }
8943 \cs_generate_variant:Nn \msg_if_more_text:NF { c }
8944 \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 docu-
      mented on page ??.)

```

`\msg_fatal:nnxxxx` For fatal errors, after the error message T_EX bails out.

```

\msg_fatal:nnxxxx
\msg_fatal:nnxx
\msg_fatal:nnx
\msg_fatal:nn
8945 \msg_class_set:nn { fatal }
8946 {

```



```

8947 \msg_interrupt:xxx
8948 { \msg_fatal_text:n {#1} : ~ "#2" }
8949 {
8950 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8951 \msg_see_documentation_text:n {#1}
8952 }
8953 { \c_msg_fatal_text_tl }
8954 \tex_end:D
8955 }

```

(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:nnxxxx 8956 \msg_class_set:nn { critical }
\msg_critical:nnxxx 8957 {
\msg_critical:nnx 8958 \msg_interrupt:xxx
\msg_critical:nn 8959 { \msg_critical_text:n {#1} : ~ "#2" }
8960 {
8961 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8962 \msg_see_documentation_text:n {#1}
8963 }
8964 { \c_msg_critical_text_tl }
8965 \tex_endinput:D
8966 }

```

(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:nnxxxx 8967 \msg_class_set:nn { error }
\msg_error:nnxxx 8968 {
\msg_error:nnx 8969 \msg_if_more_text:cTF { \c_msg_more_text_prefix_tl #1 / #2 }
\msg_error:nn 8970 {
8971 \msg_interrupt:xxx
8972 { \msg_error_text:n {#1} : ~ "#2" }
8973 {
8974 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8975 \msg_see_documentation_text:n {#1}
8976 }
8977 { \use:c { \c_msg_more_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6} }
8978 }
8979 {
8980 \msg_interrupt:xxx
8981 { \msg_error_text:n {#1} : ~ "#2" }
8982 {
8983 \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
8984 \msg_see_documentation_text:n {#1}
8985 }
8986 { }
8987 }
8988 }

```

(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:nnxxxx 8989 \msg_class_set:nn { warning }
\msg_warning:nnxx    8990 {
\msg_warning:nnx     8991   \msg_term:x
\msg_warning:nn      8992   {
                        8993     \msg_warning_text:n {#1} : ~ "#2" \\ \\
                        8994     \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
                        8995   }
                        8996 }
```

(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 8997 \msg_class_set:nn { info }
\msg_info:nnxx   8998 {
\msg_info:nnx    8999   \msg_log:x
\msg_info:nn     9000   {
                        9001     \msg_info_text:n {#1} : ~ "#2" \\ \\
                        9002     \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6}
                        9003   }
                        9004 }
```

(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 9005 \msg_class_set:nn { log }
\msg_log:nnxx   9006 {
\msg_log:nnx    9007   \msg_log:x
\msg_log:nn     9008   { \use:c { \c_msg_text_prefix_tl #1 / #2 } {#3} {#4} {#5} {#6} }
                  9009 }
```

(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 9010 \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 9011 \seq_new:N \l_msg_redirect_classes_seq
\l_msg_current_class_tl 9012 \tl_new:N \l_msg_class_tl
\l_msg_current_module_tl 9013 \tl_new:N \l_msg_current_class_tl
9014 \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
`\msg_use_aux:nnn` for the message, and the second a loop function. There is then a hand-off to the system
`\msg_use_aux:nn` for checking if redirection is needed.
`\msg_use_loop_check:nn` 9015 `\cs_new_protected:Npn \msg_use:nnnnxxxx #1#2#3#4#5#6#7#8`
`\msg_use_code:` 9016 `{`
`\msg_use_loop:n` 9017 `\cs_set_protected_nopar:Npx \msg_use_code:`
`\msg_use_loop:o` 9018 `{`
9019 `\seq_clear:N \exp_not:N \l_msg_redirect_classes_seq`
9020 `\exp_not:n {#2}`
9021 `}`
9022 `\cs_set_protected:Npx \msg_use_loop:n ##1`
9023 `{`
9024 `\seq_if_in:NnTF \exp_not:n \l_msg_redirect_classes_seq {#1}`
9025 `{ \msg_kernel_error:nn { msg } { message-loop } {#1} }`
9026 `{`
9027 `\seq_put_right:Nn \exp_not:N \l_msg_redirect_classes_seq {#1}`
9028 `\exp_not:N \cs_if_exist:cTF { msg_ ##1 :nnxxxx }`
9029 `{`
9030 `\exp_not:N \use:c { msg_ ##1 :nnxxxx }`
9031 `\exp_not:n { {#3} {#4} {#5} {#6} {#7} {#8} }`
9032 `}`
9033 `{`
9034 `\msg_kernel_error:nnx { msg } { message-class-unknown } {##1}`
9035 `}`
9036 `}`
9037 `}`
9038 `\cs_if_exist:cTF { \c_msg_text_prefix_tl #3 / #4 }`
9039 `{ \msg_use_aux:nnn {#1} {#3} {#4} }`
9040 `{ \msg_kernel_error:nnxx { msg } { message-unknown } {#3} {#4} }`
9041 `}`

The first auxiliary macro looks for a match by name: the most restrictive check.

```

9042 \cs_new_protected:Npn \msg_use_aux:nnn #1#2#3
9043 {
9044   \tl_set:Nn \l_msg_current_class_tl {#1}
9045   \tl_set:Nn \l_msg_current_module_tl {#2}
9046   \prop_if_in:NnTF \l_msg_redirect_names_prop { // #2 / #3 / }
9047   { \msg_use_loop_check:nn { names } { // #2 / #3 / } }
9048   { \msg_use_aux:nn {#1} {#2} }
9049 }

```

The second function checks for general matches by module or for all modules.

```

9050 \cs_new_protected:Npn \msg_use_aux:nn #1#2
9051 {
9052   \prop_if_in:cnTF { l_msg_redirect_ #1 _prop } {#2}
9053   { \msg_use_loop_check:nn {#1} {#2} }
9054   {
9055     \prop_if_in:cnTF { l_msg_redirect_ #1 _prop } { * }
9056     { \msg_use_loop_check:nn {#1} { * } }
9057     { \msg_use_code: }

```

```

9058     }
9059 }

```

When checking whether to loop, the same code is needed in a few places.

```

9060 \cs_new_protected:Npn \msg_use_loop_check:nn #1#2
9061 {
9062     \prop_get:cnN { l_msg_redirect_ #1 _prop } {#2} \l_msg_class_tl
9063     \tl_if_eq:NNTF \l_msg_current_class_tl \l_msg_class_tl
9064     {
9065         { \msg_use_code: }
9066         { \msg_use_loop:o \l_msg_class_tl }
9067     }
9068 }
9069 \cs_new_protected_nopar:Npn \msg_use_code: { }
9070 \cs_new_protected:Npn \msg_use_loop:n #1 { }
9071 \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.

```

9072 \cs_new_protected:Npn \msg_redirect_class:nn #1#2
9073 { \prop_put:cnn { l_msg_redirect_ #1 _prop } { * } {#2} }

```

(End definition for \msg_redirect_class:nn. This function is documented on page 147.)

`\msg_redirect_module:nnn` For when all messages of a class should be altered for a given module.

```

9074 \cs_new_protected:Npn \msg_redirect_module:nnn #1#2#3
9075 { \prop_put:cnn { l_msg_redirect_ #2 _prop } {#1} {#3} }

```

(End definition for \msg_redirect_module:nnn. This function is documented on page 147.)

`\msg_redirect_name:nnn` Named message will always use the given class.

```

9076 \cs_new_protected:Npn \msg_redirect_name:nnn #1#2#3
9077 { \prop_put:Nnn \l_msg_redirect_names_prop { // #1 / #2 / } {#3} }

```

(End definition for \msg_redirect_name:nnn. This function is documented on page 147.)

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
`\msg_kernel_set:nnnn` part.
`\msg_kernel_set:nnn`

```

9078 \cs_new_protected:Npn \msg_kernel_new:nnnn #1#2
9079 { \msg_new:nnnn { LaTeX } { #1 / #2 } }
9080 \cs_new_protected:Npn \msg_kernel_new:nnn #1#2
9081 { \msg_new:nnn { LaTeX } { #1 / #2 } }
9082 \cs_new_protected:Npn \msg_kernel_set:nnnn #1#2
9083 { \msg_set:nnnn { LaTeX } { #1 / #2 } }
9084 \cs_new_protected:Npn \msg_kernel_set:nnn #1#2
9085 { \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
\msg_kernel_fatal:nnxx
\msg_kernel_fatal:nnx
\msg_kernel_fatal:nn
9086 \cs_new_protected:Npn \msg_kernel_fatal:nnxxxx #1#2#3#4#5#6
9087 {
9088   \msg_interrupt:xxx
9089   { \msg_fatal_text:n { LaTeX } : ~ "#1 / #2" }
9090   {
9091     \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
9092     {#3} {#4} {#5} {#6}
9093     \msg_see_documentation_text:n { LaTeX3 }
9094   }
9095   { \c_msg_fatal_text_tl }
9096   \tex_end:D
9097 }
9098 \cs_new_protected:Npn \msg_kernel_fatal:nnxxx #1#2#3#4#5
9099 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
9100 \cs_new_protected:Npn \msg_kernel_fatal:nnxx #1#2#3#4
9101 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} {#4} { } { } }
9102 \cs_new_protected:Npn \msg_kernel_fatal:nnx #1#2#3
9103 { \msg_kernel_fatal:nnxxxx {#1} {#2} {#3} { } { } { } }
9104 \cs_new_protected:Npn \msg_kernel_fatal:nn #1#2
9105 { \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
\msg_kernel_error:nnxx
\msg_kernel_error:nnx
\msg_kernel_error:nn
9106 \cs_new_protected:Npn \msg_kernel_error:nnxxxx #1#2#3#4#5#6
9107 {
9108   \msg_if_more_text:cTF { \c_msg_more_text_prefix_tl LaTeX / #1 / #2 }
9109   {
9110     \msg_interrupt:xxx
9111     { \msg_error_text:n { LaTeX } : ~ " #1 / #2 " }
9112     {
9113       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
9114       {#3} {#4} {#5} {#6}
9115       \msg_see_documentation_text:n { LaTeX3 }
9116     }
9117     {
9118       \use:c { \c_msg_more_text_prefix_tl LaTeX / #1 / #2 }
9119       {#3} {#4} {#5} {#6}
9120     }
9121   }
9122   {
9123     \msg_interrupt:xxx
9124     { \msg_error_text:n { LaTeX } : ~ " #1 / #2 " }
9125     {
9126       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
9127       {#3} {#4} {#5} {#6}
9128       \msg_see_documentation_text:n { LaTeX3 }
9129     }
9130   }

```

```

9131     }
9132   }
9133   \cs_new_protected:Npn \msg_kernel_error:nnxxx #1#2#3#4#5
9134   { \msg_kernel_error:nnxxx {#1} {#2} {#3} {#4} {#5} { } }
9135   \cs_set_protected:Npn \msg_kernel_error:nnxx #1#2#3#4
9136   { \msg_kernel_error:nnxxx {#1} {#2} {#3} {#4} { } { } }
9137   \cs_set_protected:Npn \msg_kernel_error:nnx #1#2#3
9138   { \msg_kernel_error:nnxxx {#1} {#2} {#3} { } { } { } }
9139   \cs_set_protected:Npn \msg_kernel_error:nn #1#2
9140   { \msg_kernel_error:nnxxx {#1} {#2} { } { } { } { } }

```

(End definition for \msg_kernel_error:nnxxx. This function is documented on page ??.)

\msg_kernel_warning:nnxxx Kernel messages which can be redirected.

```

\msg_kernel_warning:nnxxx
\msg_kernel_warning:nnxxx 9141 \prop_new:N \l_msg_redirect_kernel_warning_prop
\msg_kernel_warning:nnxx 9142 \cs_new_protected:Npn \msg_kernel_warning:nnxxx #1#2#3#4#5#6
\msg_kernel_warning:nnx 9143 {
\msg_kernel_warning:nn 9144   \msg_use:nnnnxxxx { warning }
\msg_kernel_info:nnxxx 9145   {
\msg_kernel_info:nnxxx 9146     \msg_term:x
\msg_kernel_info:nnxx 9147     {
\msg_kernel_info:nnxx 9148       \msg_warning_text:n { LaTeX } : ~ " #1 / #2 " \\ \\
\msg_kernel_info:nnx 9149       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
\msg_kernel_info:nn 9150       {#3} {#4} {#5} {#6}
9151     }
9152   }
9153   { LaTeX } { #1 / #2 } {#3} {#4} {#5} {#6}
9154 }
9155 \cs_new_protected:Npn \msg_kernel_warning:nnxxx #1#2#3#4#5
9156 { \msg_kernel_warning:nnxxx {#1} {#2} {#3} {#4} {#5} { } }
9157 \cs_new_protected:Npn \msg_kernel_warning:nnxx #1#2#3#4
9158 { \msg_kernel_warning:nnxxx {#1} {#2} {#3} {#4} { } { } }
9159 \cs_new_protected:Npn \msg_kernel_warning:nnx #1#2#3
9160 { \msg_kernel_warning:nnxxx {#1} {#2} {#3} { } { } { } }
9161 \cs_new_protected:Npn \msg_kernel_warning:nn #1#2
9162 { \msg_kernel_warning:nnxxx {#1} {#2} { } { } { } { } }
9163 \prop_new:N \l_msg_redirect_kernel_info_prop
9164 \cs_new_protected:Npn \msg_kernel_info:nnxxx #1#2#3#4#5#6
9165 {
9166   \msg_use:nnnnxxxx { info }
9167   {
9168     \msg_log:x
9169     {
9170       \msg_info_text:n { LaTeX } : ~ " #1 / #2 " \\ \\
9171       \use:c { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
9172       {#3} {#4} {#5} {#6}
9173     }
9174   }
9175   { LaTeX } { #1 / #2 } {#3} {#4} {#5} {#6}
9176 }
9177 \cs_new_protected:Npn \msg_kernel_info:nnxxx #1#2#3#4#5

```

```

9178 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} {#4} {#5} { } }
9179 \cs_new_protected:Npn \msg_kernel_info:nnxx #1#2#3#4
9180 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} {#4} { } { } }
9181 \cs_new_protected:Npn \msg_kernel_info:nnx #1#2#3
9182 { \msg_kernel_info:nnxxxx {#1} {#2} {#3} { } { } { } }
9183 \cs_new_protected:Npn \msg_kernel_info:nn #1#2
9184 { \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.
9185 \msg_kernel_new:nnnn { msg } { message-already-defined }
9186 { Message~'#2'~for~module~'#1'~already-defined. }
9187 {
9188   \c_msg_coding_error_text_tl
9189   LaTeX~was~asked~to~define~a~new~message~called~'#2'\
9190   by~the~module~'#1':~this~message~already~exists.
9191   \c_msg_return_text_tl
9192 }
9193 \msg_kernel_new:nnnn { msg } { message-unknown }
9194 { Unknown~message~'#2'~for~module~'#1'. }
9195 {
9196   \c_msg_coding_error_text_tl
9197   LaTeX~was~asked~to~display~a~message~called~'#2'\
9198   by~the~module~'#1'~module:~this~message~does~not~exist.
9199   \c_msg_return_text_tl
9200 }
9201 \msg_kernel_new:nnnn { msg } { message-class-unknown }
9202 { Unknown~message~class~'#1'. }
9203 {
9204   LaTeX~has~been~asked~to~redirect~messages~to~a~class~'#1':\
9205   this~was~never~defined.
9206   \c_msg_return_text_tl
9207 }
9208 \msg_kernel_new:nnnn { msg } { redirect-loop }
9209 { Message~redirection~loop~for~message~class~'#1'. }
9210 {
9211   LaTeX~has~been~asked~to~redirect~messages~in~an~infinite~loop.\
9212   The~original~message~here~has~been~lost.
9213   \c_msg_return_text_tl
9214 }
      Messages for earlier kernel modules.
9215 \msg_kernel_new:nnnn { kernel } { bad-number-of-arguments }
9216 { Function~'#1'~cannot~be~defined~with~#2~arguments. }
9217 {
9218   \c_msg_coding_error_text_tl
9219   LaTeX~has~been~asked~to~define~a~function~'#1'~with~
9220   #2~arguments. \
9221   TeX~allows~between~0~and~9~arguments~for~a~single~function.
9222 }
9223 \msg_kernel_new:nnnn { kernel } { command-already-defined }

```

```

9224 { Control~sequence~#1~already~defined. }
9225 {
9226   \c_msg_coding_error_text_tl
9227   LaTeX~has~been~asked~to~create~a~new~control~sequence~'~#1'~
9228   but~this~name~has~already~been~used~elsewhere. \\ \\
9229   The~current~meaning~is:\\
9230   \\ #2
9231 }
9232 \msg_kernel_new:nnnn { kernel } { command-not-defined }
9233 { Control~sequence~#1~undefined. }
9234 {
9235   \c_msg_coding_error_text_tl
9236   LaTeX~has~been~asked~to~use~a~command~#1,~but~this~has~not~
9237   been~defined~yet.
9238 }
9239 \msg_kernel_new:nnnn { kernel } { variable-not-defined }
9240 { Variable~#1~undefined. }
9241 {
9242   \c_msg_coding_error_text_tl
9243   LaTeX~has~been~asked~to~show~a~variable~#1,~but~this~has~not~
9244   been~defined~yet.
9245 }
9246 \msg_kernel_new:nnnn { seq } { empty-sequence }
9247 { Empty~sequence~#1. }
9248 {
9249   \c_msg_coding_error_text_tl
9250   LaTeX~has~been~asked~to~recover~an~entry~from~a~sequence~that~
9251   has~no~content:~that~cannot~happen!
9252 }
9253 \msg_kernel_new:nnnn { tl } { empty-search-pattern }
9254 { Empty~search~pattern. }
9255 {
9256   \c_msg_coding_error_text_tl
9257   LaTeX~has~been~asked~to~replace~an~empty~pattern~by~'~#1':~that~%
9258   would~lead~to~an~infinite~loop!
9259 }
9260 \msg_kernel_new:nnnn { scan } { already-defined }
9261 { Scan~mark~#1~already~defined. }
9262 {
9263   \c_msg_coding_error_text_tl
9264   LaTeX~has~been~asked~to~create~a~new~scan~mark~'~#1'~
9265   but~this~name~has~already~been~used~for~a~scan~mark.
9266 }

```

Some errors only appear in expandable settings, hence don't need a “more-text” argument.

```

9267 \msg_kernel_new:nnn { seq } { misused }
9268 { A~sequence~was~misused. }
9269 \msg_kernel_new:nnn { kernel } { bad-var }
9270 { Erroneous~variable~#1~used! }

```



```

9271 \msg_kernel_new:nnn { prg } { zero-step }
9272 { Zero~step~size~for~stepwise~function~#1. }
9273 \msg_kernel_new:nnn { prg } { replicate-neg }
9274 { Negative~argument~for~\prg_replicate:nn. }

Messages used by the “show” functions.

9275 \msg_kernel_new:nnn { seq } { show }
9276 {
9277   The~sequence~\token_to_str:N #1~
9278   \seq_if_empty:NTF #1
9279   { is~empty }
9280   { contains~the~items~(without~outer~braces): }
9281 }
9282 \msg_kernel_new:nnn { prop } { show }
9283 {
9284   The~property~list~\token_to_str:N #1~
9285   \prop_if_empty:NTF #1
9286   { is~empty }
9287   { contains~the~pairs~(without~outer~braces): }
9288 }
9289 \msg_kernel_new:nnn { clist } { show }
9290 {
9291   The~comma~list~
9292   \str_if_eq:nnF {#1} { \l_clist_internal_clist } { \token_to_str:N #1~}
9293   \clist_if_empty:NTF #1
9294   { is~empty }
9295   { contains~the~items~(without~outer~braces): }
9296 }
9297 \msg_kernel_new:nnn { ior } { show-no-stream }
9298 { No~input~streams~are~open }
9299 \msg_kernel_new:nnn { ior } { show-open-streams }
9300 { The~following~input~streams~are~in~use: }
9301 \msg_kernel_new:nnn { iow } { show-no-stream }
9302 { No~output~streams~are~open }
9303 \msg_kernel_new:nnn { iow } { show-open-streams }
9304 { 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`.

```

9305 \group_begin:
9306 \char_set_catcode_math_superscript:N \^
9307 \char_set_lccode:nn {'^} {'\ }
9308 \char_set_lccode:nn {'L} {'L}
9309 \char_set_lccode:nn {'T} {'T}
9310 \char_set_lccode:nn {'X} {'X}
9311 \tl_to_lowercase:n
9312 {
9313   \cs_new:Npx \msg_expandable_error:n #1
9314   {
9315     \exp_not:n
9316     {
9317       \tex_romannumeral:D
9318       \exp_after:wN \exp_after:wN
9319       \exp_after:wN \msg_expandable_error_aux:w
9320       \exp_after:wN \exp_after:wN
9321       \exp_after:wN \c_zero
9322     }
9323     \exp_not:N \use:n { \exp_not:c { LaTeX3~error: } ^ #1 } ^
9324   }
9325   \cs_new:Npn \msg_expandable_error_aux:w #1 ^ #2 ^ { #1 }
9326 }
9327 \group_end:

```

(End definition for `\msg_expandable_error:n`. This function is documented on page 150.)

`\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 9328 \cs_new:Npn \msg_expandable_kernel_error:nnnnnn #1#2#3#4#5#6
\msg_expandable_kernel_error:nnnnnn 9329 {
\msg_expandable_kernel_error:nnnn 9330   \exp_args:Nf \msg_expandable_error:n
\msg_expandable_kernel_error:nnn 9331   {
\msg_expandable_kernel_error:nn 9332     \exp_args:Nnc \exp_after:wN \exp_stop_f:
9333     { \c_msg_text_prefix_tl LaTeX / #1 / #2 }
9334     {#3} {#4} {#5} {#6}
9335   }
9336 }
9337 \cs_new:Npn \msg_expandable_kernel_error:nnnnn #1#2#3#4#5
9338 {
9339   \msg_expandable_kernel_error:nnnnnn
9340   {#1} {#2} {#3} {#4} {#5} { }
9341 }
9342 \cs_new:Npn \msg_expandable_kernel_error:nnnn #1#2#3#4
9343 {
9344   \msg_expandable_kernel_error:nnnnnn
9345   {#1} {#2} {#3} {#4} { } { }
9346 }
9347 \cs_new:Npn \msg_expandable_kernel_error:nnn #1#2#3

```

```

9348 {
9349   \msg_expandable_kernel_error:nnnnnn
9350   {#1} {#2} {#3} { } { } { }
9351 }
9352 \cs_new:Npn \msg_expandable_kernel_error:nn #1#2
9353 {
9354   \msg_expandable_kernel_error:nnnnnn
9355   {#1} {#2} { } { } { } { }
9356 }

```

(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

\msg_aux_use:nn Print the text of a message to the terminal, without formatting.
\msg_aux_use:nnxxxx

```

9357 \cs_new_protected:Npn \msg_aux_use:nn #1#2
9358 { \msg_aux_use:nnxxxx {#1} {#2} { } { } { } { } }
9359 \cs_new_protected:Npn \msg_aux_use:nnxxxx #1#2#3#4#5#6
9360 {
9361   \iow_wrap:xnnnN
9362   {
9363     \use:c { \c_msg_text_prefix_tl #1 / #2 }
9364     {#3} {#4} {#5} {#6}
9365   }
9366   { } \c_zero { } \iow_term:x
9367 }

```

(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 T_EX 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.

```

9368 \cs_new_protected:Npn \msg_aux_show:Nnx #1#2#3
9369 {
9370   \cs_if_exist:NTF #1
9371   {

```

```

9372     \msg_aux_use:nxxxxx { LaTeX / #2 } { show } {#1} { } { } { }
9373     \msg_aux_show:x {#3}
9374   }
9375   {
9376     \msg_kernel_error:nxx { kernel } { variable-not-defined }
9377     { \token_to_str:N #1 }
9378   }
9379 }
9380 \cs_new_protected:Npn \msg_aux_show:x #1
9381 {
9382   \tl_set:Nx \l_msg_internal_tl {#1}
9383   \tl_if_empty:NT \l_msg_internal_tl
9384   { \tl_set:Nx \l_msg_internal_tl { > } }
9385   \exp_args:Nf \etex_showtokens:D
9386   {
9387     \exp_after:wN \exp_after:wN
9388     \exp_after:wN \msg_aux_show:w
9389     \exp_after:wN \l_msg_internal_tl
9390     \exp_after:wN
9391   }
9392   \prg_do_nothing:
9393 }
9394 \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`
`\msg_aux_show_unbraced:nn`

```

9395 \cs_new:Npn \msg_aux_show:n #1
9396 {
9397   \iow_newline: > \c_space_tl \c_space_tl { \exp_not:n {#1} }
9398 }
9399 \cs_new:Npn \msg_aux_show:nn #1#2
9400 {
9401   \iow_newline: > \c_space_tl \c_space_tl { \exp_not:n {#1} }
9402   \c_space_tl \c_space_tl => \c_space_tl \c_space_tl { \exp_not:n {#2} }
9403 }
9404 \cs_new:Npn \msg_aux_show_unbraced:nn #1#2
9405 {
9406   \iow_newline: > \c_space_tl \c_space_tl \exp_not:n {#1}
9407   \c_space_tl \c_space_tl => \c_space_tl \c_space_tl \exp_not:n {#2}
9408 }

```

(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.

```

9409 <*/deprecated>
9410 \cs_new_eq:NN \msg_class_new:nn \msg_class_set:nn
9411 </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:nnxxx
 \msg_trace:nnxx

```

\msg_trace:nnx
\msg_trace:nn
9412 \*deprecated)
9413 \cs_new_eq:NN \msg_trace:nnxxxx \msg_log:nnxxxx
9414 \cs_new_eq:NN \msg_trace:nnxxx \msg_log:nnxxx
9415 \cs_new_eq:NN \msg_trace:nnxx \msg_log:nnxx
9416 \cs_new_eq:NN \msg_trace:nnx \msg_log:nnx
9417 \cs_new_eq:NN \msg_trace:nn \msg_log:nn
9418 \*deprecated)

```

(End definition for \msg_trace:nnxxxx and others. These functions are documented on page ??.)

\msg_generic_new:nnn These were all too low-level.

```

\msg_generic_new:nn
\msg_generic_set:nnn
\msg_generic_set:nn
\msg_direct_interrupt:xxxxx
\msg_direct_log:xx
\msg_direct_term:xx
9419 \*deprecated)
9420 \cs_new_protected:Npn \msg_generic_new:nnn #1#2#3 { \deprecated }
9421 \cs_new_protected:Npn \msg_generic_new:nn #1#2 { \deprecated }
9422 \cs_new_protected:Npn \msg_generic_set:nnn #1#2#3 { \deprecated }
9423 \cs_new_protected:Npn \msg_generic_set:nn #1#2 { \deprecated }
9424 \cs_new_protected:Npn \msg_direct_interrupt:xxxxx #1#2#3#4#5 { \deprecated }
9425 \cs_new_protected:Npn \msg_direct_log:xx #1#2 { \deprecated }
9426 \cs_new_protected:Npn \msg_direct_term:xx #1#2 { \deprecated }
9427 \*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

```

9428 \*deprecated)
9429 \cs_set_protected:Npn \msg_kernel_bug:x #1
9430 {
9431   \msg_interrupt:xxx { \c_msg_kernel_bug_text_tl }
9432   {
9433     #1
9434     \msg_see_documentation_text:n { LaTeX3 }
9435   }
9436   { \c_msg_kernel_bug_more_text_tl }
9437 }
9438 \tl_const:Nn \c_msg_kernel_bug_text_tl
9439 { This~is~a~LaTeX~bug:~check~coding! }
9440 \tl_const:Nn \c_msg_kernel_bug_more_text_tl
9441 {
9442   There~is~a~coding~bug~somewhere~around~here. \\
9443   This~probably~needs~examining~by~an~expert.
9444   \c_msg_return_text_tl
9445 }
9446 \*deprecated)

```

(End definition for \msg_kernel_bug:x. This function is documented on page ??.)

9447 *initex | package)

201 l3keys Implementation

```

9448 <*initex | package>
9449 <*package>
9450 \ProvidesExplPackage
9451   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
9452 \package_check_loaded_expl:
9453 </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.

```

9454 \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
9455 \tl_new:N \l_keyval_key_tl
9456 \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
9457 \tl_new:N \l_keyval_sanitise_tl
9458 \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.

```

9459 \group_begin:
9460   \char_set_catcode_active:n { '\= }
9461   \char_set_catcode_active:n { '\, }
9462   \char_set_lccode:nn { '\8 } { '\= }
9463   \char_set_lccode:nn { '\9 } { '\, }
9464   \tl_to_lowercase:n
9465   {
9466     \group_end:
9467     \cs_new_protected:Npn \keyval_parse:n #1
9468     {
9469       \group_begin:
9470       \tl_clear:N \l_keyval_sanitise_tl
9471       \tl_set:Nn \l_keyval_sanitise_tl {#1}
9472       \tl_replace_all:Nnn \l_keyval_sanitise_tl { = } { 8 }
9473       \tl_replace_all:Nnn \l_keyval_sanitise_tl { , } { 9 }
9474       \tl_clear:N \l_keyval_parse_tl
9475       \exp_after:wN \keyval_parse_elt:w \exp_after:wN
9476       \q_no_value \l_keyval_sanitise_tl 9 \q_nil 9
9477       \exp_after:wN \group_end:

```

```

9478         \l_keyval_parse_tl
9479     }
9480 }

```

(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.

```

9481 \cs_new_protected:Npn \keyval_parse_elt:w #1 ,
9482 {
9483     \tl_if_blank:oTF { \use_none:n #1 }
9484     { \keyval_parse_elt:w \q_no_value }
9485     {
9486         \quark_if_nil:oF { \use_ii:nn #1 }
9487         {
9488             \keyval_split_key_value:w #1 = = \q_stop
9489             \keyval_parse_elt:w \q_no_value
9490         }
9491     }
9492 }

```

(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

```

9493 \cs_new_protected:Npn \keyval_split_key_value:w #1 = #2 \q_stop
9494 {
9495     \keyval_split_key:w #1 \q_stop
9496     \str_if_eq:nnTF {#2} { = }
9497     {
9498         \tl_put_right:Nx \l_keyval_parse_tl
9499         {
9500             \exp_not:c
9501             { keyval_key_no_value_elt_ \int_use:N \g_keyval_level_int :n }
9502             { \exp_not:o \l_keyval_key_tl }
9503         }
9504     }
9505     {
9506         \keyval_split_key_value_aux:wTF #2 \q_no_value \q_stop
9507         { \keyval_split_value:w \q_nil #2 }
9508         { \msg_kernel_error:nn { keyval } { misplaced-equals-sign } }
9509     }
9510 }
9511 \cs_new:Npn \keyval_split_key_value_aux:wTF #1 = #2#3 \q_stop
9512 { \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.

```

9513 \cs_new_protected:Npn \keyval_split_key:w #1 \q_stop
9514 {
9515   \tl_set:Nx \l_keyval_key_tl
9516   { \exp_after:wN \tl_trim_spaces:n \exp_after:wN { \use_none:n #1 } }
9517 }

```

(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.

```

9518 \cs_new_protected:Npn \keyval_split_value:w #1 = =
9519 {
9520   \tl_put_right:Nx \l_keyval_parse_tl
9521   {
9522     \exp_not:c
9523     { keyval_key_value_elt_ \int_use:N \g_keyval_level_int :nn }
9524     { \exp_not:o \l_keyval_key_tl }
9525   }
9526   \tl_set:Nx \l_keyval_value_tl
9527   { \exp_not:o { \use_none:nnn #1 \q_nil \q_nil } }
9528   \tl_if_empty:NTF \l_keyval_value_tl
9529   { \tl_put_right:Nn \l_keyval_parse_tl { { } } }
9530   {
9531     \quark_if_nil:NTF \l_keyval_value_tl
9532     {
9533       \tl_put_right:Nx \l_keyval_parse_tl
9534       { { \exp_not:o { \use_ii:nnn #1 \q_nil } } }
9535     }
9536     { \keyval_split_value_aux:w #1 \q_stop }
9537   }
9538 }

```

A similar idea to the key code: remove the spaces from each end and deal with one set of braces.

```

9539 \cs_new_protected:Npn \keyval_split_value_aux:w \q_nil #1 \q_stop
9540 {
9541   \tl_set:Nx \l_keyval_value_tl { \tl_trim_spaces:n {#1} }
9542   \tl_put_right:Nx \l_keyval_parse_tl
9543   { { \exp_not:o \l_keyval_value_tl } }
9544 }

```

(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.

```

9545 \cs_new_protected:Npn \keyval_parse:NNn #1#2#3
9546 {
9547   \int_gincr:N \g_keyval_level_int
9548   \cs_gset_eq:cN
9549     { keyval_key_no_value_elt_ \int_use:N \g_keyval_level_int :n } #1
9550   \cs_gset_eq:cN
9551     { keyval_key_value_elt_ \int_use:N \g_keyval_level_int :nn } #2
9552   \keyval_parse:n {#3}
9553   \int_gdecr:N \g_keyval_level_int
9554 }

```

(End definition for \keyval_parse:NNn. This function is documented on page 161.)

One message for the low level parsing system.

```

9555 \msg_kernel_new:nnnn { keyval } { misplaced-equals-sign }
9556 { Misplaced-equals-sign-in-key-value-input~\msg_line_number: }
9557 {
9558   LaTeX-is-attempting-to-parse-some-key-value-input-but-found~
9559   two-equals-signs-not-separated-by-a-comma.
9560 }

```

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
9561 \tl_const:Nn \c_keys_code_root_tl { key~code~>~ }
9562 \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.

```

9563 \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
9564 \tl_const:Nn \c_keys_value_forbidden_tl { forbidden }
9565 \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.

```

9566 \int_new:N \l_keys_choice_int
9567 \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.

```

9568 \tl_new:N \l_keys_key_tl

```

(End definition for \l_keys_key_tl. This function is documented on page 159.)

<code>\l_keys_module_tl</code>	The module for an entire set of keys. <div> <div>9569</div> <div><code>\tl_new:N \l_keys_module_tl</code></div> <div>(End definition for <code>\l_keys_module_tl</code>. This function is documented on page ??.)</div> </div>
<code>\l_keys_no_value_bool</code>	A marker is needed internally to show if only a key or a key plus a value was seen: this is recorded here. <div> <div>9570</div> <div><code>\bool_new:N \l_keys_no_value_bool</code></div> <div>(End definition for <code>\l_keys_no_value_bool</code>. This function is documented on page ??.)</div> </div>
<code>\l_keys_path_tl</code>	The “path” of the current key is stored here: this is available to the programmer and so is public. <div> <div>9571</div> <div><code>\tl_new:N \l_keys_path_tl</code></div> <div>(End definition for <code>\l_keys_path_tl</code>. This function is documented on page 159.)</div> </div>
<code>\l_keys_property_tl</code>	The “property” begin set for a key at definition time is stored here. <div> <div>9572</div> <div><code>\tl_new:N \l_keys_property_tl</code></div> <div>(End definition for <code>\l_keys_property_tl</code>. This function is documented on page ??.)</div> </div>
<code>\l_keys_unknown_clist</code>	Used when setting only known keys to store those left over. <div> <div>9573</div> <div><code>\tl_new:N \l_keys_unknown_clist</code></div> <div>(End definition for <code>\l_keys_unknown_clist</code>. This function is documented on page ??.)</div> </div>
<code>\l_keys_value_tl</code>	The value given for a key: may be empty if no value was given. <div> <div>9574</div> <div><code>\tl_new:N \l_keys_value_tl</code></div> <div>(End definition for <code>\l_keys_value_tl</code>. This function is documented on page 159.)</div> </div>

201.3 The key defining mechanism

<code>\keys_define:nn</code>	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).
<code>\keys_define_aux:nnn</code>	
<code>\keys_define_aux:onn</code>	
	<div> <div>9575</div> <div><code>\cs_new_protected:Npn \keys_define:nn</code></div> <div><code>{ \keys_define_aux:onn \l_keys_module_tl }</code></div> <div>9576</div> <div><code>\cs_new_protected:Npn \keys_define_aux:nnn #1#2#3</code></div> <div>9577</div> <div><code>{</code></div> <div>9578</div> <div><code> \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }</code></div> <div>9579</div> <div><code> \keyval_parse:NNn \keys_define_elt:n \keys_define_elt:nn {#3}</code></div> <div>9580</div> <div><code> \tl_set:Nn \l_keys_module_tl {#1}</code></div> <div>9581</div> <div><code>}</code></div> <div>9582</div> <div><code>\cs_generate_variant:Nn \keys_define_aux:nnn { o }</code></div> <div>9583</div> <div>(End definition for <code>\keys_define:nn</code>. This function is documented on page ??.)</div> </div>

`\keys_define_elt:n` The outer functions here record whether a value was given and then converge on a
`\keys_define_elt:nn` common internal mechanism. There is first a search for a property in the current key
`\keys_define_elt_aux:nn` name, then a check to make sure it is known before the code hands off to the next step.

```

9584 \cs_new_protected:Npn \keys_define_elt:n #1
9585 {
9586   \bool_set_true:N \l_keys_no_value_bool
9587   \keys_define_elt_aux:nn {#1} { }
9588 }
9589 \cs_new_protected:Npn \keys_define_elt:nn #1#2
9590 {
9591   \bool_set_false:N \l_keys_no_value_bool
9592   \keys_define_elt_aux:nn {#1} {#2}
9593 }
9594 \cs_new_protected:Npn \keys_define_elt_aux:nn #1#2 {
9595   \keys_property_find:n {#1}
9596   \cs_if_exist:cTF { \c_keys_props_root_tl \l_keys_property_tl }
9597   { \keys_define_key:n {#2} }
9598   {
9599     \msg_kernel_error:nxxx { keys } { property-unknown }
9600     { \l_keys_property_tl } { \l_keys_path_tl }
9601   }
9602 }

```

(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.

```

9603 \cs_new_protected:Npn \keys_property_find:n #1
9604 {
9605   \tl_set:Nx \l_keys_path_tl { \l_keys_module_tl / }
9606   \tl_if_in:nnTF {#1} { . }
9607   { \keys_property_find_aux:w #1 \q_stop }
9608   { \msg_kernel_error:nnx { keys } { key-no-property } {#1} }
9609 }
9610 \cs_new_protected:Npn \keys_property_find_aux:w #1 . #2 \q_stop
9611 {
9612   \tl_set:Nx \l_keys_path_tl { \l_keys_path_tl \tl_to_str:n {#1} }
9613   \tl_if_in:nnTF {#2} { . }
9614   {
9615     \tl_set:Nx \l_keys_path_tl { \l_keys_path_tl . }
9616     \keys_property_find_aux:w #2 \q_stop
9617   }
9618   { \tl_set:Nn \l_keys_property_tl { . #2 } }
9619 }

```

(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.

```

9620 \cs_new_protected:Npn \keys_define_key:n #1
9621 {
9622   \bool_if:NTF \l_keys_no_value_bool
9623   {
9624     \exp_after:wN \keys_define_key_aux:w
9625     \l_keys_property_tl \q_stop
9626     { \use:c { \c_keys_props_root_tl \l_keys_property_tl } }
9627     {
9628       \msg_kernel_error:nnxx { keys }
9629       { property-requires-value } { \l_keys_property_tl }
9630       { \l_keys_path_tl }
9631     }
9632   }
9633   { \use:c { \c_keys_props_root_tl \l_keys_property_tl } {#1} }
9634 }
9635 \cs_new_protected:Npn \keys_define_key_aux:w #1 : #2 \q_stop
9636 { \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.

```

9637 \cs_new:Npn \keys_bool_set:NN #1#2
9638 {
9639   \cs_if_exist:NF #1 { \bool_new:N #1 }
9640   \keys_choice_make:
9641   \keys_cmd_set:nx { \l_keys_path_tl / true }
9642   { \exp_not:c { bool_ #2 set_true:N } \exp_not:N #1 }
9643   \keys_cmd_set:nx { \l_keys_path_tl / false }
9644   { \exp_not:c { bool_ #2 set_false:N } \exp_not:N #1 }
9645   \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9646   {
9647     \msg_kernel_error:nnx { keys } { boolean-values-only }
9648     { \l_keys_key_tl }
9649   }
9650   \keys_default_set:n { true }
9651 }

```

(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.

```

9652 \cs_new:Npn \keys_bool_set_inverse:NN #1#2
9653 {
9654   \cs_if_exist:NF #1 { \bool_new:N #1 }
9655   \keys_choice_make:
9656   \keys_cmd_set:nx { \l_keys_path_tl / true }

```

```

9657     { \exp_not:c { bool_ #2 set_false:N } \exp_not:N #1 }
9658 \keys_cmd_set:nx { \l_keys_path_tl / false }
9659     { \exp_not:c { bool_ #2 set_true:N } \exp_not:N #1 }
9660 \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9661     {
9662         \msg_kernel_error:nnx { keys } { boolean-values-only }
9663         { \l_keys_key_tl }
9664     }
9665 \keys_default_set:n { true }
9666 }

```

(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.

```

9667 \cs_new_protected_nopar:Npn \keys_choice_make:
9668 {
9669     \keys_cmd_set:nn { \l_keys_path_tl }
9670     { \keys_choice_find:n {##1} }
9671     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9672     {
9673         \msg_kernel_error:nnxx { keys } { choice-unknown }
9674         { \l_keys_path_tl } {##1}
9675     }
9676 }

```

(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.

```

9677 \cs_new_protected:Npn \keys_choices_make:nn #1#2
9678 {
9679     \keys_choice_make:
9680     \int_zero:N \l_keys_choice_int
9681     \clist_map_inline:nn {#1}
9682     {
9683         \keys_cmd_set:nx { \l_keys_path_tl / ##1 }
9684         {
9685             \tl_set:Nn \exp_not:N \l_keys_choice_tl {##1}
9686             \int_set:Nn \exp_not:N \l_keys_choice_int
9687             { \int_use:N \l_keys_choice_int }
9688             \exp_not:n {#2}
9689         }
9690         \int_incr:N \l_keys_choice_int
9691     }
9692 }

```

(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.

```

9693 \cs_new_protected:Npn \keys_choices_generate:n #1
9694 {

```

```

9695 \cs_if_exist:cTF
9696 { \c_keys_vars_root_tl \l_keys_path_tl .choice~code }
9697 {
9698   \keys_choice_make:
9699   \int_zero:N \l_keys_choice_int
9700   \clist_map_function:nN {#1} \keys_choices_generate_aux:n
9701 }
9702 {
9703   \msg_kernel_error:nnx { keys }
9704   { generate-choices-before-code } { \l_keys_path_tl }
9705 }
9706 }
9707 \cs_new_protected:Npn \keys_choices_generate_aux:n #1
9708 {
9709   \keys_cmd_set:nx { \l_keys_path_tl / #1 }
9710   {
9711     \tl_set:Nn \exp_not:N \l_keys_choice_tl {#1}
9712     \int_set:Nn \exp_not:N \l_keys_choice_int
9713     { \int_use:N \l_keys_choice_int }
9714     \exp_not:v
9715     { \c_keys_vars_root_tl \l_keys_path_tl .choice~code }
9716   }
9717   \int_incr:N \l_keys_choice_int
9718 }

```

(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.

```

9719 \cs_new_protected:Npn \keys_choice_code_store:x #1
9720 {
9721   \cs_if_exist:cF
9722   { \c_keys_vars_root_tl \l_keys_path_tl .choice~code }
9723   {
9724     \tl_new:c
9725     { \c_keys_vars_root_tl \l_keys_path_tl .choice~code }
9726   }
9727   \tl_set:cx { \c_keys_vars_root_tl \l_keys_path_tl .choice~code }
9728   {#1}
9729 }

```

(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
9730 \cs_new_protected:Npn \keys_cmd_set:nn #1#2
9731 {
9732   \keys_cmd_set_aux:n {#1}
9733   \cs_set:cpn { \c_keys_code_root_tl #1 } ##1 {#2}
9734 }
9735 \cs_new_protected:Npn \keys_cmd_set:nx #1#2
9736 {

```

```

9737     \keys_cmd_set_aux:n {#1}
9738     \cs_set:cpx { \c_keys_code_root_tl #1 } ##1 {#2}
9739   }
9740   \cs_new_protected:Npn \keys_cmd_set_aux:n #1
9741   {
9742     \tl_clear_new:c { \c_keys_vars_root_tl #1 .default }
9743     \tl_set:cn { \c_keys_vars_root_tl #1 .default } { \q_no_value }
9744     \tl_clear_new:c { \c_keys_vars_root_tl #1 .req }
9745   }

```

(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
9746   \cs_new_protected:Npn \keys_default_set:n #1
9747   { \tl_set:cn { \c_keys_vars_root_tl \l_keys_path_tl .default } {#1} }
9748   \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
9749   \cs_new_protected:Npn \keys_meta_make:n #1
9750   {
9751     \exp_args:NNo \keys_cmd_set:nn \l_keys_path_tl
9752     { \exp_after:wN \keys_set:nn \exp_after:wN { \l_keys_module_tl } {#1} }
9753   }
9754   \cs_new_protected:Npn \keys_meta_make:x #1
9755   {
9756     \keys_cmd_set:nx { \l_keys_path_tl }
9757     { \exp_not:N \keys_set:nn { \l_keys_module_tl } {#1} }
9758   }

```

(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.

```

9759   \cs_new:Npn \keys_multichoice_find:n #1
9760   { \clist_map_function:nN {#1} \keys_choice_find:n }
9761   \cs_new_protected_nopar:Npn \keys_multichoice_make:
9762   {
9763     \keys_cmd_set:nn { \l_keys_path_tl }
9764     { \keys_multichoice_find:n {##1} }
9765     \keys_cmd_set:nn { \l_keys_path_tl / unknown }
9766     {
9767       \msg_kernel_error:nnxx { keys } { choice-unknown }
9768       { \l_keys_path_tl } {##1}
9769     }
9770   }
9771   \cs_new_protected:Npn \keys_multichoices_make:nn #1#2

```

```

9772 {
9773   \keys_multichoice_make:
9774   \int_zero:N \l_keys_choice_int
9775   \clist_map_inline:nn {#1}
9776   {
9777     \keys_cmd_set:nx { \l_keys_path_tl / ##1 }
9778     {
9779       \tl_set:Nn \exp_not:N \l_keys_choice_tl {##1}
9780       \int_set:Nn \exp_not:N \l_keys_choice_int
9781       { \int_use:N \l_keys_choice_int }
9782       \exp_not:n {#2}
9783     }
9784     \int_incr:N \l_keys_choice_int
9785   }
9786 }

```

(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.

```

9787 \cs_new_protected:Npn \keys_value_requirement:n #1
9788 {
9789   \tl_set_eq:cc
9790   { \c_keys_vars_root_tl \l_keys_path_tl .req }
9791   { c_keys_value_ #1 _tl }
9792 }

```

(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 variable if needed. The three-argument version is set up so that the use of { } as an N-type variable is only done once!

```

9793 \cs_new_protected:Npn \keys_variable_set:NnNN #1#2#3#4
9794 {
9795   \cs_if_exist:NF #1 { \use:c { #2 _new:N } #1 }
9796   \keys_cmd_set:nx { \l_keys_path_tl }
9797   { \exp_not:c { #2 _ #3 set:N #4 } \exp_not:N #1 {##1} }
9798 }
9799 \cs_new_protected:Npn \keys_variable_set:NnN #1#2#3
9800 { \keys_variable_set:NnNN #1 {#2} { } #3 }
9801 \cs_generate_variant:Nn \keys_variable_set:NnNN { c }
9802 \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.

```
.bool_gset:N 9803 \cs_new_protected:cpn { \c_keys_props_root_tl .bool_set:N } #1
              9804 { \keys_bool_set:NN #1 { } }
              9805 \cs_new_protected:cpn { \c_keys_props_root_tl .bool_gset:N } #1
              9806 { \keys_bool_set:NN #1 g }
              (End definition for .bool_set:N. This function is documented on page 153.)
```

`.bool_set_inverse:N` One function for this.

```
.bool_gset_inverse:N 9807 \cs_new_protected:cpn { \c_keys_props_root_tl .bool_set_inverse:N } #1
                    9808 { \keys_bool_set_inverse:NN #1 { } }
                    9809 \cs_new_protected:cpn { \c_keys_props_root_tl .bool_gset_inverse:N } #1
                    9810 { \keys_bool_set_inverse:NN #1 g }
                    (End definition for .bool_set_inverse:N. This function is documented on page 153.)
```

`.choice:` Making a choice is handled internally, as it is also needed by `.generate_choices:n`.

```
9811 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .choice: }
9812 { \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.

```
9813 \cs_new_protected:cpn { \c_keys_props_root_tl .choices:nn } #1
9814 { \keys_choices_make:nn #1 }
      (End definition for .choices:nn. This function is documented on page 153.)
```

`.code:n` Creating code is simply a case of passing through to the underlying `set` function.

```
.code:x 9815 \cs_new_protected:cpn { \c_keys_props_root_tl .code:n } #1
        9816 { \keys_cmd_set:nn { \l_keys_path_tl } {#1} }
        9817 \cs_new_protected:cpn { \c_keys_props_root_tl .code:x } #1
        9818 { \keys_cmd_set:nx { \l_keys_path_tl } {#1} }
        (End definition for .code:n and .code:x. These functions are documented on page 154.)
```

`.choice_code:n` Storing the code for choices, using `\exp_not:n` to avoid needing two internal functions.

```
.choice_code:x 9819 \cs_new_protected:cpn { \c_keys_props_root_tl .choice_code:n } #1
              9820 { \keys_choice_code_store:x { \exp_not:n {#1} } }
              9821 \cs_new_protected:cpn { \c_keys_props_root_tl .choice_code:x } #1
              9822 { \keys_choice_code_store:x {#1} }
              (End definition for .choice_code:n and .choice_code:x. These functions are documented on
              page 153.)
```

`.clist_set:N`

```
.clist_set:c 9823 \cs_new_protected:cpn { \c_keys_props_root_tl .clist_set:N } #1
              9824 { \keys_variable_set:NnN #1 { clist } n }
.clist_gset:N 9825 \cs_new_protected:cpn { \c_keys_props_root_tl .clist_set:c } #1
              9826 { \keys_variable_set:cnN {#1} { clist } n }
.clist_gset:c 9827 \cs_new_protected:cpn { \c_keys_props_root_tl .clist_gset:N } #1
              9828 { \keys_variable_set:NnNN #1 { clist } g n }
              9829 \cs_new_protected:cpn { \c_keys_props_root_tl .clist_gset:c } #1
              9830 { \keys_variable_set:cnNN {#1} { clist } g n }
```

(End definition for `.clist_set:N` and `.clist_set:c`. These functions are documented on page 153.)

`.default:n` Expansion is left to the internal functions.

```
.default:V 9831 \cs_new_protected:cpn { \c_keys_props_root_tl .default:n } #1
          9832 { \keys_default_set:n {#1} }
          9833 \cs_new_protected:cpn { \c_keys_props_root_tl .default:V } #1
          9834 { \keys_default_set:V #1 }
```

(End definition for `.default:n` and `.default:V`. These functions are documented on page 154.)

`.dim_set:N` Setting a variable is very easy: just pass the data along.

```
.dim_set:c 9835 \cs_new_protected:cpn { \c_keys_props_root_tl .dim_set:N } #1
          9836 { \keys_variable_set:NnN #1 { dim } n }
.dim_gset:N 9837 \cs_new_protected:cpn { \c_keys_props_root_tl .dim_set:c } #1
          9838 { \keys_variable_set:cnN {#1} { dim } n }
          9839 \cs_new_protected:cpn { \c_keys_props_root_tl .dim_gset:N } #1
          9840 { \keys_variable_set:NnNN #1 { dim } g n }
          9841 \cs_new_protected:cpn { \c_keys_props_root_tl .dim_gset:c } #1
          9842 { \keys_variable_set:cnNN {#1} { dim } g n }
```

(End definition for `.dim_set:N` and `.dim_set:c`. These functions are documented on page 154.)

`.fp_set:N` Setting a variable is very easy: just pass the data along.

```
.fp_set:c 9843 \cs_new_protected:cpn { \c_keys_props_root_tl .fp_set:N } #1
          9844 { \keys_variable_set:NnN #1 { fp } n }
.fp_gset:N 9845 \cs_new_protected:cpn { \c_keys_props_root_tl .fp_set:c } #1
          9846 { \keys_variable_set:cnN {#1} { fp } n }
          9847 \cs_new_protected:cpn { \c_keys_props_root_tl .fp_gset:N } #1
          9848 { \keys_variable_set:NnNN #1 { fp } g n }
          9849 \cs_new_protected:cpn { \c_keys_props_root_tl .fp_gset:c } #1
          9850 { \keys_variable_set:cnNN {#1} { fp } g n }
```

(End definition for `.fp_set:N` and `.fp_set:c`. These functions are documented on page 154.)

`.generate_choices:n` Making choices is easy.

```
9851 \cs_new_protected:cpn { \c_keys_props_root_tl .generate_choices:n } #1
9852 { \keys_choices_generate:n {#1} }
```

(End definition for `.generate_choices:n`. This function is documented on page 155.)

`.int_set:N` Setting a variable is very easy: just pass the data along.

```
.int_set:c 9853 \cs_new_protected:cpn { \c_keys_props_root_tl .int_set:N } #1
          9854 { \keys_variable_set:NnN #1 { int } n }
.int_gset:N 9855 \cs_new_protected:cpn { \c_keys_props_root_tl .int_set:c } #1
          9856 { \keys_variable_set:cnN {#1} { int } n }
          9857 \cs_new_protected:cpn { \c_keys_props_root_tl .int_gset:N } #1
          9858 { \keys_variable_set:NnNN #1 { int } g n }
          9859 \cs_new_protected:cpn { \c_keys_props_root_tl .int_gset:c } #1
          9860 { \keys_variable_set:cnNN {#1} { int } g n }
```

(End definition for `.int_set:N` and `.int_set:c`. These functions are documented on page 155.)

`.meta:n` Making a meta is handled internally.

`.meta:x`

```

9861 \cs_new_protected:cpn { \c_keys_props_root_tl .meta:n } #1
9862 { \keys_meta_make:n {#1} }
9863 \cs_new_protected:cpn { \c_keys_props_root_tl .meta:x } #1
9864 { \keys_meta_make:x {#1} }

```

(End definition for `.meta:n` and `.meta:x`. These functions are documented on page 155.)

`.multichoice:` The same idea as `.choice:` and `.choices:nn`, but where more than one choice is allowed.

`.multichoices:nn`

```

9865 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .multichoice: }
9866 { \keys_multichoice_make: }
9867 \cs_new_protected:cpn { \c_keys_props_root_tl .multichoices:nn } #1
9868 { \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`

```

9869 \cs_new_protected:cpn { \c_keys_props_root_tl .skip_set:N } #1
9870 { \keys_variable_set:NnN #1 { skip } n }
9871 \cs_new_protected:cpn { \c_keys_props_root_tl .skip_set:c } #1
9872 { \keys_variable_set:cnN {#1} { skip } n }
9873 \cs_new_protected:cpn { \c_keys_props_root_tl .skip_gset:N } #1
9874 { \keys_variable_set:NnNN #1 { skip } g n }
9875 \cs_new_protected:cpn { \c_keys_props_root_tl .skip_gset:c } #1
9876 { \keys_variable_set:cnNN {#1} { skip } g n }

```

(End definition for `.skip_set:N` and `.skip_set:c`. These functions are documented on page 155.)

`.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`

```

9877 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_set:N } #1
9878 { \keys_variable_set:NnN #1 { tl } n }
9879 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_set:c } #1
9880 { \keys_variable_set:cnN {#1} { tl } n }
9881 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_set_x:N } #1
9882 { \keys_variable_set:NnN #1 { tl } x }
9883 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_set_x:c } #1
9884 { \keys_variable_set:cnN {#1} { tl } x }
9885 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_gset:N } #1
9886 { \keys_variable_set:NnNN #1 { tl } g n }
9887 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_gset:c } #1
9888 { \keys_variable_set:cnNN {#1} { tl } g n }
9889 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_gset_x:N } #1
9890 { \keys_variable_set:NnNN #1 { tl } g x }
9891 \cs_new_protected:cpn { \c_keys_props_root_tl .tl_gset_x:c } #1
9892 { \keys_variable_set:cnNN {#1} { tl } g x }

```

(End definition for `.tl_set:N` and `.tl_set:c`. These functions are documented on page 156.)

`.value_forbidden:` These are very similar, so both call the same function.

`.value_required:`

```

9893 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .value_forbidden: }
9894 { \keys_value_requirement:n { forbidden } }
9895 \cs_new_protected_nopar:cpn { \c_keys_props_root_tl .value_required: }
9896 { \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 9897 \cs_new_protected:Npn \keys_set:nn
\keys_set:nv 9898 { \keys_set_aux:onn { \l_keys_module_tl } }
\keys_set:no 9899 \cs_new_protected:Npn \keys_set_aux:nnn #1#2#3
\keys_set_aux:nnn 9900 {
\keys_set_aux:onn 9901   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9902   \keyval_parse:NNn \keys_set_elt:n \keys_set_elt:nn {#3}
9903   \tl_set:Nn \l_keys_module_tl {#1}
9904 }
9905 \cs_generate_variant:Nn \keys_set:nn { nV , nv , no }
9906 \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 9907 \cs_new_protected:Npn \keys_set_known:nnN
\keys_set_known:nvN 9908 { \keys_set_known_aux:onnN { \l_keys_module_tl } }
\keys_set_known:noN 9909 \cs_new_protected:Npn \keys_set_known_aux:nnnN #1#2#3#4
\keys_set_known_aux:nnnN 9910 {
\keys_set_known_aux:onnN 9911   \tl_set:Nx \l_keys_module_tl { \tl_to_str:n {#2} }
9912   \clist_clear:N \l_keys_unknown_clist
9913   \cs_set_eq:NN \keys_execute_unknown: \keys_execute_unknown_alt:
9914   \keyval_parse:NNn \keys_set_elt:n \keys_set_elt:nn {#3}
9915   \cs_set_eq:NN \keys_execute_unknown: \keys_execute_unknown_std:
9916   \tl_set:Nn \l_keys_module_tl {#1}
9917   \clist_set_eq:NN #4 \l_keys_unknown_clist
9918 }
9919 \cs_generate_variant:Nn \keys_set_known:nnN { nV , nv , no }
9920 \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.

```

9921 \cs_new_protected:Npn \keys_set_elt:n #1
9922 {
9923   \bool_set_true:N \l_keys_no_value_bool
9924   \keys_set_elt_aux:nn {#1} { }
9925 }
9926 \cs_new_protected:Npn \keys_set_elt:nn #1#2
9927 {
9928   \bool_set_false:N \l_keys_no_value_bool
9929   \keys_set_elt_aux:nn {#1} {#2}
9930 }
9931 \cs_new_protected:Npn \keys_set_elt_aux:nn #1#2
9932 {
9933   \tl_set:Nx \l_keys_key_tl { \tl_to_str:n {#1} }

```

```

9934 \tl_set:Nx \l_keys_path_tl { \l_keys_module_tl / \l_keys_key_tl }
9935 \keys_value_or_default:n {#2}
9936 \bool_if:nTF
9937 {
9938   \keys_if_value_p:n { required } &&
9939   \l_keys_no_value_bool
9940 }
9941 {
9942   \msg_kernel_error:nnx { keys } { value-required }
9943   { \l_keys_path_tl }
9944 }
9945 {
9946   \bool_if:nTF
9947   {
9948     \keys_if_value_p:n { forbidden } &&
9949     ! \l_keys_no_value_bool
9950   }
9951   {
9952     \msg_kernel_error:nnxx { keys } { value-forbidden }
9953     { \l_keys_path_tl } { \l_keys_value_tl }
9954   }
9955   { \keys_execute: }
9956 }
9957 }

```

(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.

```

9958 \cs_new_protected:Npn \keys_value_or_default:n #1
9959 {
9960   \tl_set:Nn \l_keys_value_tl {#1}
9961   \bool_if:NT \l_keys_no_value_bool
9962   {
9963     \quark_if_no_value:cF { \c_keys_vars_root_tl \l_keys_path_tl .default }
9964     {
9965       \cs_if_exist:cT { \c_keys_vars_root_tl \l_keys_path_tl .default }
9966       {
9967         \tl_set_eq:Nc \l_keys_value_tl
9968         { \c_keys_vars_root_tl \l_keys_path_tl .default }
9969       }
9970     }
9971   }
9972 }

```

(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.

```

9973 \prg_new_conditional:Npnn \keys_if_value:n #1 { p }
9974 {

```

```

9975     \tl_if_eq:ccTF { c_keys_value_ #1 _tl }
9976     { \c_keys_vars_root_tl \l_keys_path_tl .req }
9977     { \prg_return_true: }
9978     { \prg_return_false: }
9979 }

```

(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
9980 \cs_new_nopar:Npn \keys_execute:
9981 { \keys_execute:nn { \l_keys_path_tl } { \keys_execute_unknown: } }
9982 \cs_new_nopar:Npn \keys_execute_unknown:
9983 {
9984   \keys_execute:nn { \l_keys_module_tl / unknown }
9985   {
9986     \msg_kernel_error:nxxx { keys } { key-unknown }
9987     { \l_keys_path_tl } { \l_keys_module_tl }
9988   }
9989 }
9990 \cs_new_eq:NN \keys_execute_unknown_std: \keys_execute_unknown:
9991 \cs_new_nopar:Npn \keys_execute_unknown_alt:
9992 {
9993   \clist_put_right:Nx \l_keys_unknown_clist
9994   {
9995     \exp_not:o \l_keys_key_tl
9996     \bool_if:NF \l_keys_no_value_bool
9997     { = { \exp_not:o \l_keys_value_tl } }
9998   }
9999 }
10000 \cs_new:Npn \keys_execute:nn #1#2
10001 {
10002   \cs_if_exist:cTF { \c_keys_code_root_tl #1 }
10003   {
10004     \exp_args:Nno \use:c { \c_keys_code_root_tl #1 }
10005     \l_keys_value_tl
10006   }
10007   {#2}
10008 }

```

(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.

```

10009 \cs_new:Npn \keys_choice_find:n #1
10010 {
10011   \keys_execute:nn { \l_keys_path_tl / \tl_to_str:n {#1} }
10012   { \keys_execute:nn { \l_keys_path_tl / unknown } { } }
10013 }

```

(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.

```
10014 \prg_new_conditional:Npnn \keys_if_exist:nn #1#2 { p , T , F , TF }
10015 {
10016   \cs_if_exist:cTF { \c_keys_code_root_tl #1 / #2 }
10017   { \prg_return_true: }
10018   { \prg_return_false: }
10019 }
```

(End definition for \keys_if_exist:nn. This function is documented on page 160.)

`\keys_if_choice_exist:nnn` Just an alternative view on `\keys_if_exist:nn(TF)`.

```
10020 \prg_new_conditional:Npnn \keys_if_choice_exist:nnn #1#2#3 { p , T , F , TF }
10021 {
10022   \cs_if_exist:cTF { \c_keys_code_root_tl #1 / #2 / #3 }
10023   { \prg_return_true: }
10024   { \prg_return_false: }
10025 }
```

(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.

```
10026 \cs_new:Npn \keys_show:nn #1#2
10027 { \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 160.)

201.8 Messages

For when there is a need to complain.

```
10028 \msg_kernel_new:nnnn { keys } { boolean-values-only }
10029 { Key~'#1'~accepts~boolean-values-only. }
10030 { The~key~'#1'~only~accepts~the~values~'true'~and~'false'. }
10031 \msg_kernel_new:nnnn { keys } { choice-unknown }
10032 { Choice~'#2'~unknown~for~key~'#1'. }
10033 {
10034   The~key~'#1'~takes~a~limited~number~of~values.\\
10035   The~input~given,~'#2',~is~not~on~the~list~accepted.
10036 }
10037 \msg_kernel_new:nnnn { keys } { generate-choices-before-code }
10038 { No~code~available~to~generate~choices~for~key~'#1'. }
10039 {
10040   \c_msg_coding_error_text_tl
10041   Before~using~.generate_choices:n~the~code~should~be~defined~
10042   with~'.choice_code:n'~or~'.choice_code:x'.
10043 }
10044 \msg_kernel_new:nnnn { keys } { key-no-property }
10045 { No~property~given~in~definition~of~key~'#1'. }
10046 {
10047   \c_msg_coding_error_text_tl
10048   Inside~\keys_define:nn~each~key~name
```

```

10049     needs-a~property:  \\  

10050     ~ ~ #1 .<property> \\  

10051     LaTeX~did~not~find~a~'. '~to~indicate~the~start~of~a~property.  

10052 }  

10053 \msg_kernel_new:nnnn { keys } { key-unknown }  

10054 { The~key~'#1'~is~unknown~and~is~being~ignored. }  

10055 {  

10056     The~module~'#2'~does~not~have~a~key~called~'#1'.\\  

10057     Check~that~you~have~spelled~the~key~name~correctly.  

10058 }  

10059 \msg_kernel_new:nnnn { keys } { option-unknown }  

10060 { Unknown~option~'#1'~for~package~#2. }  

10061 {  

10062     LaTeX~has~been~asked~to~set~an~option~called~'#1'~  

10063     but~the~#2~package~has~not~created~an~option~with~this~name.  

10064 }  

10065 \msg_kernel_new:nnnn { keys } { property-requires-value }  

10066 { The~property~'#1'~requires~a~value. }  

10067 {  

10068     \c_msg_coding_error_text_tl  

10069     LaTeX~was~asked~to~set~property~'#2'~for~key~'#1'.\\  

10070     No~value~was~given~for~the~property,~and~one~is~required.  

10071 }  

10072 \msg_kernel_new:nnnn { keys } { property-unknown }  

10073 { The~key~property~'#1'~is~unknown. }  

10074 {  

10075     \c_msg_coding_error_text_tl  

10076     LaTeX~has~been~asked~to~set~the~property~'#1'~for~key~'#2':~  

10077     this~property~is~not~defined.  

10078 }  

10079 \msg_kernel_new:nnnn { keys } { value-forbidden }  

10080 { The~key~'#1'~does~not~taken~a~value. }  

10081 {  

10082     The~key~'#1'~should~be~given~without~a~value.\\  

10083     LaTeX~will~ignore~the~given~value~'#2'.  

10084 }  

10085 \msg_kernel_new:nnnn { keys } { value-required }  

10086 { The~key~'#1'~requires~a~value. }  

10087 {  

10088     The~key~'#1'~must~have~a~value.\\  

10089     No~value~was~present:~the~key~will~be~ignored.  

10090 }

```

201.9 Deprecated functions

Deprecated on 2011-05-27, for removal by 2011-08-31.

There is just one function for this now.

```

10091 <*deprecated>

```

```

\KV_process_space_removal_sanitiz:NNn
\KV_process_space_removal_no_sanitiz:NNn
\KV_process_no_space_removal_no_sanitiz:NNn

```



```

10092 \cs_new_eq:NN \KV_process_space_removal_sanitize:NNn \keyval_parse:NNn
10093 \cs_new_eq:NN \KV_process_space_removal_no_sanitize:NNn \keyval_parse:NNn
10094 \cs_new_eq:NN \KV_process_no_space_removal_no_sanitize:NNn \keyval_parse:NNn
10095 </deprecated>
      (End definition for \KV_process_space_removal_sanitize:NNn. This function is documented on
page ??.)
10096 </initex | package>

```

202 l3file implementation

The following test files are used for this code: *m3file001*.

```

10097 <*initex | package>
10098 <*package>
10099 \ProvidesExplPackage
10100   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
10101   \package_check_loaded_expl:
10102 </package>

```

`\g_file_current_name_tl` The name of the current file should be available at all times.

```

10103 \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.

```

10104 <*initex>
10105 \tex_everyjob:D \exp_after:wN
10106   {
10107     \tex_the:D \tex_everyjob:D
10108     \tl_gset:Nx \g_file_current_name_tl { \tex_jobname:D }
10109   }
10110 </initex>
10111 <*package>
10112 \tl_gset_eq:NN \g_file_current_name_tl \@currname
10113 </package>

```

(End definition for `\g_file_current_name_tl`. This function is documented on page 162.)

`\g_file_stack_seq` The input list of files is stored as a sequence stack.

```

10114 \seq_new:N \g_file_stack_seq

```

(End definition for `\g_file_stack_seq`. This function is documented on page 163.)

`\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.

```

10115 \seq_new:N \g_file_record_seq

```

The current file name should be included in the file list!

```

10116 <*initex>
10117 \tex_everyjob:D \exp_after:wN
10118 {
10119   \tex_the:D \tex_everyjob:D
10120   \seq_gput_right:NV \g_file_record_seq \g_file_current_name_tl
10121 }
10122 </initex>
      (End definition for \g_file_record_seq. This function is documented on page 164.)

```

`\l_file_internal_name_tl` Used to return the fully-qualified name of a file.

```

10123 \tl_new:N \l_file_internal_name_tl
      (End definition for \l_file_internal_name_tl. This function is documented on page 164.)

```

`\l_file_search_path_seq` The current search path.

```

10124 \seq_new:N \l_file_search_path_seq
      (End definition for \l_file_search_path_seq. This function is documented on page 164.)

```

`\l_file_internal_saved_path_seq` The current search path has to be saved for package use.

```

10125 <*package>
10126 \seq_new:N \l_file_internal_saved_path_seq
10127 </package>
      (End definition for \l_file_internal_saved_path_seq. This function is documented on page 164.)

```

`\l_file_internal_seq` Scratch space for comma list conversion in package mode.

```

10128 <*package>
10129 \seq_new:N \l_file_internal_seq
10130 </package>
      (End definition for \l_file_internal_seq. This function is documented on page 164.)

```

`\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
`\g_file_internal_ior` report end-of-file. For files which are in the current directory, this is straight-forward.
`\file_add_path_aux:nN` For other locations, a search has to be made looking at each potential path in turn. The
`\file_add_path_search:nN` first location is of course treated as the correct one. If nothing is found, #2 is returned
empty.

```

10131 \ior_new:N \g_file_internal_ior
10132 \cs_new_protected:Npn \file_add_path:nN #1
10133 {
10134   \group_begin:
10135     \tl_to_str_active_safe:Nx \l_file_internal_name_tl {#1}
10136     \tl_if_in:NnT \l_file_internal_name_tl { ~ }
10137     {
10138       \msg_kernel_error:nmx { io } { space-in-file-name }
10139       { \l_file_internal_name_tl }
10140     }
10141     \exp_args:NNo \group_end:
10142     \file_add_path_aux:nN \l_file_internal_name_tl
10143 }

```

```

10144 \cs_new_protected:Npn \file_add_path_aux:nN #1#2
10145 {
10146   \ior_open_unsafe:Nn \g_file_internal_ior {#1}
10147   \ior_if_eof:NTF \g_file_internal_ior
10148     { \file_add_path_search:nN {#1} #2 }
10149     {
10150       \ior_close:N \g_file_internal_ior
10151       \tl_set:Nn #2 {#1}
10152     }
10153 }
10154 \cs_new_protected:Npn \file_add_path_search:nN #1#2
10155 {
10156   \tl_clear:N #2
10157   <*package>
10158   \cs_if_exist:NT \input@path
10159   {
10160     \seq_set_eq:NN \l_file_internal_saved_path_seq \l_file_search_path_seq
10161     \seq_set_from_clist:NN \l_file_internal_seq \input@path
10162     \seq_concat:NNN \l_file_search_path_seq
10163       \l_file_search_path_seq \l_file_internal_seq
10164   }
10165   </package>
10166   \seq_map_inline:Nn \l_file_search_path_seq
10167   {
10168     \ior_open_unsafe:Nn \g_file_internal_ior { ##1 #1 }
10169     \ior_if_eof:NF \g_file_internal_ior
10170     {
10171       \tl_set:Nx #2 { ##1 #1 }
10172       \seq_map_break:
10173     }
10174   }
10175   <*package>
10176   \cs_if_exist:NT \input@path
10177   { \seq_set_eq:NN \l_file_search_path_seq \l_file_internal_saved_path_seq }
10178   </package>
10179   \ior_close:N \g_file_internal_ior
10180 }

```

(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.

```

10181 \prg_new_protected_conditional:Nnn \file_if_exist:n { T , F , TF }
10182 {
10183   \file_add_path:nN {#1} \l_file_internal_name_tl
10184   \tl_if_empty:NTF \l_file_internal_name_tl
10185     { \prg_return_false: }
10186     { \prg_return_true: }
10187 }

```

(End definition for \file_if_exist:n. This function is documented on page 162.)

\file_input:n Loading a file is done in a safe way, checking first that the file exists and loading only if it does.

```

10188 \cs_new_protected:Npn \file_input:n #1
10189 {
10190   \file_add_path:nN {#1} \l_file_internal_name_tl
10191   \tl_if_empty:NF \l_file_internal_name_tl
10192   {
10193     \*initex
10194     \seq_gput_right:Nx \g_file_record_seq {#1}
10195     \*initex
10196     \*package
10197     \@addtofilelist {#1}
10198     \*package
10199     \seq_gpush:NV \g_file_stack_seq \g_file_current_name_tl
10200     \tl_gset:Nn \g_file_current_name_tl {#1}
10201     \exp_after:wN \tex_input:D \l_file_internal_name_tl \c_space_tl
10202     \seq_gpop:NN \g_file_stack_seq \g_file_current_name_tl
10203   }
10204 }

```

(End definition for \file_input:n. This function is documented on page 163.)

\file_split_path_name_ext:nNNN Splits a file name into any path part, the file name and the extension, which is considered as the part after the last ..

```

\file_split_path_name_ext:nNNN
\file_split_path:wNNN
\file_split_name_ext:wNN
10205 \cs_new_protected:Npn \file_split_path_name_ext:nNNN #1
10206 {
10207   \group_begin:
10208   \tl_to_str_active_safe:Nx \l_file_internal_name_tl {#1}
10209   \tl_if_in:NnT \l_file_internal_name_tl { ~ }
10210   {
10211     \msg_kernel_error:nxx { io } { space-in-file-name }
10212     { \l_file_internal_name_tl }
10213   }
10214   \exp_args:NNo \group_end:
10215   \file_split_path_name_ext_aux:nNNN \l_file_internal_name_tl
10216 }
10217 \cs_new_protected:Npn \file_split_path_name_ext_aux:nNNN #1#2#3#4
10218 {
10219   \tl_clear:N #2
10220   \tl_clear:N #3
10221   \file_split_path:wNNN #1 / \q_nil / \q_stop #2#3#4
10222 }
10223 \cs_new_protected:Npn \file_split_path:wNNN #1 / #2 / #3 \q_stop #4
10224 {
10225   \quark_if_nil:nTF {#2}
10226   { \file_split_name_ext:wNN #1 . \q_nil . \q_stop }
10227   {
10228     \tl_put_right:Nn #4 { #1 / }

```

```

10229         \file_split_path:wNNN #2 / #3 \q_stop #4
10230     }
10231 }
10232 \cs_new_protected:Npn \file_split_name_ext:wNN #1 . #2 . #3 \q_stop #4#5
10233 {
10234     \quark_if_nil:nTF {#2}
10235     {
10236         \tl_if_empty:NTF #4
10237         {
10238             \tl_set:Nn #4 {#1}
10239             \tl_clear:N #5
10240         }
10241         { \tl_set:Nn #5 {#1} }
10242     }
10243     {
10244         \tl_put_right:Nx #4
10245         {
10246             \tl_if_empty:NF #4 { . }
10247             #1
10248         }
10249         \file_split_name_ext:wNN #2 . #3 \q_stop #4#5
10250     }
10251 }

```

(End definition for \file_split_path_name_ext:nNNN. This function is documented on page ??.)

\file_path_include:n Wrapper functions to manage the search path.

```

\file_path_remove:n
10252 \cs_new_protected:Npn \file_path_include:n #1
10253 {
10254     \seq_if_in:NnF \l_file_search_path_seq {#1}
10255     { \seq_put_right:Nn \l_file_search_path_seq {#1} }
10256 }
10257 \cs_new_protected:Npn \file_path_remove:n #1
10258 { \seq_remove_all:Nn \l_file_search_path_seq {#1} }

```

(End definition for \file_path_include:n. This function is documented on page 163.)

\file_list: A function to list all files used to the log.

```

10259 \cs_new_protected_nopar:Npn \file_list:
10260 {
10261     \seq_remove_duplicates:N \g_file_record_seq
10262     \iow_log:n { *~File~List~* }
10263     \seq_map_inline:Nn \g_file_record_seq { \iow_log:n {##1} }
10264     \iow_log:n { ***** }
10265 }

```

(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.

```

10266 <*package>
10267 \AtBeginDocument
10268 {

```

```

10269 \seq_set_from_clist:NN \l_file_internal_seq \@filelist
10270 \seq_gconcat:NNN \g_file_record_seq \g_file_record_seq \l_file_internal_seq
10271 }
10272 </package>
10273 </initex | package>

```

203 l3fp Implementation

The following test files are used for this code: *m3fp003.lvt*.

```

10274 <*initex | package>
10275 <*package>
10276 \ProvidesExplPackage
10277 {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
10278 \package_check_loaded_expl:
10279 </package>

```

203.1 Constants

`\c_forty_four` There is some speed to gain by moving numbers into fixed positions.

```

\c_one_million 10280 \int_const:Nn \c_forty_four { 44 }
\c_one_hundred_million 10281 \int_const:Nn \c_one_million { 1 000 000 }
\c_five_hundred_million 10282 \int_const:Nn \c_one_hundred_million { 100 000 000 }
\c_one_thousand_million 10283 \int_const:Nn \c_five_hundred_million { 500 000 000 }
10284 \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 10285 \int_new:N \c_fp_pi_by_four_decimal_int
\c_fp_pi_decimal_int 10286 \int_set:Nn \c_fp_pi_by_four_decimal_int { 785 398 158 }
\c_fp_pi_extended_int 10287 \int_new:N \c_fp_pi_by_four_extended_int
\c_fp_two_pi_decimal_int 10288 \int_set:Nn \c_fp_pi_by_four_extended_int { 897 448 310 }
\c_fp_two_pi_extended_int 10289 \int_new:N \c_fp_pi_decimal_int
10290 \int_set:Nn \c_fp_pi_decimal_int { 141 592 653 }
10291 \int_new:N \c_fp_pi_extended_int
10292 \int_set:Nn \c_fp_pi_extended_int { 589 793 238 }
10293 \int_new:N \c_fp_two_pi_decimal_int
10294 \int_set:Nn \c_fp_two_pi_decimal_int { 283 185 307 }
10295 \int_new:N \c_fp_two_pi_extended_int
10296 \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”.

```

10297 \tl_const:Nn \c_e_fp { + 2.718281828 e 0 }
(End definition for \c_e_fp. This function is documented on page 170.)

```

`\c_one_fp` The constant value 1: used for fast comparisons.

```

10298 \tl_const:Nn \c_one_fp { + 1.000000000 e 0 }

```

(End definition for `\c_one_fp`. This function is documented on page 170.)

`\c_pi_fp` The value π as a “machine number”.

10299 `\tl_const:Nn \c_pi_fp { + 3.141592654 e 0 }`

(End definition for `\c_pi_fp`. This function is documented on page 170.)

`\c_undefined_fp` A marker for undefined values.

10300 `\tl_const:Nn \c_undefined_fp { X 0.000000000 e 0 }`

(End definition for `\c_undefined_fp`. This function is documented on page 171.)

`\c_zero_fp` The constant zero value.

10301 `\tl_const:Nn \c_zero_fp { + 0.000000000 e 0 }`

(End definition for `\c_zero_fp`. This function is documented on page 171.)

203.2 Variables

`\l_fp_arg_tl` A token list to store the formalised representation of the input for transcendental functions.

10302 `\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.

10303 `\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.

10304 `\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` 10305 `\int_new:N \l_fp_exp_integer_int`

`\l_fp_exp_extended_int` 10306 `\int_new:N \l_fp_exp_decimal_int`

`\l_fp_exp_exponent_int` 10307 `\int_new:N \l_fp_exp_extended_int`

10308 `\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` 10309 `\int_new:N \l_fp_input_a_sign_int`

`\l_fp_input_a_decimal_int` 10310 `\int_new:N \l_fp_input_a_integer_int`

`\l_fp_input_a_exponent_int` 10311 `\int_new:N \l_fp_input_a_decimal_int`

`\l_fp_input_b_sign_int` 10312 `\int_new:N \l_fp_input_a_exponent_int`

`\l_fp_input_b_integer_int` 10313 `\int_new:N \l_fp_input_b_sign_int`

`\l_fp_input_b_decimal_int` 10314 `\int_new:N \l_fp_input_b_integer_int`

`\l_fp_input_b_exponent_int` 10315 `\int_new:N \l_fp_input_b_decimal_int`

10316 `\int_new:N \l_fp_input_b_exponent_int`

(End definition for `\l_fp_input_a_sign_int`. This function is documented on page ??.)

<pre> \l_fp_input_a_extended_int \l_fp_input_b_extended_int </pre>	<p>For internal use, “extended” floating point numbers are needed.</p> <pre> 10317 \int_new:N \l_fp_input_a_extended_int 10318 \int_new:N \l_fp_input_b_extended_int </pre> <p><i>(End definition for \l_fp_input_a_extended_int. This function is documented on page ??.)</i></p>
<pre> \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_b_i_int \l_fp_mul_b_ii_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 </pre>	<p>Multiplication requires that the decimal part is split into parts so that there are no overflows.</p> <pre> 10319 \int_new:N \l_fp_mul_a_i_int 10320 \int_new:N \l_fp_mul_a_ii_int 10321 \int_new:N \l_fp_mul_a_iii_int 10322 \int_new:N \l_fp_mul_a_iv_int 10323 \int_new:N \l_fp_mul_a_v_int 10324 \int_new:N \l_fp_mul_a_vi_int 10325 \int_new:N \l_fp_mul_b_i_int 10326 \int_new:N \l_fp_mul_b_ii_int 10327 \int_new:N \l_fp_mul_b_iii_int 10328 \int_new:N \l_fp_mul_b_iv_int 10329 \int_new:N \l_fp_mul_b_v_int 10330 \int_new:N \l_fp_mul_b_vi_int </pre> <p><i>(End definition for \l_fp_mul_a_i_int. This function is documented on page ??.)</i></p>
<pre> \l_fp_mul_output_int \l_fp_mul_output_tl </pre>	<p>Space for multiplication results.</p> <pre> 10331 \int_new:N \l_fp_mul_output_int 10332 \tl_new:N \l_fp_mul_output_tl </pre> <p><i>(End definition for \l_fp_mul_output_int. This function is documented on page ??.)</i></p>
<pre> \l_fp_output_sign_int \l_fp_output_integer_int \l_fp_output_decimal_int \l_fp_output_exponent_int </pre>	<p>Output is stored in the same way as input.</p> <pre> 10333 \int_new:N \l_fp_output_sign_int 10334 \int_new:N \l_fp_output_integer_int 10335 \int_new:N \l_fp_output_decimal_int 10336 \int_new:N \l_fp_output_exponent_int </pre> <p><i>(End definition for \l_fp_output_sign_int. This function is documented on page ??.)</i></p>
<pre> \l_fp_output_extended_int </pre>	<p>Again, for calculations an extended part.</p> <pre> 10337 \int_new:N \l_fp_output_extended_int </pre> <p><i>(End definition for \l_fp_output_extended_int. This function is documented on page ??.)</i></p>
<pre> \l_fp_round_carry_bool </pre>	<p>To indicate that a digit needs to be carried forward.</p> <pre> 10338 \bool_new:N \l_fp_round_carry_bool </pre> <p><i>(End definition for \l_fp_round_carry_bool. This function is documented on page ??.)</i></p>
<pre> \l_fp_round_decimal_tl </pre>	<p>A temporary store when rounding, to build up the decimal part without needing to do any maths.</p> <pre> 10339 \tl_new:N \l_fp_round_decimal_tl </pre> <p><i>(End definition for \l_fp_round_decimal_tl. This function is documented on page ??.)</i></p>

<code>\l_fp_round_position_int</code>	Used to check the position for rounding.
<code>\l_fp_round_target_int</code>	10340 <code>\int_new:N \l_fp_round_position_int</code> 10341 <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. 10342 <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. 10343 <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 ??.)
<code>\l_fp_internal_int</code>	A scratch int: used only where the value is not carried forward. 10344 <code>\int_new:N \l_fp_internal_int</code> (End definition for <code>\l_fp_internal_int</code> . This function is documented on page ??.)
<code>\l_fp_internal_tl</code>	A scratch token list variable for expanding material. 10345 <code>\tl_new:N \l_fp_internal_tl</code> (End definition for <code>\l_fp_internal_tl</code> . This function is documented on page ??.)
<code>\l_fp_trig_octant_int</code>	To track which octant the trigonometric input is in. 10346 <code>\int_new:N \l_fp_trig_octant_int</code> (End definition for <code>\l_fp_trig_octant_int</code> . This function is documented on page ??.)
<code>\l_fp_trig_sign_int</code>	Used for the calculation of trigonometric values.
<code>\l_fp_trig_decimal_int</code>	10347 <code>\int_new:N \l_fp_trig_sign_int</code>
<code>\l_fp_trig_extended_int</code>	10348 <code>\int_new:N \l_fp_trig_decimal_int</code>
	10349 <code>\int_new:N \l_fp_trig_extended_int</code> (End definition for <code>\l_fp_trig_sign_int</code> . This function is documented on page ??.)

203.3 Parsing numbers

<code>\fp_read:N</code>	Reading a stored value is made easier as the format is designed to match the delimited
<code>\fp_read_aux:w</code>	function. This is always used to read the first value (register a). 10350 <code>\cs_new_protected:Npn \fp_read:N #1</code> 10351 <code>{ \exp_after:wN \fp_read_aux:w #1 \q_stop }</code> 10352 <code>\cs_new_protected:Npn \fp_read_aux:w #1#2 . #3 e #4 \q_stop</code> 10353 <code>{</code> 10354 <code> \if:w #1 -</code> 10355 <code> \l_fp_input_a_sign_int \c_minus_one</code> 10356 <code> \else:</code> 10357 <code> \l_fp_input_a_sign_int \c_one</code> 10358 <code> \fi:</code> 10359 <code> \l_fp_input_a_integer_int #2 \scan_stop:</code>

```

10360 \l_fp_input_a_decimal_int #3 \scan_stop:
10361 \l_fp_input_a_exponent_int #4 \scan_stop:
10362 }

```

(End definition for \fp_read:N. This function is documented on page ??.)

```

\fp_split:Nn
\fp_split_sign:
\fp_split_exponent:
\fp_split_aux_i:w
\fp_split_aux_ii:w
\fp_split_aux_iii:w
\fp_split_decimal:w
\fp_split_decimal_aux:w

```

The aim here is to use as much of T_EX's mechanism as possible to pick up the numerical input without any mistakes. In particular, negative numbers have to be filtered out first in case the integer part is 0 (in which case T_EX would drop the - sign). That process has to be done in a loop for cases where the sign is repeated. Finding an exponent is relatively easy, after which the next phase is to find the integer part, which will terminate with a ., and trigger the decimal-finding code. The later will allow the decimal to be too long, truncating the result.

```

10363 \cs_new_protected:Npn \fp_split:Nn #1#2
10364 {
10365   \tl_set:Nx \l_fp_internal_tl {#2}
10366   \tl_set_rescan:Nno \l_fp_internal_tl { \char_set_catcode_ignore:n { 32 } }
10367   { \l_fp_internal_tl }
10368   \l_fp_split_sign_int \c_one
10369   \fp_split_sign:
10370   \use:c { l_fp_input_ #1 _sign_int } \l_fp_split_sign_int
10371   \exp_after:wN \fp_split_exponent:w \l_fp_internal_tl e e \q_stop #1
10372 }
10373 \cs_new_protected_nopar:Npn \fp_split_sign:
10374 {
10375   \if_int_compare:w \pdfTeX_strcmp:D
10376   { \exp_after:wN \tl_head:w \l_fp_internal_tl ? \q_stop } { - }
10377   = \c_zero
10378   \tl_set:Nx \l_fp_internal_tl
10379   {
10380     \exp_after:wN
10381     \tl_tail:w \l_fp_internal_tl \prg_do_nothing: \q_stop
10382   }
10383   \l_fp_split_sign_int -\l_fp_split_sign_int
10384   \exp_after:wN \fp_split_sign:
10385   \else:
10386   \if_int_compare:w \pdfTeX_strcmp:D
10387   { \exp_after:wN \tl_head:w \l_fp_internal_tl ? \q_stop } { + }
10388   = \c_zero
10389   \tl_set:Nx \l_fp_internal_tl
10390   {
10391     \exp_after:wN
10392     \tl_tail:w \l_fp_internal_tl \prg_do_nothing: \q_stop
10393   }
10394   \exp_after:wN \exp_after:wN \exp_after:wN \fp_split_sign:
10395   \fi:
10396   \fi:
10397 }
10398 \cs_new_protected:Npn \fp_split_exponent:w #1 e #2 e #3 \q_stop #4
10399 {

```

```

10400 \use:c { l_fp_input_ #4 _exponent_int }
10401 \int_eval:w 0 #2 \scan_stop:
10402 \tex_afterassignment:D \fp_split_aux_i:w
10403 \use:c { l_fp_input_ #4 _integer_int }
10404 \int_eval:w 0 #1 . . \q_stop #4
10405 }
10406 \cs_new_protected:Npn \fp_split_aux_i:w #1 . #2 . #3 \q_stop
10407 { \fp_split_aux_ii:w #2 000000000 \q_stop }
10408 \cs_new_protected:Npn \fp_split_aux_ii:w #1#2#3#4#5#6#7#8#9
10409 { \fp_split_aux_iii:w {#1#2#3#4#5#6#7#8#9} }
10410 \cs_new_protected:Npn \fp_split_aux_iii:w #1#2 \q_stop
10411 {
10412 \l_fp_internal_int 1 #1 \scan_stop:
10413 \exp_after:wN \fp_split_decimal:w
10414 \int_use:N \l_fp_internal_int 000000000 \q_stop
10415 }
10416 \cs_new_protected:Npn \fp_split_decimal:w #1#2#3#4#5#6#7#8#9
10417 { \fp_split_decimal_aux:w {#2#3#4#5#6#7#8#9} }
10418 \cs_new_protected:Npn \fp_split_decimal_aux:w #1#2#3 \q_stop #4
10419 {
10420 \use:c { l_fp_input_ #4 _decimal_int } #1#2 \scan_stop:
10421 \if_int_compare:w
10422 \int_eval:w
10423 \use:c { l_fp_input_ #4 _integer_int } +
10424 \use:c { l_fp_input_ #4 _decimal_int }
10425 \scan_stop:
10426 = \c_zero
10427 \use:c { l_fp_input_ #4 _sign_int } \c_one
10428 \fi:
10429 \if_int_compare:w
10430 \use:c { l_fp_input_ #4 _integer_int } < \c_one_thousand_million
10431 \else:
10432 \exp_after:wN \fp_overflow_msg:
10433 \fi:
10434 }

```

(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
\fp_standardise_aux:NNNN tokens where possible, as this is faster than arithmetic.

```

\fp_standardise_aux:
\fp_standardise_aux:w
10435 \cs_new_protected:Npn \fp_standardise:NNNN #1#2#3#4
10436 {
10437 \if_int_compare:w
10438 \int_eval:w #2 + #3 = \c_zero
10439 #1 \c_one
10440 #4 \c_zero
10441 \exp_after:wN \use_none:nnnn
10442 \else:
10443 \exp_after:wN \fp_standardise_aux:NNNN
10444 \fi:

```

```

10445     #1#2#3#4
10446   }
10447 \cs_new_protected:Npn \fp_standardise_aux:NNNN #1#2#3#4
10448 {
10449   \cs_set_protected_nopar:Npn \fp_standardise_aux:
10450   {
10451     \if_int_compare:w #2 = \c_zero
10452       \tex_advance:D #3 \c_one_thousand_million
10453       \exp_after:wN \fp_standardise_aux:w
10454       \int_use:N #3 \q_stop
10455       \exp_after:wN \fp_standardise_aux:
10456     \fi:
10457   }
10458 \cs_set_protected:Npn
10459   \fp_standardise_aux:w ##1##2##3##4##5##6##7##8##9 \q_stop
10460 {
10461   #2 ##2 \scan_stop:
10462   #3 ##3##4##5##6##7##8##9 0 \scan_stop:
10463   \tex_advance:D #4 \c_minus_one
10464 }
10465 \fp_standardise_aux:
10466 \cs_set_protected_nopar:Npn \fp_standardise_aux:
10467 {
10468   \if_int_compare:w #2 > \c_nine
10469     \tex_advance:D #2 \c_one_thousand_million
10470     \exp_after:wN \use_i:nn \exp_after:wN
10471       \fp_standardise_aux:w \int_use:N #2
10472     \exp_after:wN \fp_standardise_aux:
10473   \fi:
10474 }
10475 \cs_set_protected:Npn
10476   \fp_standardise_aux:w ##1##2##3##4##5##6##7##8##9
10477 {
10478   #2 ##1##2##3##4##5##6##7##8 \scan_stop:
10479   \tex_advance:D #3 \c_one_thousand_million
10480   \tex_divide:D #3 \c_ten
10481   \tl_set:Nx \l_fp_internal_tl
10482   {
10483     ##9
10484     \exp_after:wN \use_none:n \int_use:N #3
10485   }
10486   #3 \l_fp_internal_tl \scan_stop:
10487   \tex_advance:D #4 \c_one
10488 }
10489 \fp_standardise_aux:
10490 \if_int_compare:w #4 < \c_one_hundred
10491   \if_int_compare:w #4 > -\c_one_hundred
10492   \else:
10493     #1 \c_one
10494     #2 \c_zero

```

```

10495         #3 \c_zero
10496         #4 \c_zero
10497         \fi:
10498     \else:
10499         \exp_after:wN \fp_overflow_msg:
10500     \fi:
10501 }
10502 \cs_new_protected_nopar:Npn \fp_standardise_aux: { }
10503 \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:NNNNNNNNN`
`\fp_level_input_exponents_b:`
`\fp_level_input_exponents_b:NNNNNNNNN`

```

10504 \cs_new_protected_nopar:Npn \fp_level_input_exponents:
10505 {
10506     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
10507         \exp_after:wN \fp_level_input_exponents_a:
10508     \else:
10509         \exp_after:wN \fp_level_input_exponents_b:
10510     \fi:
10511 }
10512 \cs_new_protected_nopar:Npn \fp_level_input_exponents_a:
10513 {
10514     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
10515         \tex_advance:D \l_fp_input_b_integer_int \c_one_thousand_million
10516         \exp_after:wN \use_i:nn \exp_after:wN
10517             \fp_level_input_exponents_a:NNNNNNNNN
10518         \int_use:N \l_fp_input_b_integer_int
10519         \exp_after:wN \fp_level_input_exponents_a:
10520     \fi:
10521 }
10522 \cs_new_protected:Npn \fp_level_input_exponents_a:NNNNNNNNN
10523     #1#2#3#4#5#6#7#8#9
10524 {
10525     \l_fp_input_b_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
10526     \tex_advance:D \l_fp_input_b_decimal_int \c_one_thousand_million
10527     \tex_divide:D \l_fp_input_b_decimal_int \c_ten
10528     \tl_set:Nx \l_fp_internal_tl
10529     {
10530         #9
10531         \exp_after:wN \use_none:n
10532         \int_use:N \l_fp_input_b_decimal_int
10533     }
10534     \l_fp_input_b_decimal_int \l_fp_internal_tl \scan_stop:
10535     \tex_advance:D \l_fp_input_b_exponent_int \c_one
10536 }
10537 \cs_new_protected_nopar:Npn \fp_level_input_exponents_b:

```

```

10538 {
10539   \if_int_compare:w \l_fp_input_b_exponent_int > \l_fp_input_a_exponent_int
10540     \tex_advance:D \l_fp_input_a_integer_int \c_one_thousand_million
10541     \exp_after:wN \use_i:nn \exp_after:wN
10542     \fp_level_input_exponents_b:NNNNNNNNN
10543     \int_use:N \l_fp_input_a_integer_int
10544     \exp_after:wN \fp_level_input_exponents_b:
10545   \fi:
10546 }
10547 \cs_new_protected:Npn \fp_level_input_exponents_b:NNNNNNNNN
10548 #1#2#3#4#5#6#7#8#9
10549 {
10550   \l_fp_input_a_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
10551   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10552   \tex_divide:D \l_fp_input_a_decimal_int \c_ten
10553   \tl_set:Nx \l_fp_internal_tl
10554   {
10555     #9
10556     \exp_after:wN \use_none:n
10557     \int_use:N \l_fp_input_a_decimal_int
10558   }
10559   \l_fp_input_a_decimal_int \l_fp_internal_tl \scan_stop:
10560   \tex_advance:D \l_fp_input_a_exponent_int \c_one
10561 }

```

(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.

```

10562 \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
10563 \cs_new_protected:Npn \fp_new:N #1
10564 {
10565   \tl_new:N #1
10566   \tl_gset_eq:NN #1 \c_zero_fp
10567 }
10568 \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`

```

10569 \cs_new_protected:Npn \fp_const:Nn #1#2
10570 {
10571   \fp_new:N #1
10572   \fp_gset:Nn #1 {#2}
10573 }
10574 \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`

`\fp_gzero:N`

`\fp_gzero:c`

```

10575 \cs_new_protected:Npn \fp_zero:N #1
10576 { \tl_set_eq:NN #1 \c_zero_fp }
10577 \cs_new_protected:Npn \fp_gzero:N #1
10578 { \tl_gset_eq:NN #1 \c_zero_fp }
10579 \cs_generate_variant:Nn \fp_zero:N { c }
10580 \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_zero_new:N` Create a floating point if needed, otherwise clear it.

`\fp_zero_new:c`

`\fp_gzero_new:N`

`\fp_gzero_new:c`

```

10581 \cs_new_protected:Npn \fp_zero_new:N #1
10582 { \cs_if_exist:NTF #1 { \fp_zero:N #1 } { \fp_new:N #1 } }
10583 \cs_new_protected:Npn \fp_gzero_new:N #1
10584 { \cs_if_exist:NTF #1 { \fp_gzero:N #1 } { \fp_new:N #1 } }
10585 \cs_generate_variant:Nn \fp_zero_new:N { c }
10586 \cs_generate_variant:Nn \fp_gzero_new:N { c }

```

(End definition for \fp_zero_new:N and others. 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`

```

10587 \cs_new_protected_nopar:Npn \fp_set:Nn { \fp_set_aux:NNn \tl_set:Nn }
10588 \cs_new_protected_nopar:Npn \fp_gset:Nn { \fp_set_aux:NNn \tl_gset:Nn }
10589 \cs_new_protected:Npn \fp_set_aux:NNn #1#2#3
10590 {
10591   \group_begin:
10592     \fp_split:Nn a {#3}
10593     \fp_standardise:NNNN
10594     \l_fp_input_a_sign_int
10595     \l_fp_input_a_integer_int
10596     \l_fp_input_a_decimal_int
10597     \l_fp_input_a_exponent_int
10598     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10599     \cs_set_protected_nopar:Npx \fp_tmp:w
10600     {
10601       \group_end:
10602       #1 \exp_not:N #2
10603       {
10604         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero

```

```

10605         -
10606         \else:
10607         +
10608         \fi:
10609         \int_use:N \l_fp_input_a_integer_int
10610         .
10611         \exp_after:wN \use_none:n
10612         \int_use:N \l_fp_input_a_decimal_int
10613         e
10614         \int_use:N \l_fp_input_a_exponent_int
10615     }
10616 }
10617 \fp_tmp:w
10618 }
10619 \cs_generate_variant:Nn \fp_set:Nn { c }
10620 \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 10621 \cs_new_protected_nopar:Npn \fp_set_from_dim:Nn
\fp_set_from_dim_aux:w 10622 { \fp_set_from_dim_aux:NNn \tl_set:Nx }
  \l_fp_internal_dim 10623 \cs_new_protected_nopar:Npn \fp_gset_from_dim:Nn
  \l_fp_internal_skip 10624 { \fp_set_from_dim_aux:NNn \tl_gset:Nx }
10625 \cs_new_protected:Npn \fp_set_from_dim_aux:NNn #1#2#3
10626 {
10627   \group_begin:
10628     \l_fp_internal_skip \etex_glueexpr:D #3 \scan_stop:
10629     \l_fp_internal_dim \l_fp_internal_skip
10630     \fp_split:Nn a
10631     {
10632       \exp_after:wN \fp_set_from_dim_aux:w
10633       \dim_use:N \l_fp_internal_dim
10634     }
10635     \fp_standardise:NNNN
10636     \l_fp_input_a_sign_int
10637     \l_fp_input_a_integer_int
10638     \l_fp_input_a_decimal_int
10639     \l_fp_input_a_exponent_int
10640     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
10641     \cs_set_protected_nopar:Npx \fp_tmp:w
10642     {
10643       \group_end:
10644       #1 \exp_not:N #2
10645       {
10646         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
10647         -

```



```

10648         \else:
10649             +
10650         \fi:
10651         \int_use:N \l_fp_input_a_integer_int
10652         .
10653         \exp_after:wN \use_none:n
10654         \int_use:N \l_fp_input_a_decimal_int
10655         e
10656         \int_use:N \l_fp_input_a_exponent_int
10657     }
10658 }
10659 \fp_tmp:w
10660 }
10661 \cs_set_protected_nopar:Npx \fp_set_from_dim_aux:w
10662 {
10663     \cs_set:Npn \exp_not:N \fp_set_from_dim_aux:w
10664     ##1 \tl_to_str:n { pt } {##1}
10665 }
10666 \fp_set_from_dim_aux:w
10667 \cs_generate_variant:Nn \fp_set_from_dim:Nn { c }
10668 \cs_generate_variant:Nn \fp_gset_from_dim:Nn { c }
10669 \dim_new:N \l_fp_internal_dim
10670 \skip_new:N \l_fp_internal_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 10671 \cs_new_eq:NN \fp_set_eq:NN \tl_set_eq:NN
\fp_set_eq:Nc 10672 \cs_new_eq:NN \fp_set_eq:cN \tl_set_eq:cN
\fp_set_eq:cc 10673 \cs_new_eq:NN \fp_set_eq:Nc \tl_set_eq:Nc
\fp_gset_eq:NN 10674 \cs_new_eq:NN \fp_set_eq:cc \tl_set_eq:cc
\fp_gset_eq:cN 10675 \cs_new_eq:NN \fp_gset_eq:NN \tl_gset_eq:NN
\fp_gset_eq:Nc 10676 \cs_new_eq:NN \fp_gset_eq:cN \tl_gset_eq:cN
\fp_gset_eq:Nc 10677 \cs_new_eq:NN \fp_gset_eq:Nc \tl_gset_eq:Nc
\fp_gset_eq:cc 10678 \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 10679 \cs_new_eq:NN \fp_show:N \tl_show:N
10680 \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 10681 \cs_new:Npn \fp_use:N #1
\fp_use_aux:w { \exp_after:wN \fp_use_aux:w #1 \q_stop }
\fp_use_none:w 10682
\fp_use_small:w 10683 \cs_generate_variant:Nn \fp_use:N { c }
\fp_use_large:w 10684 \cs_new:Npn \fp_use_aux:w #1#2 e #3 \q_stop
\fp_use_large_aux_i:w
\fp_use_large_aux_1:w
\fp_use_large_aux_2:w
\fp_use_large_aux_3:w
\fp_use_large_aux_4:w
\fp_use_large_aux_5:w
\fp_use_large_aux_6:w
\fp_use_large_aux_7:w
\fp_use_large_aux_8:w
\fp_use_large_aux_i:w
\fp_use_large_aux_ii:w

```

```

10685 {
10686   \if:w #1 -
10687   -
10688   \fi:
10689   \if_int_compare:w #3 > \c_zero
10690     \exp_after:wN \fp_use_large:w
10691   \else:
10692     \if_int_compare:w #3 < \c_zero
10693       \exp_after:wN \exp_after:wN \exp_after:wN
10694       \fp_use_small:w
10695     \else:
10696       \exp_after:wN \exp_after:wN \exp_after:wN \fp_use_none:w
10697     \fi:
10698   \fi:
10699   #2 e #3 \q_stop
10700 }

```

When the exponent is zero, the input is simply returned as output.

```

10701 \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.

```

10702 \cs_new:Npn \fp_use_small:w #1 . #2 e #3 \q_stop
10703 {
10704   0 .
10705   \prg_replicate:nn { -#3 - 1 } { 0 }
10706   #1#2
10707 }

```

Life is more complex for large numbers. The decimal point needs to be shuffled, with potentially some zero-filling for very large values.

```

10708 \cs_new:Npn \fp_use_large:w #1 . #2 e #3 \q_stop
10709 {
10710   \if_int_compare:w #3 < \c_ten
10711     \exp_after:wN \fp_use_large_aux_i:w
10712   \else:
10713     \exp_after:wN \fp_use_large_aux_ii:w
10714   \fi:
10715   #1#2 e #3 \q_stop
10716 }
10717 \cs_new:Npn \fp_use_large_aux_i:w #1#2 e #3 \q_stop
10718 {
10719   #1
10720   \use:c { fp_use_large_aux_ #3 :w } #2 \q_stop
10721 }
10722 \cs_new:cpn { fp_use_large_aux_1:w } #1#2 \q_stop { #1 . #2 }
10723 \cs_new:cpn { fp_use_large_aux_2:w } #1#2#3 \q_stop
10724 { #1#2 . #3 }
10725 \cs_new:cpn { fp_use_large_aux_3:w } #1#2#3#4 \q_stop
10726 { #1#2#3 . #4 }
10727 \cs_new:cpn { fp_use_large_aux_4:w } #1#2#3#4#5 \q_stop

```

```

10728 { #1#2#3#4 . #5 }
10729 \cs_new:cpn { fp_use_large_aux_5:w } #1#2#3#4#5#6 \q_stop
10730 { #1#2#3#4#5 . #6 }
10731 \cs_new:cpn { fp_use_large_aux_6:w } #1#2#3#4#5#6#7 \q_stop
10732 { #1#2#3#4#5#6 . #7 }
10733 \cs_new:cpn { fp_use_large_aux_7:w } #1#2#3#4#5#6#7#8 \q_stop
10734 { #1#2#3#4#6#7 . #8 }
10735 \cs_new:cpn { fp_use_large_aux_8:w } #1#2#3#4#5#6#7#8#9 \q_stop
10736 { #1#2#3#4#5#6#7#8 . #9 }
10737 \cs_new:cpn { fp_use_large_aux_9:w } #1 \q_stop { #1 . }
10738 \cs_new:Npn \fp_use_large_aux_ii:w #1 e #2 \q_stop
10739 {
10740     #1
10741     \prg_replicate:nn { #2 - 9 } { 0 }
10742     .
10743 }

```

(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 10744 \cs_new:Npn \fp_to_dim:N #1 { \fp_use:N #1 pt }
10745 \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 point variables, particularly in the lead-off.

```

\fp_to_int:c 10746 \cs_new:Npn \fp_to_int:N #1
\fp_to_int_aux:w { \exp_after:wN \fp_to_int_aux:w #1 \q_stop }
\fp_to_int_none:w 10747 \cs_generate_variant:Nn \fp_to_int:N { c }
\fp_to_int_small:w 10748 \cs_new:Npn \fp_to_int_aux:w #1#2 e #3 \q_stop
\fp_to_int_large:w 10749 {
\fp_to_int_large_aux_i:w 10750     \if:w #1 -
\fp_to_int_large_aux_1:w 10751     -
\fp_to_int_large_aux_2:w 10752     \fi:
\fp_to_int_large_aux_3:w 10753     \if_int_compare:w #3 < \c_zero
\fp_to_int_large_aux_4:w 10754     \exp_after:wN \fp_to_int_small:w
\fp_to_int_large_aux_5:w 10755     \else:
\fp_to_int_large_aux_6:w 10756     \exp_after:wN \fp_to_int_large:w
\fp_to_int_large_aux_7:w 10757     \fi:
\fp_to_int_large_aux_8:w 10758     #2 e #3 \q_stop
\fp_to_int_large_aux_i:w 10759 }
\fp_to_int_large_aux:nnn 10760
\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.

```

10761 \cs_new:Npn \fp_to_int_small:w #1 . #2 e #3 \q_stop
10762 {
10763   \if_int_compare:w #3 > \c_one
10764   \else:
10765     \if_int_compare:w #1 < \c_five
10766     0
10767   \else:
10768     1
10769   \fi:
10770 \fi:
10771 }

```

For large numbers, the idea is to split off the part for rounding, do the rounding and fill if needed.

```

10772 \cs_new:Npn \fp_to_int_large:w #1 . #2 e #3 \q_stop
10773 {
10774   \if_int_compare:w #3 < \c_ten
10775     \exp_after:wN \fp_to_int_large_aux_i:w
10776   \else:
10777     \exp_after:wN \fp_to_int_large_aux_ii:w
10778   \fi:
10779   #1#2 e #3 \q_stop
10780 }
10781 \cs_new:Npn \fp_to_int_large_aux_i:w #1#2 e #3 \q_stop
10782 { \use:c { fp_to_int_large_aux_#3 :w } #2 \q_stop {#1} }
10783 \cs_new:cpn { fp_to_int_large_aux_1:w } #1#2 \q_stop
10784 { \fp_to_int_large_aux:nnn { #2 0 } {#1} }
10785 \cs_new:cpn { fp_to_int_large_aux_2:w } #1#2#3 \q_stop
10786 { \fp_to_int_large_aux:nnn { #3 00 } {#1#2} }
10787 \cs_new:cpn { fp_to_int_large_aux_3:w } #1#2#3#4 \q_stop
10788 { \fp_to_int_large_aux:nnn { #4 000 } {#1#2#3} }
10789 \cs_new:cpn { fp_to_int_large_aux_4:w } #1#2#3#4#5 \q_stop
10790 { \fp_to_int_large_aux:nnn { #5 0000 } {#1#2#3#4} }
10791 \cs_new:cpn { fp_to_int_large_aux_5:w } #1#2#3#4#5#6 \q_stop
10792 { \fp_to_int_large_aux:nnn { #6 00000 } {#1#2#3#4#5} }
10793 \cs_new:cpn { fp_to_int_large_aux_6:w } #1#2#3#4#5#6#7 \q_stop
10794 { \fp_to_int_large_aux:nnn { #7 000000 } {#1#2#3#4#5#6} }
10795 \cs_new:cpn { fp_to_int_large_aux_7:w } #1#2#3#4#5#6#7#8 \q_stop
10796 { \fp_to_int_large_aux:nnn { #8 0000000 } {#1#2#3#4#5#6#7} }
10797 \cs_new:cpn { fp_to_int_large_aux_8:w } #1#2#3#4#5#6#7#8#9 \q_stop
10798 { \fp_to_int_large_aux:nnn { #9 00000000 } {#1#2#3#4#5#6#7#8} }
10799 \cs_new:cpn { fp_to_int_large_aux_9:w } #1 \q_stop {#1}
10800 \cs_new:Npn \fp_to_int_large_aux:nnn #1#2#3
10801 {
10802   \if_int_compare:w #1 < \c_five_hundred_million
10803   #3#2
10804   \else:
10805     \int_value:w \int_eval:w #3#2 + 1 \int_eval_end:

```

```

10806     \fi:
10807   }
10808   \cs_new:Npn \fp_to_int_large_aux_ii:w #1 e #2 \q_stop
10809   {
10810     #1
10811     \prg_replicate:nn { #2 - 9 } { 0 }
10812   }

```

(End definition for \fp_to_int:N and \fp_to_int:c. These functions are documented on page ??.)

```

\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:NNNNNNNNNN
\fp_use_ix:NNNNNNNNNN
\fp_use_i_to_vii:NNNNNNNNNN
\fp_use_i_to_iix:NNNNNNNNNN
10813   \cs_new:Npn \fp_to_tl:N #1
10814   { \exp_after:wN \fp_to_tl_aux:w #1 \q_stop }
10815   \cs_generate_variant:Nn \fp_to_tl:N { c }
10816   \cs_new:Npn \fp_to_tl_aux:w #1#2 e #3 \q_stop
10817   {
10818     \if:w #1 -
10819     -
10820     \fi:
10821     \if_int_compare:w #3 < \c_zero
10822     \exp_after:wN \fp_to_tl_small:w
10823     \else:
10824     \exp_after:wN \fp_to_tl_large:w
10825     \fi:
10826     #2 e #3 \q_stop
10827   }

```

Converting to integers in an expandable manner is very similar to simply using floating point variables, particularly in the lead-off.

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.

```

10828   \cs_new:Npn \fp_to_tl_large:w #1 e #2 \q_stop
10829   {
10830     \if_int_compare:w #2 < \c_ten
10831     \exp_after:wN \fp_to_tl_large_aux_i:w
10832     \else:
10833     \exp_after:wN \fp_to_tl_large_aux_ii:w
10834     \fi:
10835     #1 e #2 \q_stop
10836   }
10837   \cs_new:Npn \fp_to_tl_large_aux_i:w #1 e #2 \q_stop
10838   { \use:c { fp_to_tl_large_#2 :w } #1 \q_stop }
10839   \cs_new:Npn \fp_to_tl_large_aux_ii:w #1 . #2 e #3 \q_stop
10840   {
10841     #1
10842     \fp_to_tl_large_zeros:NNNNNNNNN #2
10843     e #3
10844   }
10845   \cs_new:cpn { fp_to_tl_large_0:w } #1 . #2 \q_stop
10846   {

```

```

10847     #1
10848     \fp_to_tl_large_zeros:NNNNNNNN #2
10849   }
10850   \cs_new:cpn { fp_to_tl_large_1:w } #1 . #2#3 \q_stop
10851   {
10852     #1#2
10853     \fp_to_tl_large_zeros:NNNNNNNN #3 0
10854   }
10855   \cs_new:cpn { fp_to_tl_large_2:w } #1 . #2#3#4 \q_stop
10856   {
10857     #1#2#3
10858     \fp_to_tl_large_zeros:NNNNNNNN #4 00
10859   }
10860   \cs_new:cpn { fp_to_tl_large_3:w } #1 . #2#3#4#5 \q_stop
10861   {
10862     #1#2#3#4
10863     \fp_to_tl_large_zeros:NNNNNNNN #5 000
10864   }
10865   \cs_new:cpn { fp_to_tl_large_4:w } #1 . #2#3#4#5#6 \q_stop
10866   {
10867     #1#2#3#4#5
10868     \fp_to_tl_large_zeros:NNNNNNNN #6 0000
10869   }
10870   \cs_new:cpn { fp_to_tl_large_5:w } #1 . #2#3#4#5#6#7 \q_stop
10871   {
10872     #1#2#3#4#5#6
10873     \fp_to_tl_large_zeros:NNNNNNNN #7 00000
10874   }
10875   \cs_new:cpn { fp_to_tl_large_6:w } #1 . #2#3#4#5#6#7#8 \q_stop
10876   {
10877     #1#2#3#4#5#6#7
10878     \fp_to_tl_large_zeros:NNNNNNNN #8 000000
10879   }
10880   \cs_new:cpn { fp_to_tl_large_7:w } #1 . #2#3#4#5#6#7#8#9 \q_stop
10881   {
10882     #1#2#3#4#5#6#7#8
10883     \fp_to_tl_large_zeros:NNNNNNNN #9 0000000
10884   }
10885   \cs_new:cpn { fp_to_tl_large_8:w } #1 .
10886   {
10887     #1
10888     \use:c { fp_to_tl_large_8_aux:w }
10889   }
10890   \cs_new:cpn { fp_to_tl_large_8_aux:w } #1#2#3#4#5#6#7#8#9 \q_stop
10891   {
10892     #1#2#3#4#5#6#7#8
10893     \fp_to_tl_large_zeros:NNNNNNNN #9 00000000
10894   }
10895   \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.

```

10896 \cs_new:Npn \fp_to_tl_small:w #1 e #2 \q_stop
10897 {
10898   \if_int_compare:w #2 = \c_minus_one
10899     \exp_after:wN \fp_to_tl_small_one:w
10900   \else:
10901     \if_int_compare:w #2 = -\c_two
10902       \exp_after:wN \exp_after:wN \exp_after:wN \fp_to_tl_small_two:w
10903     \else:
10904       \exp_after:wN \exp_after:wN \exp_after:wN \fp_to_tl_small_aux:w
10905     \fi:
10906   \fi:
10907   #1 e #2 \q_stop
10908 }
10909 \cs_new:Npn \fp_to_tl_small_one:w #1 . #2 e #3 \q_stop
10910 {
10911   \if_int_compare:w \fp_use_ix:NNNNNNNN #2 > \c_four
10912     \if_int_compare:w
10913       \int_eval:w #1 \fp_use_i_to_iix:NNNNNNNN #2 + 1
10914       < \c_one_thousand_million
10915       0.
10916     \exp_after:wN \fp_to_tl_small_zeros:NNNNNNNN
10917     \int_value:w \int_eval:w
10918       #1 \fp_use_i_to_iix:NNNNNNNN #2 + 1
10919     \int_eval_end:
10920   \else:
10921     1
10922   \fi:
10923   \else:
10924     0. #1
10925     \fp_to_tl_small_zeros:NNNNNNNN #2
10926   \fi:
10927 }
10928 \cs_new:Npn \fp_to_tl_small_two:w #1 . #2 e #3 \q_stop
10929 {
10930   \if_int_compare:w \fp_use_iix_ix:NNNNNNNN #2 > \c_forty_four
10931     \if_int_compare:w
10932       \int_eval:w #1 \fp_use_i_to_vii:NNNNNNNN #2 0 + \c_ten
10933       < \c_one_thousand_million
10934       0.0
10935     \exp_after:wN \fp_to_tl_small_zeros:NNNNNNNN
10936     \int_value:w \int_eval:w
10937       #1 \fp_use_i_to_vii:NNNNNNNN #2 0 + \c_ten
10938     \int_eval_end:
10939   \else:
10940     0.1
10941   \fi:

```

```

10942     \else:
10943         0.0
10944         #1
10945         \fp_to_tl_small_zeros:NNNNNNNN #2
10946     \fi:
10947 }
10948 \cs_new:Npn \fp_to_tl_small_aux:w #1 . #2 e #3 \q_stop
10949 {
10950     #1
10951     \fp_to_tl_large_zeros:NNNNNNNN #2
10952     e #3
10953 }

```

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.

```

10954 \cs_new:Npn \fp_to_tl_large_zeros:NNNNNNNN #1#2#3#4#5#6#7#8#9
10955 {
10956     \if_int_compare:w #9 = \c_zero
10957     \if_int_compare:w #8 = \c_zero
10958     \if_int_compare:w #7 = \c_zero
10959     \if_int_compare:w #6 = \c_zero
10960     \if_int_compare:w #5 = \c_zero
10961     \if_int_compare:w #4 = \c_zero
10962     \if_int_compare:w #3 = \c_zero
10963     \if_int_compare:w #2 = \c_zero
10964     \if_int_compare:w #1 = \c_zero
10965     \else:
10966         . #1
10967     \fi:
10968     \else:
10969         . #1#2
10970     \fi:
10971     \else:
10972         . #1#2#3
10973     \fi:
10974     \else:
10975         . #1#2#3#4
10976     \fi:
10977     \else:
10978         . #1#2#3#4#5
10979     \fi:
10980     \else:
10981         . #1#2#3#4#5#6
10982     \fi:
10983     \else:
10984         . #1#2#3#4#5#6#7
10985     \fi:
10986     \else:
10987         . #1#2#3#4#5#6#7#8
10988     \fi:

```



```

10989     \else:
10990         . #1#2#3#4#5#6#7#8#9
10991     \fi:
10992 }
10993 \cs_new:Npn \fp_to_tl_small_zeros:NNNNNNNNN #1#2#3#4#5#6#7#8#9
10994 {
10995     \if_int_compare:w #9 = \c_zero
10996     \if_int_compare:w #8 = \c_zero
10997     \if_int_compare:w #7 = \c_zero
10998     \if_int_compare:w #6 = \c_zero
10999     \if_int_compare:w #5 = \c_zero
11000     \if_int_compare:w #4 = \c_zero
11001     \if_int_compare:w #3 = \c_zero
11002     \if_int_compare:w #2 = \c_zero
11003     \if_int_compare:w #1 = \c_zero
11004     \else:
11005         #1
11006     \fi:
11007     \else:
11008         #1#2
11009     \fi:
11010     \else:
11011         #1#2#3
11012     \fi:
11013     \else:
11014         #1#2#3#4
11015     \fi:
11016     \else:
11017         #1#2#3#4#5
11018     \fi:
11019     \else:
11020         #1#2#3#4#5#6
11021     \fi:
11022     \else:
11023         #1#2#3#4#5#6#7
11024     \fi:
11025     \else:
11026         #1#2#3#4#5#6#7#8
11027     \fi:
11028     \else:
11029         #1#2#3#4#5#6#7#8#9
11030     \fi:
11031 }

```

Some quick “return a few” functions.

```

11032 \cs_new:Npn \fp_use_iix_ix:NNNNNNNNN #1#2#3#4#5#6#7#8#9 {#8#9}
11033 \cs_new:Npn \fp_use_ix:NNNNNNNNN #1#2#3#4#5#6#7#8#9 {#9}
11034 \cs_new:Npn \fp_use_i_to_vii:NNNNNNNNN #1#2#3#4#5#6#7#8#9
11035     {#1#2#3#4#5#6#7}
11036 \cs_new:Npn \fp_use_i_to_iix:NNNNNNNNN #1#2#3#4#5#6#7#8#9

```

```
11037 {#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.

```
\fp_round_figures:Nn Rounding to figures needs only an adjustment to the target by one (as the target is in
\fp_round_figures:cn decimal places).
\fp_ground_figures:Nn
\fp_ground_figures:cn
\fp_round_figures_aux:NNn
11038 \cs_new_protected_nopar:Npn \fp_round_figures:Nn
11039 { \fp_round_figures_aux:NNn \tl_set:Nn }
11040 \cs_generate_variant:Nn \fp_round_figures:Nn { c }
11041 \cs_new_protected_nopar:Npn \fp_ground_figures:Nn
11042 { \fp_round_figures_aux:NNn \tl_gset:Nn }
11043 \cs_generate_variant:Nn \fp_ground_figures:Nn { c }
11044 \cs_new_protected:Npn \fp_round_figures_aux:NNn #1#2#3
11045 {
11046   \group_begin:
11047   \fp_read:N #2
11048   \int_set:Nn \l_fp_round_target_int { #3 - 1 }
11049   \if_int_compare:w \l_fp_round_target_int < \c_ten
11050     \exp_after:wN \fp_round:
11051   \fi:
11052   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11053   \cs_set_protected_nopar:Npx \fp_tmp:w
11054   {
11055     \group_end:
11056     #1 \exp_not:N #2
11057     {
11058       \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11059         -
11060       \else:
11061         +
11062       \fi:
11063       \int_use:N \l_fp_input_a_integer_int
11064       .
11065       \exp_after:wN \use_none:n
11066       \int_use:N \l_fp_input_a_decimal_int
11067       e
11068       \int_use:N \l_fp_input_a_exponent_int
11069     }
11070   }
11071   \fp_tmp:w
11072 }
```

(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
`\fp_round_places:cn` everything should be correct.
`\fp_ground_places:Nn`
`\fp_ground_places:cn`
`\fp_round_places_aux:NNn`

```

11073 \cs_new_protected_nopar:Npn \fp_round_places:Nn
11074 { \fp_round_places_aux:NNn \tl_set:Nn }
11075 \cs_generate_variant:Nn \fp_round_places:Nn { c }
11076 \cs_new_protected_nopar:Npn \fp_ground_places:Nn
11077 { \fp_round_places_aux:NNn \tl_gset:Nn }
11078 \cs_generate_variant:Nn \fp_ground_places:Nn { c }
11079 \cs_new_protected:Npn \fp_round_places_aux:NNn #1#2#3
11080 {
11081   \group_begin:
11082     \fp_read:N #2
11083     \int_set:Nn \l_fp_round_target_int
11084       { #3 + \l_fp_input_a_exponent_int }
11085     \if_int_compare:w \l_fp_round_target_int < \c_ten
11086       \exp_after:wN \fp_round:
11087     \fi:
11088     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11089     \cs_set_protected_nopar:Npx \fp_tmp:w
11090     {
11091       \group_end:
11092       #1 \exp_not:N #2
11093       {
11094         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11095           -
11096         \else:
11097           +
11098         \fi:
11099         \int_use:N \l_fp_input_a_integer_int
11100         .
11101         \exp_after:wN \use_none:n
11102         \int_use:N \l_fp_input_a_decimal_int
11103         e
11104         \int_use:N \l_fp_input_a_exponent_int
11105       }
11106     }
11107     \fp_tmp:w
11108   }

```

(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
`\fp_round_aux:NNNNNNNNN` always nine decimal digits to round, so the code can be written to account for this. The
`\fp_round_loop:N` 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.

```

11109 \cs_new_protected_nopar:Npn \fp_round:
11110 {
11111   \bool_set_false:N \l_fp_round_carry_bool
11112   \l_fp_round_position_int \c_eight

```

```

11113 \tl_clear:N \l_fp_round_decimal_tl
11114 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11115 \exp_after:wN \use_i:nn \exp_after:wN
11116 \fp_round_aux:NNNNNNNN \int_use:N \l_fp_input_a_decimal_int
11117 }
11118 \cs_new_protected:Npn \fp_round_aux:NNNNNNNN #1#2#3#4#5#6#7#8#9
11119 {
11120 \fp_round_loop:N #9#8#7#6#5#4#3#2#1
11121 \bool_if:NT \l_fp_round_carry_bool
11122 { \tex_advance:D \l_fp_input_a_integer_int \c_one }
11123 \l_fp_input_a_decimal_int \l_fp_round_decimal_tl \scan_stop:
11124 \if_int_compare:w \l_fp_input_a_integer_int < \c_ten
11125 \else:
11126 \l_fp_input_a_integer_int \c_one
11127 \tex_divide:D \l_fp_input_a_decimal_int \c_ten
11128 \tex_advance:D \l_fp_input_a_exponent_int \c_one
11129 \fi:
11130 }
11131 \cs_new_protected:Npn \fp_round_loop:N #1
11132 {
11133 \if_int_compare:w \l_fp_round_position_int < \l_fp_round_target_int
11134 \bool_if:NTF \l_fp_round_carry_bool
11135 { \l_fp_internal_int \int_eval:w #1 + \c_one \scan_stop: }
11136 { \l_fp_internal_int \int_eval:w #1 \scan_stop: }
11137 \if_int_compare:w \l_fp_internal_int = \c_ten
11138 \l_fp_internal_int \c_zero
11139 \else:
11140 \bool_set_false:N \l_fp_round_carry_bool
11141 \fi:
11142 \tl_set:Nx \l_fp_round_decimal_tl
11143 { \int_use:N \l_fp_internal_int \l_fp_round_decimal_tl }
11144 \else:
11145 \tl_set:Nx \l_fp_round_decimal_tl { 0 \l_fp_round_decimal_tl }
11146 \if_int_compare:w \l_fp_round_position_int = \l_fp_round_target_int
11147 \if_int_compare:w #1 > \c_four
11148 \bool_set_true:N \l_fp_round_carry_bool
11149 \fi:
11150 \fi:
11151 \fi:
11152 \tex_advance:D \l_fp_round_position_int \c_minus_one
11153 \if_int_compare:w \l_fp_round_position_int > \c_minus_one
11154 \exp_after:wN \fp_round_loop:N
11155 \fi:
11156 }

```

(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
 \fp_gabs:N
 \fp_gabs:c
 \fp_abs_aux:NN

```

11157 \cs_new_protected_nopar:Npn \fp_abs:N { \fp_abs_aux:NN \tl_set:Nn }
11158 \cs_new_protected_nopar:Npn \fp_gabs:N { \fp_abs_aux:NN \tl_gset:Nn }
11159 \cs_generate_variant:Nn \fp_abs:N { c }
11160 \cs_generate_variant:Nn \fp_gabs:N { c }
11161 \cs_new_protected:Npn \fp_abs_aux:NN #1#2
11162 {
11163   \group_begin:
11164   \fp_read:N #2
11165   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11166   \cs_set_protected_nopar:Npx \fp_tmp:w
11167   {
11168     \group_end:
11169     #1 \exp_not:N #2
11170     {
11171       +
11172       \int_use:N \l_fp_input_a_integer_int
11173       .
11174       \exp_after:wN \use_none:n
11175       \int_use:N \l_fp_input_a_decimal_int
11176       e
11177       \int_use:N \l_fp_input_a_exponent_int
11178     }
11179   }
11180   \fp_tmp:w
11181 }

```

(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 11182 \cs_new_protected_nopar:Npn \fp_neg:N { \fp_neg_aux:NN \tl_set:Nn }
\fp_gneg:N 11183 \cs_new_protected_nopar:Npn \fp_gneg:N { \fp_neg_aux:NN \tl_gset:Nn }
\fp_gneg:c 11184 \cs_generate_variant:Nn \fp_neg:N { c }
\fp_neg:NN 11185 \cs_generate_variant:Nn \fp_gneg:N { c }
11186 \cs_new_protected:Npn \fp_neg_aux:NN #1#2
11187 {
11188   \group_begin:
11189   \fp_read:N #2
11190   \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11191   \tl_set:Nx \l_fp_internal_tl
11192   {
11193     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
11194     +
11195     \else:
11196     -
11197     \fi:
11198     \int_use:N \l_fp_input_a_integer_int
11199     .
11200     \exp_after:wN \use_none:n
11201     \int_use:N \l_fp_input_a_decimal_int
11202     e
11203     \int_use:N \l_fp_input_a_exponent_int

```

```

11204     }
11205     \exp_after:wN \group_end: \exp_after:wN
11206     #1 \exp_after:wN #2 \exp_after:wN { \l_fp_internal_tl }
11207 }

```

(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` 11208 `\cs_new_protected_nopar:Npn \fp_add:Nn { \fp_add_aux:NNn \tl_set:Nn }`
`\fp_gadd:cn` 11209 `\cs_new_protected_nopar:Npn \fp_gadd:Nn { \fp_add_aux:NNn \tl_gset:Nn }`
`\fp_add_aux:NNn` 11210 `\cs_generate_variant:Nn \fp_add:Nn { c }`
`\fp_add_core:` 11211 `\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.

```

11212 \cs_new_protected:Npn \fp_add_aux:NNn #1#2#3
11213 {
11214   \group_begin:
11215   \fp_read:N #2
11216   \fp_split:Nn b {#3}
11217   \fp_standardise:NNNN
11218   \l_fp_input_b_sign_int
11219   \l_fp_input_b_integer_int
11220   \l_fp_input_b_decimal_int
11221   \l_fp_input_b_exponent_int
11222   \fp_add_core:
11223   \fp_tmp:w #1#2
11224 }
11225 \cs_new_protected_nopar:Npn \fp_add_core:
11226 {
11227   \fp_level_input_exponents:
11228   \if_int_compare:w
11229     \int_eval:w
11230       \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
11231       > \c_zero
11232     \exp_after:wN \fp_add_sum:
11233   \else:
11234     \exp_after:wN \fp_add_difference:
11235   \fi:
11236   \l_fp_output_exponent_int \l_fp_input_a_exponent_int
11237   \fp_standardise:NNNN
11238   \l_fp_output_sign_int
11239   \l_fp_output_integer_int
11240   \l_fp_output_decimal_int
11241   \l_fp_output_exponent_int
11242   \cs_set_protected:Npx \fp_tmp:w ##1##2

```

```

11243 {
11244   \group_end:
11245   ##1 ##2
11246   {
11247     \if_int_compare:w \l_fp_output_sign_int < \c_zero
11248     -
11249     \else:
11250     +
11251     \fi:
11252     \int_use:N \l_fp_output_integer_int
11253     .
11254     \exp_after:wN \use_none:n
11255     \int_value:w \int_eval:w
11256     \l_fp_output_decimal_int + \c_one_thousand_million
11257     e
11258     \int_use:N \l_fp_output_exponent_int
11259   }
11260 }
11261 }

```

Finding the sum of two numbers is trivially easy.

```

11262 \cs_new_protected_nopar:Npn \fp_add_sum:
11263 {
11264   \l_fp_output_sign_int \l_fp_input_a_sign_int
11265   \l_fp_output_integer_int
11266   \int_eval:w
11267   \l_fp_input_a_integer_int + \l_fp_input_b_integer_int
11268   \scan_stop:
11269   \l_fp_output_decimal_int
11270   \int_eval:w
11271   \l_fp_input_a_decimal_int + \l_fp_input_b_decimal_int
11272   \scan_stop:
11273   \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
11274   \else:
11275     \tex_advance:D \l_fp_output_integer_int \c_one
11276     \tex_advance:D \l_fp_output_decimal_int -\c_one_thousand_million
11277   \fi:
11278 }

```

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.

```

11279 \cs_new_protected_nopar:Npn \fp_add_difference:
11280 {
11281   \l_fp_output_integer_int
11282   \int_eval:w
11283   \l_fp_input_a_integer_int - \l_fp_input_b_integer_int
11284   \scan_stop:
11285   \l_fp_output_decimal_int

```

```

11286 \int_eval:w
11287 \l_fp_input_a_decimal_int - \l_fp_input_b_decimal_int
11288 \scan_stop:
11289 \if_int_compare:w \l_fp_output_decimal_int < \c_zero
11290 \tex_advance:D \l_fp_output_integer_int \c_minus_one
11291 \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
11292 \fi:
11293 \if_int_compare:w \l_fp_output_integer_int < \c_zero
11294 \l_fp_output_sign_int \l_fp_input_b_sign_int
11295 \if_int_compare:w \l_fp_output_decimal_int = \c_zero
11296 \l_fp_output_integer_int -\l_fp_output_integer_int
11297 \else:
11298 \l_fp_output_decimal_int
11299 \int_eval:w
11300 \c_one_thousand_million - \l_fp_output_decimal_int
11301 \scan_stop:
11302 \l_fp_output_integer_int
11303 \int_eval:w
11304 - \l_fp_output_integer_int - \c_one
11305 \scan_stop:
11306 \fi:
11307 \else:
11308 \l_fp_output_sign_int \l_fp_input_a_sign_int
11309 \fi:
11310 }

```

(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` here.
`\fp_gsub:cn`

```

\fp_sub_aux:NNn 11311 \cs_new_protected_nopar:Npn \fp_sub:Nn { \fp_sub_aux:NNn \tl_set:Nn }
11312 \cs_new_protected_nopar:Npn \fp_gsub:Nn { \fp_sub_aux:NNn \tl_gset:Nn }
11313 \cs_generate_variant:Nn \fp_sub:Nn { c }
11314 \cs_generate_variant:Nn \fp_gsub:Nn { c }
11315 \cs_new_protected:Npn \fp_sub_aux:NNn #1#2#3
11316 {
11317 \group_begin:
11318 \fp_read:N #2
11319 \fp_split:Nn b {#3}
11320 \fp_standardise:NNNN
11321 \l_fp_input_b_sign_int
11322 \l_fp_input_b_integer_int
11323 \l_fp_input_b_decimal_int
11324 \l_fp_input_b_exponent_int
11325 \tex_multiply:D \l_fp_input_b_sign_int \c_minus_one
11326 \fp_add_core:
11327 \fp_tmp:w #1#2
11328 }

```

(End definition for \fp_sub:Nn and \fp_sub:cn. These functions are documented on page ??.)

\fp_mul:Nn \fp_mul:cn \fp_gmul:Nn \fp_gmul:cn \fp_mul_aux:NNn \fp_mul_internal: \fp_mul_split:NNNN \fp_mul_split:w \fp_mul_end_level: \fp_mul_end_level:NNNNNNNN	The pattern is much the same for multiplication. 11329 \cs_new_protected_nopar:Npn \fp_mul:Nn { \fp_mul_aux:NNn \tl_set:Nn } 11330 \cs_new_protected_nopar:Npn \fp_gmul:Nn { \fp_mul_aux:NNn \tl_gset:Nn } 11331 \cs_generate_variant:Nn \fp_mul:Nn { c } 11332 \cs_generate_variant:Nn \fp_gmul:Nn { c } The approach to multiplication is as follows. First, the two numbers are split into blocks of three digits. These are then multiplied together to find products for each group of three output digits. This is all written out in full for speed reasons. Between each block of three digits in the output, there is a carry step. The very lowest digits are not calculated, while
---	---

```

11333 \cs_new_protected:Npn \fp_mul_aux:NNn #1#2#3
11334 {
11335   \group_begin:
11336   \fp_read:N #2
11337   \fp_split:Nn b {#3}
11338   \fp_standardise:NNNN
11339   \l_fp_input_b_sign_int
11340   \l_fp_input_b_integer_int
11341   \l_fp_input_b_decimal_int
11342   \l_fp_input_b_exponent_int
11343   \fp_mul_internal:
11344   \l_fp_output_exponent_int
11345   \int_eval:w
11346     \l_fp_input_a_exponent_int + \l_fp_input_b_exponent_int
11347   \scan_stop:
11348   \fp_standardise:NNNN
11349   \l_fp_output_sign_int
11350   \l_fp_output_integer_int
11351   \l_fp_output_decimal_int
11352   \l_fp_output_exponent_int
11353   \cs_set_protected_nopar:Npx \fp_tmp:w
11354   {
11355     \group_end:
11356     #1 \exp_not:N #2
11357     {
11358       \if_int_compare:w
11359         \int_eval:w
11360           \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
11361           < \c_zero
11362       \if_int_compare:w
11363         \int_eval:w
11364           \l_fp_output_integer_int + \l_fp_output_decimal_int
11365           = \c_zero
11366       +
11367       \else:
11368       -
11369       \fi:
11370     \else:
11371     +
11372     \fi:

```

```

11373         \int_use:N \l_fp_output_integer_int
11374         .
11375         \exp_after:wN \use_none:n
11376         \int_value:w \int_eval:w
11377         \l_fp_output_decimal_int + \c_one_thousand_million
11378         e
11379         \int_use:N \l_fp_output_exponent_int
11380     }
11381 }
11382 \fp_tmp:w
11383 }

```

Done separately so that the internal use is a bit easier.

```

11384 \cs_new_protected_nopar:Npn \fp_mul_internal:
11385 {
11386     \fp_mul_split:NNNN \l_fp_input_a_decimal_int
11387     \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11388     \fp_mul_split:NNNN \l_fp_input_b_decimal_int
11389     \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11390     \l_fp_mul_output_int \c_zero
11391     \tl_clear:N \l_fp_mul_output_tl
11392     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_iii_int
11393     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_ii_int
11394     \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_mul_b_i_int
11395     \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11396     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_iii_int
11397     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_ii_int
11398     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_i_int
11399     \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_input_b_integer_int
11400     \fp_mul_end_level:
11401     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_ii_int
11402     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_i_int
11403     \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_input_b_integer_int
11404     \fp_mul_end_level:
11405     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_mul_b_i_int
11406     \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_input_b_integer_int
11407     \fp_mul_end_level:
11408     \l_fp_output_decimal_int 0 \l_fp_mul_output_tl \scan_stop:
11409     \tl_clear:N \l_fp_mul_output_tl
11410     \fp_mul_product:NN \l_fp_input_a_integer_int \l_fp_input_b_integer_int
11411     \fp_mul_end_level:
11412     \l_fp_output_integer_int 0 \l_fp_mul_output_tl \scan_stop:
11413 }

```

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.

```

11414 \cs_new_protected:Npn \fp_mul_split:NNNN #1#2#3#4
11415 {

```

```

11416 \tex_advance:D #1 \c_one_thousand_million
11417 \cs_set_protected:Npn \fp_mul_split_aux:w
11418   ##1##2##3##4##5##6##7##8##9 \q_stop {
11419     #2 ##2##3##4 \scan_stop:
11420     #3 ##5##6##7 \scan_stop:
11421     #4 ##8##9 \scan_stop:
11422   }
11423 \exp_after:wN \fp_mul_split_aux:w \int_use:N #1 \q_stop
11424 \tex_advance:D #1 -\c_one_thousand_million
11425 }
11426 \cs_new_protected:Npn \fp_mul_product:NN #1#2
11427 {
11428   \l_fp_mul_output_int
11429   \int_eval:w \l_fp_mul_output_int + #1 * #2 \scan_stop:
11430 }

```

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.

```

11431 \cs_new_protected_nopar:Npn \fp_mul_end_level:
11432 {
11433   \tex_advance:D \l_fp_mul_output_int \c_one_thousand_million
11434   \exp_after:wN \use_i:nn \exp_after:wN
11435   \fp_mul_end_level:NNNNNNNN \int_use:N \l_fp_mul_output_int
11436 }
11437 \cs_new_protected:Npn \fp_mul_end_level:NNNNNNNN #1#2#3#4#5#6#7#8#9
11438 {
11439   \tl_set:Nx \l_fp_mul_output_tl { #7#8#9 \l_fp_mul_output_tl }
11440   \l_fp_mul_output_int #1#2#3#4#5#6 \scan_stop:
11441 }

```

(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 11442 \cs_new_protected_nopar:Npn \fp_div:Nn { \fp_div_aux:NNn \tl_set:Nn }
\fp_gdiv:Nn 11443 \cs_new_protected_nopar:Npn \fp_gdiv:Nn { \fp_div_aux:NNn \tl_gset:Nn }
\fp_gdiv:cn 11444 \cs_generate_variant:Nn \fp_div:Nn { c }
\fp_div_aux:NNn 11445 \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 issued. On the other hand, if the numerator is zero then the result must be 0.0 and can be given with no further work.

```

\fp_div_divide_aux: 11446 \cs_new_protected:Npn \fp_div_aux:NNn #1#2#3
\fp_div_store: 11447 {
\fp_div_store_integer: 11448   \group_begin:
\fp_div_store_decimal: 11449     \fp_read:N #2
11450     \fp_split:Nn b {#3}
11451     \fp_standardise:NNNN
11452     \l_fp_input_b_sign_int
11453     \l_fp_input_b_integer_int
11454     \l_fp_input_b_decimal_int

```

```

11455         \l_fp_input_b_exponent_int
11456     \if_int_compare:w
11457         \int_eval:w
11458         \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
11459         = \c_zero
11460     \cs_set_protected:Npx \fp_tmp:w ##1##2
11461     {
11462         \group_end:
11463         #1 \exp_not:N #2 { \c_undefined_fp }
11464     }
11465 \else:
11466     \if_int_compare:w
11467         \int_eval:w
11468         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
11469         = \c_zero
11470     \cs_set_protected:Npx \fp_tmp:w ##1##2
11471     {
11472         \group_end:
11473         #1 \exp_not:N #2 { \c_zero_fp }
11474     }
11475 \else:
11476     \exp_after:wN \exp_after:wN \exp_after:wN \fp_div_internal:
11477 \fi:
11478 \fi:
11479 \fp_tmp:w #1#2
11480 }

```

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.

```

11481 \cs_new_protected_nopar:Npn \fp_div_internal: {
11482     \l_fp_output_integer_int \c_zero
11483     \l_fp_output_decimal_int \c_zero
11484     \cs_set_eq:NN \fp_div_store: \fp_div_store_integer:
11485     \l_fp_div_offset_int \c_one_hundred_million
11486     \fp_div_loop:
11487     \l_fp_output_exponent_int
11488     \int_eval:w
11489         \l_fp_input_a_exponent_int - \l_fp_input_b_exponent_int
11490     \scan_stop:
11491     \fp_standardise:NNNN
11492     \l_fp_output_sign_int
11493     \l_fp_output_integer_int
11494     \l_fp_output_decimal_int
11495     \l_fp_output_exponent_int
11496     \cs_set_protected:Npx \fp_tmp:w ##1##2
11497     {
11498         \group_end:

```

```

11499     ##1 ##2
11500     {
11501         \if_int_compare:w
11502             \int_eval:w
11503             \l_fp_input_a_sign_int * \l_fp_input_b_sign_int
11504             < \c_zero
11505         \if_int_compare:w
11506             \int_eval:w
11507             \l_fp_output_integer_int + \l_fp_output_decimal_int
11508             = \c_zero
11509         +
11510         \else:
11511             -
11512         \fi:
11513     \else:
11514         +
11515         \fi:
11516         \int_use:N \l_fp_output_integer_int
11517         .
11518         \exp_after:wN \use_none:n
11519         \int_value:w \int_eval:w
11520         \l_fp_output_decimal_int + \c_one_thousand_million
11521         \int_eval_end:
11522     e
11523     \int_use:N \l_fp_output_exponent_int
11524 }
11525 }
11526 }

```

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.

```

11527 \cs_new_protected_nopar:Npn \fp_div_loop:
11528 {
11529     \l_fp_count_int \c_zero
11530     \fp_div_divide:
11531     \fp_div_store:
11532     \tex_multiply:D \l_fp_input_a_integer_int \c_ten
11533     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11534     \exp_after:wN \fp_div_loop_step:w
11535     \int_use:N \l_fp_input_a_decimal_int \q_stop
11536     \if_int_compare:w
11537         \int_eval:w \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
11538         > \c_zero
11539         \if_int_compare:w \l_fp_div_offset_int > \c_zero
11540             \exp_after:wN \exp_after:wN \exp_after:wN
11541             \fp_div_loop:
11542         \fi:
11543     \fi:
11544 }

```

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.

```

11545 \cs_new_protected_nopar:Npn \fp_div_divide:
11546 {
11547   \if_int_compare:w \l_fp_input_a_integer_int > \l_fp_input_b_integer_int
11548     \exp_after:wN \fp_div_divide_aux:
11549   \else:
11550     \if_int_compare:w \l_fp_input_a_integer_int < \l_fp_input_b_integer_int
11551     \else:
11552       \if_int_compare:w
11553         \l_fp_input_a_decimal_int < \l_fp_input_b_decimal_int
11554       \else:
11555         \exp_after:wN \exp_after:wN \exp_after:wN
11556         \exp_after:wN \exp_after:wN \exp_after:wN
11557         \exp_after:wN \fp_div_divide_aux:
11558       \fi:
11559     \fi:
11560   \fi:
11561 }
11562 \cs_new_protected_nopar:Npn \fp_div_divide_aux:
11563 {
11564   \tex_advance:D \l_fp_count_int \c_one
11565   \tex_advance:D \l_fp_input_a_integer_int -\l_fp_input_b_integer_int
11566   \tex_advance:D \l_fp_input_a_decimal_int -\l_fp_input_b_decimal_int
11567   \if_int_compare:w \l_fp_input_a_decimal_int < \c_zero
11568     \tex_advance:D \l_fp_input_a_integer_int \c_minus_one
11569     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11570   \fi:
11571   \fp_div_divide:
11572 }

```

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.

```

11573 \cs_new_protected_nopar:Npn \fp_div_store: { }
11574 \cs_new_protected_nopar:Npn \fp_div_store_integer:
11575 {
11576   \l_fp_output_integer_int \l_fp_count_int
11577   \cs_set_eq:NN \fp_div_store: \fp_div_store_decimal:
11578 }
11579 \cs_new_protected_nopar:Npn \fp_div_store_decimal:
11580 {
11581   \l_fp_output_decimal_int
11582   \int_eval:w
11583     \l_fp_output_decimal_int +
11584     \l_fp_count_int * \l_fp_div_offset_int
11585   \int_eval_end:
11586   \tex_divide:D \l_fp_div_offset_int \c_ten
11587 }

```

```

11588 \cs_new_protected:Npn \fp_div_loop_step:w #1#2#3#4#5#6#7#8#9 \q_stop
11589 {
11590   \l_fp_input_a_integer_int
11591   \int_eval:w #2 + \l_fp_input_a_integer_int \int_eval_end:
11592   \l_fp_input_a_decimal_int #3#4#5#6#7#8#9 0 \scan_stop:
11593 }

```

(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.

```

11594 \cs_new_protected:Npn \fp_add:NNNNNNNNN #1#2#3#4#5#6#7#8#9
11595 {
11596   #7 \int_eval:w #1 + #4 \int_eval_end:
11597   #8 \int_eval:w #2 + #5 \int_eval_end:
11598   #9 \int_eval:w #3 + #6 \int_eval_end:
11599   \if_int_compare:w #9 < \c_one_thousand_million
11600   \else:
11601     \tex_advance:D #8 \c_one
11602     \tex_advance:D #9 -\c_one_thousand_million
11603   \fi:
11604   \if_int_compare:w #8 < \c_one_thousand_million
11605   \else:
11606     \tex_advance:D #7 \c_one

```

```

11607     \tex_advance:D #8 -\c_one_thousand_million
11608     \fi:
11609   }
      (End definition for \fp_add:NNNNNNNN. This function is documented on page ??.)

```

`\fp_sub:NNNNNNNN` 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.

```

11610 \cs_new_protected:Npn \fp_sub:NNNNNNNN #1#2#3#4#5#6#7#8#9
11611 {
11612   #7 \int_eval:w #1 - #4 \int_eval_end:
11613   #8 \int_eval:w #2 - #5 \int_eval_end:
11614   #9 \int_eval:w #3 - #6 \int_eval_end:
11615   \if_int_compare:w #9 < \c_zero
11616     \tex_advance:D #8 \c_minus_one
11617     \tex_advance:D #9 \c_one_thousand_million
11618   \fi:
11619   \if_int_compare:w #8 < \c_zero
11620     \tex_advance:D #7 \c_minus_one
11621     \tex_advance:D #8 \c_one_thousand_million
11622   \fi:
11623   \if_int_compare:w #7 < \c_zero
11624     \if_int_compare:w \int_eval:w #8 + #9 = \c_zero
11625       #7 -#7
11626   \else:
11627     \tex_advance:D #7 \c_one
11628     #8 \int_eval:w \c_one_thousand_million - #8 \int_eval_end:
11629     #9 \int_eval:w \c_one_thousand_million - #9 \int_eval_end:
11630   \fi:
11631   \fi:
11632 }
      (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.

```

11633 \cs_new_protected:Npn \fp_mul:NNNNNN #1#2#3#4#5#6
11634 {
11635   \fp_mul_split:NNNN #1
11636   \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11637   \fp_mul_split:NNNN #2
11638   \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_a_vi_int
11639   \fp_mul_split:NNNN #3
11640   \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11641   \fp_mul_split:NNNN #4
11642   \l_fp_mul_b_iv_int \l_fp_mul_b_v_int \l_fp_mul_b_vi_int
11643   \l_fp_mul_output_int \c_zero
11644   \tl_clear:N \l_fp_mul_output_tl
11645   \fp_mul_product:NN \l_fp_mul_a_i_int          \l_fp_mul_b_vi_int
11646   \fp_mul_product:NN \l_fp_mul_a_ii_int         \l_fp_mul_b_v_int
11647   \fp_mul_product:NN \l_fp_mul_a_iii_int        \l_fp_mul_b_iv_int

```



```

11648 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_iii_int
11649 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_ii_int
11650 \fp_mul_product:NN \l_fp_mul_a_vi_int \l_fp_mul_b_i_int
11651 \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11652 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_v_int
11653 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iv_int
11654 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iii_int
11655 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_ii_int
11656 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_i_int
11657 \fp_mul_end_level:
11658 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iv_int
11659 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iii_int
11660 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_ii_int
11661 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_i_int
11662 \fp_mul_end_level:
11663 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11664 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11665 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11666 \fp_mul_end_level:
11667 #6 0 \l_fp_mul_output_tl \scan_stop:
11668 \tl_clear:N \l_fp_mul_output_tl
11669 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11670 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int
11671 \fp_mul_end_level:
11672 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11673 \fp_mul_end_level:
11674 \fp_mul_end_level:
11675 #5 0 \l_fp_mul_output_tl \scan_stop:
11676 }

```

(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.

```

11677 \cs_new_protected:Npn \fp_mul:NNNNNNNN #1#2#3#4#5#6#7#8#9
11678 {
11679   \fp_mul_split:NNNN #2
11680   \l_fp_mul_a_i_int \l_fp_mul_a_ii_int \l_fp_mul_a_iii_int
11681   \fp_mul_split:NNNN #3
11682   \l_fp_mul_a_iv_int \l_fp_mul_a_v_int \l_fp_mul_a_vi_int
11683   \fp_mul_split:NNNN #5
11684   \l_fp_mul_b_i_int \l_fp_mul_b_ii_int \l_fp_mul_b_iii_int
11685   \fp_mul_split:NNNN #6
11686   \l_fp_mul_b_iv_int \l_fp_mul_b_v_int \l_fp_mul_b_vi_int
11687   \l_fp_mul_output_int \c_zero
11688   \tl_clear:N \l_fp_mul_output_tl
11689   \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_vi_int
11690   \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_v_int
11691   \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iv_int
11692   \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_iii_int

```

```

11693 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_ii_int
11694 \fp_mul_product:NN \l_fp_mul_a_vi_int \l_fp_mul_b_i_int
11695 \tex_divide:D \l_fp_mul_output_int \c_one_thousand
11696 \fp_mul_product:NN #1 \l_fp_mul_b_vi_int
11697 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_v_int
11698 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iv_int
11699 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_iii_int
11700 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_ii_int
11701 \fp_mul_product:NN \l_fp_mul_a_v_int \l_fp_mul_b_i_int
11702 \fp_mul_product:NN \l_fp_mul_a_vi_int #4
11703 \fp_mul_end_level:
11704 \fp_mul_product:NN #1 \l_fp_mul_b_v_int
11705 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iv_int
11706 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_iii_int
11707 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_ii_int
11708 \fp_mul_product:NN \l_fp_mul_a_iv_int \l_fp_mul_b_i_int
11709 \fp_mul_product:NN \l_fp_mul_a_v_int #4
11710 \fp_mul_end_level:
11711 \fp_mul_product:NN #1 \l_fp_mul_b_iv_int
11712 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_iii_int
11713 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_ii_int
11714 \fp_mul_product:NN \l_fp_mul_a_iii_int \l_fp_mul_b_i_int
11715 \fp_mul_product:NN \l_fp_mul_a_iv_int #4
11716 \fp_mul_end_level:
11717 #9 0 \l_fp_mul_output_tl \scan_stop:
11718 \tl_clear:N \l_fp_mul_output_tl
11719 \fp_mul_product:NN #1 \l_fp_mul_b_iii_int
11720 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_ii_int
11721 \fp_mul_product:NN \l_fp_mul_a_ii_int \l_fp_mul_b_i_int
11722 \fp_mul_product:NN \l_fp_mul_a_iii_int #4
11723 \fp_mul_end_level:
11724 \fp_mul_product:NN #1 \l_fp_mul_b_ii_int
11725 \fp_mul_product:NN \l_fp_mul_a_i_int \l_fp_mul_b_i_int
11726 \fp_mul_product:NN \l_fp_mul_a_ii_int #4
11727 \fp_mul_end_level:
11728 \fp_mul_product:NN #1 \l_fp_mul_b_i_int
11729 \fp_mul_product:NN \l_fp_mul_a_i_int #4
11730 \fp_mul_end_level:
11731 #8 0 \l_fp_mul_output_tl \scan_stop:
11732 \tl_clear:N \l_fp_mul_output_tl
11733 \fp_mul_product:NN #1 #4
11734 \fp_mul_end_level:
11735 #7 0 \l_fp_mul_output_tl \scan_stop:
11736 }

```

(End definition for \fp_mul:NNNNNNNN. This function is documented on page ??.)

\fp_div_integer:NNNN 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.

```

11737 \cs_new_protected:Npn \fp_div_integer:NNNNN #1#2#3#4#5
11738 {
11739   \l_fp_internal_int #1
11740   \tex_divide:D \l_fp_internal_int #3
11741   \l_fp_internal_int \int_eval:w #1 - \l_fp_internal_int * #3 \int_eval_end:
11742   #4 #1
11743   \tex_divide:D #4 #3
11744   #5 #2
11745   \tex_divide:D #5 #3
11746   \tex_multiply:D \l_fp_internal_int \c_one_thousand
11747   \tex_divide:D \l_fp_internal_int #3
11748   #5 \int_eval:w #5 + \l_fp_internal_int * \c_one_million \int_eval_end:
11749   \if_int_compare:w #5 > \c_one_thousand_million
11750     \tex_advance:D #4 \c_one
11751     \tex_advance:D #5 -\c_one_thousand_million
11752   \fi:
11753 }

```

(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.

```

\fp_extended_normalise_aux_i:
\fp_extended_normalise_aux_i:w
\fp_extended_normalise_aux_ii:w
\fp_extended_normalise_aux_ii:
\fp_extended_normalise_aux:NNNNNNNN
11754 \cs_new_protected_nopar:Npn \fp_extended_normalise:
11755 {
11756   \fp_extended_normalise_aux_i:
11757   \fp_extended_normalise_aux_ii:
11758 }
11759 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_i:
11760 {
11761   \if_int_compare:w \l_fp_input_a_exponent_int > \c_zero
11762     \tex_multiply:D \l_fp_input_a_integer_int \c_ten
11763     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11764     \exp_after:wN \fp_extended_normalise_aux_i:w
11765     \int_use:N \l_fp_input_a_decimal_int \q_stop
11766     \exp_after:wN \fp_extended_normalise_aux_i:
11767   \fi:
11768 }
11769 \cs_new_protected:Npn \fp_extended_normalise_aux_i:w
11770 #1#2#3#4#5#6#7#8#9 \q_stop
11771 {
11772   \l_fp_input_a_integer_int
11773   \int_eval:w \l_fp_input_a_integer_int + #2 \scan_stop:
11774   \l_fp_input_a_decimal_int #3#4#5#6#7#8#9 0 \scan_stop:
11775   \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
11776   \exp_after:wN \fp_extended_normalise_aux_ii:w
11777   \int_use:N \l_fp_input_a_extended_int \q_stop
11778 }

```

```

11779 \cs_new_protected:Npn \fp_extended_normalise_aux_ii:w
11780   #1#2#3#4#5#6#7#8#9 \q_stop
11781   {
11782     \l_fp_input_a_decimal_int
11783     \int_eval:w \l_fp_input_a_decimal_int + #2 \scan_stop:
11784     \l_fp_input_a_extended_int #3#4#5#6#7#8#9 0 \scan_stop:
11785     \tex_advance:D \l_fp_input_a_exponent_int \c_minus_one
11786   }
11787 \cs_new_protected_nopar:Npn \fp_extended_normalise_aux_ii:
11788   {
11789     \if_int_compare:w \l_fp_input_a_exponent_int < \c_zero
11790     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11791     \exp_after:wN \use_i:nn \exp_after:wN
11792     \fp_extended_normalise_ii_aux:NNNNNNNNN
11793     \int_use:N \l_fp_input_a_decimal_int
11794     \exp_after:wN \fp_extended_normalise_aux_ii:
11795     \fi:
11796   }
11797 \cs_new_protected:Npn \fp_extended_normalise_ii_aux:NNNNNNNNN
11798   #1#2#3#4#5#6#7#8#9
11799   {
11800     \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
11801     \l_fp_input_a_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
11802     \else:
11803       \tl_set:Nx \l_fp_internal_tl
11804       {
11805         \int_use:N \l_fp_input_a_integer_int
11806         #1#2#3#4#5#6#7#8
11807       }
11808       \l_fp_input_a_integer_int \c_zero
11809       \l_fp_input_a_decimal_int \l_fp_internal_tl \scan_stop:
11810     \fi:
11811     \tex_divide:D \l_fp_input_a_extended_int \c_ten
11812     \tl_set:Nx \l_fp_internal_tl
11813     {
11814       #9
11815       \int_use:N \l_fp_input_a_extended_int
11816     }
11817     \l_fp_input_a_extended_int \l_fp_internal_tl \scan_stop:
11818     \tex_advance:D \l_fp_input_a_exponent_int \c_one
11819   }

```

(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
11820 \cs_new_protected_nopar:Npn \fp_extended_normalise_output:
11821   {
11822     \if_int_compare:w \l_fp_output_integer_int > \c_nine

```

```

11823     \tex_advance:D \l_fp_output_integer_int \c_one_thousand_million
11824     \exp_after:wN \use_i:nn \exp_after:wN
11825     \fp_extended_normalise_output_aux_i:NNNNNNNNN
11826     \int_use:N \l_fp_output_integer_int
11827     \exp_after:wN \fp_extended_normalise_output:
11828     \fi:
11829   }
11830 \cs_new_protected:Npn \fp_extended_normalise_output_aux_i:NNNNNNNNN
11831 #1#2#3#4#5#6#7#8#9
11832 {
11833   \l_fp_output_integer_int #1#2#3#4#5#6#7#8 \scan_stop:
11834   \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
11835   \tl_set:Nx \l_fp_internal_tl
11836   {
11837     #9
11838     \exp_after:wN \use_none:n
11839     \int_use:N \l_fp_output_decimal_int
11840   }
11841   \exp_after:wN \fp_extended_normalise_output_aux_ii:NNNNNNNNN
11842   \l_fp_internal_tl
11843 }
11844 \cs_new_protected:Npn \fp_extended_normalise_output_aux_ii:NNNNNNNNN
11845 #1#2#3#4#5#6#7#8#9
11846 {
11847   \l_fp_output_decimal_int #1#2#3#4#5#6#7#8#9 \scan_stop:
11848   \fp_extended_normalise_output_aux:N
11849 }
11850 \cs_new_protected:Npn \fp_extended_normalise_output_aux:N #1
11851 {
11852   \tex_advance:D \l_fp_output_extended_int \c_one_thousand_million
11853   \tex_divide:D \l_fp_output_extended_int \c_ten
11854   \tl_set:Nx \l_fp_internal_tl
11855   {
11856     #1
11857     \exp_after:wN \use_none:n
11858     \int_use:N \l_fp_output_extended_int
11859   }
11860   \l_fp_output_extended_int \l_fp_internal_tl \scan_stop:
11861   \tex_advance:D \l_fp_output_exponent_int \c_one
11862 }

```

(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 $-\pi < x \leq \pi$.

```

11863 \cs_new_protected_nopar:Npn \fp_trig_normalise:
11864 {

```

```

11865 \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
11866 \l_fp_input_a_extended_int \c_zero
11867 \fp_extended_normalise:
11868 \fp_trig_normalise_aux:
11869 \if_int_compare:w \l_fp_input_a_integer_int < \c_zero
11870 \l_fp_input_a_sign_int -\l_fp_input_a_sign_int
11871 \l_fp_input_a_integer_int -\l_fp_input_a_integer_int
11872 \fi:
11873 \exp_after:wN \fp_trig_octant:
11874 \else:
11875 \l_fp_input_a_sign_int \c_one
11876 \l_fp_output_integer_int \c_zero
11877 \l_fp_output_decimal_int \c_zero
11878 \l_fp_output_exponent_int \c_zero
11879 \exp_after:wN \fp_trig_overflow_msg:
11880 \fi:
11881 }
11882 \cs_new_protected_nopar:Npn \fp_trig_normalise_aux:
11883 {
11884 \if_int_compare:w \l_fp_input_a_integer_int > \c_three
11885 \fp_trig_sub:NNN
11886 \c_six \c_fp_two_pi_decimal_int \c_fp_two_pi_extended_int
11887 \exp_after:wN \fp_trig_normalise_aux:
11888 \else:
11889 \if_int_compare:w \l_fp_input_a_integer_int > \c_two
11890 \if_int_compare:w \l_fp_input_a_decimal_int > \c_fp_pi_decimal_int
11891 \fp_trig_sub:NNN
11892 \c_six \c_fp_two_pi_decimal_int \c_fp_two_pi_extended_int
11893 \exp_after:wN \exp_after:wN \exp_after:wN
11894 \exp_after:wN \exp_after:wN \exp_after:wN
11895 \exp_after:wN \fp_trig_normalise_aux:
11896 \fi:
11897 \fi:
11898 \fi:
11899 }

```

Here, there may be a sign change but there will never be any variation in the input. So a dedicated function can be used.

```

11900 \cs_new_protected:Npn \fp_trig_sub:NNN #1#2#3
11901 {
11902 \l_fp_input_a_integer_int
11903 \int_eval:w \l_fp_input_a_integer_int - #1 \int_eval_end:
11904 \l_fp_input_a_decimal_int
11905 \int_eval:w \l_fp_input_a_decimal_int - #2 \int_eval_end:
11906 \l_fp_input_a_extended_int
11907 \int_eval:w \l_fp_input_a_extended_int - #3 \int_eval_end:
11908 \if_int_compare:w \l_fp_input_a_extended_int < \c_zero
11909 \tex_advance:D \l_fp_input_a_decimal_int \c_minus_one
11910 \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
11911 \fi:

```

```

11912 \if_int_compare:w \l_fp_input_a_decimal_int < \c_zero
11913 \tex_advance:D \l_fp_input_a_integer_int \c_minus_one
11914 \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
11915 \fi:
11916 \if_int_compare:w \l_fp_input_a_integer_int < \c_zero
11917 \l_fp_input_a_sign_int -\l_fp_input_a_sign_int
11918 \if_int_compare:w
11919 \int_eval:w
11920 \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
11921 = \c_zero
11922 \l_fp_input_a_integer_int -\l_fp_input_a_integer_int
11923 \else:
11924 \l_fp_input_a_integer_int
11925 \int_eval:w
11926 - \l_fp_input_a_integer_int - \c_one
11927 \int_eval_end:
11928 \l_fp_input_a_decimal_int
11929 \int_eval:w
11930 \c_one_thousand_million - \l_fp_input_a_decimal_int
11931 \int_eval_end:
11932 \l_fp_input_a_extended_int
11933 \int_eval:w
11934 \c_one_thousand_million - \l_fp_input_a_extended_int
11935 \int_eval_end:
11936 \fi:
11937 \fi:
11938 }

```

(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!

```

11939 \cs_new_protected_nopar:Npn \fp_trig_octant:
11940 {
11941 \l_fp_trig_octant_int \c_one
11942 \fp_trig_octant_aux_i:
11943 \if_int_compare:w \l_fp_input_a_decimal_int < \c_ten
11944 \l_fp_input_a_decimal_int \c_zero
11945 \l_fp_input_a_extended_int \c_zero
11946 \fi:
11947 \if_int_odd:w \l_fp_trig_octant_int
11948 \else:
11949 \fp_sub:NNNNNNNNN
11950 \c_zero \c_fp_pi_by_four_decimal_int \c_fp_pi_by_four_extended_int
11951 \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11952 \l_fp_input_a_extended_int
11953 \l_fp_input_a_integer_int \l_fp_input_a_decimal_int

```

```

11954         \l_fp_input_a_extended_int
11955     \fi:
11956 }
11957 \cs_new_protected_nopar:Npn \fp_trig_octant_aux_i:
11958 {
11959     \if_int_compare:w \l_fp_trig_octant_int > \c_four
11960         \l_fp_trig_octant_int \c_four
11961         \l_fp_input_a_decimal_int \c_fp_pi_by_four_decimal_int
11962         \l_fp_input_a_extended_int \c_fp_pi_by_four_extended_int
11963     \else:
11964         \exp_after:wN \fp_trig_octant_aux_ii:
11965     \fi:
11966 }
11967 \cs_new_protected_nopar:Npn \fp_trig_octant_aux_ii:
11968 {
11969     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
11970         \fp_sub:NNNNNNNNN
11971         \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11972         \l_fp_input_a_extended_int
11973         \c_zero \c_fp_pi_by_four_decimal_int \c_fp_pi_by_four_extended_int
11974         \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11975         \l_fp_input_a_extended_int
11976         \tex_advance:D \l_fp_trig_octant_int \c_one
11977         \exp_after:wN \fp_trig_octant_aux_i:
11978     \else:
11979         \if_int_compare:w
11980             \l_fp_input_a_decimal_int > \c_fp_pi_by_four_decimal_int
11981         \fp_sub:NNNNNNNNN
11982             \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11983             \l_fp_input_a_extended_int
11984             \c_zero \c_fp_pi_by_four_decimal_int
11985             \c_fp_pi_by_four_extended_int
11986             \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
11987             \l_fp_input_a_extended_int
11988             \tex_advance:D \l_fp_trig_octant_int \c_one
11989         \exp_after:wN \exp_after:wN \exp_after:wN
11990         \fp_trig_octant_aux_i:
11991     \fi:
11992 \fi:
11993 }

```

(End definition for \fp_trig_octant:. This function is documented on page ??.)

<pre> \fp_sin:Nn \fp_sin:cn \fp_gsin:Nn \fp_gsin:cn \fp_sin_aux:NNn \fp_sin_aux_i: \fp_sin_aux_ii: </pre>	<p>Calculating the sine starts off in the usual way. There is a check to see if the value has already been worked out before proceeding further.</p> <pre> 11994 \cs_new_protected_nopar:Npn \fp_sin:Nn { \fp_sin_aux:NNn \tl_set:Nn } 11995 \cs_new_protected_nopar:Npn \fp_gsin:Nn { \fp_sin_aux:NNn \tl_gset:Nn } 11996 \cs_generate_variant:Nn \fp_sin:Nn { c } 11997 \cs_generate_variant:Nn \fp_gsin:Nn { c } </pre>
---	--

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} .

```

11998 \cs_new_protected:Npn \fp_sin_aux:NNn #1#2#3
11999 {
12000   \group_begin:
12001   \fp_split:Nn a {#3}
12002   \fp_standardise:NNNN
12003   \l_fp_input_a_sign_int
12004   \l_fp_input_a_integer_int
12005   \l_fp_input_a_decimal_int
12006   \l_fp_input_a_exponent_int
12007   \tl_set:Nx \l_fp_arg_tl
12008   {
12009     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12010     -
12011     \else:
12012     +
12013     \fi:
12014     \int_use:N \l_fp_input_a_integer_int
12015     .
12016     \exp_after:wN \use_none:n
12017     \int_value:w \int_eval:w
12018     \l_fp_input_a_decimal_int + \c_one_thousand_million
12019     e
12020     \int_use:N \l_fp_input_a_exponent_int
12021   }
12022   \if_int_compare:w \l_fp_input_a_exponent_int < -\c_five
12023   \cs_set_protected_nopar:Npx \fp_tmp:w
12024   {
12025     \group_end:
12026     #1 \exp_not:N #2 { \l_fp_arg_tl }
12027   }
12028   \else:
12029     \if_cs_exist:w
12030     c_fp_sin ( \l_fp_arg_tl ) _fp
12031     \cs_end:
12032     \else:
12033     \exp_after:wN \exp_after:wN \exp_after:wN
12034     \fp_sin_aux_i:
12035     \fi:
12036     \cs_set_protected_nopar:Npx \fp_tmp:w
12037     {
12038       \group_end:
12039       #1 \exp_not:N #2
12040       { \use:c { c_fp_sin ( \l_fp_arg_tl ) _fp } }
12041     }
12042     \fi:
12043     \fp_tmp:w

```

```
12044 }
```

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.

```
12045 \cs_new_protected_nopar:Npn \fp_sin_aux_i:
12046 {
12047   \fp_trig_normalise:
12048   \fp_sin_aux_ii:
12049   \if_int_compare:w \l_fp_output_integer_int = \c_one
12050     \l_fp_output_exponent_int \c_zero
12051   \else:
12052     \l_fp_output_integer_int \l_fp_output_decimal_int
12053     \l_fp_output_decimal_int \l_fp_output_extended_int
12054     \l_fp_output_exponent_int -\c_nine
12055   \fi:
12056   \fp_standardise:NNNN
12057   \l_fp_input_a_sign_int
12058   \l_fp_output_integer_int
12059   \l_fp_output_decimal_int
12060   \l_fp_output_exponent_int
12061   \tl_new:c { c_fp_sin ( \l_fp_arg_tl ) _fp }
12062   \tl_gset:cx { c_fp_sin ( \l_fp_arg_tl ) _fp }
12063   {
12064     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12065       +
12066     \else:
12067       -
12068     \fi:
12069     \int_use:N \l_fp_output_integer_int
12070     .
12071     \exp_after:wN \use_none:n
12072     \int_value:w \int_eval:w
12073     \l_fp_output_decimal_int + \c_one_thousand_million
12074     e
12075     \int_use:N \l_fp_output_exponent_int
12076   }
12077 }
12078 \cs_new_protected_nopar:Npn \fp_sin_aux_ii:
12079 {
12080   \if_case:w \l_fp_trig_octant_int
12081   \or:
12082     \exp_after:wN \fp_trig_calc_sin:
12083   \or:
12084     \exp_after:wN \fp_trig_calc_cos:
12085   \or:
12086     \exp_after:wN \fp_trig_calc_cos:
12087   \or:
12088     \exp_after:wN \fp_trig_calc_sin:
12089   \fi:
```

```

12090 }
      (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 12091 \cs_new_protected_nopar:Npn \fp_cos:Nn { \fp_cos_aux:NNn \tl_set:Nn }
\fp_gcos:Nn 12092 \cs_new_protected_nopar:Npn \fp_gcos:Nn { \fp_cos_aux:NNn \tl_gset:Nn }
\fp_gcos:cn 12093 \cs_generate_variant:Nn \fp_cos:Nn { c }
\fp_cos_aux:NNn 12094 \cs_generate_variant:Nn \fp_gcos:Nn { c }
\fp_cos_aux_i: 12095 \cs_new_protected:Npn \fp_cos_aux:NNn #1#2#3
\fp_cos_aux_ii: 12096 {
12097   \group_begin:
12098   \fp_split:Nn a {#3}
12099   \fp_standardise:NNNN
12100   \l_fp_input_a_sign_int
12101   \l_fp_input_a_integer_int
12102   \l_fp_input_a_decimal_int
12103   \l_fp_input_a_exponent_int
12104   \tl_set:Nx \l_fp_arg_tl
12105   {
12106     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12107     -
12108     \else:
12109     +
12110     \fi:
12111     \int_use:N \l_fp_input_a_integer_int
12112     .
12113     \exp_after:wN \use_none:n
12114     \int_value:w \int_eval:w
12115     \l_fp_input_a_decimal_int + \c_one_thousand_million
12116     e
12117     \int_use:N \l_fp_input_a_exponent_int
12118   }
12119   \if_cs_exist:w c_fp_cos ( \l_fp_arg_tl ) _fp \cs_end:
12120   \else:
12121     \exp_after:wN \fp_cos_aux_i:
12122   \fi:
12123   \cs_set_protected_nopar:Npx \fp_tmp:w
12124   {
12125     \group_end:
12126     #1 \exp_not:N #2
12127     { \use:c { c_fp_cos ( \l_fp_arg_tl ) _fp } }
12128   }
12129   \fp_tmp:w
12130 }

```

Almost the same as for sine: just a bit of correction for the sign of the output.

```

12131 \cs_new_protected_nopar:Npn \fp_cos_aux_i:
12132 {
12133   \fp_trig_normalise:
12134   \fp_cos_aux_ii:

```

```

12135 \if_int_compare:w \l_fp_output_integer_int = \c_one
12136 \l_fp_output_exponent_int \c_zero
12137 \else:
12138 \l_fp_output_integer_int \l_fp_output_decimal_int
12139 \l_fp_output_decimal_int \l_fp_output_extended_int
12140 \l_fp_output_exponent_int -\c_nine
12141 \fi:
12142 \fp_standardise:NNNN
12143 \l_fp_input_a_sign_int
12144 \l_fp_output_integer_int
12145 \l_fp_output_decimal_int
12146 \l_fp_output_exponent_int
12147 \tl_new:c { c_fp_cos ( \l_fp_arg_tl ) _fp }
12148 \tl_gset:cx { c_fp_cos ( \l_fp_arg_tl ) _fp }
12149 {
12150 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12151 +
12152 \else:
12153 -
12154 \fi:
12155 \int_use:N \l_fp_output_integer_int
12156 .
12157 \exp_after:wN \use_none:n
12158 \int_value:w \int_eval:w
12159 \l_fp_output_decimal_int + \c_one_thousand_million
12160 e
12161 \int_use:N \l_fp_output_exponent_int
12162 }
12163 }
12164 \cs_new_protected_nopar:Npn \fp_cos_aux_ii:
12165 {
12166 \if_case:w \l_fp_trig_octant_int
12167 \or:
12168 \exp_after:wN \fp_trig_calc_cos:
12169 \or:
12170 \exp_after:wN \fp_trig_calc_sin:
12171 \or:
12172 \exp_after:wN \fp_trig_calc_sin:
12173 \or:
12174 \exp_after:wN \fp_trig_calc_cos:
12175 \fi:
12176 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12177 \if_int_compare:w \l_fp_trig_octant_int > \c_two
12178 \l_fp_input_a_sign_int \c_minus_one
12179 \fi:
12180 \else:
12181 \if_int_compare:w \l_fp_trig_octant_int > \c_two
12182 \else:
12183 \l_fp_input_a_sign_int \c_one
12184 \fi:

```

```

12185     \fi:
12186   }
      (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:
\fp_trig_calc_Taylor:
12187 \cs_new_protected_nopar:Npn \fp_trig_calc_cos:
12188 {
12189   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12190     \l_fp_output_integer_int \c_one
12191     \l_fp_output_decimal_int \c_zero
12192   \else:
12193     \l_fp_trig_sign_int \c_minus_one
12194     \fp_mul:NNNNNN
12195     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12196     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12197     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12198     \fp_div_integer:NNNNN
12199     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12200     \c_two
12201     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12202     \l_fp_count_int \c_three
12203     \if_int_compare:w \l_fp_trig_extended_int = \c_zero
12204       \if_int_compare:w \l_fp_trig_decimal_int = \c_zero
12205         \l_fp_output_integer_int \c_one
12206         \l_fp_output_decimal_int \c_zero
12207         \l_fp_output_extended_int \c_zero
12208       \else:
12209         \l_fp_output_integer_int \c_zero
12210         \l_fp_output_decimal_int \c_one_thousand_million
12211         \l_fp_output_extended_int \c_zero
12212     \fi:
12213   \else:
12214     \l_fp_output_integer_int \c_zero
12215     \l_fp_output_decimal_int 999999999 \scan_stop:
12216     \l_fp_output_extended_int \c_one_thousand_million
12217   \fi:
12218   \tex_advance:D \l_fp_output_extended_int -\l_fp_trig_extended_int
12219   \tex_advance:D \l_fp_output_decimal_int -\l_fp_trig_decimal_int
12220   \exp_after:wN \fp_trig_calc_Taylor:
12221   \fi:
12222 }
12223 \cs_new_protected_nopar:Npn \fp_trig_calc_sin:
12224 {
12225   \l_fp_output_integer_int \c_zero
12226   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12227     \l_fp_output_decimal_int \c_zero
12228   \else:
12229     \l_fp_output_decimal_int \l_fp_input_a_decimal_int
12230     \l_fp_output_extended_int \l_fp_input_a_extended_int
12231     \l_fp_trig_sign_int \c_one

```

```

12232     \l_fp_trig_decimal_int \l_fp_input_a_decimal_int
12233     \l_fp_trig_extended_int \l_fp_input_a_extended_int
12234     \l_fp_count_int \c_two
12235     \exp_after:wN \fp_trig_calc_Taylor:
12236   \fi:
12237 }

```

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.

```

12238 \cs_new_protected_nopar:Npn \fp_trig_calc_Taylor:
12239 {
12240   \l_fp_trig_sign_int -\l_fp_trig_sign_int
12241   \fp_mul:NNNNNN
12242     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12243     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12244     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12245   \fp_mul:NNNNNN
12246     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12247     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12248     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12249   \fp_div_integer:NNNNN
12250     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12251     \l_fp_count_int
12252     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12253   \tex_advance:D \l_fp_count_int \c_one
12254   \fp_div_integer:NNNNN
12255     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12256     \l_fp_count_int
12257     \l_fp_trig_decimal_int \l_fp_trig_extended_int
12258   \tex_advance:D \l_fp_count_int \c_one
12259   \if_int_compare:w \l_fp_trig_decimal_int > \c_zero
12260     \if_int_compare:w \l_fp_trig_sign_int > \c_zero
12261       \tex_advance:D \l_fp_output_decimal_int \l_fp_trig_decimal_int
12262       \tex_advance:D \l_fp_output_extended_int
12263         \l_fp_trig_extended_int
12264       \if_int_compare:w \l_fp_output_extended_int < \c_one_thousand_million
12265         \else:
12266           \tex_advance:D \l_fp_output_decimal_int \c_one
12267           \tex_advance:D \l_fp_output_extended_int
12268             -\c_one_thousand_million
12269         \fi:
12270       \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12271         \else:
12272           \tex_advance:D \l_fp_output_integer_int \c_one
12273           \tex_advance:D \l_fp_output_decimal_int
12274             -\c_one_thousand_million
12275         \fi:
12276       \else:
12277         \tex_advance:D \l_fp_output_decimal_int -\l_fp_trig_decimal_int
12278         \tex_advance:D \l_fp_output_extended_int

```

```

12279         -\l_fp_input_a_extended_int
12280     \if_int_compare:w \l_fp_output_extended_int < \c_zero
12281         \tex_advance:D \l_fp_output_decimal_int \c_minus_one
12282         \tex_advance:D \l_fp_output_extended_int \c_one_thousand_million
12283     \fi:
12284     \if_int_compare:w \l_fp_output_decimal_int < \c_zero
12285         \tex_advance:D \l_fp_output_integer_int \c_minus_one
12286         \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
12287     \fi:
12288     \fi:
12289     \exp_after:wN \fp_trig_calc_Taylor:
12290     \fi:
12291 }

```

(End definition for \fp_trig_calc_cos:. This function is documented on page ??.)

As might be expected, tangents are calculated from the sine and cosine by division. So there is a bit of set up, the two subsidiary pieces of work are done and then a division takes place. For small numbers, the same approach is used as for sines, with the input value simply returned as is.

```

\fp_tan:Nn
\fp_tan:cn
\fp_gtan:Nn
\fp_gtan:cn
\fp_tan_aux:NNn
\fp_tan_aux_i:
\fp_tan_aux_ii:
\fp_tan_aux_iii:
\fp_tan_aux_iv:
12292 \cs_new_protected_nopar:Npn \fp_tan:Nn { \fp_tan_aux:NNn \tl_set:Nn }
12293 \cs_new_protected_nopar:Npn \fp_gtan:Nn { \fp_tan_aux:NNn \tl_gset:Nn }
12294 \cs_generate_variant:Nn \fp_tan:Nn { c }
12295 \cs_generate_variant:Nn \fp_gtan:Nn { c }
12296 \cs_new_protected:Npn \fp_tan_aux:NNn #1#2#3
12297 {
12298     \group_begin:
12299     \fp_split:Nn a {#3}
12300     \fp_standardise:NNNN
12301     \l_fp_input_a_sign_int
12302     \l_fp_input_a_integer_int
12303     \l_fp_input_a_decimal_int
12304     \l_fp_input_a_exponent_int
12305     \tl_set:Nx \l_fp_arg_tl
12306     {
12307         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12308             -
12309         \else:
12310             +
12311         \fi:
12312         \int_use:N \l_fp_input_a_integer_int
12313         .
12314         \exp_after:wN \use_none:n
12315         \int_value:w \int_eval:w
12316         \l_fp_input_a_decimal_int + \c_one_thousand_million
12317         e
12318         \int_use:N \l_fp_input_a_exponent_int
12319     }
12320     \if_int_compare:w \l_fp_input_a_exponent_int < -\c_five
12321         \cs_set_protected_nopar:Npx \fp_tmp:w

```

```

12322     {
12323         \group_end:
12324         #1 \exp_not:N #2 { \l_fp_arg_tl }
12325     }
12326 \else:
12327     \if_cs_exist:w
12328         c_fp_tan ( \l_fp_arg_tl ) _fp
12329     \cs_end:
12330 \else:
12331     \exp_after:wN \exp_after:wN \exp_after:wN
12332         \fp_tan_aux_i:
12333 \fi:
12334 \cs_set_protected_nopar:Npx \fp_tmp:w
12335     {
12336         \group_end:
12337         #1 \exp_not:N #2
12338         { \use:c { c_fp_tan ( \l_fp_arg_tl ) _fp } }
12339     }
12340 \fi:
12341 \fp_tmp:w
12342 }

```

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.

```

12343 \cs_new_protected_nopar:Npn \fp_tan_aux_i:
12344 {
12345     \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
12346         \exp_after:wN \fp_tan_aux_ii:
12347     \else:
12348         \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
12349         \c_zero_fp
12350         \exp_after:wN \fp_trig_overflow_msg:
12351     \fi:
12352 }
12353 \cs_new_protected_nopar:Npn \fp_tan_aux_ii:
12354 {
12355     \fp_trig_normalise:
12356     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12357         \if_int_compare:w \l_fp_trig_octant_int > \c_two
12358             \l_fp_output_sign_int \c_minus_one
12359         \else:
12360             \l_fp_output_sign_int \c_one
12361         \fi:
12362     \else:
12363         \if_int_compare:w \l_fp_trig_octant_int > \c_two
12364             \l_fp_output_sign_int \c_one
12365         \else:
12366             \l_fp_output_sign_int \c_minus_one

```



```

12367     \fi:
12368 \fi:
12369 \fp_cos_aux_ii:
12370 \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12371   \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
12372     \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
12373     \c_undefined_fp
12374   \else:
12375     \exp_after:wN \exp_after:wN \exp_after:wN
12376     \fp_tan_aux_iii:
12377   \fi:
12378 \else:
12379   \exp_after:wN \fp_tan_aux_iii:
12380 \fi:
12381 }

```

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.

```

12382 \cs_new_protected_nopar:Npn \fp_tan_aux_iii:
12383 {
12384   \l_fp_input_b_integer_int \l_fp_output_decimal_int
12385   \l_fp_input_b_decimal_int \l_fp_output_extended_int
12386   \l_fp_input_b_exponent_int -\c_nine
12387   \fp_standardise:NNNN
12388     \l_fp_input_b_sign_int
12389     \l_fp_input_b_integer_int
12390     \l_fp_input_b_decimal_int
12391     \l_fp_input_b_exponent_int
12392   \fp_sin_aux_ii:
12393   \l_fp_input_a_integer_int \l_fp_output_decimal_int
12394   \l_fp_input_a_decimal_int \l_fp_output_extended_int
12395   \l_fp_input_a_exponent_int -\c_nine
12396   \fp_standardise:NNNN
12397     \l_fp_input_a_sign_int
12398     \l_fp_input_a_integer_int
12399     \l_fp_input_a_decimal_int
12400     \l_fp_input_a_exponent_int
12401   \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
12402     \if_int_compare:w \l_fp_input_a_integer_int = \c_zero
12403       \cs_new_eq:cN { c_fp_tan ( \l_fp_arg_tl ) _fp }
12404       \c_zero_fp
12405     \else:
12406       \exp_after:wN \exp_after:wN \exp_after:wN \fp_tan_aux_iv:
12407     \fi:
12408   \else:
12409     \exp_after:wN \fp_tan_aux_iv:
12410   \fi:
12411 }
12412 \cs_new_protected_nopar:Npn \fp_tan_aux_iv:
12413 {

```

```

12414 \l_fp_output_integer_int \c_zero
12415 \l_fp_output_decimal_int \c_zero
12416 \cs_set_eq:NN \fp_div_store: \fp_div_store_integer:
12417 \l_fp_div_offset_int \c_one_hundred_million
12418 \fp_div_loop:
12419 \l_fp_output_exponent_int
12420 \int_eval:w
12421 \l_fp_input_a_exponent_int - \l_fp_input_b_exponent_int
12422 \int_eval_end:
12423 \fp_standardise:NNNN
12424 \l_fp_output_sign_int
12425 \l_fp_output_integer_int
12426 \l_fp_output_decimal_int
12427 \l_fp_output_exponent_int
12428 \tl_new:c { c_fp_tan ( \l_fp_arg_tl ) _fp }
12429 \tl_gset:cx { c_fp_tan ( \l_fp_arg_tl ) _fp }
12430 {
12431 \if_int_compare:w \l_fp_output_sign_int > \c_zero
12432 +
12433 \else:
12434 -
12435 \fi:
12436 \int_use:N \l_fp_output_integer_int
12437 .
12438 \exp_after:wN \use_none:n
12439 \int_value:w \int_eval:w
12440 \l_fp_output_decimal_int + \c_one_thousand_million
12441 e
12442 \int_use:N \l_fp_output_exponent_int
12443 }
12444 }

```

(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.

<code>\c_fp_exp_2_tl</code>	12445	<code>\tl_const:cn { c_fp_exp_1_tl }</code>	{ { 2 } { 718281828 } { 459045235 } { 0 } }
<code>\c_fp_exp_3_tl</code>	12446	<code>\tl_const:cn { c_fp_exp_2_tl }</code>	{ { 7 } { 389056098 } { 930650227 } { 0 } }
<code>\c_fp_exp_4_tl</code>	12447	<code>\tl_const:cn { c_fp_exp_3_tl }</code>	{ { 2 } { 008553692 } { 318766774 } { 1 } }
<code>\c_fp_exp_5_tl</code>	12448	<code>\tl_const:cn { c_fp_exp_4_tl }</code>	{ { 5 } { 459815003 } { 314423908 } { 1 } }
<code>\c_fp_exp_6_tl</code>	12449	<code>\tl_const:cn { c_fp_exp_5_tl }</code>	{ { 1 } { 484131591 } { 025766034 } { 2 } }
<code>\c_fp_exp_7_tl</code>	12450	<code>\tl_const:cn { c_fp_exp_6_tl }</code>	{ { 4 } { 034287934 } { 927351226 } { 2 } }
<code>\c_fp_exp_8_tl</code>	12451	<code>\tl_const:cn { c_fp_exp_7_tl }</code>	{ { 1 } { 096633158 } { 428458599 } { 3 } }
<code>\c_fp_exp_9_tl</code>	12452	<code>\tl_const:cn { c_fp_exp_8_tl }</code>	{ { 2 } { 980957987 } { 041728275 } { 3 } }
<code>\c_fp_exp_10_tl</code>	12453	<code>\tl_const:cn { c_fp_exp_9_tl }</code>	{ { 8 } { 103083927 } { 575384008 } { 3 } }
<code>\c_fp_exp_20_tl</code>	12454	<code>\tl_const:cn { c_fp_exp_10_tl }</code>	{ { 2 } { 202646579 } { 480671652 } { 4 } }
<code>\c_fp_exp_30_tl</code>	12455	<code>\tl_const:cn { c_fp_exp_20_tl }</code>	{ { 4 } { 851651954 } { 097902280 } { 8 } }
<code>\c_fp_exp_40_tl</code>	12456	<code>\tl_const:cn { c_fp_exp_30_tl }</code>	{ { 1 } { 068647458 } { 152446215 } { 13 } }
<code>\c_fp_exp_50_tl</code>			
<code>\c_fp_exp_60_tl</code>			
<code>\c_fp_exp_70_tl</code>			
<code>\c_fp_exp_80_tl</code>			
<code>\c_fp_exp_90_tl</code>			
<code>\c_fp_exp_100_tl</code>			
<code>\c_fp_exp_200_tl</code>			

```

12457 \tl_const:cn { c_fp_exp_40_tl } { { 2 } { 353852668 } { 370199854 } { 17 } }
12458 \tl_const:cn { c_fp_exp_50_tl } { { 5 } { 184705528 } { 587072464 } { 21 } }
12459 \tl_const:cn { c_fp_exp_60_tl } { { 1 } { 142007389 } { 815684284 } { 26 } }
12460 \tl_const:cn { c_fp_exp_70_tl } { { 2 } { 515438670 } { 919167006 } { 30 } }
12461 \tl_const:cn { c_fp_exp_80_tl } { { 5 } { 540622384 } { 393510053 } { 34 } }
12462 \tl_const:cn { c_fp_exp_90_tl } { { 1 } { 220403294 } { 317840802 } { 39 } }
12463 \tl_const:cn { c_fp_exp_100_tl } { { 2 } { 688117141 } { 816135448 } { 43 } }
12464 \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 12465 \tl_const:cn { c_fp_exp_-1_tl } { { 3 } { 678794411 } { 71442322 } { -1 } }
\c_fp_exp_-3_tl 12466 \tl_const:cn { c_fp_exp_-2_tl } { { 1 } { 353352832 } { 366132692 } { -1 } }
\c_fp_exp_-4_tl 12467 \tl_const:cn { c_fp_exp_-3_tl } { { 4 } { 978706836 } { 786394298 } { -2 } }
\c_fp_exp_-5_tl 12468 \tl_const:cn { c_fp_exp_-4_tl } { { 1 } { 831563888 } { 873418029 } { -2 } }
\c_fp_exp_-6_tl 12469 \tl_const:cn { c_fp_exp_-5_tl } { { 6 } { 737946999 } { 085467097 } { -3 } }
\c_fp_exp_-7_tl 12470 \tl_const:cn { c_fp_exp_-6_tl } { { 2 } { 478752176 } { 666358423 } { -3 } }
\c_fp_exp_-8_tl 12471 \tl_const:cn { c_fp_exp_-7_tl } { { 9 } { 118819655 } { 545162080 } { -4 } }
\c_fp_exp_-9_tl 12472 \tl_const:cn { c_fp_exp_-8_tl } { { 3 } { 354626279 } { 025118388 } { -4 } }
\c_fp_exp_-10_tl 12473 \tl_const:cn { c_fp_exp_-9_tl } { { 1 } { 234098040 } { 866795495 } { -4 } }
\c_fp_exp_-20_tl 12474 \tl_const:cn { c_fp_exp_-10_tl } { { 4 } { 539992976 } { 248451536 } { -5 } }
\c_fp_exp_-30_tl 12475 \tl_const:cn { c_fp_exp_-20_tl } { { 2 } { 061153622 } { 438557828 } { -9 } }
\c_fp_exp_-40_tl 12476 \tl_const:cn { c_fp_exp_-30_tl } { { 9 } { 357622968 } { 840174605 } { -14 } }
\c_fp_exp_-50_tl 12477 \tl_const:cn { c_fp_exp_-40_tl } { { 4 } { 248354255 } { 291588995 } { -18 } }
\c_fp_exp_-60_tl 12478 \tl_const:cn { c_fp_exp_-50_tl } { { 1 } { 928749847 } { 963917783 } { -22 } }
\c_fp_exp_-70_tl 12479 \tl_const:cn { c_fp_exp_-60_tl } { { 8 } { 756510762 } { 696520338 } { -27 } }
\c_fp_exp_-80_tl 12480 \tl_const:cn { c_fp_exp_-70_tl } { { 3 } { 975449735 } { 908646808 } { -31 } }
\c_fp_exp_-90_tl 12481 \tl_const:cn { c_fp_exp_-80_tl } { { 1 } { 804851387 } { 845415172 } { -35 } }
\c_fp_exp_-100_tl 12482 \tl_const:cn { c_fp_exp_-90_tl } { { 8 } { 194012623 } { 990515430 } { -40 } }
\c_fp_exp_-200_tl 12483 \tl_const:cn { c_fp_exp_-100_tl } { { 3 } { 720075976 } { 020835963 } { -44 } }
12484 \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 ??.)

\fp_exp:Nn The calculation of an exponent starts off starts in much the same way as the trigonometric
\fp_exp:cn functions: normalise the input, look for a pre-defined value and if one is not found hand
\fp_gexp:Nn off to the real workhorse function. The test for a definition of the result is used so that
\fp_gexp:cn overflows do not result in any outcome being defined.

```

\fp_exp_aux:NNn 12485 \cs_new_protected_nopar:Npn \fp_exp:Nn { \fp_exp_aux:NNn \tl_set:Nn }
\fp_exp_internal: 12486 \cs_new_protected_nopar:Npn \fp_gexp:Nn { \fp_exp_aux:NNn \tl_gset:Nn }
\fp_exp_aux: 12487 \cs_generate_variant:Nn \fp_exp:Nn { c }
\fp_exp_integer: 12488 \cs_generate_variant:Nn \fp_gexp:Nn { c }
\fp_exp_integer_tens: 12489 \cs_new_protected:Npn \fp_exp_aux:NNn #1#2#3
\fp_exp_integer_units: 12490 {
\fp_exp_integer_const:n 12491 \group_begin:
\fp_exp_integer_const:nnnn 12492 \fp_split:Nn a {#3}
12493 \fp_standardise:NNNN
\fp_exp_decimal: 12494 \l_fp_input_a_sign_int
\fp_exp_Taylor: 12495 \l_fp_input_a_integer_int
\fp_exp_const:Nx 12496 \l_fp_input_a_decimal_int
\fp_exp_const:cx

```

```

12497         \l_fp_input_a_exponent_int
12498     \l_fp_input_a_extended_int \c_zero
12499     \tl_set:Nx \l_fp_arg_tl
12500     {
12501         \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12502         -
12503         \else:
12504         +
12505         \fi:
12506         \int_use:N \l_fp_input_a_integer_int
12507         .
12508         \exp_after:wN \use_none:n
12509         \int_value:w \int_eval:w
12510         \l_fp_input_a_decimal_int + \c_one_thousand_million
12511         e
12512         \int_use:N \l_fp_input_a_exponent_int
12513     }
12514     \if_cs_exist:w c_fp_exp ( \l_fp_arg_tl ) _fp \cs_end:
12515     \else:
12516         \exp_after:wN \fp_exp_internal:
12517     \fi:
12518     \cs_set_protected_nopar:Npx \fp_tmp:w
12519     {
12520         \group_end:
12521         #1 \exp_not:N #2
12522         {
12523             \if_cs_exist:w c_fp_exp ( \l_fp_arg_tl ) _fp
12524             \cs_end:
12525             \use:c { c_fp_exp ( \l_fp_arg_tl ) _fp }
12526         \else:
12527             \c_zero_fp
12528         \fi:
12529         }
12530     }
12531     \fp_tmp:w
12532 }

```

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

```

12533 \cs_new_protected_nopar:Npn \fp_exp_internal:
12534 {
12535     \if_int_compare:w \l_fp_input_a_exponent_int < \c_three
12536     \fp_extended_normalise:
12537     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12538     \if_int_compare:w \l_fp_input_a_integer_int < 230 \scan_stop:
12539     \exp_after:wN \exp_after:wN \exp_after:wN
12540     \exp_after:wN \exp_after:wN \exp_after:wN
12541     \exp_after:wN \fp_exp_aux:
12542     \else:

```

```

12543         \exp_after:wN \exp_after:wN \exp_after:wN
12544         \exp_after:wN \exp_after:wN \exp_after:wN
12545         \exp_after:wN \fp_exp_overflow_msg:
12546         \fi:
12547     \else:
12548         \if_int_compare:w \l_fp_input_a_integer_int < 230 \scan_stop:
12549         \exp_after:wN \exp_after:wN \exp_after:wN
12550         \exp_after:wN \exp_after:wN \exp_after:wN
12551         \exp_after:wN \fp_exp_aux:
12552     \else:
12553         \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12554         { \c_zero_fp }
12555     \fi:
12556 \fi:
12557 \else:
12558     \exp_after:wN \fp_exp_overflow_msg:
12559 \fi:
12560 }

```

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.

```

12561 \cs_new_protected_nopar:Npn \fp_exp_aux:
12562 {
12563     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
12564     \exp_after:wN \fp_exp_integer:
12565 \else:
12566     \l_fp_output_integer_int \c_one
12567     \l_fp_output_decimal_int \c_zero
12568     \l_fp_output_extended_int \c_zero
12569     \l_fp_output_exponent_int \c_zero
12570     \exp_after:wN \fp_exp_decimal:
12571 \fi:
12572 }

```

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).

```

12573 \cs_new_protected_nopar:Npn \fp_exp_integer:
12574 {
12575     \if_int_compare:w \l_fp_input_a_integer_int < \c_one_hundred
12576     \l_fp_exp_integer_int \c_one

```

```

12577     \l_fp_exp_decimal_int  \c_zero
12578     \l_fp_exp_extended_int \c_zero
12579     \l_fp_exp_exponent_int \c_zero
12580     \exp_after:wN \fp_exp_integer_tens:
12581 \else:
12582     \tl_set:Nx \l_fp_internal_tl
12583     {
12584         \exp_after:wN \use_i:nnn
12585         \int_use:N \l_fp_input_a_integer_int
12586     }
12587     \l_fp_input_a_integer_int
12588     \int_eval:w
12589     \l_fp_input_a_integer_int - \l_fp_internal_tl 00
12590     \int_eval_end:
12591     \if_int_compare:w \l_fp_input_a_sign_int < \c_zero
12592     \if_int_compare:w \l_fp_output_integer_int > 200 \scan_stop:
12593         \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12594         { \c_zero_fp }
12595     \else:
12596         \fp_exp_integer_const:n { - \l_fp_internal_tl 00 }
12597         \exp_after:wN \exp_after:wN \exp_after:wN
12598         \exp_after:wN \exp_after:wN \exp_after:wN
12599         \exp_after:wN \fp_exp_integer_tens:
12600     \fi:
12601 \else:
12602     \fp_exp_integer_const:n { \l_fp_internal_tl 00 }
12603     \exp_after:wN \exp_after:wN \exp_after:wN
12604     \exp_after:wN \fp_exp_integer_tens:
12605 \fi:
12606 \fi:
12607 }

```

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.

```

12608 \cs_new_protected_nopar:Npn \fp_exp_integer_tens:
12609 {
12610     \l_fp_output_integer_int  \l_fp_exp_integer_int
12611     \l_fp_output_decimal_int  \l_fp_exp_decimal_int
12612     \l_fp_output_extended_int \l_fp_exp_extended_int
12613     \l_fp_output_exponent_int \l_fp_exp_exponent_int
12614     \if_int_compare:w \l_fp_input_a_integer_int > \c_nine
12615         \tl_set:Nx \l_fp_internal_tl
12616         {
12617             \exp_after:wN \use_i:nn
12618             \int_use:N \l_fp_input_a_integer_int
12619         }
12620     \l_fp_input_a_integer_int
12621     \int_eval:w
12622     \l_fp_input_a_integer_int - \l_fp_internal_tl 0

```

```

12623         \int_eval_end:
12624         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12625             \fp_exp_integer_const:n { \l_fp_internal_tl 0 }
12626         \else:
12627             \fp_exp_integer_const:n { - \l_fp_internal_tl 0 }
12628         \fi:
12629         \fp_mul:NNNNNNNNN
12630             \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12631             \l_fp_output_integer_int \l_fp_output_decimal_int
12632             \l_fp_output_extended_int
12633             \l_fp_output_integer_int \l_fp_output_decimal_int
12634             \l_fp_output_extended_int
12635         \tex_advance:D \l_fp_output_exponent_int \l_fp_exp_exponent_int
12636         \fp_extended_normalise_output:
12637     \fi:
12638     \fp_exp_integer_units:
12639 }
12640 \cs_new_protected_nopar:Npn \fp_exp_integer_units:
12641 {
12642     \if_int_compare:w \l_fp_input_a_integer_int > \c_zero
12643         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12644             \fp_exp_integer_const:n { \int_use:N \l_fp_input_a_integer_int }
12645         \else:
12646             \fp_exp_integer_const:n
12647                 { - \int_use:N \l_fp_input_a_integer_int }
12648         \fi:
12649         \fp_mul:NNNNNNNNN
12650             \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12651             \l_fp_output_integer_int \l_fp_output_decimal_int
12652             \l_fp_output_extended_int
12653             \l_fp_output_integer_int \l_fp_output_decimal_int
12654             \l_fp_output_extended_int
12655         \tex_advance:D \l_fp_output_exponent_int \l_fp_exp_exponent_int
12656         \fp_extended_normalise_output:
12657     \fi:
12658     \fp_exp_decimal:
12659 }

```

Recovery of the stored constant values into the separate registers is done with a simple expansion then assignment.

```

12660 \cs_new_protected:Npn \fp_exp_integer_const:n #1
12661 {
12662     \exp_after:wN \exp_after:wN \exp_after:wN
12663     \fp_exp_integer_const:nnnn
12664     \cs:w c_fp_exp_ #1 _tl \cs_end:
12665 }
12666 \cs_new_protected:Npn \fp_exp_integer_const:nnnn #1#2#3#4
12667 {
12668     \l_fp_exp_integer_int #1 \scan_stop:
12669     \l_fp_exp_decimal_int #2 \scan_stop:

```

```

12670     \l_fp_exp_extended_int #3 \scan_stop:
12671     \l_fp_exp_exponent_int #4 \scan_stop:
12672 }

```

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.

```

12673 \cs_new_protected_nopar:Npn \fp_exp_decimal:
12674 {
12675   \if_int_compare:w \l_fp_input_a_decimal_int > \c_zero
12676   \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
12677     \l_fp_exp_integer_int \c_one
12678     \l_fp_exp_decimal_int \l_fp_input_a_decimal_int
12679     \l_fp_exp_extended_int \l_fp_input_a_extended_int
12680   \else:
12681     \l_fp_exp_integer_int \c_zero
12682     \if_int_compare:w \l_fp_exp_extended_int = \c_zero
12683       \l_fp_exp_decimal_int
12684       \int_eval:w
12685         \c_one_thousand_million - \l_fp_input_a_decimal_int
12686       \int_eval_end:
12687     \l_fp_exp_extended_int \c_zero
12688   \else:
12689     \l_fp_exp_decimal_int
12690     \int_eval:w
12691       999999999 - \l_fp_input_a_decimal_int
12692     \scan_stop:
12693     \l_fp_exp_extended_int
12694     \int_eval:w
12695       \c_one_thousand_million - \l_fp_input_a_extended_int
12696     \int_eval_end:
12697   \fi:
12698   \fi:
12699   \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12700   \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
12701   \l_fp_input_b_extended_int \l_fp_input_a_extended_int
12702   \l_fp_count_int \c_one
12703   \fp_exp_Taylor:
12704   \fp_mul:NNNNNNNNN
12705     \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12706     \l_fp_output_integer_int \l_fp_output_decimal_int
12707     \l_fp_output_extended_int
12708     \l_fp_output_integer_int \l_fp_output_decimal_int
12709     \l_fp_output_extended_int
12710   \fi:
12711   \if_int_compare:w \l_fp_output_extended_int < \c_five_hundred_million
12712   \else:
12713     \tex_advance:D \l_fp_output_decimal_int \c_one
12714     \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million

```



```

12715 \else:
12716 \l_fp_output_decimal_int \c_zero
12717 \tex_advance:D \l_fp_output_integer_int \c_one
12718 \fi:
12719 \fi:
12720 \fp_standardise:NNNN
12721 \l_fp_output_sign_int
12722 \l_fp_output_integer_int
12723 \l_fp_output_decimal_int
12724 \l_fp_output_exponent_int
12725 \fp_exp_const:cx { c_fp_exp ( \l_fp_arg_tl ) _fp }
12726 {
12727 +
12728 \int_use:N \l_fp_output_integer_int
12729 .
12730 \exp_after:wN \use_none:n
12731 \int_value:w \int_eval:w
12732 \l_fp_output_decimal_int + \c_one_thousand_million
12733 e
12734 \int_use:N \l_fp_output_exponent_int
12735 }
12736 }

```

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`.

```

12737 \cs_new_protected_nopar:Npn \fp_exp_Taylor:
12738 {
12739 \tex_advance:D \l_fp_count_int \c_one
12740 \tex_multiply:D \l_fp_input_b_sign_int \l_fp_input_a_sign_int
12741 \fp_mul:NNNNNN
12742 \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
12743 \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12744 \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12745 \fp_div_integer:NNNNN
12746 \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12747 \l_fp_count_int
12748 \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
12749 \if_int_compare:w
12750 \int_eval:w
12751 \l_fp_input_b_decimal_int + \l_fp_input_b_extended_int
12752 > \c_zero
12753 \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
12754 \tex_advance:D \l_fp_exp_decimal_int \l_fp_input_b_decimal_int

```

```

12755 \tex_advance:D \l_fp_exp_extended_int
12756 \l_fp_input_b_extended_int
12757 \if_int_compare:w \l_fp_exp_extended_int < \c_one_thousand_million
12758 \else:
12759 \tex_advance:D \l_fp_exp_decimal_int \c_one
12760 \tex_advance:D \l_fp_exp_extended_int
12761 -\c_one_thousand_million
12762 \fi:
12763 \if_int_compare:w \l_fp_exp_decimal_int < \c_one_thousand_million
12764 \else:
12765 \tex_advance:D \l_fp_exp_integer_int \c_one
12766 \tex_advance:D \l_fp_exp_decimal_int
12767 -\c_one_thousand_million
12768 \fi:
12769 \else:
12770 \tex_advance:D \l_fp_exp_decimal_int -\l_fp_input_b_decimal_int
12771 \tex_advance:D \l_fp_exp_extended_int
12772 -\l_fp_input_a_extended_int
12773 \if_int_compare:w \l_fp_exp_extended_int < \c_zero
12774 \tex_advance:D \l_fp_exp_decimal_int \c_minus_one
12775 \tex_advance:D \l_fp_exp_extended_int \c_one_thousand_million
12776 \fi:
12777 \if_int_compare:w \l_fp_exp_decimal_int < \c_zero
12778 \tex_advance:D \l_fp_exp_integer_int \c_minus_one
12779 \tex_advance:D \l_fp_exp_decimal_int \c_one_thousand_million
12780 \fi:
12781 \fi:
12782 \exp_after:wN \fp_exp_Taylor:
12783 \fi:
12784 }

```

This is set up as a function so that the power code can redirect the effect.

```

12785 \cs_new_protected:Npn \fp_exp_const:Nx #1#2
12786 {
12787 \tl_new:N #1
12788 \tl_gset:Nx #1 {#2}
12789 }
12790 \cs_generate_variant:Nn \fp_exp_const:Nx { c }

```

(End definition for \fp_exp:Nn and \fp_exp:cn. These functions are documented on page ??.)

\c_fp_ln_10_1_tl Constants for working out logarithms: first those for the powers of ten.

```

12791 \tl_const:cn { c_fp_ln_10_1_tl } { { 2 } { 302585092 } { 994045684 } { 0 } }
12792 \tl_const:cn { c_fp_ln_10_2_tl } { { 4 } { 605170185 } { 988091368 } { 0 } }
12793 \tl_const:cn { c_fp_ln_10_3_tl } { { 6 } { 907755278 } { 982137052 } { 0 } }
12794 \tl_const:cn { c_fp_ln_10_4_tl } { { 9 } { 210340371 } { 976182736 } { 0 } }
12795 \tl_const:cn { c_fp_ln_10_5_tl } { { 1 } { 151292546 } { 497022842 } { 1 } }
12796 \tl_const:cn { c_fp_ln_10_6_tl } { { 1 } { 381551055 } { 796427410 } { 1 } }
12797 \tl_const:cn { c_fp_ln_10_7_tl } { { 1 } { 611809565 } { 095831979 } { 1 } }
12798 \tl_const:cn { c_fp_ln_10_8_tl } { { 1 } { 842068074 } { 395226547 } { 1 } }
12799 \tl_const:cn { c_fp_ln_10_9_tl } { { 2 } { 072326583 } { 694641116 } { 1 } }

```

(End definition for \c_fp_ln_10_1_t1. This function is documented on page ??.)

\c_fp_ln_2_1_t1 The smaller set for powers of two.

```

\c_fp_ln_2_2_t1 12800 \tl_const:cn { c_fp_ln_2_1_t1 } { { 0 } { 693147180 } { 559945309 } { 0 } }
\c_fp_ln_2_3_t1 12801 \tl_const:cn { c_fp_ln_2_2_t1 } { { 1 } { 386294361 } { 119890618 } { 0 } }
12802 \tl_const:cn { c_fp_ln_2_3_t1 } { { 2 } { 079441541 } { 679835928 } { 0 } }

```

(End definition for \c_fp_ln_2_1_t1. This function is documented on page ??.)

\fp_ln:Nn The approach for logarithms is again based on a mix of tables and Taylor series. Here,
\fp_ln:cn the initial validation is a bit easier and so it is set up earlier, meaning less need to escape
\fp_gln:Nn later on.

```

\fp_gln:cn 12803 \cs_new_protected_nopar:Npn \fp_ln:Nn { \fp_ln_aux:NNn \tl_set:Nn }
\fp_ln_aux:NNn 12804 \cs_new_protected_nopar:Npn \fp_gln:Nn { \fp_ln_aux:NNn \tl_gset:Nn }
\fp_ln_aux: 12805 \cs_generate_variant:Nn \fp_ln:Nn { c }
\fp_ln_exponent: 12806 \cs_generate_variant:Nn \fp_gln:Nn { c }
\fp_ln_internal: 12807 \cs_new_protected:Npn \fp_ln_aux:NNn #1#2#3
\fp_ln_exponent_tens: 12808 {
\fp_ln_exponent_units: 12809 \group_begin:
\fp_ln_normalise: 12810 \fp_split:Nn a {#3}
12811 \fp_standardise:NNNN
12812 \l_fp_input_a_sign_int
12813 \l_fp_input_a_integer_int
\fp_ln_mantissa: 12814 \l_fp_input_a_decimal_int
\fp_ln_mantissa_aux: 12815 \l_fp_input_a_exponent_int
\fp_ln_mantissa_divide_two: 12816 \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
\fp_ln_integer_const:nn 12817 \if_int_compare:w
\fp_ln_Taylor: 12818 \int_eval:w
\fp_ln_fixed: 12819 \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
\fp_ln_fixed_aux:NNNNNNNN 12820 > \c_zero
\fp_ln_Taylor_aux: 12821 \exp_after:wN \exp_after:wN \exp_after:wN \fp_ln_aux:
12822 \else:
12823 \cs_set_protected:Npx \fp_tmp:w ##1##2
12824 {
12825 \group_end:
12826 ##1 \exp_not:N ##2 { \c_zero_fp }
12827 }
12828 \exp_after:wN \exp_after:wN \exp_after:wN \fp_ln_error_msg:
12829 \fi:
12830 \else:
12831 \cs_set_protected:Npx \fp_tmp:w ##1##2
12832 {
12833 \group_end:
12834 ##1 \exp_not:N ##2 { \c_zero_fp }
12835 }
12836 \exp_after:wN \fp_ln_error_msg:
12837 \fi:
12838 \fp_tmp:w #1 #2
12839 }

```

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.

```

12840 \cs_new_protected_nopar:Npn \fp_ln_aux:
12841 {
12842   \tl_set:Nx \l_fp_arg_tl
12843   {
12844     +
12845     \int_use:N \l_fp_input_a_integer_int
12846     .
12847     \exp_after:wN \use_none:n
12848     \int_value:w \int_eval:w
12849     \l_fp_input_a_decimal_int + \c_one_thousand_million
12850     e
12851     \int_use:N \l_fp_input_a_exponent_int
12852   }
12853   \if_cs_exist:w c_fp_ln ( \l_fp_arg_tl ) _fp \cs_end:
12854   \else:
12855     \exp_after:wN \fp_ln_exponent:
12856     \fi:
12857     \cs_set_protected:Npx \fp_tmp:w ##1##2
12858     {
12859       \group_end:
12860       ##1 \exp_not:N ##2
12861       { \use:c { c_fp_ln ( \l_fp_arg_tl ) _fp } }
12862     }
12863   }

```

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.

```

12864 \cs_new_protected_nopar:Npn \fp_ln_exponent:
12865 {
12866   \fp_ln_internal:
12867   \if_int_compare:w \l_fp_output_extended_int < \c_five_hundred_million
12868   \else:
12869     \tex_advance:D \l_fp_output_decimal_int \c_one

```

```

12870 \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
12871 \else:
12872 \l_fp_output_decimal_int \c_zero
12873 \tex_advance:D \l_fp_output_integer_int \c_one
12874 \fi:
12875 \fi:
12876 \fp_standardise:NNNN
12877 \l_fp_output_sign_int
12878 \l_fp_output_integer_int
12879 \l_fp_output_decimal_int
12880 \l_fp_output_exponent_int
12881 \tl_const:cx { c_fp_ln ( \l_fp_arg_tl ) _fp }
12882 {
12883 \if_int_compare:w \l_fp_output_sign_int > \c_zero
12884 +
12885 \else:
12886 -
12887 \fi:
12888 \int_use:N \l_fp_output_integer_int
12889 .
12890 \exp_after:wN \use_none:n
12891 \int_value:w \int_eval:w
12892 \l_fp_output_decimal_int + \c_one_thousand_million
12893 \scan_stop:
12894 e
12895 \int_use:N \l_fp_output_exponent_int
12896 }
12897 }
12898 \cs_new_protected_nopar:Npn \fp_ln_internal:
12899 {
12900 \if_int_compare:w \l_fp_input_a_exponent_int < \c_zero
12901 \l_fp_input_a_exponent_int -\l_fp_input_a_exponent_int
12902 \l_fp_output_sign_int \c_minus_one
12903 \else:
12904 \l_fp_output_sign_int \c_one
12905 \fi:
12906 \if_int_compare:w \l_fp_input_a_exponent_int > \c_nine
12907 \exp_after:wN \fp_ln_exponent_tens:NN
12908 \int_use:N \l_fp_input_a_exponent_int
12909 \else:
12910 \l_fp_output_integer_int \c_zero
12911 \l_fp_output_decimal_int \c_zero
12912 \l_fp_output_extended_int \c_zero
12913 \l_fp_output_exponent_int \c_zero
12914 \fi:
12915 \fp_ln_exponent_units:
12916 }
12917 \cs_new_protected:Npn \fp_ln_exponent_tens:NN #1 #2
12918 {
12919 \l_fp_input_a_exponent_int #2 \scan_stop:

```

```

12920 \fp_ln_const:nn { 10 } { #1 }
12921 \tex_advance:D \l_fp_exp_exponent_int \c_one
12922 \l_fp_output_integer_int \l_fp_exp_integer_int
12923 \l_fp_output_decimal_int \l_fp_exp_decimal_int
12924 \l_fp_output_extended_int \l_fp_exp_extended_int
12925 \l_fp_output_exponent_int \l_fp_exp_exponent_int
12926 }

```

Next the smaller powers of ten, which will need to be combined with the above: always an additive process.

```

12927 \cs_new_protected_nopar:Npn \fp_ln_exponent_units:
12928 {
12929 \if_int_compare:w \l_fp_input_a_exponent_int > \c_zero
12930 \fp_ln_const:nn { 10 } { \int_use:N \l_fp_input_a_exponent_int }
12931 \fp_ln_normalise:
12932 \fp_add:NNNNNNNNN
12933 \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12934 \l_fp_output_integer_int \l_fp_output_decimal_int
12935 \l_fp_output_extended_int
12936 \l_fp_output_integer_int \l_fp_output_decimal_int
12937 \l_fp_output_extended_int
12938 \fi:
12939 \fp_ln_mantissa:
12940 }

```

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.

```

12941 \cs_new_protected_nopar:Npn \fp_ln_normalise:
12942 {
12943 \if_int_compare:w \l_fp_exp_exponent_int < \l_fp_output_exponent_int
12944 \tex_advance:D \l_fp_exp_decimal_int \c_one_thousand_million
12945 \exp_after:wN \use_i:nn \exp_after:wN
12946 \fp_ln_normalise_aux:NNNNNNNNN
12947 \int_use:N \l_fp_exp_decimal_int
12948 \exp_after:wN \fp_ln_normalise:
12949 \else:
12950 \l_fp_output_exponent_int \l_fp_exp_exponent_int
12951 \fi:
12952 }
12953 \cs_new_protected:Npn \fp_ln_normalise_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
12954 {
12955 \if_int_compare:w \l_fp_exp_integer_int = \c_zero
12956 \l_fp_exp_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
12957 \else:
12958 \tl_set:Nx \l_fp_internal_tl
12959 {
12960 \int_use:N \l_fp_exp_integer_int
12961 #1#2#3#4#5#6#7#8

```

```

12962     }
12963     \l_fp_exp_integer_int \c_zero
12964     \l_fp_exp_decimal_int \l_fp_internal_tl \scan_stop:
12965 \fi:
12966 \tex_divide:D \l_fp_exp_extended_int \c_ten
12967 \tl_set:Nx \l_fp_internal_tl
12968 {
12969     #9
12970     \int_use:N \l_fp_exp_extended_int
12971 }
12972 \l_fp_exp_extended_int \l_fp_internal_tl \scan_stop:
12973 \tex_advance:D \l_fp_exp_exponent_int \c_one
12974 }

```

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*.

```

12975 \cs_new_protected_nopar:Npn \fp_ln_mantissa:
12976 {
12977     \l_fp_count_int \c_zero
12978     \l_fp_input_a_extended_int \c_zero
12979     \fp_ln_mantissa_aux:
12980 \if_int_compare:w \l_fp_count_int > \c_zero
12981     \fp_ln_const:nn { 2 } { \int_use:N \l_fp_count_int }
12982     \fp_ln_normalise:
12983 \if_int_compare:w \l_fp_output_sign_int > \c_zero
12984     \exp_after:wN \fp_add:NNNNNNNNN
12985 \else:
12986     \exp_after:wN \fp_sub:NNNNNNNNN
12987 \fi:
12988 \l_fp_output_integer_int \l_fp_output_decimal_int
12989 \l_fp_output_extended_int
12990 \l_fp_exp_integer_int \l_fp_exp_decimal_int \l_fp_exp_extended_int
12991 \l_fp_output_integer_int \l_fp_output_decimal_int
12992 \l_fp_output_extended_int
12993 \fi:
12994 \if_int_compare:w
12995     \int_eval:w
12996     \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int > \c_one
12997 \exp_after:wN \fp_ln_Taylor:
12998 \fi:
12999 }
13000 \cs_new_protected_nopar:Npn \fp_ln_mantissa_aux:
13001 {
13002 \if_int_compare:w \l_fp_input_a_integer_int > \c_one
13003     \tex_advance:D \l_fp_count_int \c_one
13004     \fp_ln_mantissa_divide_two:
13005 \exp_after:wN \fp_ln_mantissa_aux:
13006 \fi:
13007 }

```

A fast one-shot division by two.

```

13008 \cs_new_protected_nopar:Npn \fp_ln_mantissa_divide_two:
13009 {
13010   \if_int_odd:w \l_fp_input_a_decimal_int
13011     \tex_advance:D \l_fp_input_a_extended_int \c_one_thousand_million
13012   \fi:
13013   \if_int_odd:w \l_fp_input_a_integer_int
13014     \tex_advance:D \l_fp_input_a_decimal_int \c_one_thousand_million
13015   \fi:
13016   \tex_divide:D \l_fp_input_a_integer_int \c_two
13017   \tex_divide:D \l_fp_input_a_decimal_int \c_two
13018   \tex_divide:D \l_fp_input_a_extended_int \c_two
13019 }

```

Recovering constants makes use of the same auxiliary code as for exponents.

```

13020 \cs_new_protected:Npn \fp_ln_const:nn #1#2
13021 {
13022   \exp_after:wN \exp_after:wN \exp_after:wN
13023   \fp_exp_integer_const:nnnn
13024   \cs:w c_fp_ln_ #1 _ #2 _t1 \cs_end:
13025 }

```

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”.

```

13026 \cs_new_protected_nopar:Npn \fp_ln_Taylor:
13027 {

```



```

13028 \group_begin:
13029   \l_fp_input_a_integer_int \c_zero
13030   \l_fp_input_a_exponent_int \c_zero
13031   \l_fp_input_b_integer_int \c_two
13032   \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
13033   \l_fp_input_b_exponent_int \c_zero
13034   \fp_div_internal:
13035   \fp_ln_fixed:
13036   \l_fp_input_a_integer_int \l_fp_output_integer_int
13037   \l_fp_input_a_decimal_int \l_fp_output_decimal_int
13038   \l_fp_input_a_extended_int \c_zero
13039   \l_fp_input_a_exponent_int \l_fp_output_exponent_int
13040   \l_fp_output_decimal_int \c_zero %^^A Bug?
13041   \l_fp_output_decimal_int \l_fp_input_a_decimal_int
13042   \l_fp_output_extended_int \l_fp_input_a_extended_int
13043   \fp_mul:NNNNNN
13044     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
13045     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
13046     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
13047   \l_fp_count_int \c_one
13048   \fp_ln_Taylor_aux:
13049   \cs_set_protected_nopar:Npx \fp_tmp:w
13050   {
13051     \group_end:
13052     \l_fp_exp_integer_int \c_zero
13053     \exp_not:N \l_fp_exp_decimal_int
13054       \int_use:N \l_fp_output_decimal_int \scan_stop:
13055     \exp_not:N \l_fp_exp_extended_int
13056       \int_use:N \l_fp_output_extended_int \scan_stop:
13057     \exp_not:N \l_fp_exp_exponent_int
13058       \int_use:N \l_fp_output_exponent_int \scan_stop:
13059   }
13060   \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.

```

13061   \tex_advance:D \l_fp_exp_decimal_int \l_fp_exp_decimal_int
13062   \if_int_compare:w \l_fp_exp_extended_int < \c_five_hundred_million
13063   \else:
13064     \tex_advance:D \l_fp_exp_extended_int -\c_five_hundred_million
13065     \tex_advance:D \l_fp_exp_decimal_int \c_one
13066   \fi:
13067   \tex_advance:D \l_fp_exp_extended_int \l_fp_exp_extended_int
13068   \fp_ln_normalise:
13069   \if_int_compare:w \l_fp_output_sign_int > \c_zero
13070     \exp_after:wN \fp_add:NNNNNNNNN
13071   \else:
13072     \exp_after:wN \fp_sub:NNNNNNNNN
13073   \fi:
13074   \l_fp_output_integer_int \l_fp_output_decimal_int

```

```

13075     \l_fp_output_extended_int
13076     \c_zero \l_fp_exp_decimal_int \l_fp_exp_extended_int
13077     \l_fp_output_integer_int \l_fp_output_decimal_int
13078     \l_fp_output_extended_int
13079 }

```

The usual shifts to move to fixed-point working. This is done using the output registers as this saves a reassignment here.

```

13080 \cs_new_protected_nopar:Npn \fp_ln_fixed:
13081 {
13082     \if_int_compare:w \l_fp_output_exponent_int < \c_zero
13083     \tex_advance:D \l_fp_output_decimal_int \c_one_thousand_million
13084     \exp_after:wN \use_i:nn \exp_after:wN
13085     \fp_ln_fixed_aux:NNNNNNNNN
13086     \int_use:N \l_fp_output_decimal_int
13087     \exp_after:wN \fp_ln_fixed:
13088     \fi:
13089 }
13090 \cs_new_protected:Npn \fp_ln_fixed_aux:NNNNNNNNN #1#2#3#4#5#6#7#8#9
13091 {
13092     \if_int_compare:w \l_fp_output_integer_int = \c_zero
13093     \l_fp_output_decimal_int #1#2#3#4#5#6#7#8 \scan_stop:
13094     \else:
13095     \tl_set:Nx \l_fp_internal_tl
13096     {
13097         \int_use:N \l_fp_output_integer_int
13098         #1#2#3#4#5#6#7#8
13099     }
13100     \l_fp_output_integer_int \c_zero
13101     \l_fp_output_decimal_int \l_fp_internal_tl \scan_stop:
13102     \fi:
13103     \tex_advance:D \l_fp_output_exponent_int \c_one
13104 }

```

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.

```

13105 \cs_new_protected_nopar:Npn \fp_ln_Taylor_aux:
13106 {
13107     \tex_advance:D \l_fp_count_int \c_two
13108     \fp_mul:NNNNNN
13109     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
13110     \l_fp_input_b_decimal_int \l_fp_input_b_extended_int
13111     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int
13112     \if_int_compare:w
13113     \int_eval:w
13114     \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
13115     > \c_zero
13116     \fp_div_integer:NNNNN
13117     \l_fp_input_a_decimal_int \l_fp_input_a_extended_int

```

```

13118 \l_fp_count_int
13119 \l_fp_exp_decimal_int \l_fp_exp_extended_int
13120 \tex_advance:D \l_fp_output_decimal_int \l_fp_exp_decimal_int
13121 \tex_advance:D \l_fp_output_extended_int \l_fp_exp_extended_int
13122 \if_int_compare:w \l_fp_output_extended_int < \c_one_thousand_million
13123 \else:
13124 \tex_advance:D \l_fp_output_decimal_int \c_one
13125 \tex_advance:D \l_fp_output_extended_int
13126 -\c_one_thousand_million
13127 \fi:
13128 \if_int_compare:w \l_fp_output_decimal_int < \c_one_thousand_million
13129 \else:
13130 \tex_advance:D \l_fp_output_integer_int \c_one
13131 \tex_advance:D \l_fp_output_decimal_int
13132 -\c_one_thousand_million
13133 \fi:
13134 \exp_after:wN \fp_ln_Taylor_aux:
13135 \fi:
13136 }

```

(End definition for `\fp_ln:Nn` and `\fp_ln:cn`. These functions are documented on page ??.)

`\fp_pow:Nn` The approach used for working out powers is to first filter out the various special cases and
`\fp_pow:cn` then do most of the work using the logarithm and exponent functions. The two storage
`\fp_gpow:Nn` areas are used in the reverse of the ‘natural’ logic as this avoids some re-assignment in
`\fp_gpow:cn` the sanity checking code.

```

\fp_pow_aux:NNn 13137 \cs_new_protected_nopar:Npn \fp_pow:Nn { \fp_pow_aux:NNn \tl_set:Nn }
\fp_pow_aux_i: 13138 \cs_new_protected_nopar:Npn \fp_gpow:Nn { \fp_pow_aux:NNn \tl_gset:Nn }
\fp_pow_positive: 13139 \cs_generate_variant:Nn \fp_pow:Nn { c }
\fp_pow_negative: 13140 \cs_generate_variant:Nn \fp_gpow:Nn { c }
\fp_pow_aux_ii: 13141 \cs_new_protected:Npn \fp_pow_aux:NNn #1#2#3
\fp_pow_aux_iii: 13142 {
\fp_pow_aux_iv: 13143 \group_begin:
13144 \fp_read:N #2
13145 \l_fp_input_b_sign_int \l_fp_input_a_sign_int
13146 \l_fp_input_b_integer_int \l_fp_input_a_integer_int
13147 \l_fp_input_b_decimal_int \l_fp_input_a_decimal_int
13148 \l_fp_input_b_exponent_int \l_fp_input_a_exponent_int
13149 \fp_split:Nn a {#3}
13150 \fp_standardise:NNNN
13151 \l_fp_input_a_sign_int
13152 \l_fp_input_a_integer_int
13153 \l_fp_input_a_decimal_int
13154 \l_fp_input_a_exponent_int
13155 \if_int_compare:w
13156 \int_eval:w
13157 \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
13158 = \c_zero
13159 \if_int_compare:w
13160 \int_eval:w

```

```

13161         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
13162         = \c_zero
13163         \cs_set_protected:Npx \fp_tmp:w ##1##2
13164         {
13165             \group_end:
13166             ##1 ##2 { \c_undefined_fp }
13167         }
13168     \else:
13169         \cs_set_protected:Npx \fp_tmp:w ##1##2
13170         {
13171             \group_end:
13172             ##1 ##2 { \c_zero_fp }
13173         }
13174     \fi:
13175 \else:
13176     \if_int_compare:w
13177         \int_eval:w
13178         \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
13179         = \c_zero
13180         \cs_set_protected:Npx \fp_tmp:w ##1##2
13181         {
13182             \group_end:
13183             ##1 ##2 { \c_one_fp }
13184         }
13185     \else:
13186         \exp_after:wN \exp_after:wN \exp_after:wN
13187         \fp_pow_aux_i:
13188     \fi:
13189     \fi:
13190     \fp_tmp:w #1 #2
13191 }

```

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.

```

13192 \cs_new_protected_nopar:Npn \fp_pow_aux_i:
13193 {
13194     \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
13195         \tl_set:Nn \l_fp_sign_tl { + }
13196         \exp_after:wN \fp_pow_aux_ii:
13197     \else:
13198         \l_fp_input_a_extended_int \c_zero
13199         \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
13200             \group_begin:
13201             \fp_extended_normalise:
13202             \if_int_compare:w
13203                 \int_eval:w
13204                 \l_fp_input_a_decimal_int + \l_fp_input_a_extended_int
13205                 = \c_zero
13206             \group_end:

```

```

13207         \tl_set:Nn \l_fp_sign_tl { - }
13208         \exp_after:wN \exp_after:wN \exp_after:wN
13209         \exp_after:wN \exp_after:wN \exp_after:wN
13210         \exp_after:wN \fp_pow_aux_ii:
13211     \else:
13212         \group_end:
13213         \cs_set_protected:Npx \fp_tmp:w ##1##2
13214         {
13215             \group_end:
13216             ##1 ##2 { \c_undefined_fp }
13217         }
13218     \fi:
13219 \else:
13220     \cs_set_protected:Npx \fp_tmp:w ##1##2
13221     {
13222         \group_end:
13223         ##1 ##2 { \c_undefined_fp }
13224     }
13225 \fi:
13226 \fi:
13227 }

```

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.

```

13228 \cs_new_protected_nopar:Npn \fp_pow_aux_ii:
13229 {
13230     \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
13231     \exp_after:wN \fp_pow_aux_iv:
13232 \else:
13233     \if_int_compare:w \l_fp_input_a_exponent_int < \c_ten
13234     \group_begin:
13235     \l_fp_input_a_extended_int \c_zero
13236     \fp_extended_normalise:
13237     \if_int_compare:w \l_fp_input_a_decimal_int = \c_zero
13238     \if_int_compare:w \l_fp_input_a_integer_int > \c_ten
13239     \group_end:
13240     \exp_after:wN \exp_after:wN \exp_after:wN
13241     \exp_after:wN \exp_after:wN \exp_after:wN
13242     \exp_after:wN \exp_after:wN \exp_after:wN
13243     \exp_after:wN \exp_after:wN \exp_after:wN
13244     \exp_after:wN \exp_after:wN \exp_after:wN
13245     \fp_pow_aux_iv:
13246 \else:
13247     \group_end:
13248     \exp_after:wN \exp_after:wN \exp_after:wN
13249     \exp_after:wN \exp_after:wN \exp_after:wN
13250     \exp_after:wN \exp_after:wN \exp_after:wN

```

```

13251         \exp_after:wN \exp_after:wN \exp_after:wN
13252         \exp_after:wN \exp_after:wN \exp_after:wN
13253         \exp_after:wN \fp_pow_aux_iii:
13254     \fi:
13255 \else:
13256     \group_end:
13257     \exp_after:wN \exp_after:wN \exp_after:wN
13258     \exp_after:wN \exp_after:wN \exp_after:wN
13259     \exp_after:wN \fp_pow_aux_iv:
13260 \fi:
13261 \else:
13262     \exp_after:wN \exp_after:wN \exp_after:wN
13263     \fp_pow_aux_iv:
13264 \fi:
13265 \fi:
13266 \cs_set_protected:Npx \fp_tmp:w ##1##2
13267 {
13268     \group_end:
13269     ##1 ##2
13270     {
13271         \l_fp_sign_tl
13272         \int_use:N \l_fp_output_integer_int
13273         .
13274         \exp_after:wN \use_none:n
13275         \int_value:w \int_eval:w
13276         \l_fp_output_decimal_int + \c_one_thousand_million
13277         e
13278         \int_use:N \l_fp_output_exponent_int
13279     }
13280 }
13281 }

```

For the small negative integer powers, the calculation is done for the positive power and the reciprocal is then taken.

```

13282 \cs_new_protected_nopar:Npn \fp_pow_aux_iii:
13283 {
13284     \l_fp_input_a_sign_int \c_one
13285     \fp_pow_aux_iv:
13286     \l_fp_input_a_integer_int \c_one
13287     \l_fp_input_a_decimal_int \c_zero
13288     \l_fp_input_a_exponent_int \c_zero
13289     \l_fp_input_b_integer_int \l_fp_output_integer_int
13290     \l_fp_input_b_decimal_int \l_fp_output_decimal_int
13291     \l_fp_input_b_exponent_int \l_fp_output_exponent_int
13292     \fp_div_internal:
13293 }

```

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.

```

13294 \cs_new_protected_nopar:Npn \fp_pow_aux_iv:
13295 {
13296   \group_begin:
13297     \l_fp_input_a_integer_int \l_fp_input_b_integer_int
13298     \l_fp_input_a_decimal_int \l_fp_input_b_decimal_int
13299     \l_fp_input_a_exponent_int \l_fp_input_b_exponent_int
13300     \fp_ln_internal:
13301     \cs_set_protected_nopar:Npx \fp_tmp:w
13302     {
13303       \group_end:
13304       \exp_not:N \l_fp_input_b_sign_int
13305       \int_use:N \l_fp_output_sign_int \scan_stop:
13306       \exp_not:N \l_fp_input_b_integer_int
13307       \int_use:N \l_fp_output_integer_int \scan_stop:
13308       \exp_not:N \l_fp_input_b_decimal_int
13309       \int_use:N \l_fp_output_decimal_int \scan_stop:
13310       \exp_not:N \l_fp_input_b_extended_int
13311       \int_use:N \l_fp_output_extended_int \scan_stop:
13312       \exp_not:N \l_fp_input_b_exponent_int
13313       \int_use:N \l_fp_output_exponent_int \scan_stop:
13314     }
13315     \fp_tmp:w
13316     \l_fp_input_a_extended_int \c_zero
13317     \fp_mul:NNNNNNNNN
13318     \l_fp_input_a_integer_int \l_fp_input_a_decimal_int
13319     \l_fp_input_a_extended_int
13320     \l_fp_input_b_integer_int \l_fp_input_b_decimal_int
13321     \l_fp_input_b_extended_int
13322     \l_fp_output_integer_int \l_fp_output_decimal_int
13323     \l_fp_output_extended_int
13324     \l_fp_output_exponent_int
13325     \int_eval:w
13326     \l_fp_input_a_exponent_int + \l_fp_input_b_exponent_int
13327     \scan_stop:
13328     \fp_extended_normalise_output:
13329     \tex_multiply:D \l_fp_input_a_sign_int \l_fp_input_b_sign_int
13330     \l_fp_input_a_integer_int \l_fp_output_integer_int
13331     \l_fp_input_a_decimal_int \l_fp_output_decimal_int
13332     \l_fp_input_a_extended_int \l_fp_output_extended_int
13333     \l_fp_input_a_exponent_int \l_fp_output_exponent_int
13334     \l_fp_output_integer_int \c_zero
13335     \l_fp_output_decimal_int \c_zero
13336     \l_fp_output_extended_int \c_zero
13337     \l_fp_output_exponent_int \c_zero
13338     \cs_set_eq:NN \fp_exp_const:Nx \use_none:nn

```

```

13339     \fp_exp_internal:
13340 }

```

(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.

```

13341 \prg_new_conditional:Npnn \fp_if_undefined:N #1 { p , T , F , TF }
13342 {
13343     \if_meaning:w #1 \c_undefined_fp
13344     \prg_return_true:
13345 }else:
13346     \prg_return_false:
13347 \fi:
13348 }

```

(End definition for \fp_if_undefined:N. This function is documented on page 168.)

`\fp_if_zero:N` Testing for a zero fixed-point is also easy.

```

13349 \prg_new_conditional:Npnn \fp_if_zero:N #1 { p , T , F , TF }
13350 {
13351     \if_meaning:w #1 \c_zero_fp
13352     \prg_return_true:
13353 }else:
13354     \prg_return_false:
13355 \fi:
13356 }

```

(End definition for \fp_if_zero:N. This function is documented on page 168.)

203.14 Floating-point conditionals

`\fp_compare:nNn` 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:NnNn
\fp_compare:NnnN
\fp_compare_aux:N
\fp_compare_=:
\fp_compare_<:
\fp_compare_<_aux:
\fp_compare_absolute_a>b:
\fp_compare_absolute_a<b:
\fp_compare_>:

```

```

13357 \prg_new_protected_conditional:Npnn \fp_compare:nNn #1#2#3 { T , F , TF }
13358 {
13359     \group_begin:
13360     \fp_split:Nn a {#1}
13361     \fp_standardise:NnnN
13362     \l_fp_input_a_sign_int
13363     \l_fp_input_a_integer_int
13364     \l_fp_input_a_decimal_int
13365     \l_fp_input_a_exponent_int
13366     \fp_split:Nn b {#3}
13367     \fp_standardise:NnnN
13368     \l_fp_input_b_sign_int
13369     \l_fp_input_b_integer_int
13370     \l_fp_input_b_decimal_int
13371     \l_fp_input_b_exponent_int

```



```

13372         \fp_compare_aux:N #2
13373     }
13374 \prg_new_protected_conditional:Npnn \fp_compare:NNN #1#2#3 { T , F , TF }
13375 {
13376     \group_begin:
13377     \fp_read:N #3
13378     \l_fp_input_b_sign_int      \l_fp_input_a_sign_int
13379     \l_fp_input_b_integer_int   \l_fp_input_a_integer_int
13380     \l_fp_input_b_decimal_int   \l_fp_input_a_decimal_int
13381     \l_fp_input_b_exponent_int  \l_fp_input_a_exponent_int
13382     \fp_read:N #1
13383     \fp_compare_aux:N #2
13384 }
13385 \cs_new_protected:Npn \fp_compare_aux:N #1
13386 {
13387     \cs_if_exist:cTF { fp_compare_#1: }
13388     { \use:c { fp_compare_#1: } }
13389     {
13390         \group_end:
13391         \prg_return_false:
13392     }
13393 }

```

For equality, the test is pretty easy as things are either equal or they are not.

```

13394 \cs_new_protected_nopar:cpn { fp_compare_=: }
13395 {
13396     \if_int_compare:w \l_fp_input_a_sign_int = \l_fp_input_b_sign_int
13397     \if_int_compare:w \l_fp_input_a_integer_int = \l_fp_input_b_integer_int
13398     \if_int_compare:w \l_fp_input_a_decimal_int = \l_fp_input_b_decimal_int
13399     \if_int_compare:w
13400         \l_fp_input_a_exponent_int = \l_fp_input_b_exponent_int
13401     \group_end:
13402     \prg_return_true:
13403     \else:
13404         \group_end:
13405         \prg_return_false:
13406     \fi:
13407     \else:
13408         \group_end:
13409         \prg_return_false:
13410     \fi:
13411     \else:
13412         \group_end:
13413         \prg_return_false:
13414     \fi:
13415     \else:
13416         \group_end:
13417         \prg_return_false:
13418     \fi:
13419 }

```

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.

```

13420 \cs_new_protected_nopar:cpn { fp_compare_>: }
13421 {
13422   \if_int_compare:w \int_eval:w
13423     \l_fp_input_a_integer_int + \l_fp_input_a_decimal_int
13424     = \c_zero
13425   \if_int_compare:w \int_eval:w
13426     \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
13427     = \c_zero
13428   \group_end:
13429   \prg_return_false:
13430 \else:
13431   \if_int_compare:w \l_fp_input_b_sign_int > \c_zero
13432   \group_end:
13433   \prg_return_false:
13434 \else:
13435   \group_end:
13436   \prg_return_true:
13437 \fi:
13438 \fi:
13439 \else:
13440   \if_int_compare:w \int_eval:w
13441     \l_fp_input_b_integer_int + \l_fp_input_b_decimal_int
13442     = \c_zero
13443   \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
13444   \group_end:
13445   \prg_return_true:
13446 \else:
13447   \group_end:
13448   \prg_return_false:
13449 \fi:
13450 \else:
13451   \use:c { fp_compare_>_aux: }
13452 \fi:
13453 \fi:
13454 }

```

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.

```

13455 \cs_new_protected_nopar:cpn { fp_compare_>_aux: }
13456 {
13457   \if_int_compare:w \l_fp_input_a_sign_int > \l_fp_input_b_sign_int
13458   \group_end:
13459   \prg_return_true:
13460 \else:
13461   \if_int_compare:w \l_fp_input_a_sign_int < \l_fp_input_b_sign_int
13462   \group_end:
13463   \prg_return_false:
13464 \else:

```

```

13465         \if_int_compare:w \l_fp_input_a_sign_int > \c_zero
13466         \use:c { fp_compare_absolute_a>b: }
13467     \else:
13468         \use:c { fp_compare_absolute_a<b: }
13469     \fi:
13470 \fi:
13471 \fi:
13472 }

```

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.

```

13473 \cs_new_protected_nopar:cpn { fp_compare_absolute_a>b: }
13474 {
13475     \if_int_compare:w \l_fp_input_a_exponent_int > \l_fp_input_b_exponent_int
13476     \group_end:
13477     \prg_return_true:
13478 \else:
13479     \if_int_compare:w \l_fp_input_a_exponent_int < \l_fp_input_b_exponent_int
13480     \group_end:
13481     \prg_return_false:
13482 \else:
13483     \if_int_compare:w \l_fp_input_a_integer_int > \l_fp_input_b_integer_int
13484     \group_end:
13485     \prg_return_true:
13486 \else:
13487     \if_int_compare:w
13488         \l_fp_input_a_integer_int < \l_fp_input_b_integer_int
13489     \group_end:
13490     \prg_return_false:
13491 \else:
13492     \if_int_compare:w
13493         \l_fp_input_a_decimal_int > \l_fp_input_b_decimal_int
13494     \group_end:
13495     \prg_return_true:
13496 \else:
13497     \group_end:
13498     \prg_return_false:
13499 \fi:
13500 \fi:
13501 \fi:
13502 \fi:
13503 \fi:
13504 }
13505 \cs_new_protected_nopar:cpn { fp_compare_absolute_a<b: }
13506 {
13507     \if_int_compare:w \l_fp_input_b_exponent_int > \l_fp_input_a_exponent_int
13508     \group_end:
13509     \prg_return_true:
13510 \else:

```

```

13511 \if_int_compare:w \l_fp_input_b_exponent_int < \l_fp_input_a_exponent_int
13512 \group_end:
13513 \prg_return_false:
13514 \else:
13515 \if_int_compare:w \l_fp_input_b_integer_int > \l_fp_input_a_integer_int
13516 \group_end:
13517 \prg_return_true:
13518 \else:
13519 \if_int_compare:w
13520 \l_fp_input_b_integer_int < \l_fp_input_a_integer_int
13521 \group_end:
13522 \prg_return_false:
13523 \else:
13524 \if_int_compare:w
13525 \l_fp_input_b_decimal_int > \l_fp_input_a_decimal_int
13526 \group_end:
13527 \prg_return_true:
13528 \else:
13529 \group_end:
13530 \prg_return_false:
13531 \fi:
13532 \fi:
13533 \fi:
13534 \fi:
13535 \fi:
13536 }

```

This is just a case of reversing the two input values and then running the tests already defined.

```

13537 \cs_new_protected_nopar:cpn { fp_compare_<: }
13538 {
13539 \tl_set:Nx \l_fp_internal_tl
13540 {
13541 \int_set:Nn \exp_not:N \l_fp_input_a_sign_int
13542 { \int_use:N \l_fp_input_b_sign_int }
13543 \int_set:Nn \exp_not:N \l_fp_input_a_integer_int
13544 { \int_use:N \l_fp_input_b_integer_int }
13545 \int_set:Nn \exp_not:N \l_fp_input_a_decimal_int
13546 { \int_use:N \l_fp_input_b_decimal_int }
13547 \int_set:Nn \exp_not:N \l_fp_input_a_exponent_int
13548 { \int_use:N \l_fp_input_b_exponent_int }
13549 \int_set:Nn \exp_not:N \l_fp_input_b_sign_int
13550 { \int_use:N \l_fp_input_a_sign_int }
13551 \int_set:Nn \exp_not:N \l_fp_input_b_integer_int
13552 { \int_use:N \l_fp_input_a_integer_int }
13553 \int_set:Nn \exp_not:N \l_fp_input_b_decimal_int
13554 { \int_use:N \l_fp_input_a_decimal_int }
13555 \int_set:Nn \exp_not:N \l_fp_input_b_exponent_int
13556 { \int_use:N \l_fp_input_a_exponent_int }
13557 }

```

```

13558 \l_fp_internal_tl
13559 \use:c { fp_compare_>: }
13560 }

```

(End definition for \fp_compare:nNn. This function is documented on page ??.)

As T_EX 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:n
\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
13561 \prg_new_protected_conditional:Npnn \fp_compare:n #1 { T , F , TF }
13562 {
13563   \group_begin:
13564   \tl_set:Nx \l_fp_internal_tl
13565   {
13566     \group_end:
13567     \fp_compare_aux_i:w #1 \exp_not:n { == \q_nil == \q_stop }
13568   }
13569   \l_fp_internal_tl
13570 }
13571 \cs_new_protected:Npn \fp_compare_aux_i:w #1 == #2 == #3 \q_stop
13572 {
13573   \quark_if_nil:nTF {#2}
13574   { \fp_compare_aux_ii:w #1 != \q_nil != \q_stop }
13575   { \fp_compare:nNnTF {#1} = {#2} \prg_return_true: \prg_return_false: }
13576 }
13577 \cs_new_protected:Npn \fp_compare_aux_ii:w #1 != #2 != #3 \q_stop
13578 {
13579   \quark_if_nil:nTF {#2}
13580   { \fp_compare_aux_iii:w #1 <= \q_nil <= \q_stop }
13581   { \fp_compare:nNnTF {#1} = {#2} \prg_return_false: \prg_return_true: }
13582 }
13583 \cs_new_protected:Npn \fp_compare_aux_iii:w #1 <= #2 <= #3 \q_stop
13584 {
13585   \quark_if_nil:nTF {#2}
13586   { \fp_compare_aux_iv:w #1 >= \q_nil >= \q_stop }
13587   { \fp_compare:nNnTF {#1} > {#2} \prg_return_false: \prg_return_true: }
13588 }
13589 \cs_new_protected:Npn \fp_compare_aux_iv:w #1 >= #2 >= #3 \q_stop
13590 {
13591   \quark_if_nil:nTF {#2}
13592   { \fp_compare_aux_v:w #1 = \q_nil \q_stop }
13593   { \fp_compare:nNnTF {#1} < {#2} \prg_return_false: \prg_return_true: }
13594 }
13595 \cs_new_protected:Npn \fp_compare_aux_v:w #1 = #2 = #3 \q_stop
13596 {
13597   \quark_if_nil:nTF {#2}
13598   { \fp_compare_aux_vi:w #1 < \q_nil < \q_stop }
13599   { \fp_compare:nNnTF {#1} = {#2} \prg_return_true: \prg_return_false: }
13600 }
13601 \cs_new_protected:Npn \fp_compare_aux_vi:w #1 < #2 < #3 \q_stop

```

```

13602 {
13603   \quark_if_nil:nTF {#2}
13604   { \fp_compare_aux_vii:w #1 > \q_nil > \q_stop }
13605   { \fp_compare:nNnTF {#1} < {#2} \prg_return_true: \prg_return_false: }
13606 }
13607 \cs_new_protected:Npn \fp_compare_aux_vii:w #1 > #2 > #3 \q_stop
13608 {
13609   \quark_if_nil:nTF {#2}
13610   { \prg_return_false: }
13611   { \fp_compare:nNnTF {#1} > {#2} \prg_return_true: \prg_return_false: }
13612 }

```

(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.

```

13613 \msg_kernel_new:nnnn { fpu } { overflow }
13614 { Number~too~big. }
13615 {
13616   The~input~given~is~too~big~for~the~LaTeX~floating~point~unit. \\
13617   Further~errors~may~well~occur!
13618 }
13619 \cs_new_protected_nopar:Npn \fp_overflow_msg:
13620 { \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.

```

13621 \msg_kernel_new:nnnn { fpu } { exponent-overflow }
13622 { Number~too~big~for~exponent~unit. }
13623 {
13624   The~exponent~of~the~input~given~is~too~big~for~the~floating~point~
13625   unit:~the~maximum~input~value~for~an~exponent~is~230.
13626 }
13627 \cs_new_protected_nopar:Npn \fp_exp_overflow_msg:
13628 { \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

```

13629 \msg_kernel_new:nnnn { fpu } { logarithm-input-error }
13630 { Invalid~input~to~ln~function. }
13631 { Logarithms~can~only~be~calculated~for~positive~numbers. }
13632 \cs_new_protected_nopar:Npn \fp_ln_error_msg: {
13633   \msg_kernel_error:nn { fpu } { logarithm-input-error }
13634 }

```

(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.

```

13635 \msg_kernel_new:nnnn { fpu } { trigonometric-overflow }
13636 { Number~too~big~for~trigonometry~unit. }
13637 {
13638   The~trigonometry~code~can~only~work~with~numbers~smaller~
13639   than~1000000000.
13640 }
13641 \cs_new_protected_nopar:Npn \fp_trig_overflow_msg:
13642 { \msg_kernel_error:nn { fpu } { trigonometric-overflow } }
      (End definition for \fp_trig_overflow_msg:. This function is documented on page ??.)
13643 </initex | package>

```

204 l3luatex implementation

```

13644 <*initex | package>

      Announce and ensure that the required packages are loaded.
13645 <*package>
13646 \ProvidesExplPackage
13647   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
13648 \package_check_loaded_expl:
13649 </package>

      An error message.
13650 \msg_kernel_new:nnnn { luatex } { bad-engine }
13651 { LuaTeX~engine~not~in~use!~Ignoring~#1. }
13652 {
13653   The~feature~you~are~using~is~only~available~
13654   with~the~LuaTeX~engine.~LaTeX3~ignored~‘#1#2’.
13655 }

\lua_now:n When LuaTeX is in use, this is all a question of primitives with new names. On the other
\lua_now:x hand, for pdfTeX and XeTeX the argument should be removed from the input stream
\lua_shipout_x:n before issuing an error. This is expandable, using \msg_expandable_kernel_error:nnn
\lua_shipout_x:x as done for V-type expansion in l3expan.
\lua_shipout:n
\lua_shipout:x
13656 \luatex_if_engine:TF
13657 {
13658   \cs_new_eq:NN \lua_now:x \luatex_directlua:D
13659   \cs_new_eq:NN \lua_shipout_x:n \luatex_latelua:D
13660 }
13661 {
13662   \cs_new:Npn \lua_now:x #1
13663   {
13664     \msg_expandable_kernel_error:nnn
13665     { luatex } { bad-engine } { \lua_now:x }
13666   }
13667   \cs_new_protected:Npn \lua_shipout_x:n #1
13668   {
13669     \msg_expandable_kernel_error:nnn

```

```

13670         { luatex } { bad-engine } { \lua_shipout_x:n }
13671     }
13672 }
13673 \cs_new:Npn \lua_now:n #1
13674 { \lua_now:x { \exp_not:n {#1} } }
13675 \cs_generate_variant:Nn \lua_shipout_x:n { x }
13676 \cs_new_protected:Npn \lua_shipout:n #1
13677 { \lua_shipout_x:n { \exp_not:n {#1} } }
13678 \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`

```

13679 \int_new:N \g_cctab_allocate_int
13680 \int_set:Nn \g_cctab_allocate_int { \c_minus_one }
13681 \int_new:N \g_cctab_stack_int
13682 \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.

```

13683 \cs_new_protected:Npn \cctab_new:N #1
13684 {
13685     \chk_if_free_cs:N #1
13686     \int_gadd:Nn \g_cctab_allocate_int { \c_two }
13687     \int_compare:nNnTF
13688         \g_cctab_allocate_int < { \c_max_register_int + \c_one }
13689     {
13690         \tex_global:D \tex_chardef:D #1 \g_cctab_allocate_int
13691         \luatex_initcatcodetable:D #1
13692     }
13693     { \msg_kernel_fatal:nxx { alloc } { out-of-registers } { cctab } }
13694 }
13695 \luatex_if_engine:F
13696 {
13697     \cs_set_protected:Npn \cctab_new:N #1
13698     {
13699         \msg_kernel_error:nxx { luatex } { bad-engine }
13700         { \exp_not:N \cctab_new:N }
13701     }
13702 }
13703 <*package>
13704 \luatex_if_engine:T

```



```

13705 {
13706   \cs_set_protected:Npn \cctab_new:N #1
13707   {
13708     \chk_if_free_cs:N #1
13709     \newcatcodetable #1
13710     \luatex_initcatcodetable:D #1
13711   }
13712 }
13713 </package>

```

(End definition for \cctab_new:N. This function is documented on page 173.)

\cctab_begin:N The aim here is to ensure that the saved tables are read-only. This is done by using a stack of tables which are not read only, and actually having them as “in use” copies.

\cctab_end:

\l_cctab_internal_tl

```

13714 \cs_new_protected:Npn \cctab_begin:N #1
13715 {
13716   \seq_gpush:Nx \g_cctab_stack_seq { \tex_the:D \luatex_catcodetable:D }
13717   \luatex_catcodetable:D #1
13718   \int_gadd:Nn \g_cctab_stack_int { \c_two }
13719   \int_compare:nNnT \g_cctab_stack_int > \c_max_register_int
13720   { \msg_kernel_fatal:nn { code } { cctab-stack-full } }
13721   \luatex_savecatcodetable:D \g_cctab_stack_int
13722   \luatex_catcodetable:D \g_cctab_stack_int
13723 }
13724 \cs_new_protected_nopar:Npn \cctab_end:
13725 {
13726   \int_gsub:Nn \g_cctab_stack_int { \c_two }
13727   \seq_if_empty:NTF \g_cctab_stack_seq
13728   { \tl_set:Nn \l_cctab_internal_tl { 0 } }
13729   { \seq_gpop:NN \g_cctab_stack_seq \l_cctab_internal_tl }
13730   \luatex_catcodetable:D \l_cctab_internal_tl \scan_stop:
13731 }
13732 \luatex_if_engine:F
13733 {
13734   \cs_set_protected:Npn \cctab_begin:N #1
13735   {
13736     \msg_kernel_error:nxxx { luatex } { bad-engine }
13737     { \exp_not:N \cctab_begin:N } {#1}
13738   }
13739   \cs_set_protected_nopar:Npn \cctab_end:
13740   {
13741     \msg_kernel_error:nxx { luatex } { bad-engine }
13742     { \exp_not:N \cctab_end: }
13743   }
13744 }
13745 <*package>
13746 \luatex_if_engine:T
13747 {
13748   \cs_set_protected:Npn \cctab_begin:N #1 { \BeginCatcodeRegime #1 }
13749   \cs_set_protected_nopar:Npn \cctab_end: { \EndCatcodeRegime }

```

```

13750 }
13751 </package>
13752 \tl_new:N \l_cctab_internal_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.

```

13753 \cs_new_protected:Npn \cctab_gset:Nn #1#2
13754 {
13755   \group_begin:
13756     #2
13757     \luatex_savecatcodetable:D #1
13758   \group_end:
13759 }
13760 \luatex_if_engine:F
13761 {
13762   \cs_set_protected:Npn \cctab_gset:Nn #1#2
13763   {
13764     \msg_kernel_error:nnxx { luatex } { bad-engine }
13765     { \exp_not:N \cctab_gset:Nn } { #1 {#2} }
13766   }
13767 }
      (End definition for \cctab_gset:Nn. This function is documented on page 173.)

```

\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

```

13768 \luatex_if_engine:T
13769 {
13770   \cctab_new:N \c_code_cctab
13771   \cctab_gset:Nn \c_code_cctab { }
13772 }
13773 <*package>
13774 \luatex_if_engine:T
13775 {
13776   \cs_new_eq:NN \c_document_cctab \CatcodeTableLaTeX
13777   \cs_new_eq:NN \c_initex_cctab \CatcodeTableIniTeX
13778   \cs_new_eq:NN \c_other_cctab \CatcodeTableOther
13779   \cs_new_eq:NN \c_str_cctab \CatcodeTableString
13780 }
13781 </package>
13782 <*initex>
13783 \luatex_if_engine:T
13784 {
13785   \cctab_new:N \c_document_cctab
13786   \cctab_new:N \c_other_cctab
13787   \cctab_new:N \c_str_cctab

```

```

13788 \cctab_gset:Nn \c_document_cctab
13789 {
13790   \char_set_catcode_space:n { 9 }
13791   \char_set_catcode_space:n { 32 }
13792   \char_set_catcode_other:n { 58 }
13793   \char_set_catcode_math_subscript:n { 95 }
13794   \char_set_catcode_active:n { 126 }
13795 }
13796 \cctab_gset:Nn \c_other_cctab
13797 {
13798   \prg_stepwise_inline:nnnn { 0 } { 1 } { 127 }
13799   { \char_set_catcode_other:n {#1} }
13800 }
13801 \cctab_gset:Nn \c_str_cctab
13802 {
13803   \prg_stepwise_inline:nnnn { 0 } { 1 } { 127 }
13804   { \char_set_catcode_other:n {#1} }
13805   \char_set_catcode_space:n { 32 }
13806 }
13807 }
13808 </initex>

```

(End definition for \c_code_cctab. This function is documented on page 174.)

204.2 Deprecated functions

Deprecated 2011-12-21, for removal by 2012-03-31.

\c_string_cctab

```

13809 \cs_new_eq:NN \c_string_cctab \c_str_cctab

```

(End definition for \c_string_cctab. This function is documented on page ??.)

```

13810 </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	
\!	8876
\"	2634, 2649
\#	5, 2649, 8549
\\$	2635, 2649
\%	2649, 8551
\	2276, 2277, 2291, 2292, 2636, 2649, 8876, 8877
*	2614, 2616, 2620, 2627, 8506, 8507
\,	9461, 9463
\-	348
\.	8824, 8829
\.bool_gset:N	153, 9803
\.bool_gset_inverse:N	153, 9807
\.bool_set:N	152, 9803
\.bool_set_inverse:N	153, 9807
\.choice:	153, 9811
\.choice_code:n	153, 9819
\.choice_code:x	153, 9819
\.choices:nn	153, 9813
\.clist_gset:N	153, 9823
\.clist_gset:c	153, 9823
\.clist_set:N	153, 9823
\.clist_set:c	153, 9823
\.code:n	154, 9815
\.code:x	154, 9815
\.default:V	154, 9831
\.default:n	154, 9831
\.dim_gset:N	154, 9835
\.dim_gset:c	154, 9835
\.dim_set:N	154, 9835
\.dim_set:c	154, 9835
\.fp_gset:N	154, 9843
\.fp_gset:c	154, 9843
\.fp_set:N	154, 9843
\.fp_set:c	154, 9843
\.generate_choices:n	155, 9851
\.int_gset:N	155, 9853
\.int_gset:c	155, 9853
\.int_set:N	155, 9853
\.int_set:c	155, 9853
\.meta:n	155, 9861
\.meta:x	155, 9861
\.multichoice:	155, 9865
\.multichoice:nn	155
\.multichoices:nn	9865
\.skip_gset:N	155, 9869
\.skip_gset:c	155, 9869
\.skip_set:N	155, 9869
\.skip_set:c	155, 9869
\.tl_gset:N	156, 9877
\.tl_gset:c	156, 9877
\.tl_gset_x:N	156, 9877
\.tl_gset_x:c	156, 9877
\.tl_set:N	156, 9877
\.tl_set:c	156, 9877
\.tl_set_x:N	156, 9877
\.tl_set_x:c	156, 9877
\.value_forbidden:	156, 9893
\.value_required:	156, 9893
\/	347
\:	1053, 2726, 2904
\::	32, 1522, 1523, 1524, 1524–1527, 1529, 1531, 1538, 1543, 1549, 1670–1697, 1699, 1704, 1706, 1711, 1716, 1753–1756
\::N	32, 1526, 1526, 1680, 1686, 1687, 1691, 1692
\::V	32, 1543, 1543, 1676
\::V_unbraced	1698, 1706
\::c	32, 1527, 1527, 1671, 1679, 1681, 1688, 1695, 1696
\::f	32, 1531, 1531, 1672–1674, 1678, 1755
\::f_unbraced	1698, 1699
\::n	32, 1525, 1525, 1671, 1674–1676, 1682, 1686, 1688, 1689, 1691, 1693, 1694, 1696, 1753
\::o	32, 1529, 1529, 1672, 1675, 1677–1679, 1683, 1684, 1686, 1687, 1689, 1690, 1692, 1694, 1697, 1754
\::o_unbraced	1698, 1704, 1753–1755
\::v	32, 1543, 1549
\::v_unbraced	1698, 1711
\::x	32, 1538, 1538, 1670, 1680–1685, 1691–1697
\::x_unbraced	1698, 1716, 1756
\;	2726, 2903, 2904
\=	9460, 9462

\? 1829
 \@ 1053, 1054, 4483, 4484
 \@@end 766
 \@@hyph 769
 \@@input 770
 \@@italiccorr 771
 \@@underline 772
 \@addtofilelist 10197
 \@currname 10112
 \@filelist 10269
 \@ifpackageloaded 216
 \@namedef 198
 \@nil 193, 201
 \@popfilename 163, 181, 183
 \@pushfilename 163, 164, 179
 \@tempa 54, 56, 64
 \@tempboxa 6642
 \\ 1829,
 2649, 8188–8191, 8196, 8464, 8471,
 8554, 8564, 8612, 8631, 8731, 8786,
 8804, 8806, 8811, 8812, 8843, 8845,
 8851, 8853, 8917, 8993, 9001, 9148,
 9170, 9197, 9204, 9211, 9220, 9228,
 9229, 9442, 10034, 10049, 10050,
 10056, 10069, 10082, 10088, 13616
 \{ 3, 2649, 8548
 \} 4, 2649, 8550, 8875
 \^ 6, 9,
 249, 2637, 2649, 3166, 3173, 8513, 9306
 _ 2291, 2638, 2649
 \| 3999
 \~ 2639, 2649, 8552

Numbers

\8 9462
 \9 9463
 _ 346, 1046, 2649, 2803,
 2825, 2845, 2863, 2876, 2887, 2897,
 8507, 8555, 8875, 9189, 9230, 9307

A

\A 2725, 4483, 4485
 \above 467
 \abovedisplayskip 480
 \abovedisplayskip 481
 \abovewithdelims 468
 \accent 518
 \adjdemerits 555

\advance 362
 \afterassignment 372
 \aftergroup 373
 \alloc_reg:nnn 8265, 8268
 \alloc_setup_type:nnn 8263, 8266
 \AtBeginDocument 10267
 \atop 469
 \atopwithdelims 470

B

\B 4484, 4486
 \badness 617
 \baselineskip 545
 \batchmode 438
 \BeginCatcodeRegime 13748
 \begingroup . 12, 61, 100, 228, 232, 338, 376
 \beginL 730
 \beginR 732
 \belowdisplayskip 482
 \belowdisplayskip 483
 \binoppenalty 506
 \bool_{:w 1930
 \bool_)_0:w 1930
 \bool_)_1:w 1930
 \bool_8_0:w 1930
 \bool_8_1:w 1930
 \bool_:w 1930
 \bool_choose:NN 1930, 2019, 2027
 \bool_cleanup:N .. 1930, 2012, 2015, 2017
 \bool_do_until:cn 2075
 \bool_do_until:Nn 2075, 2077, 2078, 2080
 \bool_do_until:nn 2081, 2102, 2105
 \bool_do_while:cn 2075
 \bool_do_while:Nn 2075, 2075, 2076, 2079
 \bool_do_while:nn 2081, 2089, 2092
 \bool_eval_skip_to_end:Nw 1930,
 2039, 2040, 2042, 2044, 2046, 2060
 \bool_eval_skip_to_end_aux:Nw
 1930, 2048, 2052
 \bool_eval_skip_to_end_aux_ii:Nw ...
 1930, 2056, 2058
 \bool_get_next:N ... 1930, 1941, 1943,
 1963, 1992, 2012, 2014, 2029–2032
 \bool_get_next:NN 1963, 1971
 \bool_get_not_next:N ... 1953, 1964, 1991
 \bool_get_not_next:NN 1964, 1984
 \bool_gset:cn 1910
 \bool_gset:Nn 36, 1910, 1912, 1915
 \bool_gset_eq:cc 1902, 1909
 \bool_gset_eq:cN 1902, 1908

- \bool_gset_eq:Nc [1902](#), [1907](#)
- \bool_gset_eq:NN [36](#), [1902](#), [1906](#)
- \bool_gset_false:c [1890](#)
- \bool_gset_false:N .. [35](#), [1890](#), [1896](#), [1901](#)
- \bool_gset_true:c [1890](#)
- \bool_gset_true:N .. [35](#), [1890](#), [1894](#), [1900](#)
- \bool_I_0:w [1930](#)
- \bool_I_1:w [1930](#)
- \bool_if:c [1916](#)
- \bool_if:N [1916](#), [1916](#)
- \bool_if:n [1930](#), [1930](#)
- \bool_if:NF
[294](#), [1926](#), [2072](#), [2078](#), [3826](#), [8076](#), [9996](#)
- \bool_if:nF [2096](#), [2105](#)
- \bool_if:NT [1925](#), [2070](#),
[2076](#), [3826](#), [3827](#), [7444](#), [9961](#), [11121](#)
- \bool_if:nT [2083](#), [2092](#), [5730](#)
- \bool_if:NTF
... [36](#), [1927](#), [3812](#), [8595](#), [9622](#), [11134](#)
- \bool_if:nTF [37](#), [3036](#), [3186](#), [4278](#), [9936](#), [9946](#)
- \bool_if_p:N [1924](#)
- \bool_if_p:n
... [1911](#), [1913](#), [1932](#), [1938](#), [2062](#), [2065](#)
- \bool_new:c [1888](#)
- \bool_new:N
... [35](#), [1888](#), [1888](#), [1889](#), [1928](#), [1929](#),
[7138](#), [8504](#), [9570](#), [9639](#), [9654](#), [10338](#)
- \bool_Not:N [1973](#), [1993](#)
- \bool_Not:w [1930](#), [1968](#), [1991](#)
- \bool_not_choose:NN [2024](#), [2028](#)
- \bool_not_cleanup:N [2014](#), [2016](#), [2022](#)
- \bool_not_Not:N [1986](#), [2002](#)
- \bool_not_Not:w [1981](#), [1992](#)
- \bool_not_p:n [37](#), [2062](#), [2062](#)
- \bool_p:w [1930](#), [1995](#), [2004](#)
- \bool_S_0:w [1930](#)
- \bool_S_1:w [1930](#)
- \bool_set:cn [1910](#)
- \bool_set:Nn [36](#), [1910](#), [1910](#), [1914](#)
- \bool_set_eq:cc [1902](#), [1905](#)
- \bool_set_eq:cN [1902](#), [1904](#)
- \bool_set_eq:Nc [1902](#), [1903](#)
- \bool_set_eq:NN [36](#), [1902](#), [1902](#)
- \bool_set_false:c [1890](#)
- \bool_set_false:N
... [35](#), [309](#), [1890](#), [1892](#), [1899](#), [7440](#),
[8074](#), [8597](#), [9591](#), [9928](#), [11111](#), [11140](#)
- \bool_set_true:c [1890](#)
- \bool_set_true:N [35](#),
[323](#), [1890](#), [1890](#), [1898](#), [7458](#), [7473](#),
[7506](#), [8546](#), [8634](#), [9586](#), [9923](#), [11148](#)
- \bool_until_do:cn [2069](#)
- \bool_until_do:Nn [37](#), [2069](#), [2071](#), [2072](#), [2074](#)
- \bool_until_do:nn .. [38](#), [2081](#), [2094](#), [2099](#)
- \bool_while_do:cn [2069](#)
- \bool_while_do:Nn [37](#), [2069](#), [2069](#), [2070](#), [2073](#)
- \bool_while_do:nn .. [38](#), [2081](#), [2081](#), [2086](#)
- \bool_xor_p:nn [37](#), [2063](#), [2063](#)
- \botmark [453](#)
- \botmarks [679](#)
- \box [661](#)
- \box_clear:c [6549](#)
- \box_clear:N
[121](#), [6549](#), [6549](#), [6553](#), [7181](#), [7237](#), [7282](#)
- \box_clear_new:c [6555](#)
- \box_clear_new:N .. [121](#), [6555](#), [6555](#), [6567](#)
- \box_clip:c [7048](#)
- \box_clip:N [125](#), [7048](#), [7048](#), [7050](#)
- \box_dp:c [6581](#), [7302](#)
- \box_dp:N [122](#),
[6581](#), [6582](#), [6585](#), [6588](#), [6791](#), [6904](#),
[6951](#), [6972](#), [6994](#), [7059](#), [7060](#), [7064](#),
[7301](#), [7349](#), [7350](#), [7398](#), [7400](#), [7417](#),
[7431](#), [7593](#), [7611](#), [7759](#), [8146](#), [8190](#)
- \box_gclear:c [6549](#)
- \box_gclear:N [121](#), [6549](#), [6551](#), [6554](#)
- \box_gclear_new:c [6555](#)
- \box_gclear_new:N .. [121](#), [6555](#), [6561](#), [6568](#)
- \box_gset_eq:cc [6569](#)
- \box_gset_eq:cN [6569](#)
- \box_gset_eq:Nc [6569](#)
- \box_gset_eq:NN
... [121](#), [6552](#), [6564](#), [6569](#), [6571](#), [6574](#)
- \box_gset_eq_clear:cc [6575](#)
- \box_gset_eq_clear:cN [6575](#)
- \box_gset_eq_clear:Nc [6575](#)
- \box_gset_eq_clear:NN [121](#), [6575](#), [6577](#), [6580](#)
- \box_gset_to_last:c [6629](#)
- \box_gset_to_last:N [126](#), [6629](#), [6631](#), [6634](#)
- \box_ht:c [6581](#), [7304](#)
- \box_ht:N [122](#), [6581](#), [6581](#), [6584](#),
[6590](#), [6790](#), [6903](#), [6950](#), [6971](#), [6993](#),
[7069](#), [7070](#), [7074](#), [7233](#), [7277](#), [7303](#),
[7348](#), [7350](#), [7394](#), [7396](#), [7417](#), [7424](#),
[7592](#), [7610](#), [7756](#), [7758](#), [8144](#), [8189](#)
- \box_if_empty:c [6623](#)
- \box_if_empty:N [6623](#), [6623](#)
- \box_if_empty:NF [6627](#)

<code>\box_if_empty:NT</code>	6626	<code>\box_scale:cnn</code>	6986
<code>\box_if_empty:NTF</code>	125, 6628	<code>\box_scale:Nnn</code> 124, 6986, 6986, 7007, 7881	
<code>\box_if_empty_p:N</code>	6625	<code>\box_scale_aux:Nnn</code> 6986, 7001, 7003, 7008	
<code>\box_if_horizontal:c</code>	6611	<code>\box_set_dp:cn</code>	6587
<code>\box_if_horizontal:N</code>	6611, 6611	<code>\box_set_dp:Nn</code>	123, 6587,
<code>\box_if_horizontal:NF</code>	6617	6587, 6594, 6817, 7033, 7060, 7067,	
<code>\box_if_horizontal:NT</code>	6616	7092, 7094, 7593, 7611, 7744, 8145	
<code>\box_if_horizontal:NTF</code>	125, 6618	<code>\box_set_eq:cc</code>	6569
<code>\box_if_horizontal_p:N</code>	6615	<code>\box_set_eq:cN</code>	6569
<code>\box_if_vertical:c</code>	6611	<code>\box_set_eq:Nc</code>	6569
<code>\box_if_vertical:N</code>	6611, 6613	<code>\box_set_eq:NN</code> . 121, 6550, 6558, 6569,	
<code>\box_if_vertical:NF</code>	6621	6569, 6572, 6573, 7292, 7613, 8149	
<code>\box_if_vertical:NT</code>	6620	<code>\box_set_eq_clear:cc</code>	6575
<code>\box_if_vertical:NTF</code>	126, 6622	<code>\box_set_eq_clear:cN</code>	6575
<code>\box_if_vertical_p:N</code>	6619	<code>\box_set_eq_clear:Nc</code>	6575
<code>\box_move_down:nn</code>		<code>\box_set_eq_clear:NN</code>	
... 122, 6600, 6606, 7064, 7091, 7739	 121, 6575, 6575, 6578, 6579	
<code>\box_move_left:nn</code>	122, 6600, 6600	<code>\box_set_ht:cn</code>	6587
<code>\box_move_right:nn</code>	122, 6600, 6602	<code>\box_set_ht:Nn</code>	123, 6587,
<code>\box_move_up:nn</code>		6589, 6593, 6816, 7032, 7070, 7077,	
122, 6600, 6604, 7074, 7099, 7631, 8141		7096, 7100, 7592, 7610, 7742, 8143	
<code>\box_new:c</code>	6541	<code>\box_set_to_last:c</code>	6629
<code>\box_new:N</code>		<code>\box_set_to_last:N</code>	
121, 6541, 6542, 6548, 6559, 6565,	 126, 6629, 6629, 6632, 6633	
6639, 6645, 6647, 6765, 7112, 7188		<code>\box_set_wd:cn</code>	6587
<code>\box_resize:cnn</code>	6898	<code>\box_set_wd:Nn</code>	
<code>\box_resize:Nnn</code> 123, 6898, 6898, 6924, 7854		123, 6587, 6591, 6595, 6818, 7044,	
<code>\box_resize_aux:Nnn</code>		7053, 7083, 7594, 7612, 7745, 8147	
.. 6898, 6918, 6920, 6925, 6961, 6981		<code>\box_show:c</code>	6648
<code>\box_resize_common:N</code> 6943, 7022, 7024, 7024		<code>\box_show:cnn</code>	6658
<code>\box_resize_to_ht_plus_dp:cn</code>	6945	<code>\box_show:N</code> ... 126, 6648, 6648, 6657, 6664	
<code>\box_resize_to_ht_plus_dp:Nn</code>		<code>\box_show:Nnn</code> . 130, 6658, 6658, 6667, 6669	
..... 124, 6945, 6945, 6965		<code>\box_show_full:c</code>	6658
<code>\box_resize_to_wd:cn</code>	6945	<code>\box_show_full:N</code> .. 130, 6658, 6668, 6670	
<code>\box_resize_to_wd:Nn</code> 124, 6945, 6966, 6985		<code>\box_trim:cnnnn</code>	7051
<code>\box_rotate:Nn</code> 124, 6771, 6771, 7734		<code>\box_trim:Nnnnn</code> ... 125, 7051, 7051, 7080	
<code>\box_rotate_aux:N</code> 6771, 6782, 6784, 6788		<code>\box_use:c</code>	6596
<code>\box_rotate_quadrant_four:</code>		<code>\box_use:N</code> 122, 6596, 6597, 6599,	
..... 6771, 6803, 6885		6781, 6805, 6812, 6820, 6917, 6960,	
<code>\box_rotate_quadrant_one:</code> 6771, 6797, 6852		6980, 7000, 7029, 7039, 7045, 7057,	
<code>\box_rotate_quadrant_three:</code>		7065, 7075, 7087, 7091, 7099, 7628,	
..... 6771, 6802, 6874		7631, 7709, 7740, 8068, 8138, 8141	
<code>\box_rotate_quadrant_two:</code> 6771, 6798, 6863		<code>\box_use_clear:c</code>	6596
<code>\box_rotate_set_sin_cos:</code> 6771, 6777, 6822		<code>\box_use_clear:N</code> .. 122, 6596, 6596, 6598	
<code>\box_rotate_x:nnN</code>		<code>\box_viewport:cnnnn</code>	7081
..... 6771, 6830, 6858, 6860,		<code>\box_viewport:Nnnnn</code> 125, 7081, 7081, 7103	
6869, 6871, 6880, 6882, 6891, 6893		<code>\box_wd:c</code>	6581, 7306
<code>\box_rotate_y:nnN</code>		<code>\box_wd:N</code> 123, 6581, 6583, 6586,	
..... 6771, 6841, 6854, 6856,		6592, 6792, 6905, 6952, 6973, 6995,	
6865, 6867, 6876, 6878, 6887, 6889		7053, 7305, 7351, 7396, 7400, 7406,	

7411, 7561, 7594, 7612, 7629, 7758, 7763, 7923, 7930, 8139, 8148, 8191	\c_fp_exp-100_tl	12465
\boxmaxdepth 623	\c_fp_exp-10_tl	12465
\brokenpenalty 580	\c_fp_exp-1_tl	12465
C	\c_fp_exp-200_tl	12465
\C 1835	\c_fp_exp-20_tl	12465
\c_active_char_token 3235, 3236	\c_fp_exp-2_tl	12465
\c_alignment_tab_token 3229, 3230	\c_fp_exp-30_tl	12465
\c_alignment_token	\c_fp_exp-3_tl	12465
. 51, 2611, 2617, 2667, 3230	\c_fp_exp-40_tl	12465
\c_catcode_active_tl	\c_fp_exp-4_tl	12465
. 51, 2626, 2628, 2705, 3236	\c_fp_exp-50_tl	12465
\c_catcode_letter_token	\c_fp_exp-5_tl	12465
. 51, 2611, 2623, 2695, 3232	\c_fp_exp-60_tl	12465
\c_catcode_other_space_tl	\c_fp_exp-6_tl	12465
139, 8505, 8508, 8518, 8520, 8522, 8555	\c_fp_exp-70_tl	12465
\c_catcode_other_token	\c_fp_exp-7_tl	12465
. 51, 2611, 2624, 2700, 3233	\c_fp_exp-80_tl	12465
\c_code_cctab . . . 173, 13768, 13770, 13771	\c_fp_exp-8_tl	12465
\c_coffin_corners_prop	\c_fp_exp-90_tl	12465
. 7116, 7116-7120, 7192, 7320	\c_fp_exp-9_tl	12465
\c_coffin_poles_prop 7121, 7121, 7123-7125, 7127-7132, 7194, 7322	\c_fp_exp_100_tl	12445
\c_document_cctab	\c_fp_exp_10_tl	12445
. . . . 174, 13768, 13776, 13785, 13788	\c_fp_exp_1_tl	12445
\c_e_fp 170, 10297, 10297	\c_fp_exp_200_tl	12445
\c_eight 69, 2539, 2571, 3888, 3953, 3958, 8240, 11112	\c_fp_exp_20_tl	12445
\c_eleven 69, 2545, 2577, 3953, 3961, 8243	\c_fp_exp_2_tl	12445
\c_empty_box 126, 6550, 6552, 6558, 6564, 6635, 6636, 6639, 7708	\c_fp_exp_30_tl	12445
\c_empty_coffin . . 7297, 7297, 7298, 7706	\c_fp_exp_3_tl	12445
\c_empty_prop . 120, 6237, 6237-6243, 6356	\c_fp_exp_40_tl	12445
\c_empty_tl 95, 3701, 4380, 4395, 4395, 4397, 4399, 4602, 5456, 5551	\c_fp_exp_4_tl	12445
\c_false_bool	\c_fp_exp_50_tl	12445
. . 21, 972, 1001, 1041, 1042, 1071, 1429, 1431, 1440, 1452, 1888, 1893, 1897, 1997, 2008, 2033, 2036, 2037, 2039, 2042, 2066, 3801, 3806, 3815	\c_fp_exp_5_tl	12445
\c_fifteen 69, 2553, 2585, 3953, 3964, 8247	\c_fp_exp_60_tl	12445
\c_five 69, 2533, 2565, 3953, 3957, 8237, 10765, 12022, 12320	\c_fp_exp_6_tl	12445
\c_five_hundred_million 10280, 10283, 10802, 12711, 12867, 13062, 13064	\c_fp_exp_70_tl	12445
\c_forty_four 10280, 10280, 10930	\c_fp_exp_7_tl	12445
\c_four 69, 2531, 2563, 3953, 3956, 8236, 8638, 8644, 10911, 11147, 11959, 11960	\c_fp_exp_80_tl	12445
\c_fourteen 69, 2551, 2583, 3953, 3963, 8246	\c_fp_exp_8_tl	12445
	\c_fp_exp_90_tl	12445
	\c_fp_exp_9_tl	12445
	\c_fp_ln_10_1_tl	12791
	\c_fp_ln_10_2_tl	12791
	\c_fp_ln_10_3_tl	12791
	\c_fp_ln_10_4_tl	12791
	\c_fp_ln_10_5_tl	12791
	\c_fp_ln_10_6_tl	12791
	\c_fp_ln_10_7_tl	12791
	\c_fp_ln_10_8_tl	12791
	\c_fp_ln_10_9_tl	12791
	\c_fp_ln_2_1_tl	12800

- \c_fp_ln_2_2_tl [12800](#)
- \c_fp_ln_2_3_tl [12800](#)
- \c_fp_pi_by_four_decimal_int
..... [10285](#), [10285](#), [10286](#),
[11950](#), [11961](#), [11973](#), [11980](#), [11984](#)
- \c_fp_pi_by_four_extended_int
..... [10285](#), [10287](#),
[10288](#), [11950](#), [11962](#), [11973](#), [11985](#)
- \c_fp_pi_decimal_int
..... [10285](#), [10289](#), [10290](#), [11890](#)
- \c_fp_pi_extended_int [10285](#), [10291](#), [10292](#)
- \c_fp_two_pi_decimal_int
.. [10285](#), [10293](#), [10294](#), [11886](#), [11892](#)
- \c_fp_two_pi_extended_int
.. [10285](#), [10295](#), [10296](#), [11886](#), [11892](#)
- \c_group_begin_token
..... [51](#), [2611](#), [2611](#), [2652](#),
[3038](#), [3188](#), [4853](#), [4887](#), [6683](#), [6731](#)
- \c_group_end_token ... [51](#), [2611](#), [2612](#),
[2657](#), [3039](#), [3189](#), [6688](#), [6689](#), [6739](#)
- \c_initex_cctab [174](#), [13768](#), [13777](#)
- \c_int_from_roman_C_int [3889](#)
- \c_int_from_roman_c_int [3889](#)
- \c_int_from_roman_D_int [3889](#)
- \c_int_from_roman_d_int [3889](#)
- \c_int_from_roman_I_int [3889](#)
- \c_int_from_roman_i_int [3889](#)
- \c_int_from_roman_L_int [3889](#)
- \c_int_from_roman_l_int [3889](#)
- \c_int_from_roman_M_int [3889](#)
- \c_int_from_roman_m_int [3889](#)
- \c_int_from_roman_V_int [3889](#)
- \c_int_from_roman_v_int [3889](#)
- \c_int_from_roman_X_int [3889](#)
- \c_int_from_roman_x_int [3889](#)
- \c_iior_streams_tl [8230](#), [8249](#), [8310](#)
- \c_iow_streams_tl [8230](#), [8230](#), [8249](#), [8323](#)
- \c_iow_wrap_end_marker_tl ... [8510](#), [8569](#)
- \c_iow_wrap_indent_marker_tl [8510](#), [8533](#)
- \c_iow_wrap_marker_tl
..... [8510](#), [8512](#), [8519](#), [8579](#), [8624](#)
- \c_iow_wrap_newline_marker_tl [8510](#), [8554](#)
- \c_iow_wrap_unindent_marker_tl
..... [8510](#), [8535](#)
- \c_job_name_tl [95](#), [4943](#), [4954](#)
- \c_keys_code_root_tl
..... [9561](#), [9561](#), [9733](#), [9738](#),
[10002](#), [10004](#), [10016](#), [10022](#), [10027](#)
- \c_keys_props_root_tl
..... [9563](#), [9563](#), [9596](#), [9626](#),
[9633](#), [9803](#), [9805](#), [9807](#), [9809](#), [9811](#),
[9813](#), [9815](#), [9817](#), [9819](#), [9821](#), [9823](#),
[9825](#), [9827](#), [9829](#), [9831](#), [9833](#), [9835](#),
[9837](#), [9839](#), [9841](#), [9843](#), [9845](#), [9847](#),
[9849](#), [9851](#), [9853](#), [9855](#), [9857](#), [9859](#),
[9861](#), [9863](#), [9865](#), [9867](#), [9869](#), [9871](#),
[9873](#), [9875](#), [9877](#), [9879](#), [9881](#), [9883](#),
[9885](#), [9887](#), [9889](#), [9891](#), [9893](#), [9895](#)
- \c_keys_value_forbidden_tl .. [9564](#), [9564](#)
- \c_keys_value_required_tl ... [9564](#), [9565](#)
- \c_keys_vars_root_tl [9561](#), [9562](#), [9696](#),
[9715](#), [9722](#), [9725](#), [9727](#), [9742](#)–[9744](#),
[9747](#), [9790](#), [9963](#), [9965](#), [9968](#), [9976](#)
- \c_letter_token [3229](#), [3232](#)
- \c_log_iow ... [140](#), [8228](#), [8228](#), [8483](#), [8484](#)
- \c_luatex_is_engine_bool [1500](#), [1501](#)
- \c_math_shift_token [3229](#), [3231](#)
- \c_math_subscript_token
..... [51](#), [2611](#), [2621](#), [2685](#)
- \c_math_superscript_token
..... [51](#), [2611](#), [2619](#), [2680](#)
- \c_math_toggle_token
..... [51](#), [2611](#), [2615](#), [2662](#), [3231](#)
- \c_max_const_int . [3383](#), [3387](#), [3407](#), [3411](#)
- \c_max_dim [76](#), [4213](#), [4215](#),
[4216](#), [4220](#), [4303](#), [7808](#)–[7811](#), [7824](#)
- \c_max_int [69](#), [3971](#), [3971](#), [6669](#)
- \c_max_register_int
..... [69](#), [845](#), [846](#), [848](#), [13688](#), [13719](#)
- \c_max_skip [79](#), [4302](#), [4303](#)
- \c_minus_one [69](#), [833](#),
[834](#), [837](#), [838](#), [1154](#), [3385](#), [3444](#),
[3953](#), [4502](#), [4503](#), [8228](#), [8426](#), [8439](#),
[8511](#), [8547](#), [10355](#), [10463](#), [10898](#),
[11152](#), [11153](#), [11290](#), [11325](#), [11568](#),
[11616](#), [11620](#), [11785](#), [11909](#), [11913](#),
[12178](#), [12193](#), [12281](#), [12285](#), [12358](#),
[12366](#), [12774](#), [12778](#), [12902](#), [13680](#)
- \c_msg_coding_error_text_tl
... [8171](#), [8182](#), [8783](#), [8783](#), [9188](#),
[9196](#), [9218](#), [9226](#), [9235](#), [9242](#), [9249](#),
[9256](#), [9263](#), [10040](#), [10047](#), [10068](#), [10075](#)
- \c_msg_continue_text_tl [8783](#), [8788](#), [8845](#)
- \c_msg_critical_text_tl [8783](#), [8790](#), [8964](#)
- \c_msg_fatal_text_tl [8783](#), [8792](#), [8953](#), [9095](#)
- \c_msg_help_text_tl [8783](#), [8794](#), [8853](#)
- \c_msg_hide_tl [8824](#), [8826](#)–[8828](#), [8890](#)
- \c_msg_hide_tl<dots> [8824](#)
- \c_msg_kernel_bug_more_text_tl
..... [9428](#), [9436](#), [9440](#)

- \c_msg_kernel_bug_text_tl [9428](#), [9431](#), [9438](#)
- \c_msg_more_text_prefix_tl [8752](#), [8753](#),
[8769](#), [8778](#), [8969](#), [8977](#), [9108](#), [9118](#)
- \c_msg_no_info_text_tl . [8783](#), [8796](#), [8843](#)
- \c_msg_on_line_text_tl [8801](#), [8820](#)
- \c_msg_on_line_tl [8783](#)
- \c_msg_return_text_tl [144](#), [8783](#), [8799](#),
[8802](#), [9191](#), [9199](#), [9206](#), [9213](#), [9444](#)
- \c_msg_text_prefix_tl . . . [8752](#), [8752](#),
[8756](#), [8767](#), [8776](#), [8950](#), [8961](#), [8974](#),
[8983](#), [8994](#), [9002](#), [9008](#), [9038](#), [9091](#),
[9113](#), [9126](#), [9149](#), [9171](#), [9333](#), [9363](#)
- \c_msg_trouble_text_tl . [144](#), [8783](#), [8809](#)
- \c_nine [69](#), [2541](#), [2573](#), [3953](#),
[3959](#), [8241](#), [10468](#), [11822](#), [12054](#),
[12140](#), [12386](#), [12395](#), [12614](#), [12906](#)
- \c_one [69](#), [2525](#), [2557](#),
[3442](#), [3953](#), [3953](#), [4741](#), [6102](#), [6663](#),
[8233](#), [10357](#), [10368](#), [10427](#), [10439](#),
[10487](#), [10493](#), [10535](#), [10560](#), [10763](#),
[11122](#), [11126](#), [11128](#), [11135](#), [11275](#),
[11304](#), [11564](#), [11601](#), [11606](#), [11627](#),
[11750](#), [11818](#), [11861](#), [11875](#), [11926](#),
[11941](#), [11976](#), [11988](#), [12049](#), [12135](#),
[12183](#), [12190](#), [12205](#), [12231](#), [12253](#),
[12258](#), [12266](#), [12272](#), [12360](#), [12364](#),
[12566](#), [12576](#), [12677](#), [12702](#), [12713](#),
[12717](#), [12739](#), [12759](#), [12765](#), [12869](#),
[12873](#), [12904](#), [12921](#), [12973](#), [12996](#),
[13002](#), [13003](#), [13047](#), [13065](#), [13103](#),
[13124](#), [13130](#), [13284](#), [13286](#), [13688](#)
- \c_one_fp . [170](#), [6780](#), [6914](#), [6916](#), [6959](#),
[6979](#), [6997](#), [6999](#), [10298](#), [10298](#), [13183](#)
- \c_one_hundred
. [69](#), [3968](#), [3968](#), [10490](#), [10491](#), [12575](#)
- \c_one_hundred_million
. [10280](#), [10282](#), [11485](#), [12417](#)
- \c_one_million [10280](#), [10281](#), [11748](#)
- \c_one_thousand [69](#),
[3968](#), [3969](#), [11395](#), [11651](#), [11695](#), [11746](#)
- \c_one_thousand_million
. [10280](#), [10284](#), [10430](#),
[10452](#), [10469](#), [10479](#), [10515](#), [10526](#),
[10540](#), [10551](#), [10598](#), [10640](#), [10914](#),
[10933](#), [11052](#), [11088](#), [11114](#), [11165](#),
[11190](#), [11256](#), [11273](#), [11276](#), [11291](#),
[11300](#), [11377](#), [11416](#), [11424](#), [11433](#),
[11520](#), [11533](#), [11569](#), [11599](#), [11602](#),
[11604](#), [11607](#), [11617](#), [11621](#), [11628](#),
[11629](#), [11749](#), [11751](#), [11763](#), [11775](#),
[11790](#), [11823](#), [11834](#), [11852](#), [11910](#),
[11914](#), [11930](#), [11934](#), [12018](#), [12073](#),
[12115](#), [12159](#), [12210](#), [12216](#), [12264](#),
[12268](#), [12270](#), [12274](#), [12282](#), [12286](#),
[12316](#), [12440](#), [12510](#), [12685](#), [12695](#),
[12714](#), [12732](#), [12757](#), [12761](#), [12763](#),
[12767](#), [12775](#), [12779](#), [12849](#), [12870](#),
[12892](#), [12944](#), [13011](#), [13014](#), [13083](#),
[13122](#), [13126](#), [13128](#), [13132](#), [13276](#)
- \c_other_cctab
. [174](#), [13768](#), [13778](#), [13786](#), [13796](#)
- \c_other_char_token [3229](#), [3233](#)
- \c_parameter_token
. [51](#), [2611](#), [2618](#), [2671](#), [2674](#)
- \c_pdftex_is_engine_bool [1500](#), [1502](#)
- \c_pi_fp [170](#), [6826](#), [7722](#), [10299](#), [10299](#)
- \c_seven [69](#), [833](#), [843](#), [2537](#), [2569](#), [3953](#), [8239](#)
- \c_six [69](#), [833](#), [842](#),
[2535](#), [2567](#), [3953](#), [8238](#), [11886](#), [11892](#)
- \c_sixteen [69](#), [833](#), [840](#),
[1156](#), [3886](#), [3953](#), [8227](#), [8229](#), [8263](#),
[8266](#), [8309](#), [8311](#), [8322](#), [8324](#), [8357](#),
[8375](#), [8393](#), [8411](#), [8428](#), [8441](#), [8676](#)
- \c_space_tl [95](#), [4955](#), [4955](#),
[5017](#), [6095](#), [6103](#), [8570](#), [8640](#), [8821](#),
[9397](#), [9401](#), [9402](#), [9406](#), [9407](#), [10201](#)
- \c_space_token [51](#), [2611](#), [2622](#),
[2690](#), [3040](#), [3059](#), [3190](#), [4854](#), [4888](#)
- \c_str_cctab
[174](#), [13768](#), [13779](#), [13787](#), [13801](#), [13809](#)
- \c_string_cctab [13809](#), [13809](#)
- \c_ten [69](#),
[2543](#), [2575](#), [3726](#), [3953](#), [3960](#), [8242](#),
[10480](#), [10527](#), [10552](#), [10710](#), [10774](#),
[10830](#), [10932](#), [10937](#), [11049](#), [11085](#),
[11124](#), [11127](#), [11137](#), [11532](#), [11586](#),
[11762](#), [11811](#), [11853](#), [11865](#), [11943](#),
[12345](#), [12966](#), [13199](#), [13233](#), [13238](#)
- \c_ten_thousand [69](#), [3968](#), [3970](#)
- \c_term_ior [140](#), [8227](#), [8227](#), [8276](#), [8431](#)
- \c_term_iow
[140](#), [8228](#), [8229](#), [8278](#), [8444](#), [8485](#), [8486](#)
- \c_thirteen [69](#), [2549](#), [2581](#), [3953](#), [3962](#), [8245](#)
- \c_thirty_two [69](#), [3965](#), [3965](#)
- \c_three [69](#), [2529](#), [2561](#),
[3953](#), [3955](#), [8235](#), [11884](#), [12202](#), [12535](#)
- \c_tl_act_lowercase_tl . [5093](#), [5098](#), [5116](#)
- \c_tl_act_uppercase_tl . [5093](#), [5093](#), [5108](#)
- \c_tl_rescan_marker_tl
. [4482](#), [4490](#), [4501](#), [4519](#)

- `\c_token_A_int` 2901, 2936
- `\c_true_bool` 21, 972, 1001, 1041, 1041, 1075, 1281, 1430, 1441, 1451, 1891, 1895, 1918, 1996, 1999, 2005, 2006, 2034, 2035, 2038, 2040, 2044, 2067, 3801, 3806, 3819
- `\c_twelve` 69, 833, 844, 2277, 2292, 2547, 2579, 2767, 3953, 8244
- `\c_two` 69, 2527, 2559, 3884, 3953, 3954, 8234, 10901, 11889, 12177, 12181, 12200, 12234, 12357, 12363, 13016–13018, 13031, 13107, 13686, 13718, 13726
- `\c_two_hundred_fifty_five` 69, 3966, 3966
- `\c_two_hundred_fifty_six` . 69, 3966, 3967
- `\c_undefined:D` 1267, 1275
- `\c_undefined_fp` 171, 10300, 10300, 11463, 12373, 13166, 13216, 13223, 13343
- `\c_xetex_is_engine_bool` 1500, 1503
- `\c_zero` 69, 833, 841, 971, 979, 987, 994, 1000, 1008, 1016, 1023, 1049, 1051, 1458, 1463, 1564, 1573, 2107, 2199–2209, 2217, 2220, 2271, 2273, 2422, 2429, 2446, 2473, 2482, 2523, 2555, 2739, 3316, 3346, 3350, 3351, 3357, 3413, 3414, 3699, 3953, 4203, 4270, 4280, 4281, 4924, 4925, 4962, 5009, 5146, 5158, 5633, 5646, 6116, 6131, 6151, 8232, 8263, 8266, 8736, 8738, 9321, 9366, 10377, 10388, 10426, 10438, 10440, 10451, 10494–10496, 10604, 10646, 10689, 10692, 10754, 10821, 10956–10964, 10995–11003, 11058, 11094, 11138, 11193, 11231, 11247, 11289, 11293, 11295, 11361, 11365, 11390, 11459, 11469, 11482, 11483, 11504, 11508, 11529, 11538, 11539, 11567, 11615, 11619, 11623, 11624, 11643, 11687, 11761, 11789, 11800, 11808, 11866, 11869, 11876–11878, 11908, 11912, 11916, 11921, 11944, 11945, 11950, 11969, 11973, 11984, 12009, 12050, 12064, 12106, 12136, 12150, 12176, 12189, 12191, 12203, 12204, 12206, 12207, 12209, 12211, 12214, 12225–12227, 12259, 12260, 12280, 12284, 12307, 12356, 12370, 12371, 12401, 12402, 12414, 12415, 12431, 12498, 12501, 12537, 12563, 12567–12569, 12577–12579, 12591, 12624, 12642, 12643, 12675, 12676, 12681, 12682, 12687, 12716, 12752, 12753, 12773, 12777, 12816, 12820, 12872, 12883, 12900, 12910–12913, 12929, 12955, 12963, 12977, 12978, 12980, 12983, 13029, 13030, 13033, 13038, 13040, 13052, 13069, 13076, 13082, 13092, 13100, 13115, 13158, 13162, 13179, 13194, 13198, 13205, 13230, 13235, 13237, 13287, 13288, 13316, 13334–13337, 13424, 13427, 13431, 13442, 13443, 13465
- `\c_zero_dim` 76, 4039, 4088, 4213, 4214, 4219, 4302, 6712, 6927, 6929, 6930, 7067, 7077, 7089, 7092, 7095, 7100, 7454, 7457, 7460, 7469, 7472, 7475, 7484, 7491, 7557, 7562, 7569
- `\c_zero_fp` 171, 6778, 6794, 6796, 6801, 7010, 7019, 7034, 7870, 7885, 7888, 10301, 10301, 10566, 10576, 10578, 11473, 12349, 12404, 12527, 12554, 12594, 12826, 12834, 13172, 13351
- `\c_zero_muskip` 4318
- `\c_zero_skip` 79, 4235, 4302, 4302, 4361, 4362, 6698
- `\catcode` 3–6, 9, 70–78, 84–91, 101, 281–288, 665
- `\catcodetable` 758
- `\CatcodeTableIniTeX` 13777
- `\CatcodeTableLaTeX` 13776
- `\CatcodeTableOther` 13778
- `\CatcodeTableString` 13779
- `\cctab_begin:N` 173, 13714, 13714, 13734, 13737, 13748
- `\cctab_end` 173
- `\cctab_end:` 13714, 13724, 13739, 13742, 13749
- `\cctab_gset:Nn` 173, 13753, 13753, 13762, 13765, 13771, 13788, 13796, 13801
- `\cctab_new:N` 173, 13683, 13683, 13697, 13700, 13706, 13770, 13785–13787
- `\char` 519, 2778
- `\char_gset_active:Npn` 59, 3179
- `\char_gset_active:Npx` 3180
- `\char_gset_active_eq:NN` . 60, 3165, 3182
- `\char_make_active:N` 3238, 3254
- `\char_make_active:n` 3238, 3272
- `\char_make_alignment:N` 3238
- `\char_make_alignment:n` 3238

\char_make_alignment_tab:N	3243	\char_set_catcode_active:n	48, 2554, 2580, 3171, 3272, 9460, 9461, 13794
\char_make_alignment_tab:n	3261	\char_set_catcode_alignment:N	48, 2522, 2530, 2616, 3243
\char_make_begin_group:N	3240	\char_set_catcode_alignment:n	48, 316, 2554, 2562, 3261
\char_make_begin_group:n	3258	\char_set_catcode_comment:N	48, 2522, 2550, 3255
\char_make_comment:N	3238, 3255	\char_set_catcode_comment:n	48, 2554, 2582, 3273
\char_make_comment:n	3238, 3273	\char_set_catcode_end_line:N	48, 2522, 2532, 3244
\char_make_end_group:N	3241	\char_set_catcode_end_line:n	48, 2554, 2564, 3262
\char_make_end_group:n	3259	\char_set_catcode_escape:N	48, 2522, 2522, 3239
\char_make_end_line:N	3238, 3244	\char_set_catcode_escape:n	48, 2554, 2554, 3257
\char_make_end_line:n	3238, 3262	\char_set_catcode_group_begin:N	48, 2522, 2524, 3240
\char_make_escape:N	3238, 3239	\char_set_catcode_group_begin:n	48, 2554, 2556, 3258
\char_make_escape:n	3238, 3257	\char_set_catcode_group_end:N	48, 2522, 2526, 3241
\char_make_group_begin:N	3238	\char_set_catcode_group_end:n	48, 2554, 2558, 3259
\char_make_group_begin:n	3238	\char_set_catcode_ignore:N	48, 2522, 2540, 3250
\char_make_group_end:N	3238	\char_set_catcode_ignore:n	48, 313, 314, 2554, 2572, 3268, 10366
\char_make_group_end:n	3238	\char_set_catcode_invalid:N	48, 2522, 2552, 3256
\char_make_ignore:N	3238, 3250	\char_set_catcode_invalid:n	48, 2554, 2584, 3274
\char_make_ignore:n	3238, 3268	\char_set_catcode_letter:N	48, 2522, 2544, 3252, 8824
\char_make_invalid:N	3238, 3256	\char_set_catcode_letter:n	48, 317, 319, 2554, 2576, 3270
\char_make_invalid:n	3238, 3274	\char_set_catcode_math_subscript:N	48, 2522, 2538, 2620, 3249
\char_make_letter:N	3238, 3252	\char_set_catcode_math_subscript:n	48, 2554, 2570, 3267, 13793
\char_make_letter:n	3238, 3270	\char_set_catcode_math_superscript:N	48, 2522, 2536, 3247, 9306
\char_make_math_shift:N	3242	\char_set_catcode_math_superscript:n	48, 318, 2554, 2568, 3265
\char_make_math_shift:n	3260	\char_set_catcode_math_toggle:N	48, 2522, 2528, 2614, 3242
\char_make_math_subscript:N	3238, 3248	\char_set_catcode_math_toggle:n	48, 2554, 2560, 3260
\char_make_math_subscript:n	3238, 3266		
\char_make_math_superscript:N	3238, 3246		
\char_make_math_superscript:n	3238, 3264		
\char_make_math_toggle:N	3238		
\char_make_math_toggle:n	3238		
\char_make_other:N	3238, 3253		
\char_make_other:n	3238, 3271		
\char_make_parameter:N	3238, 3245		
\char_make_parameter:n	3238, 3263		
\char_make_space:N	3238, 3251		
\char_make_space:n	3238, 3269		
\char_set_active:Npn	59, 3165, 3177		
\char_set_active:Npx	3165, 3178		
\char_set_active_eq:NN	59, 3165, 3181		
\char_set_catcode:nn	49, 298–306, 2516, 2516, 2523, 2525, 2527, 2529, 2531, 2533, 2535, 2537, 2539, 2541, 2543, 2545, 2547, 2549, 2551, 2553, 2555, 2557, 2559, 2561, 2563, 2565, 2567, 2569, 2571, 2573, 2575, 2577, 2579, 2581, 2583, 2585, 2767		
\char_set_catcode:w	3201, 3202, 3209, 3211		
\char_set_catcode_active:N	48, 2522, 2548, 2627, 2634–2639, 3166, 3254, 8877		

- \char_set_catcode_other:N 48, [2522](#), 2546,
2724, 2725, 2903, 3253, 8506, 8829
- \char_set_catcode_other:n 48, 315, 320,
[2554](#), 2578, 3271, 13792, 13799, 13804
- \char_set_catcode_parameter:N
..... 48, [2522](#), 2534, 3245
- \char_set_catcode_parameter:n
..... 48, [2554](#), 2566, 3263
- \char_set_catcode_space:N
..... 48, [2522](#), 2542, 3251
- \char_set_catcode_space:n .. 48, 321,
[2554](#), 2574, 3269, 13790, 13791, 13805
- \char_set_lccode:nn
..... 49, [2586](#), 2592, 2726–2728,
2761–2765, 2904–2906, 3173, 8507,
8875, 8876, 9307–9310, 9462, 9463
- \char_set_lccode:w [3201](#), 3204, 3215, 3217
- \char_set_mathcode:nn ... 50, [2586](#), 2586
- \char_set_mathcode:w [3201](#), 3203, 3212, 3214
- \char_set_sfcode:nn 50, [2586](#), 2604
- \char_set_sfcode:w [3201](#), 3206, 3221, 3223
- \char_set_uccode:nn 50, [2586](#), 2598
- \char_set_uccode:w [3201](#), 3205, 3218, 3220
- \char_show_value_catcode:n 49, [2516](#), 2520
- \char_show_value_catcode:w .. [3208](#), 3210
- \char_show_value_lccode:n 49, [2586](#), 2596
- \char_show_value_lccode:w ... [3208](#), 3216
- \char_show_value_mathcode:n
..... 50, [2586](#), 2590
- \char_show_value_mathcode:w . [3208](#), 3213
- \char_show_value_sfcode:n 51, [2586](#), 2608
- \char_show_value_sfcode:w ... [3208](#), 3222
- \char_show_value_uccode:n 50, [2586](#), 2602
- \char_show_value_uccode:w ... [3208](#), 3219
- \char_tmp:NN 3167, 3177–3182
- \char_value_catcode:n
..... 49, 298–306, [2516](#), 2518
- \char_value_catcode:w [3208](#), 3209
- \char_value_lccode:n 49, [2586](#), 2594
- \char_value_lccode:w [3208](#), 3215
- \char_value_mathcode:n .. 50, [2586](#), 2588
- \char_value_mathcode:w [3208](#), 3212
- \char_value_sfcode:n 50, [2586](#), 2606
- \char_value_sfcode:w [3208](#), 3221
- \char_value_uccode:n 50, [2586](#), 2600
- \char_value_uccode:w [3208](#), 3218
- \chardef 80, 93, 96, 328, 354
- \chk_if_exist_cs:c [1198](#), 1206
- \chk_if_exist_cs:N
..... 23, [1198](#), 1198, 1207, 1777
- \chk_if_free_cs:c [1175](#), 1196
- \chk_if_free_cs:N
.. 24, [1175](#), 1175, 1185, 1197, 1212,
1256, 3378, 3393, 4034, 4230, 4312,
4379, 4385, 4390, 6544, 13685, 13708
- \cleaders 537
- \clist_clear:c [5763](#), 5764
- \clist_clear:N
... 107, [5763](#), 5763, 5915, 6167, 9912
- \clist_clear_new:c [5767](#), 5768
- \clist_clear_new:N 107, [5767](#), 5767
- \clist_concat:ccc [5779](#)
- \clist_concat:NNN
... 108, [5779](#), 5779, 5792, 5842, 5855
- \clist_concat_aux:NNNN
..... [5779](#), 5780, 5782, 5783
- \clist_const:cn [6194](#)
- \clist_const:cx [6194](#)
- \clist_const:Nn ... 114, [6194](#), 6194, 6196
- \clist_const:Nx [6194](#)
- \clist_display:c [6219](#), 6221
- \clist_display:N [6219](#), 6220
- \clist_gclear:c [5763](#), 5766
- \clist_gclear:N ... 107, [5763](#), 5765, 6169
- \clist_gclear_new:c [5767](#), 5770
- \clist_gclear_new:N 107, [5767](#), 5769
- \clist_gconcat:ccc [5779](#)
- \clist_gconcat:NNN
... 108, [5779](#), 5781, 5793, 5844, 5857
- \clist_get:cN [5867](#), 6213
- \clist_get:NN . 112, [5867](#), 5867, 5871, 6212
- \clist_get_aux:wN [5867](#), 5868, 5869
- \clist_gpop:cN [5872](#)
- \clist_gpop:NN 113, [5872](#), 5874, 5889
- \clist_gpush:cn [5890](#), 5902
- \clist_gpush:co [5890](#), 5904
- \clist_gpush:cV [5890](#), 5903
- \clist_gpush:cx [5890](#), 5905
- \clist_gpush:Nn 113, [5890](#), 5898
- \clist_gpush:No [5890](#), 5900
- \clist_gpush:NV [5890](#), 5899
- \clist_gpush:Nx [5890](#), 5901
- \clist_gput_left:cn [5841](#), 5902
- \clist_gput_left:co [5841](#), 5904
- \clist_gput_left:cV [5841](#), 5903
- \clist_gput_left:cx [5841](#), 5905
- \clist_gput_left:Nn
... 108, [5841](#), 5843, 5852, 5853, 5898

\clist_gput_left:No	5841, 5900	\clist_if_eq:Nc	5958, 5959
\clist_gput_left:Nv	5841, 5899	\clist_if_eq:NN	5958, 5958
\clist_gput_left:Nx	5841, 5901	\clist_if_eq:NNTF	109
\clist_gput_right:cn	5854	\clist_if_in:cn	5962
\clist_gput_right:co	5854	\clist_if_in:co	5962
\clist_gput_right:cV	5854	\clist_if_in:cV	5962
\clist_gput_right:cx	5854	\clist_if_in:Nn	5962, 5962
\clist_gput_right:Nn	108, 5854, 5856, 5865, 5866	\clist_if_in:nn	5962, 5966
\clist_gput_right:No	5854	\clist_if_in:NnF	5918, 5980, 5981
\clist_gput_right:Nv	5854	\clist_if_in:nnF	5985
\clist_gput_right:Nx	5854	\clist_if_in:NnT	5978, 5979
\clist_gremove_all:cn	5925	\clist_if_in:nnT	5984
\clist_gremove_all:Nn	109, 5925, 5927, 5955, 6217	\clist_if_in:NnTF	110, 5982, 5983
\clist_gremove_duplicates:c	5909	\clist_if_in:nnTF	5986
\clist_gremove_duplicates:N	109, 5909, 5911, 5924	\clist_if_in:No	5962
\clist_gremove_element:Nn	6215, 6217	\clist_if_in:no	5962
\clist_gset:cn	5835	\clist_if_in:Nv	5962
\clist_gset:co	5835	\clist_if_in:nv	5962
\clist_gset:cV	5835	\clist_if_in_return:nn	5962, 5964, 5969, 5971
\clist_gset:cx	5835	\clist_item:cn	6106
\clist_gset:Nn	108, 5835, 5837, 5840, 5857	\clist_item:Nn	114, 6106, 6106, 6135
\clist_gset:No	5835, 6225	\clist_item:nn	6136, 6136
\clist_gset:Nv	5835	\clist_item_aux:nnNn	6106, 6108, 6114, 6138
\clist_gset:Nx	5835	\clist_item_n_aux:nw	6136, 6141, 6144
\clist_gset_eq:cc	5771, 5778	\clist_item_n_end:n	6136, 6152, 6160
\clist_gset_eq:cN	5771, 5777	\clist_item_N_loop:nw	6106, 6111, 6129, 6133
\clist_gset_eq:Nc	5771, 5776	\clist_item_n_loop:nw	6136, 6145, 6146, 6149, 6154
\clist_gset_eq:NN	107, 5771, 5775, 5912	\clist_item_n_strip:w	6136, 6162, 6165
\clist_gset_from_seq:cc	6166	\clist_length:c	6081
\clist_gset_from_seq:cN	6166	\clist_length:N	113, 6081, 6081, 6105, 6109
\clist_gset_from_seq:Nc	6166	\clist_length:n	6081, 6090, 6139
\clist_gset_from_seq:NN	114, 6166, 6168, 6192, 6193	\clist_length_aux:n	6086, 6089
\clist_gtrim_spaces:c	6223	\clist_length_aux:w	6081
\clist_gtrim_spaces:N	6223, 6225, 6227	\clist_length_n_aux:w	6095, 6099, 6103
\clist_if_empty:c	5956, 5957	\clist_map_aux_unbrace:Nw	6003, 6012, 6016
\clist_if_empty:N	5956, 5956	\clist_map_break	111
\clist_if_empty:n	6197, 6197	\clist_map_break:	6059, 6059
\clist_if_empty:Nf	5788, 5942, 5989, 6019, 6037	\clist_map_break:n	112, 6059, 6060
\clist_if_empty:Ntf	109, 9293	\clist_map_function:cN	5987
\clist_if_empty:nTF	114	\clist_map_function:NN	110, 5682, 5692, 5987, 5987, 6002, 6066, 6074, 6086
\clist_if_empty_n_aux:w	6197, 6199, 6204, 6207	\clist_map_function:nN	5687, 5697, 6003, 6003, 9700, 9760
\clist_if_empty_n_aux:wNw	6197, 6208, 6210	\clist_map_function_aux:Nw	5987, 5991, 5996, 6000, 6023
\clist_if_eq:cc	5958, 5961	\clist_map_function_n_aux:Nn	6003, 6005, 6009, 6013
\clist_if_eq:cN	5958, 5960		

\clist_map_inline:cn	6017	\clist_remove_all:cn	5925
\clist_map_inline:Nn		\clist_remove_all:Nn	
... 110, 5916, 6017, 6017, 6032, 6034		... 109, 5925, 5925, 5954, 6216	
\clist_map_inline:nn	6017, 6029, 9681, 9775	\clist_remove_all_aux:	
\clist_map_variable:cNn	6035	... 5925, 5935, 5939, 5951	
\clist_map_variable:NNn		\clist_remove_all_aux:NNn	
... 111, 6035, 6035, 6049, 6058		... 5925, 5926, 5928, 5929	
\clist_map_variable:nNn	6035, 6046	\clist_remove_all_aux:w	5925, 5952, 5953
\clist_map_variable_aux:Nnw		\clist_remove_duplicates:c	5909
... 6035, 6040, 6051, 6056		\clist_remove_duplicates:N	
\clist_new:c	5761, 5762	... 109, 5909, 5909, 5923	
\clist_new:N		\clist_remove_duplicates_aux:NN	
... 107, 5761, 5761, 5908, 6077-6080		... 5909, 5910, 5912, 5913	
\clist_pop:cn	5872	\clist_remove_element:Nn	6215, 6216
\clist_pop:NN	112, 5872, 5872, 5888	\clist_set:cn	5835
\clist_pop_aux:NNN	5872, 5873, 5875, 5876	\clist_set:co	5835
\clist_pop_aux:NwNNN	5872	\clist_set:cV	5835
\clist_pop_aux:w	5885, 5887	\clist_set:cx	5835
\clist_pop_aux:wNN	5872	\clist_set:Nn	
\clist_pop_aux:wNNN	5878, 5880	... 108, 5835, 5835, 5839, 5842,	
\clist_push:cn	5890, 5894	5844, 5855, 5968, 6031, 6048, 6070	
\clist_push:co	5890, 5896	\clist_set:No	5835, 6224
\clist_push:cV	5890, 5895	\clist_set:Nv	5835
\clist_push:cx	5890, 5897	\clist_set:Nx	5835
\clist_push:Nn	113, 5890, 5890	\clist_set_eq:cc	5771, 5774
\clist_push:No	5890, 5892	\clist_set_eq:cN	5771, 5773
\clist_push:Nv	5890, 5891	\clist_set_eq:Nc	5771, 5772
\clist_push:Nx	5890, 5893	\clist_set_eq:NN	107, 5771, 5771, 5910, 9917
\clist_put_left:cn	5841, 5894	\clist_set_from_seq:cc	6166
\clist_put_left:co	5841, 5896	\clist_set_from_seq:cN	6166
\clist_put_left:cV	5841, 5895	\clist_set_from_seq:Nc	6166
\clist_put_left:cx	5841, 5897	\clist_set_from_seq:NN	
\clist_put_left:Nn		... 114, 6166, 6166, 6190, 6191	
... 108, 5841, 5841, 5850, 5851, 5890		\clist_set_from_seq_aux:NNN	
\clist_put_left:No	5841, 5892	... 6166, 6167, 6169, 6170	
\clist_put_left:Nv	5841, 5891	\clist_set_from_seq_aux:w	6185, 6189
\clist_put_left:Nx	5841, 5893	\clist_show:c	6061, 6221
\clist_put_left_aux:NNNn		\clist_show:N	113, 6061, 6061, 6076, 6220
... 5841, 5842, 5844, 5845		\clist_show:n	113, 6061, 6068
\clist_put_right:cn	5854	\clist_tmp:w	5760, 5760,
\clist_put_right:co	5854	5794, 5810, 5931, 5952, 5973, 5975	
\clist_put_right:cV	5854	\clist_top:cN	6211, 6213
\clist_put_right:cx	5854	\clist_top:NN	6211, 6212
\clist_put_right:Nn		\clist_trim_spaces:c	6223
... 108, 5854, 5854, 5863, 5864, 5919		\clist_trim_spaces:N	6223, 6224, 6226
\clist_put_right:No	5854	\clist_trim_spaces:n	
\clist_put_right:Nv	5854	... 114, 5814, 5814, 5836, 5838, 6195	
\clist_put_right:Nx	5854, 9993	\clist_trim_spaces_aux:nn	
\clist_put_right_aux:NNNn		... 5814, 5817, 5821, 5827, 5832	
... 5854, 5855, 5857, 5858			

\clist_trim_spaces_generic:nw	5794 , 5796 , 5816 , 5826 , 5831 , 6005 , 6013	\coffin_find_corner_maxima_aux:nn 7806 , 7813 , 7815
\clist_trim_spaces_generic_aux:w 5794 , 5805 , 5811	\coffin_get_pole:NnN 7307 , 7307 , 7438 , 7439 , 7671 , 7672 , 7675 , 7676 , 8055 , 8056 , 8059
\clist_trim_spaces_generic_aux_ii:nn 5794 , 5812 , 5813	\coffin_gset_eq_structure:NN	7324 , 7331
\clist_use:c 5906 , 5907	\coffin_ht:c 7301 , 7304
\clist_use:N 108 , 5906 , 5906	\coffin_ht:N 134 , 7301 , 7303
\clist_wrap_item:n 6166 , 6178 , 6182	\coffin_if_exist:NT 7161 , 7161 , 7179 , 7199 , 7216 , 7243 , 7260 , 7290 , 7362 , 7375
\closein 413	\coffin_join:cnncnnnn 7551
\closeout 408	\coffin_join:cnnNnnnn 7551
\clubpenalties 721	\coffin_join:Nnncnnnn 7551
\clubpenalty 548	\coffin_join:NnnNnnnn	133 , 7551 , 7551 , 7587
\coffin_align:NnnNnnnnN 7553 , 7590 , 7608 , 7616 , 7616 , 7706	\coffin_mark_handle:cnnn 7986
\coffin_attach:cnncnnnn 7588	\coffin_mark_handle:Nnnn 134 , 7986 , 7986 , 8040
\coffin_attach:cnnNnnnn 7588	\coffin_mark_handle_aux:nnnnNnn 7986 , 8024 , 8029 , 8033
\coffin_attach:Nnncnnnn 7588	\coffin_new:c 7186
\coffin_attach:NnnNnnnn 133 , 7588 , 7588 , 7615	\coffin_new:N 131 , 7186 , 7186 , 7196 , 7297 , 7299 , 7300 , 7934 – 7936
\coffin_attach_mark:NnnNnnnn 7588 , 7606 , 7998 , 8019 , 8035	\coffin_offset_corner:Nnnnn	.. 7657 , 7659
\coffin_calculate_intersection:Nnn 7436 , 7436 , 7618 , 7621 , 8129	\coffin_offset_corners:Nnn 7573 , 7574 , 7580 , 7581 , 7654 , 7654
\coffin_calculate_intersection:nnnnnnnn 7436 , 7442 , 7451 , 8075	\coffin_offset_corners:Nnnnn 7654
\coffin_calculate_intersection_aux:nnnnnn 7436 , 7463 , 7478 , 7487 , 7494 , 7528 , 7537	\coffin_offset_pole:Nnnnnnn 7635 , 7638 , 7640
\coffin_clear:c 7177	\coffin_offset_poles:Nnn	.. 7571 , 7572 , 7577 , 7578 , 7600 , 7601 , 7635 , 7635
\coffin_clear:N	... 131 , 7177 , 7177 , 7185	\coffin_reset_structure:N 7182 , 7208 , 7226 , 7250 , 7269 , 7317 , 7317 , 7565 , 7595
\coffin_display_attach:Nnnnn 8041 , 8080 , 8102 , 8121 , 8127	\coffin_resize:cn 7851
\coffin_display_handles:cn	... 134 , 8041	\coffin_resize:Nnn	.. 132 , 7851 , 7851 , 7863
\coffin_display_handles:Nn 8041 , 8041 , 8126	\coffin_resize_common:Nnn 7861 , 7864 , 7864 , 7891
\coffin_display_handles_aux:nnnn 8041 , 8108 , 8113 , 8119	\coffin_rotate:cn 7718
\coffin_display_handles_aux:nnnnnn 8041 , 8066 , 8070	\coffin_rotate:Nn	.. 132 , 7718 , 7718 , 7752
\coffin_dp:c 7301 , 7302	\coffin_rotate_bounding:nnn 7731 , 7765 , 7765
\coffin_dp:N 134 , 7301 , 7301	\coffin_rotate_corner:Nnnn 7726 , 7765 , 7771
\coffin_end_user_dimensions: 7338 , 7353 , 7370 , 7383 , 7877	\coffin_rotate_pole:Nnnnnn 7728 , 7777 , 7777
\coffin_find_bounding_shift: 7733 , 7822 , 7822	\coffin_rotate_vector:nnNN 7767 , 7773 , 7779 , 7780 , 7789 , 7789
\coffin_find_bounding_shift_aux:nn 7822 , 7826 , 7828	\coffin_saved_Depth:	7157 , 7157 , 7341 , 7356
\coffin_find_corner_maxima:N 7732 , 7806 , 7806		

- \coffin_saved_Height: [7157](#), [7158](#), [7340](#), [7355](#)
- \coffin_saved_TotalHeight: [7157](#), [7159](#), [7342](#), [7357](#)
- \coffin_saved_Width: [7157](#), [7160](#), [7343](#), [7358](#)
- \coffin_scale:cnn [7879](#)
- \coffin_scale:Nnn . [133](#), [7879](#), [7879](#), [7894](#)
- \coffin_scale_corner:Nnnn [7867](#), [7904](#), [7904](#)
- \coffin_scale_pole:Nnnnnn [7869](#), [7904](#), [7910](#)
- \coffin_scale_vector:nnNN [7895](#), [7895](#), [7906](#), [7912](#)
- \coffin_set_bounding:N . [7729](#), [7753](#), [7753](#)
- \coffin_set_eq:cc [7288](#)
- \coffin_set_eq:cN [7288](#)
- \coffin_set_eq:Nc [7288](#)
- \coffin_set_eq:NN [131](#), [7288](#), [7288](#), [7296](#), [7585](#), [7604](#), [7633](#), [8062](#)
- \coffin_set_eq_structure:NN [7293](#), [7324](#), [7324](#)
- \coffin_set_horizontal_pole:cnn .. [7360](#)
- \coffin_set_horizontal_pole:Nnn [132](#), [7360](#), [7360](#), [7388](#)
- \coffin_set_pole:Nnn ... [7360](#), [7386](#), [7390](#)
- \coffin_set_pole:Nnx [7230](#), [7273](#), [7360](#), [7365](#), [7378](#), [7647](#), [7684](#), [7688](#), [7696](#), [7700](#), [7782](#), [7913](#)
- \coffin_set_user_dimensions:N [7338](#), [7338](#), [7364](#), [7377](#), [7853](#), [7882](#)
- \coffin_set_vertical_pole:cnn ... [7360](#)
- \coffin_set_vertical_pole:Nnn [132](#), [7360](#), [7373](#), [7389](#)
- \coffin_shift_corner:Nnnn [7748](#), [7830](#), [7830](#)
- \coffin_shift_pole:Nnnnnn [7750](#), [7830](#), [7838](#)
- \coffin_show_structure:c [8151](#)
- \coffin_show_structure:N [134](#), [8151](#), [8151](#), [8167](#)
- \coffin_typeset:cnnnn [7704](#)
- \coffin_typeset:Nnnnn [133](#), [7704](#), [7704](#), [7711](#)
- \coffin_update_B:nnnnnnnnN [7669](#), [7677](#), [7692](#)
- \coffin_update_corners:N [7210](#), [7228](#), [7252](#), [7271](#), [7391](#), [7391](#)
- \coffin_update_poles:N .. [7209](#), [7227](#), [7251](#), [7270](#), [7402](#), [7402](#), [7568](#), [7599](#)
- \coffin_update_T:nnnnnnnnN [7669](#), [7673](#), [7680](#)
- \coffin_update_vertical_poles:NNN .. [7584](#), [7603](#), [7669](#), [7669](#)
- \coffin_wd:c [7301](#), [7306](#)
- \coffin_wd:N [134](#), [7301](#), [7305](#)
- \coffin_x_shift_corner:Nnnn [7873](#), [7919](#), [7919](#)
- \coffin_x_shift_pole:Nnnnnn [7875](#), [7919](#), [7926](#)
- \color [7994](#), [8006](#), [8049](#), [8089](#)
- \color_ensure_current [135](#)
- \color_ensure_current: [7204](#), [7222](#), [7245](#), [7264](#), [8212](#), [8213](#), [8217](#)
- \color_group_begin [135](#)
- \color_group_begin: [7203](#), [7221](#), [7245](#), [7264](#), [8206](#), [8206](#)
- \color_group_end [135](#)
- \color_group_end: [7206](#), [7224](#), [7248](#), [7267](#), [8206](#), [8207](#)
- \copy [605](#)
- \count [656](#)
- \countdef [355](#)
- \cr [380](#)
- \crr [381](#)
- \cs [46](#)
- \cs:w [16](#), [804](#), [806](#), [819](#), [878](#), [1101](#), [1129](#), [1332](#), [1364](#), [1528](#), [1567](#), [1581](#), [1583](#), [1585](#), [1589–1591](#), [1626](#), [1632](#), [1652](#), [1654](#), [1659](#), [1666](#), [1667](#), [1729](#), [1733](#), [1762](#), [2177](#), [2179](#), [3459](#), [5179](#), [12664](#), [13024](#)
- \cs_end [16](#)
- \cs_end: [804](#), [807](#), [819](#), [823](#), [878](#), [1095](#), [1101](#), [1123](#), [1129](#), [1270](#), [1332](#), [1364](#), [1528](#), [1567](#), [1581](#), [1583](#), [1585](#), [1589–1591](#), [1626](#), [1632](#), [1652](#), [1654](#), [1659](#), [1666](#), [1667](#), [1729](#), [1733](#), [1762](#), [2174](#), [2180–2183](#), [2185](#), [2187](#), [2189](#), [2191](#), [2193](#), [2195](#), [2197](#), [3459](#), [5179](#), [12031](#), [12119](#), [12329](#), [12514](#), [12524](#), [12664](#), [12853](#), [13024](#)
- \cs_generate_from_arg_count:cNnn ... [998](#), [1006](#), [1014](#), [1022](#), [1295](#), [1318](#), [1363](#)
- \cs_generate_from_arg_count:Ncnn ... [1295](#), [1320](#)
- \cs_generate_from_arg_count:NNnn ... [15](#), [1295](#), [1295](#), [1319](#), [1321](#), [1331](#)
- \cs_generate_from_arg_count_aux:nwn [1295](#), [1298–1307](#), [1316](#)
- \cs_generate_from_arg_count_error_msg:Nn [1295](#), [1309](#), [1322](#)
- \cs_generate_internal_variant:n [32](#), [1816](#), [1855](#), [1855](#)
- \cs_generate_internal_variant_aux:N [1855](#), [1860](#), [1863](#), [1869](#)

<code>\cs_generate_variant:Nn</code>	7050, 7080, 7103, 7185, 7196, 7213,
..... 26, 1775 , 1775, 1871–1878,	7240, 7257, 7287, 7296, 7388–7390,
1889, 1898–1901, 1914, 1915, 1924–	7587, 7615, 7711, 7752, 7863, 7894,
1927, 2073, 2074, 2079, 2080, 2145,	8040, 8126, 8167, 8274, 8275, 8277,
2168, 2435, 2436, 2466–2469, 2488–	8279, 8292, 8305, 8448, 8449, 8476,
2491, 3382, 3403, 3415, 3416, 3421,	8479, 8941–8944, 9071, 9583, 9748,
3422, 3424, 3425, 3427, 3428, 3437–	9801, 9802, 9905, 9906, 9919, 9920,
3440, 3449–3452, 3456, 3457, 3831,	10568, 10574, 10579, 10580, 10585,
4038, 4041, 4042, 4047, 4048, 4052,	10586, 10619, 10620, 10667, 10668,
4053, 4055, 4056, 4058, 4059, 4070–	10683, 10745, 10748, 10815, 11040,
4073, 4077, 4078, 4082, 4083, 4208,	11043, 11075, 11078, 11159, 11160,
4210, 4234, 4237, 4238, 4243, 4244,	11184, 11185, 11210, 11211, 11313,
4248, 4249, 4251, 4252, 4254, 4255,	11314, 11331, 11332, 11444, 11445,
4259, 4260, 4264, 4265, 4289, 4296,	11996, 11997, 12093, 12094, 12294,
4297, 4299, 4316, 4320, 4321, 4326,	12295, 12487, 12488, 12790, 12805,
4327, 4331, 4332, 4334, 4335, 4337,	12806, 13139, 13140, 13675, 13678
4338, 4342, 4343, 4347, 4348, 4352,	<code>\cs_generate_variant_aux:N</code>
4354, 4382, 4393, 4394, 4400, 4401, 1778 , 1827 , 1840
4406, 4407, 4428–4433, 4450–4457,	<code>\cs_generate_variant_aux:NNn</code>
4474–4481, 4522–4525, 4536–4539, 1775 , 1789, 1811
4582, 4583, 4588, 4589, 4592–4599,	<code>\cs_generate_variant_aux:nnNn</code>
4608–4611, 4620–4623, 4643–4646, 1775 , 1779 , 1782
4665–4667, 4674–4676, 4690, 4703,	<code>\cs_generate_variant_aux:Nnnw</code>
4719, 4724, 4730, 4742, 4743, 4803, 1775 , 1783 , 1784 , 1809
4804, 4812, 4813, 4841–4844, 4941,	<code>\cs_generate_variant_aux:w</code>
5070, 5075, 5076, 5163, 5172, 5173, 1827 , 1842 , 1849
5212, 5283, 5284, 5289–5292, 5297–	<code>\cs_get_arg_count_from_signature:c</code> .
5300, 5317, 5318, 5343, 5344, 5366– 1293, 1365
5371, 5380, 5396, 5397, 5421, 5448,	<code>\cs_get_arg_count_from_signature:N</code> .
5449, 5473, 5502, 5513, 5514, 5548, 19, 945, 1277 , 1277, 1294, 1333
5571–5576, 5605–5616, 5626, 5650,	<code>\cs_get_arg_count_from_signature_aux:nnN</code>
5652, 5677, 5678, 5699–5704, 5722, 1277 , 1278, 1279
5723, 5792, 5793, 5839, 5840, 5850–	<code>\cs_get_arg_count_from_signature_auxii:w</code>
5853, 5863–5866, 5871, 5888, 5889, 1277 , 1287, 1292
5923, 5924, 5954, 5955, 5978–5986,	<code>\cs_get_function_name:N</code> . 19, 1077 , 1077
6002, 6034, 6058, 6076, 6105, 6135,	<code>\cs_get_function_signature:N</code>
6190–6193, 6196, 6226, 6227, 6246, 19, 1077 , 1079
6249, 6282–6285, 6294, 6295, 6313–	<code>\cs_gnew:cpn</code>
6316, 6331, 6333, 6335, 6337, 6352, 1479
6353, 6362–6365, 6389–6396, 6408–	<code>\cs_gnew:cpx</code>
6413, 6428, 6429, 6441, 6451, 6470– 1483
6475, 6490, 6504, 6514, 6515, 6521–	<code>\cs_gnew:Npn</code>
6523, 6526, 6548, 6553, 6554, 6567, 1471
6568, 6573, 6574, 6579, 6580, 6584–	<code>\cs_gnew:Npx</code>
6586, 6593–6595, 6598, 6599, 6615– 1475
6622, 6625–6628, 6633, 6634, 6657,	<code>\cs_gnew_eq:cc</code>
6667, 6670, 6674, 6675, 6680, 6681, 1491
6686, 6687, 6705, 6706, 6716, 6717,	<code>\cs_gnew_eq:cN</code>
6722, 6723, 6728, 6729, 6734, 6735, 1489
6750, 6751, 6924, 6965, 6985, 7007,	<code>\cs_gnew_eq:Nc</code>
 1490
	<code>\cs_gnew_eq:NN</code>
 1488
	<code>\cs_gnew_nopar:cpn</code>
 1478
	<code>\cs_gnew_nopar:cpx</code>
 1482
	<code>\cs_gnew_nopar:Npn</code>
 1470
	<code>\cs_gnew_nopar:Npx</code>
 1474
	<code>\cs_gnew_protected:cpn</code>
 1481

\cs_gnew_protected:cpx	1485	\cs_gset_protected_nopar:cx	1368
\cs_gnew_protected:Npn	1473	\cs_gset_protected_nopar:Nn	14, 1327
\cs_gnew_protected:Npx	1477	\cs_gset_protected_nopar:Npn	12, 864, 870, 1220, 1240
\cs_gnew_protected_nopar:cpn	1480	\cs_gset_protected_nopar:Npx	864, 872, 1221, 1241
\cs_gnew_protected_nopar:cpx	1484	\cs_gset_protected_nopar:Nx	1327
\cs_gnew_protected_nopar:Npn	1472	\cs_gundefine:c	1495
\cs_gnew_protected_nopar:Npx	1476	\cs_gundefine:N	1494
\cs_gset:cn	1368	\cs_if_eq:cc	1392
\cs_gset:cpn	1232, 1234, 4694, 6022, 6433, 8776, 8778	\cs_if_eq:ccF	1408
\cs_gset:cpx	1232, 1235	\cs_if_eq:ccT	1407
\cs_gset:cx	1368	\cs_if_eq:ccTF	1406
\cs_gset:Nn	14, 1327	\cs_if_eq:cN	1392
\cs_gset:Npn	12, 864, 866, 1218, 1234, 3179, 5477	\cs_if_eq:cNF	1400
\cs_gset:Npx	864, 868, 1219, 1235, 3180, 5482	\cs_if_eq:cNT	1399
\cs_gset:Nx	1327	\cs_if_eq:cNTF	1398
\cs_gset_eq:cc	1262, 1265, 1909, 4415	\cs_if_eq:Nc	1392
\cs_gset_eq:cN	1262, 1264, 1275, 1908, 4413, 5486, 6243, 8361, 8397, 9548, 9550	\cs_if_eq:NcF	1404
\cs_gset_eq:Nc	1262, 1263, 1907, 4414, 5493, 8353, 8368, 8389, 8404	\cs_if_eq:NcT	1403
\cs_gset_eq:NN	15, 1262, 1262-1265, 1267, 1895, 1897, 1906, 3182, 4380, 4412, 6242, 8431, 8444	\cs_if_eq:NcTF	1402
\cs_gset_nopar:cn	1368	\cs_if_eq:NN	1392, 1392
\cs_gset_nopar:cpn	1224, 1228	\cs_if_eq:NNF	1400, 1404, 1408
\cs_gset_nopar:cpx	1224, 1229, 3077	\cs_if_eq:NNT	1399, 1403, 1407
\cs_gset_nopar:cx	1368	\cs_if_eq:NNTF	21, 1398, 1402, 1406, 8936
\cs_gset_nopar:Nn	14, 1327	\cs_if_eq_p:cc	1405
\cs_gset_nopar:Npn	12, 864, 864, 867, 871, 875, 1216, 1228, 2241	\cs_if_eq_p:cN	1397
\cs_gset_nopar:Npx	864, 865, 869, 873, 877, 1217, 1229, 2247, 4386, 4391, 4423, 4425, 4427, 4443, 4445, 4447, 4449, 4467, 4469, 4471, 4473	\cs_if_eq_p:Nc	1401
\cs_gset_nopar:Nx	1327	\cs_if_eq_p:NN	1397, 1401, 1405
\cs_gset_protected:cn	1368	\cs_if_exist:c	1081, 1093
\cs_gset_protected:cpn	1244, 1246	\cs_if_exist:cF	3919, 3926, 3928, 9721
\cs_gset_protected:cpx	1244, 1247	\cs_if_exist:cT	8756, 9965
\cs_gset_protected:cx	1368	\cs_if_exist:cTF	1146, 1148, 1150, 1152, 7165, 8153, 8352, 8377, 8388, 8413, 9028, 9038, 9596, 9695, 10002, 10016, 10022, 13387
\cs_gset_protected:Nn	14, 1327	\cs_if_exist:N	1081, 1081
\cs_gset_protected:Npn	12, 864, 874, 1222, 1246	\cs_if_exist:NF	1200, 8708, 9639, 9654, 9795
\cs_gset_protected:Npx	864, 876, 1223, 1247	\cs_if_exist:NT	1432, 1443, 8424, 8437, 10158, 10176
\cs_gset_protected:Nx	1327	\cs_if_exist:NTF	21, 1138, 1140, 1142, 1144, 1411, 2753, 3418, 3420, 4044, 4046, 4240, 4242, 4323, 4325, 4403, 4405, 4727, 6245, 6248, 6557, 6563, 6650, 7163, 8674, 9370, 10582, 10584
\cs_gset_protected_nopar:cn	1368	\cs_if_exist_use:c	1137, 1151
\cs_gset_protected_nopar:cpn	1238, 1240	\cs_if_exist_use:cF	1147
\cs_gset_protected_nopar:cpx	1238, 1241	\cs_if_exist_use:cT	1149
		\cs_if_exist_use:cTF	1145
		\cs_if_exist_use:N	1137, 1143

- \cs_if_exist_use:NF 1139
- \cs_if_exist_use:NT 1141
- \cs_if_exist_use:NTF 24, 1137
- \cs_if_free:c 1109, 1121
- \cs_if_free:cT 1857
- \cs_if_free:N 1109, 1109
- \cs_if_free:NF 1177, 1187
- \cs_if_free:NTF 21, 1813, 8488, 8490
- \cs_meaning:c 820, 821
- \cs_meaning:N 16, 804, 808, 828
- \cs_new:cn 1384
- \cs_new:cpn ... 1232, 1236, 1479, 1965,
1978, 2011, 2013, 2015, 2016, 2181–
2184, 2186, 2188, 2190, 2192, 2194,
2196, 2198, 2200–2209, 3474, 3482,
3490, 3498, 3506, 3514, 3522, 4125–
4131, 10722, 10723, 10725, 10727,
10729, 10731, 10733, 10735, 10737,
10783, 10785, 10787, 10789, 10791,
10793, 10795, 10797, 10799, 10845,
10850, 10855, 10860, 10865, 10870,
10875, 10880, 10885, 10890, 10895
- \cs_new:cpx 1232, 1237, 1483, 1859
- \cs_new:cx 1384
- \cs_new:Nn 12, 1352
- \cs_new:Npn
... 10, 920, 933, 1208, 1218, 1236,
1277, 1279, 1292, 1409, 1467, 1468,
1471, 1522–1527, 1529, 1531, 1543,
1549, 1555, 1566, 1568, 1575, 1576,
1578, 1580, 1582, 1584, 1586, 1593,
1595, 1600, 1605, 1611, 1617, 1623,
1629, 1635, 1642, 1649, 1656, 1663,
1698, 1699, 1704, 1706, 1711, 1721,
1723, 1725, 1726, 1728, 1730, 1736,
1742, 1744, 1751, 1757, 1759, 1761–
1763, 1765, 1770, 1863, 1938, 1943,
1953, 1963, 1964, 1991–1993, 2002,
2017, 2022, 2027, 2028, 2046, 2052,
2058, 2062, 2063, 2069, 2071, 2075,
2077, 2081, 2089, 2094, 2102, 2107,
2108, 2113, 2115, 2121, 2126, 2128,
2134, 2139, 2146, 2151, 2157, 2162,
2169, 2176, 2178, 2180, 2210, 2215,
2229, 2281, 2287, 2296, 2301, 2405,
2411, 2419, 2426, 2437, 2443, 2508,
2518, 2588, 2594, 2600, 2606, 2737,
2789, 2807, 2831, 2849, 2867, 2878,
2899, 2918, 2925, 2926, 2934, 2943,
2952, 2967, 3048, 3135, 3138, 3147,
3156, 3169, 3311, 3313, 3321, 3332,
3343, 3368, 3369, 3459, 3462, 3472,
3554, 3562, 3570, 3576, 3582, 3590,
3598, 3604, 3610, 3611, 3625, 3631,
3663, 3695, 3697, 3703, 3715, 3723,
3756, 3758, 3760, 3762, 3767, 3772,
3777, 3797, 3798, 3803, 3808, 3832,
3840, 3842, 3851, 3853, 3862, 3864,
3874, 3883, 3885, 3887, 3903, 3912,
3946, 3948, 3990, 4005, 4084, 4094,
4096, 4113, 4188, 4190, 4192, 4199,
4286, 4291, 4294, 4349, 4357, 4518,
4565, 4566, 4576, 4624, 4677, 4685,
4723, 4725, 4731, 4736, 4741, 4744,
4751, 4758, 4762, 4776, 4784, 4789,
4795, 4805–4807, 4809, 4811, 4814,
4820, 4822, 4828, 4868, 4876, 4921,
4960, 4968–4971, 4980, 4981, 4988,
4999, 5008, 5010, 5017, 5023, 5025,
5027, 5032, 5045, 5047, 5049, 5055,
5068, 5077, 5089–5091, 5103, 5111,
5119, 5126, 5127, 5135, 5140, 5155,
5211, 5213, 5222, 5271, 5277, 5301,
5416, 5461, 5467, 5549, 5617, 5625,
5627, 5643, 5651, 5653, 5662, 5670,
5717, 5796, 5811, 5813, 5814, 5821,
5951, 5953, 5987, 5996, 6003, 6009,
6016, 6081, 6089, 6106, 6114, 6129,
6136, 6144, 6146, 6160, 6165, 6182,
6189, 6204, 6210, 6268, 6374, 6380,
6414, 6420, 6476, 6482, 6491, 6498,
8656, 8663, 8911–8916, 8940, 9325,
9328, 9337, 9342, 9347, 9352, 9394,
9395, 9399, 9404, 9511, 9637, 9652,
9759, 10000, 10009, 10026, 10681,
10684, 10701, 10702, 10708, 10717,
10738, 10744, 10746, 10749, 10761,
10772, 10781, 10800, 10808, 10813,
10816, 10828, 10837, 10839, 10896,
10909, 10928, 10948, 10954, 10993,
11032–11034, 11036, 13662, 13673
- \cs_new:Npx 1208, 1219,
1237, 1475, 6090, 6099, 8531, 9313
- \cs_new:Nx 1352
- \cs_new_eq:cc 950, 1254, 1261, 1491
- \cs_new_eq:cN 1254,
1259, 1489, 6239, 12348, 12372, 12403
- \cs_new_eq:Nc 1254, 1260, 1490
- \cs_new_eq:NN ... 15, 1254, 1254, 1259–
1261, 1420–1431, 1466, 1470–1485,

- 1488–1491, 1494, 1495, 1498, 1501–1503, 1537, 1888, 1902–1909, 2504, 2610–2612, 2960–2962, 3202–3206, 3226, 3227, 3230–3233, 3236, 3239–3246, 3248, 3250–3264, 3266, 3268–3274, 3277–3292, 3301–3306, 3406, 3410, 3458, 3949, 3950, 3978–3980, 4028–4030, 4207, 4209, 4219, 4220, 4288, 4290, 4293, 4298, 4302, 4303, 4351, 4353, 4408–4415, 4526, 4527, 4720–4722, 4942, 5182–5189, 5192–5199, 5202–5206, 5209, 5210, 5229–5246, 5450–5453, 5515–5540, 5745, 5746, 5749, 5750, 5761–5778, 5890–5907, 6059, 6060, 6212, 6213, 6216, 6217, 6220, 6221, 6238, 6250–6257, 6442, 6443, 6506, 6507, 6518, 6581–6583, 6596, 6597, 6608–6610, 6636, 6642, 6688–6695, 6703, 6704, 6741–6749, 7104, 7301–7306, 8206, 8226–8229, 8249, 8261, 8276, 8278, 8475, 8492, 8741, 8742, 9410, 9413–9417, 9990, 10092–10094, 10671–10680, 13658, 13659, 13776–13779, 13809
- `\cs_new_nopar:cn` [1384](#)
- `\cs_new_nopar:cpn`
- [1224](#), [1230](#), [1478](#), [2029](#)–[2041](#), [2043](#)
- `\cs_new_nopar:cpx` [1224](#), [1231](#), [1482](#)
- `\cs_new_nopar:cx` [1384](#)
- `\cs_new_nopar:Nn` [13](#), [1352](#)
- `\cs_new_nopar:Npn` [11](#), [1208](#),
1216, 1230, 1293, 1397–1408, 1418,
1454, 1470, 1521, 1671–1679, 1686–
1690, 1753–1755, 2270, 2272, 2964–
2966, 3017, 3026, 3034, 3209, 3210,
3212, 3213, 3215, 3216, 3218, 3219,
3221, 3222, 3782–3796, 3983, 4571,
4683, 5162, 5365, 7157–7160, 8491,
8654, 8815–8817, 9980, 9982, 9991
- `\cs_new_nopar:Npx`
 [1208](#), [1217](#), [1231](#), [1474](#), [1846](#)
- `\cs_new_nopar:Nx` [1352](#)
- `\cs_new_protected:cn` [1384](#)
- `\cs_new_protected:cpn`
 [1244](#), [1248](#), [1481](#), [9803](#),
9805, 9807, 9809, 9813, 9815, 9817,
9819, 9821, 9823, 9825, 9827, 9829,
9831, 9833, 9835, 9837, 9839, 9841,
9843, 9845, 9847, 9849, 9851, 9853,
9855, 9857, 9859, 9861, 9863, 9867,
9869, 9871, 9873, 9875, 9877, 9879,
9881, 9883, 9885, 9887, 9889, 9891
- `\cs_new_protected:cpx` [1244](#), [1249](#), [1485](#)
- `\cs_new_protected:cx` [1384](#)
- `\cs_new_protected:Nn` [13](#), [1352](#)
- `\cs_new_protected:Npn` [11](#),
924, 937, [1208](#), [1222](#), [1248](#), [1250](#),
[1254](#), [1266](#), [1268](#), [1295](#), [1315](#), [1322](#),
[1473](#), [1538](#), [1716](#), [1775](#), [1782](#), [1784](#),
[1811](#), [1840](#), [1849](#), [1855](#), [1888](#), [1890](#),
[1892](#), [1894](#), [1896](#), [1910](#), [1912](#), [2238](#),
[2244](#), [2255](#), [2309](#), [2388](#), [2389](#), [2398](#),
[2495](#), [2516](#), [2520](#), [2522](#), [2524](#), [2526](#),
[2528](#), [2530](#), [2532](#), [2534](#), [2536](#), [2538](#),
[2540](#), [2542](#), [2544](#), [2546](#), [2548](#), [2550](#),
[2552](#), [2554](#), [2556](#), [2558](#), [2560](#), [2562](#),
[2564](#), [2566](#), [2568](#), [2570](#), [2572](#), [2574](#),
[2576](#), [2578](#), [2580](#), [2582](#), [2584](#), [2586](#),
[2590](#), [2592](#), [2596](#), [2598](#), [2602](#), [2604](#),
[2608](#), [2610](#), [2972](#), [2978](#), [2995](#), [2997](#),
[2999](#), [3013](#), [3015](#), [3376](#), [3383](#), [3413](#),
[3414](#), [3417](#), [3419](#), [3423](#), [3426](#), [3429](#),
[3431](#), [3441](#), [3443](#), [3453](#), [3951](#), [4032](#),
[4039](#), [4040](#), [4043](#), [4045](#), [4049](#), [4051](#),
[4054](#), [4057](#), [4068](#), [4074](#), [4076](#), [4079](#),
[4081](#), [4211](#), [4228](#), [4235](#), [4236](#), [4239](#),
[4241](#), [4245](#), [4247](#), [4250](#), [4253](#), [4256](#),
[4258](#), [4261](#), [4263](#), [4300](#), [4310](#), [4317](#),
[4319](#), [4322](#), [4324](#), [4328](#), [4330](#), [4333](#),
[4336](#), [4339](#), [4341](#), [4344](#), [4346](#), [4355](#),
[4377](#), [4383](#), [4388](#), [4396](#), [4398](#), [4402](#),
[4404](#), [4416](#), [4418](#), [4420](#), [4422](#), [4424](#),
[4426](#), [4434](#), [4436](#), [4438](#), [4440](#), [4442](#),
[4444](#), [4446](#), [4448](#), [4458](#), [4460](#), [4462](#),
[4464](#), [4466](#), [4468](#), [4470](#), [4472](#), [4498](#),
[4540](#), [4578](#), [4580](#), [4584](#), [4586](#), [4691](#),
[4701](#), [4704](#), [4712](#), [4799](#), [4801](#), [4928](#),
[4940](#), [5071](#), [5073](#), [5167](#), [5178](#), [5251](#),
[5279](#), [5281](#), [5285](#), [5287](#), [5293](#), [5295](#),
[5303](#), [5305](#), [5307](#), [5319](#), [5321](#), [5323](#),
[5372](#), [5378](#), [5385](#), [5391](#), [5398](#), [5404](#),
[5426](#), [5432](#), [5454](#), [5474](#), [5479](#), [5484](#),
[5496](#), [5503](#), [5541](#), [5679](#), [5684](#), [5689](#),
[5694](#), [5710](#), [5728](#), [5738](#), [5760](#), [5783](#),
[5835](#), [5837](#), [5845](#), [5858](#), [5867](#), [5869](#),
[5876](#), [5880](#), [5887](#), [5909](#), [5911](#), [5913](#),
[5925](#), [5927](#), [5929](#), [5971](#), [6017](#), [6029](#),
[6035](#), [6046](#), [6051](#), [6061](#), [6068](#), [6166](#),
[6168](#), [6170](#), [6194](#), [6224](#), [6225](#), [6238](#)–
[6244](#), [6247](#), [6258](#), [6260](#), [6269](#), [6270](#),

6276, 6278, 6280, 6286, 6292, 6296,
 6302, 6308, 6317–6319, 6323, 6343,
 6403, 6430, 6444, 6464, 6510, 6512,
 6542, 6549, 6551, 6555, 6561, 6569,
 6571, 6575, 6577, 6587, 6589, 6591,
 6600, 6602, 6604, 6606, 6629, 6631,
 6648, 6658, 6668, 6671–6673, 6676,
 6678, 6682, 6684, 6696, 6698, 6699,
 6701, 6707–6709, 6711, 6713, 6715,
 6718, 6720, 6724, 6726, 6730, 6732,
 6736, 6752, 6771, 6788, 6822, 6830,
 6841, 6852, 6863, 6874, 6885, 6898,
 6925, 6945, 6966, 6986, 7008, 7024,
 7048, 7051, 7081, 7161, 7177, 7186,
 7197, 7214, 7241, 7258, 7288, 7307,
 7317, 7324, 7331, 7338, 7360, 7373,
 7386, 7391, 7402, 7436, 7451, 7537,
 7551, 7588, 7606, 7616, 7635, 7640,
 7654, 7659, 7669, 7680, 7692, 7704,
 7718, 7753, 7765, 7771, 7777, 7789,
 7806, 7815, 7828, 7830, 7838, 7851,
 7864, 7879, 7895, 7904, 7910, 7919,
 7926, 7986, 8033, 8041, 8070, 8119,
 8127, 8151, 8264, 8267, 8276, 8278,
 8280, 8293, 8306, 8319, 8332, 8340,
 8350, 8386, 8422, 8435, 8454, 8477,
 8481, 8487, 8489, 8530, 8537, 8576,
 8621, 8688, 8690, 8692, 8694, 8706,
 8719, 8735, 8737, 8754, 8763, 8765,
 8772, 8774, 8781, 8831, 8840, 8848,
 8856, 8858, 8881, 8896, 8903, 8920,
 9015, 9042, 9050, 9060, 9070, 9072,
 9074, 9076, 9078, 9080, 9082, 9084,
 9086, 9098, 9100, 9102, 9104, 9106,
 9133, 9142, 9155, 9157, 9159, 9161,
 9164, 9177, 9179, 9181, 9183, 9357,
 9359, 9368, 9380, 9420–9426, 9467,
 9481, 9493, 9513, 9518, 9539, 9545,
 9575, 9577, 9584, 9589, 9594, 9603,
 9610, 9620, 9635, 9677, 9693, 9707,
 9719, 9730, 9735, 9740, 9746, 9749,
 9754, 9771, 9787, 9793, 9799, 9897,
 9899, 9907, 9909, 9921, 9926, 9931,
 9958, 10132, 10144, 10154, 10188,
 10205, 10217, 10223, 10232, 10252,
 10257, 10350, 10352, 10363, 10398,
 10406, 10408, 10410, 10416, 10418,
 10435, 10447, 10522, 10547, 10562,
 10563, 10569, 10575, 10577, 10581,
 10583, 10589, 10625, 11044, 11079,
 11118, 11131, 11161, 11186, 11212,
 11315, 11333, 11414, 11426, 11437,
 11446, 11588, 11594, 11610, 11633,
 11677, 11737, 11769, 11779, 11797,
 11830, 11844, 11850, 11900, 11998,
 12095, 12296, 12489, 12660, 12666,
 12785, 12807, 12917, 12953, 13020,
 13090, 13141, 13385, 13571, 13577,
 13583, 13589, 13595, 13601, 13607,
 13667, 13676, 13683, 13714, 13753
 \cs_new_protected:Npx
 [1208](#), [1223](#), [1249](#), [1477](#)
 \cs_new_protected:Nx [1352](#)
 \cs_new_protected_nopar:cn [1384](#)
 \cs_new_protected_nopar:cpn
 [1238](#), [1242](#),
[1480](#), [9811](#), [9865](#), [9893](#), [9895](#), [13394](#),
[13420](#), [13455](#), [13473](#), [13505](#), [13537](#)
 \cs_new_protected_nopar:cpx
 [1238](#), [1243](#), [1484](#)
 \cs_new_protected_nopar:cx [1384](#)
 \cs_new_protected_nopar:Nn [13](#), [1352](#)
 \cs_new_protected_nopar:Npn
 [11](#), [1208](#), [1220](#), [1225](#), [1242](#),
[1251–1253](#), [1259–1265](#), [1318](#), [1320](#),
[1472](#), [1670](#), [1680–1685](#), [1691–1697](#),
[1756](#), [2274](#), [2968](#), [2970](#), [3057](#), [3066](#),
[3184](#), [3195](#), [3197](#), [3199](#), [3433](#), [3435](#),
[3445](#), [3447](#), [3455](#), [4060](#), [4062](#), [4064](#),
[4066](#), [4492](#), [4494](#), [4496](#), [4528](#), [4530](#),
[4532](#), [4534](#), [4662–4664](#), [4710](#), [5176](#),
[5247](#), [5249](#), [5381](#), [5383](#), [5422](#), [5424](#),
[5490](#), [5705](#), [5706](#), [5708](#), [5724](#), [5726](#),
[5734](#), [5736](#), [5779](#), [5781](#), [5841](#), [5843](#),
[5854](#), [5856](#), [5872](#), [5874](#), [6339](#), [6341](#),
[7256](#), [7286](#), [7353](#), [7822](#), [8207](#), [8213](#),
[8217](#), [8372](#), [8408](#), [8450](#), [8452](#), [8480](#),
[8484](#), [8486](#), [8583](#), [8593](#), [8609](#), [8628](#),
[8642](#), [8648](#), [8696](#), [8698](#), [8700](#), [9069](#),
[9667](#), [9761](#), [10259](#), [10373](#), [10502–](#)
[10504](#), [10512](#), [10537](#), [10587](#), [10588](#),
[10621](#), [10623](#), [11038](#), [11041](#), [11073](#),
[11076](#), [11109](#), [11157](#), [11158](#), [11182](#),
[11183](#), [11208](#), [11209](#), [11225](#), [11262](#),
[11279](#), [11311](#), [11312](#), [11329](#), [11330](#),
[11384](#), [11431](#), [11442](#), [11443](#), [11481](#),
[11527](#), [11545](#), [11562](#), [11573](#), [11574](#),
[11579](#), [11754](#), [11759](#), [11787](#), [11820](#),
[11863](#), [11882](#), [11939](#), [11957](#), [11967](#),
[11994](#), [11995](#), [12045](#), [12078](#), [12091](#),

- 12092, 12131, 12164, 12187, 12223,
 12238, 12292, 12293, 12343, 12353,
 12382, 12412, 12485, 12486, 12533,
 12561, 12573, 12608, 12640, 12673,
 12737, 12803, 12804, 12840, 12864,
 12898, 12927, 12941, 12975, 13000,
 13008, 13026, 13080, 13105, 13137,
 13138, 13192, 13228, 13282, 13294,
 13619, 13627, 13632, 13641, 13724
 \cs_new_protected_nopar:Npx
 .. 1208, 1221, 1243, 1476, 1844, 8636
 \cs_new_protected_nopar:Nx 1352
 \cs_set:cn 1368
 \cs_set:cpn .. 1232, 1232, 8767, 8769, 9733
 \cs_set:cpx 1232, 1233,
 2310, 2314, 2318, 2322, 2326, 2335,
 2344, 2353, 2362, 2364, 2366, 2368,
 2370, 2372, 2374, 2376, 2378, 9738
 \cs_set:cx 1368
 \cs_set:Nn 13, 1327
 \cs_set:Npn 11,
 850, 852, 878, 884–912, 918, 931,
 1037–1040, 1049, 1050, 1058, 1064,
 1074, 1077, 1079, 1137, 1139, 1141,
 1143, 1145, 1147, 1149, 1151, 1208,
 1224, 1232, 1327, 1360, 1506–1509,
 2385, 2386, 3069, 3075, 3167, 3177,
 3308, 4132, 4140, 4148, 4154, 4160,
 4168, 4176, 4182, 4670, 4760, 5794,
 5931, 5973, 8514, 8564, 8709, 10663
 \cs_set:Npx 850, 854, 1233, 3178, 4549
 \cs_set:Nx 1327
 \cs_set_eq:cc . 948, 1250, 1253, 1905, 4411
 \cs_set_eq:cN 1250, 1251, 1904, 4409, 6241
 \cs_set_eq:Nc 1250, 1252, 1903, 4410
 \cs_set_eq:NN 15, 1250, 1250–1253, 1257,
 1262, 1434–1441, 1445–1452, 1852,
 1891, 1893, 1902, 2671, 2976, 2980,
 3001, 3003, 3062, 3080, 3181, 4408,
 5712, 5713, 5715, 6240, 7340–7347,
 7355–7358, 8271, 8272, 8554–8556,
 9913, 9915, 11484, 11577, 12416, 13338
 \cs_set_eq:NwN 1511, 1512
 \cs_set_nopar:cn 1368
 \cs_set_nopar:cpn 1224, 1226
 \cs_set_nopar:cpx 1224, 1227
 \cs_set_nopar:cx 1368
 \cs_set_nopar:Nn 13, 1327
 \cs_set_nopar:Npn
 .. 11, 850, 850, 852–854, 856–858,
 861, 913, 915, 1043, 1173, 1226, 8818
 \cs_set_nopar:Npx 850, 851, 855,
 859, 863, 881, 1227, 1540, 1718,
 2982, 2987, 3004, 3005, 4417, 4419,
 4421, 4435, 4437, 4439, 4441, 4459,
 4461, 4463, 4465, 4932, 8548–8552
 \cs_set_nopar:Nx 1327
 \cs_set_protected:cn 1368
 \cs_set_protected:cpn .. 1244, 1244, 8923
 \cs_set_protected:cpx ... 1244, 1245,
 1329, 1362, 8925, 8927, 8929, 8931
 \cs_set_protected:cx 1368
 \cs_set_protected:Nn 13, 1327
 \cs_set_protected:Npn .. 11, 850, 860,
 879, 922, 925, 935, 938, 947, 949,
 951, 959, 967, 975, 983, 991, 996,
 1004, 1012, 1020, 1025, 1027, 1157,
 1169, 1171, 1175, 1185, 1198, 1210,
 1244, 5354, 6262, 7265, 9135, 9137,
 9139, 9429, 10458, 10475, 11417,
 13697, 13706, 13734, 13748, 13762
 \cs_set_protected:Npx 850, 862,
 1245, 9022, 11242, 11460, 11470,
 11496, 12823, 12831, 12857, 13163,
 13169, 13180, 13213, 13220, 13266
 \cs_set_protected:Nx 1327
 \cs_set_protected_nopar:cn 1368
 \cs_set_protected_nopar:cpn . 1238, 1238
 \cs_set_protected_nopar:cpx . 1238, 1239
 \cs_set_protected_nopar:cx 1368
 \cs_set_protected_nopar:Nn 14, 1327
 \cs_set_protected_nopar:Npn
 12, 310, 850, 856, 860, 862,
 866, 868, 870, 872, 874, 876, 917,
 919, 921, 923, 930, 932, 934, 936,
 1153, 1155, 1196, 1206, 1238, 7246,
 8483, 8485, 10449, 10466, 13739, 13749
 \cs_set_protected_nopar:Npx
 .. 296, 850, 858, 1239, 8891, 9017,
 10599, 10641, 10661, 11053, 11089,
 11166, 11353, 12023, 12036, 12123,
 12321, 12334, 12518, 13049, 13301
 \cs_set_protected_nopar:Nx 1327
 \cs_show:c 820, 831, 10027
 \cs_show:N 16, 804, 809, 832, 4940
 \cs_split_function:NN
 20, 927, 942, 1033, 1034,
 1052, 1058, 1078, 1080, 1278, 1779
 \cs_split_function_aux:w 1052, 1061, 1064

- \cs_split_function_auxii:w 1052, 1072, 1074
- \cs_tmp:w 1208, 1216–1224, 1226–1249, 1327, 1336–1360, 1368–1391, 1815, 1852
- \cs_to_str:N 4, 17, 1043, 1043, 1062, 2285, 8492
- \cs_to_str_aux:N . 1043, 1047, 1049, 1050
- \cs_to_str_aux:w 1043, 1046, 1050
- \cs_undefine:c 1266, 1268, 1495
- \cs_undefine:N 15, 1266, 1266, 1494
- \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
- \currentgroupstype 696
- \currentifbranch 692
- \currentiflevel 691
- \currentifttype 693
- D**
- \d 1828
- \dagger 3995, 4001
- \day 651
- \ddagger 3996, 4002
- \deadcycles 585
- \def 54, 56, 98, 104, 106, 107, 109, 112–115, 118, 126, 128–130, 133, 141–144, 147, 152, 157, 203, 213, 292, 325, 339, 350
- \defaultthyphenchar 635
- \defaultskewchar 636
- \delcode 666
- \delimiter 460
- \delimiterfactor 509
- \delimitershortfall 508
- \deprecated 2388, 2389, 9420–9426
- \Depth 7338, 7341, 7345, 7349, 7356
- \detokenize 32, 35, 80, 93, 96, 167, 170, 176, 185, 190, 192, 198, 201, 204, 214, 269, 271, 276, 278, 683
- \dim_abs:n 73, 4084, 4084
- \dim_add:cn 4074
- \dim_add:Nn 72, 4074, 4074, 4076, 4077
- \dim_compare:n 4103, 4103
- \dim_compare:nF 4142, 4157
- \dim_compare:nNn 4098, 4098
- \dim_compare:nNnF 4170, 4185
- \dim_compare:nNnT 4069, 4162, 4179, 7557, 7562
- \dim_compare:nNnTF 74, 2130, 6927, 6930, 7059, 7069, 7089, 7095, 7454, 7457, 7460, 7469, 7472, 7475, 7484, 7491, 7569, 7682, 7694
- \dim_compare:nT 4134, 4151
- \dim_compare:nTF 74
- \dim_compare_<:nw 4103
- \dim_compare_=:nw 4103
- \dim_compare_>:nw 4103
- \dim_compare_aux:wNn ... 4103, 4105, 4113
- \dim_compare_p:nNn 4098
- \dim_do_until:nn ... 75, 4132, 4154, 4158
- \dim_do_until:nNn .. 75, 4160, 4182, 4186
- \dim_do_while:nn ... 75, 4132, 4148, 4152
- \dim_do_while:nNn .. 75, 4160, 4176, 4180
- \dim_eval:n 76, 2124, 4188, 4188, 7053, 7056, 7060, 7064, 7070, 7074, 7083, 7086, 7094, 7099, 7233, 7277, 7367, 7380, 7398, 7400, 7406, 7417, 7431, 7665, 7666, 7834, 7835, 7842, 7843, 7923, 7930
- \dim_eval:w 82, 4028, 4029, 4050, 4075, 4080, 4087, 4088, 4091, 4097, 4100, 4105, 4125–4131, 4189, 4191, 4195, 4212, 6588, 6590, 6592, 6601, 6603, 6605, 6607, 6677, 6697, 6710, 6725, 6753
- \dim_eval_end 82
- \dim_eval_end: 4028, 4030, 4050, 4075, 4080, 4091, 4092, 4097, 4100, 4106, 4189, 4191, 4195, 4212, 6588, 6590, 6592, 6601, 6603, 6605, 6607, 6677, 6697, 6710, 6725, 6753
- \dim_gadd:cn 4074
- \dim_gadd:Nn 72, 4074, 4076, 4078
- \dim_gset:cn 4049
- \dim_gset:Nn 73, 4049, 4051, 4053, 4063, 4067
- \dim_gset_eq:cc 4054
- \dim_gset_eq:cN 4054
- \dim_gset_eq:Nc 4054
- \dim_gset_eq:NN 73, 4054, 4057–4059
- \dim_gset_max:cn 4060
- \dim_gset_max:Nn ... 73, 4060, 4062, 4071
- \dim_gset_min:cn 4060
- \dim_gset_min:Nn ... 73, 4060, 4066, 4073
- \dim_gsub:cn 4074
- \dim_gsub:Nn 73, 4074, 4081, 4083
- \dim_gzero:c 4039
- \dim_gzero:N ... 72, 4039, 4040, 4042, 4046
- \dim_gzero_new:c 4043

<code>\dim_gzero_new:N</code> . . .	72, 4043 , 4045, 4048
<code>\dim_new:c</code>	4031
<code>\dim_new:N</code>	72, 4031 , 4032, 4038, 4044, 4046, 4214, 4215, 4222– 4226, 6757–6764, 7113, 7139, 7140, 7145–7148, 7153–7156, 7713–7717, 7849, 7850, 7974, 7976, 7977, 10669
<code>\dim_ratio:nn</code>	74, 4094 , 4094
<code>\dim_ratio_aux:n</code>	4094 , 4095, 4096
<code>\dim_set:cn</code>	4049
<code>\dim_set:Nn</code>	73, 4049 , 4049, 4051, 4052, 4061, 4065, 4216, 6790– 6792, 6839, 6850, 6903–6905, 6928, 6929, 6932, 6934, 6938, 6940, 6950– 6952, 6971–6973, 6993–6995, 7012, 7013, 7016, 7017, 7220, 7263, 7348– 7351, 7456, 7461, 7471, 7476, 7486, 7493, 7526, 7549, 7560, 7619, 7620, 7622, 7624, 7642, 7643, 7759, 7803, 7804, 7808–7811, 7824, 7899, 7902, 7975, 8078, 8079, 8130–8132, 8134
<code>\dim_set_eq:cc</code>	4054
<code>\dim_set_eq:cN</code>	4054
<code>\dim_set_eq:Nc</code>	4054
<code>\dim_set_eq:NN</code>	73, 4054 , 4054–4056
<code>\dim_set_max:cn</code>	4060
<code>\dim_set_max:Nn</code> 73, 4060 , 4060, 4070, 7818, 7820
<code>\dim_set_max_aux:NNNn</code> 4060 , 4061, 4063, 4065, 4067, 4068
<code>\dim_set_min:cn</code>	4060
<code>\dim_set_min:Nn</code>	73, 4060 , 4064, 4072, 7817, 7819, 7829
<code>\dim_show:c</code>	4209
<code>\dim_show:N</code>	76, 4209 , 4209, 4210
<code>\dim_show:n</code>	76, 4211 , 4211
<code>\dim_strip_bp:n</code>	83, 4190 , 4190
<code>\dim_strip_pt:n</code>	83, 4191, 4192 , 4192
<code>\dim_strip_pt:w</code>	4192 , 4195, 4199
<code>\dim_sub:cn</code>	4074
<code>\dim_sub:Nn</code>	73, 4074 , 4079, 4081, 4082
<code>\dim_until_do:nn</code> . . .	75, 4132 , 4140, 4145
<code>\dim_until_do:nNnn</code> . .	75, 4160 , 4168, 4173
<code>\dim_use:c</code>	4207
<code>\dim_use:N</code>	76, 4086, 4105, 4189, 4195, 4207 , 4207, 4208, 7394, 7396, 7400, 7411, 7424, 7650, 7756, 7758, 7761, 7763, 7769, 7775, 7784– 7786, 7908, 7915, 8189–8191, 10633
<code>\dim_while_do:nn</code> . . .	76, 4132 , 4132, 4137
<code>\dim_while_do:nNnn</code> . .	75, 4160 , 4160, 4165
<code>\dim_zero:c</code>	4039
<code>\dim_zero:N</code> 72, 4039 , 4039–4041, 4044, 6793, 6906, 6953, 6974, 6996, 7447, 7448
<code>\dim_zero_new:c</code>	4043
<code>\dim_zero_new:N</code>	72, 4043 , 4043, 4047
<code>\dimen</code>	657
<code>\dimendef</code>	356
<code>\dimexpr</code>	710
<code>\directlua</code>	15, 759
<code>\discretionary</code>	520
<code>\displayindent</code>	485
<code>\displaylimits</code>	495
<code>\displaystyle</code>	473
<code>\displaywidowpenalties</code>	723
<code>\displaywidowpenalty</code>	484
<code>\displaywidth</code>	486
<code>\divide</code>	363
<code>\doublehyphendemerits</code>	553
<code>\dp</code>	664
<code>\driver_box_rotate_begin:</code>	6811
<code>\driver_box_rotate_end:</code>	6813
<code>\driver_box_scale_begin:</code>	7028
<code>\driver_box_scale_end:</code>	7030
<code>\driver_box_use_clip:N</code>	7049
<code>\driver_color_ensure_current:</code> . . .	8214
<code>\dump</code>	647
E	
<code>\E</code>	1834
<code>\edef</code>	33, 68, 82, 164, 166, 181, 201, 266, 273, 351
<code>\else</code>	14, 23, 63, 117, 137, 172, 187, 207, 404
<code>\else:</code>	785, 788, 825, 1068, 1085, 1088, 1097, 1103, 1113, 1116, 1125, 1131, 1272, 1283, 1308, 1395, 1459, 1464, 1507, 1561, 1920, 1934, 1948, 1958, 1969, 1972, 1982, 1985, 1998, 2007, 2263, 2265, 2267, 2269, 2415, 2431, 2454, 2462, 2475, 2484, 2653, 2658, 2663, 2668, 2675, 2681, 2686, 2691, 2696, 2701, 2706, 2711, 2716, 2721, 2741, 2749, 2756, 2794, 2797, 2816, 2819, 2836, 2839, 2854, 2857, 2930, 2939, 2947, 2956, 3022, 3030, 3052, 3327, 3338, 3348, 3353, 3356, 3359, 3467, 3478, 3486, 3494, 3502, 3510, 3518, 3526, 3534, 3542, 3550, 3753, 4101, 4108, 4272,

4604, 4616, 4629, 4639, 4655, 4837,	\errhelp 250, 424
4857, 4872, 4880, 4890, 4912, 4964,	\errmessage 418
6358, 6384, 6612, 6614, 6624, 8667,	\ERROR 2385, 2386
8678, 8681, 10356, 10385, 10431,	\errorcontextlines 425
10442, 10492, 10498, 10508, 10606,	\errorstopmode 439
10648, 10691, 10695, 10712, 10756,	\escapechar 457
10764, 10767, 10776, 10804, 10823,	\etex_beginL:D 730
10832, 10900, 10903, 10920, 10923,	\etex_beginR:D 732
10939, 10942, 10965, 10968, 10971,	\etex_botmarks:D 679
10974, 10977, 10980, 10983, 10986,	\etex_clubpenalties:D 721
10989, 11004, 11007, 11010, 11013,	\etex_currentgrouplevel:D 695
11016, 11019, 11022, 11025, 11028,	\etex_currentgroupstype:D 696
11060, 11096, 11125, 11139, 11144,	\etex_currentifbranch:D 692
11195, 11233, 11249, 11274, 11297,	\etex_currentiflevel:D 691
11307, 11367, 11370, 11465, 11475,	\etex_currentiftype:D 693
11510, 11513, 11549, 11551, 11554,	\etex_detokenize:D 683, 4722, 4723
11600, 11605, 11626, 11802, 11874,	\etex_dimexpr:D 710, 4029
11888, 11923, 11948, 11963, 11978,	\etex_displaywidowpenalties:D 723
12011, 12028, 12032, 12051, 12066,	\etex_endL:D 731
12108, 12120, 12137, 12152, 12180,	\etex_endR:D 733
12182, 12192, 12208, 12213, 12228,	\etex_eTeXrevision:D 675
12265, 12271, 12276, 12309, 12326,	\etex_eTeXversion:D 674
12330, 12347, 12359, 12362, 12365,	\etex_everyeof:D 735, 4501
12374, 12378, 12405, 12408, 12433,	\etex_firstmarks:D 678
12503, 12515, 12526, 12542, 12547,	\etex_fontcharhp:D 703
12552, 12557, 12565, 12581, 12595,	\etex_fontcharht:D 702
12601, 12626, 12645, 12680, 12688,	\etex_fontcharic:D 705
12712, 12715, 12758, 12764, 12769,	\etex_fontcharwd:D 704
12822, 12830, 12854, 12868, 12871,	\etex_glueexpr:D 711, 4246, 4257,
12885, 12903, 12909, 12949, 12957,	4262, 4287, 4292, 4295, 4301, 10628
12985, 13063, 13071, 13094, 13123,	\etex_glueshrink:D 714, 4367
13129, 13168, 13175, 13185, 13197,	\etex_glueshrinkorder:D 716, 4281
13211, 13219, 13232, 13246, 13255,	\etex_gluestretch:D 713, 4366
13261, 13345, 13353, 13403, 13407,	\etex_gluestretchorder:D 715, 4280
13411, 13415, 13430, 13434, 13439,	\etex_gluetomu:D 717
13446, 13450, 13460, 13464, 13467,	\etex_ifcsname:D 672, 800
13478, 13482, 13486, 13491, 13496,	\etex_ifdefined:D 671, 799, 845
13510, 13514, 13518, 13523, 13528	\etex_iffontchar:D 701
\emergencystretch 568	\etex_interactionmode:D 699
\end 442	\etex_interlinepenalties:D 720
\EndCatcodeRegime 13749	\etex_lastlinefit:D 719
\endcsname 13, 32, 35, 62, 80, 93, 96, 167,	\etex_lastnodetype:D 700
170, 176, 185, 190, 192, 201, 204,	\etex_marks:D 676
214, 229, 233, 269, 271, 276, 278, 444	\etex_middle:D 724
\endgroup 12, 61, 111, 120, 228, 232, 377	\etex_muexpr:D
\endinput 264, 416 712, 4329, 4340, 4345, 4350, 4356
\endL 731	\etex_mutoglue:D 718
\endlinechar 79, 92, 289, 458	\etex_numexpr:D 709, 3302
\endR 733	\etex_pagediscards:D 727
\eqno 478	\etex_parshapedimen:D 708

<code>\etex_parshapeindent:D</code>	706	1705, 1708, 1709, 1713, 1714, 1722,
<code>\etex_parshapelength:D</code>	707	1724, 1725, 1727, 1729, 1732, 1733,
<code>\etex_predisplaydirection:D</code>	734	1738, 1739, 1743, 1746–1748, 1752,
<code>\etex_protected:D</code>	736, 818	1758, 1760–1762, 1764, 1767, 1772,
<code>\etex_readline:D</code>	686, 8693, 8695	1787, 1793, 1842, 1867, 1968, 1971,
<code>\etex_savinghyphcodes:D</code>	725	1973, 1981, 1984, 1986, 1991, 1992,
<code>\etex_savingvdiscards:D</code>	726	1995, 2004, 2012, 2014–2016, 2019,
<code>\etex_scantokens:D</code>	684, 4512	2024, 2172, 2283, 2284, 2298, 2408,
<code>\etex_showgroups:D</code>	697	2414, 2416, 2423, 2430, 2432, 2440,
<code>\etex_showifs:D</code>	698	2447, 2734, 2755, 2775, 2784, 2800,
<code>\etex_showtokens:D</code>	685, 4942, 9385	2822, 2842, 2860, 2873, 2884, 2894,
<code>\etex_splitbotmarks:D</code>	681	2914, 2937, 2938, 2940, 2946, 2949,
<code>\etex_splitdiscards:D</code>	728	3021, 3023, 3029, 3031, 3044, 3051,
<code>\etex_splitfirstmarks:D</code>	680	3053, 3142, 3151, 3160, 3461, 3464,
<code>\etex_TeXXETstate:D</code>	729	3725, 3753, 3764, 3774, 3907, 4105,
<code>\etex_topmarks:D</code>	677	4194, 4510, 4511, 4560, 4568, 4573,
<code>\etex_tracingassigns:D</code>	687	4614, 4626, 4627, 4723, 4796, 4800,
<code>\etex_tracinggroups:D</code>	694	4802, 4808, 4816, 4824, 4834, 4850,
<code>\etex_tracingifs:D</code>	690	4870, 4879, 4882, 4904–4906, 4924–
<code>\etex_tracingnesting:D</code>	689	4926, 4944, 5002, 5030, 5034, 5057,
<code>\etex_tracingscantokens:D</code>	688	5105, 5113, 5137, 5138, 5358, 5375,
<code>\etex_unexpanded:D</code>	682,	5388, 5407, 5408, 5436, 5437, 5458,
	803, 1761, 1764, 1767, 1772, 4764,	5463, 5553, 5559, 5580, 5587, 5655–
	4808, 4810, 5034, 5057, 5105, 5113	5657, 5664, 5665, 5868, 5878, 5939,
<code>\etex_unless:D</code>	673, 790	5947, 5952, 6162, 6265, 6423, 6485,
<code>\etex_widowpenalties:D</code>	722	8659, 8666, 8669, 8872, 9318–9321,
<code>\eTeXrevision</code>	675	9332, 9387–9390, 9475, 9477, 9516,
<code>\eTeXversion</code>	674	9624, 9752, 10105, 10117, 10201,
<code>\everycr</code>	386	10351, 10371, 10376, 10380, 10384,
<code>\everydisplay</code>	487	10387, 10391, 10394, 10413, 10432,
<code>\everyeof</code>	735	10441, 10443, 10453, 10455, 10470,
<code>\everyhbox</code>	626	10472, 10484, 10499, 10507, 10509,
<code>\everyjob</code>	31, 655	10516, 10519, 10531, 10541, 10544,
<code>\everymath</code>	511	10556, 10611, 10632, 10653, 10682,
<code>\everypar</code>	574	10690, 10693, 10696, 10711, 10713,
<code>\everyvbox</code>	627	10747, 10755, 10757, 10775, 10777,
<code>\exhyphenpenalty</code>	550	10814, 10822, 10824, 10831, 10833,
<code>\exp_after:wN</code>	30, 801, 801,	10899, 10902, 10904, 10916, 10935,
	819, 824, 826, 914, 916, 962, 1030,	11050, 11065, 11086, 11101, 11115,
	1047, 1051, 1060, 1061, 1067, 1069,	11154, 11174, 11200, 11205, 11206,
	1096, 1098, 1101, 1124, 1126, 1129,	11232, 11234, 11254, 11375, 11423,
	1271, 1273, 1282, 1284, 1287, 1332,	11434, 11476, 11518, 11534, 11540,
	1364, 1521, 1528, 1530, 1533, 1534,	11548, 11555–11557, 11764, 11766,
	1541, 1545, 1546, 1551, 1552, 1557,	11776, 11791, 11794, 11824, 11827,
	1562, 1564, 1567, 1575, 1577, 1579,	11838, 11841, 11857, 11873, 11879,
	1581, 1583, 1585, 1588–1590, 1594,	11887, 11893–11895, 11964, 11977,
	1597, 1602, 1607–1609, 1613–1615,	11989, 12016, 12033, 12071, 12082,
	1619–1621, 1625–1627, 1631–1633,	12084, 12086, 12088, 12113, 12121,
	1637–1640, 1644–1647, 1651–1653,	12157, 12168, 12170, 12172, 12174,
	1658–1661, 1665–1668, 1701, 1702,	12220, 12235, 12289, 12314, 12331,

- 12346, 12350, 12375, 12379, 12406,
 12409, 12438, 12508, 12516, 12539–
 12541, 12543–12545, 12549–12551,
 12558, 12564, 12570, 12580, 12584,
 12597–12599, 12603, 12604, 12617,
 12662, 12730, 12782, 12821, 12828,
 12836, 12847, 12855, 12890, 12907,
 12945, 12948, 12984, 12986, 12997,
 13005, 13022, 13070, 13072, 13084,
 13087, 13134, 13186, 13196, 13208–
 13210, 13231, 13240–13244, 13248–
 13253, 13257–13259, 13262, 13274
 \exp_arg_last_unbraced:nn
 .. 1698, 1698, 1701, 1705, 1708, 1713
 \exp_arg_next:NNn 1522, 1523, 1528
 \exp_arg_next:nnn 1522,
 1522, 1530, 1533, 1541, 1545, 1551
 \exp_args:cc 1580, 1580
 \exp_args:Nc 27, 819, 819,
 820, 828, 969, 977, 985, 993, 1197,
 1207, 1225, 1251, 1259, 1264, 1294,
 1319, 1397–1400, 1419, 1580, 4696
 \exp_args:Ncc 1253, 1261,
 1265, 1405–1408, 1580, 1584, 8702
 \exp_args:Nccc 1580, 1586
 \exp_args:Ncco 1642, 1663
 \exp_args:Nccx 1686, 1695
 \exp_args:Ncf 1605, 1629
 \exp_args:NcNc 1642, 1649
 \exp_args:NcNo 1642, 1656
 \exp_args:Ncnx 1686, 1696
 \exp_args:Nco 1605, 1623
 \exp_args:Ncx 1671, 1681
 \exp_args:Nf 28, 1593, 1593,
 2111, 2124, 3627, 3696, 3708, 3717,
 3810, 3823, 3837, 3847, 3858, 3869,
 5029, 5129, 5142, 5160, 5648, 6101,
 6120, 6133, 6138, 6154, 9330, 9385
 \exp_args:Nff 1671, 1673
 \exp_args:Nfo 1671, 1672, 6108
 \exp_args:NNc 832, 1033, 1034,
 1252, 1260, 1263, 1321, 1401–1404,
 1580, 1582, 1789, 2240, 2246, 9332
 \exp_args:Nnc 1671, 1671
 \exp_args:NNf 1605, 1605, 2218, 2225, 2234
 \exp_args:Nnf 940, 1671, 1674
 \exp_args:Nnnc 1686, 1688
 \exp_args:NNNo . 29, 1575, 1578, 8289, 8302
 \exp_args:NNno 1686, 1686
 \exp_args:Nnno 1686, 1689
 \exp_args:NNNV 1642, 1642
 \exp_args:NNnx 29, 1686, 1691
 \exp_args:Nnnx 1686, 1693
 \exp_args:NNo ... 28, 1575, 1576, 3615,
 5131, 6259, 8573, 9751, 10141, 10214
 \exp_args:Nno 28, 1671,
 1675, 3134, 4112, 6039, 6274, 10004
 \exp_args:NNoo 29, 1686, 1687
 \exp_args:NNox 1686, 1692
 \exp_args:Nnox 1686, 1694
 \exp_args:NNV 1605, 1617
 \exp_args:NNv 1605, 1611
 \exp_args:NnV 1671, 1676
 \exp_args:NNx 29, 1671, 1680
 \exp_args:Nnx 1671, 1682
 \exp_args:No 27,
 1575, 1575, 3615, 3700, 4501, 4662–
 4664, 4684, 4702, 4711, 4968–4971,
 5072, 5074, 5162, 5274, 5812, 5946,
 5964, 5969, 6148, 6152, 8655, 8724
 \exp_args:Noc 1671, 1679
 \exp_args:Nof 1671, 1678
 \exp_args:Noo 1671, 1677
 \exp_args:Nooo 1686, 1690
 \exp_args:Noox 1686, 1697
 \exp_args:Nox 1671, 1683
 \exp_args:NV 28, 1593, 1600
 \exp_args:Nv 28, 1593, 1595
 \exp_args:NVV 1605, 1635
 \exp_args:Nx 28, 1670, 1670
 \exp_args:Nxo 1671, 1684
 \exp_args:Nxx 1671, 1685
 \exp_eval_error_msg:w .. 1555, 1559, 1568
 \exp_eval_register:c 1552, 1555,
 1566, 1598, 1615, 1714, 1724, 1773
 \exp_eval_register:N
 32, 1546, 1555, 1555,
 1567, 1603, 1621, 1639, 1640, 1647,
 1709, 1722, 1734, 1740, 1749, 1768
 \exp_last_two_unbraced:Noo
 30, 1757, 1757, 7441, 7673, 7677
 \exp_last_two_unbraced_aux:noN
 1758, 1759
 \exp_last_unbraced:Nco
 1721, 1728, 6023, 6435
 \exp_last_unbraced:NcV 1721, 1730
 \exp_last_unbraced:Nf
 30, 1721, 1726, 3706, 6177
 \exp_last_unbraced:Nfo . 1721, 1755, 5629
 \exp_last_unbraced:NNNo 1721, 1751

- \exp_last_unbraced:NNNV [1721](#), [1744](#)
- \exp_last_unbraced:NNo
- .. [1721](#), [1742](#), [5016](#), [5991](#), [6416](#), [7647](#)
- \exp_last_unbraced:Nno . [1721](#), [1753](#), [6478](#)
- \exp_last_unbraced:NNV [1721](#), [1736](#)
- \exp_last_unbraced:No
- .. [1721](#), [1725](#), [8024](#), [8029](#), [8107](#), [8113](#)
- \exp_last_unbraced:Noo
- [1721](#), [1754](#), [6368](#), [6493](#)
- \exp_last_unbraced:NV [1721](#), [1721](#)
- \exp_last_unbraced:Nv [1721](#), [1723](#)
- \exp_last_unbraced:Nx [30](#), [1721](#), [1756](#)
- \exp_not:c [31](#), [1761](#),
[1762](#), [1815](#), [1865](#), [2311](#), [2315](#), [2320](#),
[2324](#), [2327](#), [2329](#)–[2331](#), [2336](#), [2338](#)–
[2340](#), [2345](#), [2347](#)–[2349](#), [2354](#), [2356](#)–
[2358](#), [2363](#), [2365](#), [2367](#), [2369](#), [2371](#),
[2373](#), [2375](#), [2377](#), [2379](#)–[2381](#), [3081](#),
[8926](#), [8928](#), [8930](#), [8932](#), [9323](#), [9500](#),
[9522](#), [9642](#), [9644](#), [9657](#), [9659](#), [9797](#)
- \exp_not:f [31](#), [1761](#), [1763](#)
- \exp_not:N [31](#), [801](#),
[802](#), [1331](#)–[1333](#), [1363](#)–[1365](#), [1521](#),
[1557](#), [1762](#), [1815](#), [2251](#), [2316](#), [2319](#),
[2323](#), [2327](#), [2336](#), [2345](#), [2354](#), [2422](#),
[2429](#), [2446](#), [2473](#), [2482](#), [2628](#), [2652](#),
[2657](#), [2662](#), [2667](#), [2674](#), [2680](#), [2685](#),
[2690](#), [2695](#), [2700](#), [2705](#), [2715](#), [2720](#),
[2748](#), [2755](#), [2984](#), [2989](#), [3007](#), [3020](#),
[3050](#), [4199](#), [4200](#), [4203](#), [4501](#), [4508](#),
[4518](#), [4520](#), [4552](#), [4553](#), [4832](#), [4834](#),
[4848](#), [4850](#), [4878](#), [4885](#), [4936](#), [5273](#),
[5275](#), [5507](#), [5740](#), [6092](#), [6095](#), [6102](#),
[6103](#), [8571](#), [8639](#), [9019](#), [9027](#), [9028](#),
[9030](#), [9323](#), [9642](#), [9644](#), [9657](#), [9659](#),
[9685](#), [9686](#), [9711](#), [9712](#), [9757](#), [9779](#),
[9780](#), [9797](#), [10602](#), [10644](#), [10663](#),
[11056](#), [11092](#), [11169](#), [11356](#), [11463](#),
[11473](#), [12026](#), [12039](#), [12126](#), [12324](#),
[12337](#), [12521](#), [12826](#), [12834](#), [12860](#),
[13053](#), [13055](#), [13057](#), [13304](#), [13306](#),
[13308](#), [13310](#), [13312](#), [13541](#), [13543](#),
[13545](#), [13547](#), [13549](#), [13551](#), [13553](#),
[13555](#), [13700](#), [13737](#), [13742](#), [13765](#)
- \exp_not:n [31](#), [801](#), [803](#), [1457](#), [1521](#), [1718](#),
[2252](#), [2312](#), [2422](#), [2429](#), [2446](#), [2473](#),
[2482](#), [2985](#), [2990](#), [3004](#), [3008](#), [3080](#),
[3082](#), [4386](#), [4417](#), [4423](#), [4435](#), [4443](#),
[4459](#), [4467](#), [4554](#), [4899](#), [4978](#), [5159](#),
[5301](#), [5508](#), [5647](#), [5714](#), [5717](#), [5720](#),
[5830](#), [5953](#), [6096](#), [6101](#), [6132](#), [6165](#),
[6186](#), [6187](#), [6327](#), [6328](#), [6349](#), [6501](#),
[8478](#), [8482](#), [8736](#), [9020](#), [9024](#), [9031](#),
[9315](#), [9397](#), [9401](#), [9402](#), [9406](#), [9407](#),
[9688](#), [9782](#), [9820](#), [13567](#), [13674](#), [13677](#)
- \exp_not:o
- . [31](#), [1761](#), [1761](#), [4419](#), [4425](#), [4435](#),
[4437](#), [4439](#), [4441](#), [4443](#), [4445](#), [4447](#),
[4449](#), [4459](#), [4461](#), [4463](#), [4465](#), [4467](#),
[4469](#), [4471](#), [4473](#), [4520](#), [4565](#), [4577](#),
[4759](#), [4898](#), [4973](#), [4977](#), [5280](#), [5282](#),
[5333](#), [5787](#), [5789](#), [5946](#), [8892](#), [9502](#),
[9524](#), [9527](#), [9534](#), [9543](#), [9995](#), [9997](#)
- \exp_not:v
- [31](#), [1761](#), [1765](#), [4437](#), [4445](#), [4461](#), [4469](#)
- \exp_not:v [31](#), [1761](#), [1770](#), [9714](#)
- \exp_stop_f [31](#)
- \exp_stop_f:
- .. [1531](#), [1537](#), [2285](#), [5132](#), [5885](#), [9332](#)
- \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](#), [1518](#),
[1883](#), [2395](#), [2513](#), [3298](#), [4025](#), [4374](#),
[5219](#), [5756](#), [6233](#), [6538](#), [7109](#), [8203](#),
[8223](#), [8748](#), [9451](#), [10100](#), [10277](#), [13647](#)
- \ExplFileDescription
- [113](#), [130](#), [334](#), [782](#), [1518](#),
[1883](#), [2395](#), [2513](#), [3298](#), [4025](#), [4374](#),
[5219](#), [5756](#), [6233](#), [6538](#), [7109](#), [8203](#),
[8223](#), [8748](#), [9451](#), [10100](#), [10277](#), [13647](#)
- \ExplFileName [114](#), [128](#), [334](#), [782](#), [1518](#),
[1883](#), [2395](#), [2513](#), [3298](#), [4025](#), [4374](#),
[5219](#), [5756](#), [6233](#), [6538](#), [7109](#), [8203](#),
[8223](#), [8748](#), [9451](#), [10100](#), [10277](#), [13647](#)
- \ExplFileVersion
- [49](#), [115](#), [129](#), [334](#), [782](#), [1518](#),
[1883](#), [2395](#), [2513](#), [3298](#), [4025](#), [4374](#),
[5219](#), [5756](#), [6233](#), [6538](#), [7109](#), [8203](#),
[8223](#), [8748](#), [9451](#), [10100](#), [10277](#), [13647](#)
- \ExplSyntaxNamesOff [6](#), [266](#), [273](#)
- \ExplSyntaxNamesOn [6](#), [266](#), [266](#)
- \ExplSyntaxOff [4](#), [6](#),
[67](#), [68](#), [178](#), [186](#), [208](#), [291](#), [296](#), [310](#), [325](#)
- \ExplSyntaxOn [4](#),
[6](#), [67](#), [82](#), [150](#), [155](#), [160](#), [206](#), [291](#), [292](#)

F

- \F 2728, 2906
- \fam 366
- \fi 23, 44, 65,
123, 139, 174, 194, 209, 231, 265, 405
- \fi: 785, 789, 827, 963, 1031, 1046,
1051, 1070, 1090, 1091, 1099, 1105,
1118, 1119, 1127, 1133, 1194, 1274,
1285, 1311, 1316, 1317, 1395, 1459,
1464, 1506–1509, 1560, 1563, 1570,
1571, 1788, 1868, 1922, 1936, 1948,
1958, 1974, 1975, 1987, 1988, 2000,
2009, 2263, 2265, 2267, 2269, 2271,
2273, 2409, 2417, 2424, 2433, 2441,
2448, 2456, 2464, 2477, 2486, 2653,
2658, 2663, 2668, 2675, 2681, 2686,
2691, 2696, 2701, 2706, 2711, 2716,
2721, 2743, 2749, 2756, 2804, 2805,
2826, 2827, 2846, 2847, 2864, 2865,
2932, 2941, 2950, 2958, 3024, 3032,
3054, 3318, 3329, 3340, 3355, 3361,
3362, 3364, 3469, 3473, 3480, 3488,
3496, 3504, 3512, 3520, 3528, 3536,
3544, 3552, 3754, 4090, 4101, 4110,
4120, 4274, 4606, 4618, 4631, 4641,
4658, 4828, 4839, 4859, 4874, 4883,
4892, 4914, 4918, 4926, 4966, 5003,
5329, 5332, 5359, 5435, 5439, 5459,
5554, 6360, 6386, 6424, 6486, 6612,
6614, 6624, 8670, 8683, 8684, 10358,
10395, 10396, 10428, 10433, 10444,
10456, 10473, 10497, 10500, 10510,
10520, 10545, 10608, 10650, 10688,
10697, 10698, 10714, 10753, 10758,
10769, 10770, 10778, 10806, 10820,
10825, 10834, 10905, 10906, 10922,
10926, 10941, 10946, 10967, 10970,
10973, 10976, 10979, 10982, 10985,
10988, 10991, 11006, 11009, 11012,
11015, 11018, 11021, 11024, 11027,
11030, 11051, 11062, 11087, 11098,
11129, 11141, 11149–11151, 11155,
11197, 11235, 11251, 11277, 11292,
11306, 11309, 11369, 11372, 11477,
11478, 11512, 11515, 11542, 11543,
11558–11560, 11570, 11603, 11608,
11618, 11622, 11630, 11631, 11752,
11767, 11795, 11810, 11828, 11872,
11880, 11896–11898, 11911, 11915,
11936, 11937, 11946, 11955, 11965,
11991, 11992, 12013, 12035, 12042,
12055, 12068, 12089, 12110, 12122,
12141, 12154, 12175, 12179, 12184,
12185, 12212, 12217, 12221, 12236,
12269, 12275, 12283, 12287, 12288,
12290, 12311, 12333, 12340, 12351,
12361, 12367, 12368, 12377, 12380,
12407, 12410, 12435, 12505, 12517,
12528, 12546, 12555, 12556, 12559,
12571, 12600, 12605, 12606, 12628,
12637, 12648, 12657, 12697, 12698,
12710, 12718, 12719, 12762, 12768,
12776, 12780, 12781, 12783, 12829,
12837, 12856, 12874, 12875, 12887,
12905, 12914, 12938, 12951, 12965,
12987, 12993, 12998, 13006, 13012,
13015, 13066, 13073, 13088, 13102,
13127, 13133, 13135, 13174, 13188,
13189, 13218, 13225, 13226, 13254,
13260, 13264, 13265, 13347, 13355,
13406, 13410, 13414, 13418, 13437,
13438, 13449, 13452, 13453, 13469–
13471, 13499–13503, 13531–13535
- \file_add_path:nN
.... 162, 10131, 10132, 10183, 10190
- \file_add_path_aux:nN 10131, 10142, 10144
- \file_add_path_search:nN
..... 10131, 10148, 10154
- \file_if_exist:n 10181, 10181
- \file_if_exist:nTF 162
- \file_input:n 163, 10188, 10188
- \file_list 163
- \file_list: 10259, 10259
- \file_path_include:n .. 163, 10252, 10252
- \file_path_remove:n ... 163, 10252, 10257
- \file_split_name_ext:wNN
..... 10205, 10226, 10232, 10249
- \file_split_path:wNNN
..... 10205, 10221, 10223, 10229
- \file_split_path_name_ext:nNNN
..... 163, 10205, 10205
- \file_split_path_name_ext_aux:nNNN .
..... 10205, 10215, 10217
- \finalhyphendemerits 554
- \firstmark 452
- \firstmarks 678
- \floatingpenalty 599
- \font 365
- \fontcharhp 703
- \fontcharht 702

\fontcharic	705	\fp_div:cn	<u>11442</u>
\fontcharwd	704	\fp_div:Nn	169, 6825, 6909, 6913, 6957, 6977, 7502, 7503, 7524, 7546, 7721, 7857, 7860, <u>11442</u> , <u>11442</u> , <u>11444</u>
\fontdimen	632	\fp_div_aux:Nn	<u>11442</u> , <u>11442</u> , <u>11443</u> , <u>11446</u>
\fontname	456	\fp_div_divide:	<u>11442</u> , <u>11530</u> , <u>11545</u> , <u>11571</u>
\fp_abs:c	<u>11157</u>	\fp_div_divide_aux: <u>11442</u> , <u>11548</u> , <u>11557</u> , <u>11562</u>
\fp_abs:N	169, <u>11157</u> , <u>11157</u> , <u>11159</u>	\fp_div_integer:NNNNN	. <u>11737</u> , <u>11737</u> , 12198, 12249, 12254, 12745, 13116
\fp_abs_aux:NN	<u>11157</u> , <u>11157</u> , <u>11158</u> , <u>11161</u>	\fp_div_internal: <u>11442</u> , <u>11476</u> , <u>11481</u> , <u>13034</u> , <u>13292</u>
\fp_add:cn	<u>11208</u>	\fp_div_loop: <u>11442</u> , <u>11486</u> , <u>11527</u> , <u>11541</u> , <u>12418</u>
\fp_add:Nn	169, 6849, 7515, 7548, 7802, <u>11208</u> , <u>11208</u> , <u>11210</u>	\fp_div_loop_step:w 11534, <u>11588</u>
\fp_add:NNNNNNNN <u>11594</u> , <u>11594</u> , <u>12932</u> , <u>12984</u> , <u>13070</u>	\fp_div_store: <u>11442</u> , 11484, 11531, 11573, 11577, 12416
\fp_add_aux:Nn	<u>11208</u> , <u>11208</u> , <u>11209</u> , <u>11212</u>	\fp_div_store_decimal:	<u>11442</u> , <u>11577</u> , <u>11579</u>
\fp_add_core:	. <u>11208</u> , <u>11222</u> , <u>11225</u> , <u>11326</u>	\fp_div_store_integer: <u>11442</u> , <u>11484</u> , <u>11574</u> , <u>12416</u>
\fp_add_difference:	. <u>11208</u> , <u>11234</u> , <u>11279</u>	\fp_exp:cn	<u>12485</u>
\fp_add_sum: <u>11208</u> , <u>11232</u> , <u>11262</u>	\fp_exp:Nn	170, <u>12485</u> , <u>12485</u> , <u>12487</u>
\fp_compare:n	<u>13561</u> , <u>13561</u>	\fp_exp_aux:	.. <u>12485</u> , <u>12541</u> , <u>12551</u> , <u>12561</u>
\fp_compare:NNN <u>13357</u> , <u>13374</u>	\fp_exp_aux:Nn	<u>12485</u> , <u>12485</u> , <u>12486</u> , <u>12489</u>
\fp_compare:nNn <u>13357</u> , <u>13357</u>	\fp_exp_const:cx	<u>12485</u> , <u>12553</u> , <u>12593</u> , <u>12725</u>
\fp_compare:NNNT 7870	\fp_exp_const:Nx	<u>12485</u> , <u>12785</u> , <u>12790</u> , <u>13338</u>
\fp_compare:NNNTF 6778, 6780, 6794, 6796, 6801, 6914, 6916, 6959, 6979, 6997, 6999, 7010, 7019, 7034, 7885, 7888	\fp_exp_decimal:	<u>12485</u> , <u>12570</u> , <u>12658</u> , <u>12673</u>
\fp_compare:nNnTF 168, 7504, 13575, 13581, 13587, 13593, 13599, 13605, 13611	\fp_exp_integer:	... <u>12485</u> , <u>12564</u> , <u>12573</u>
\fp_compare:nTF 168	\fp_exp_integer_const:n <u>12485</u> , <u>12596</u> , <u>12602</u> , 12625, 12627, 12644, 12646, 12660
\fp_compare_<: <u>13357</u>	\fp_exp_integer_const:nnnn <u>12485</u> , <u>12663</u> , <u>12666</u> , <u>13023</u>
\fp_compare_<_aux: <u>13357</u>	\fp_exp_integer_tens: <u>12485</u> , <u>12580</u> , <u>12599</u> , <u>12604</u> , <u>12608</u>
\fp_compare_>: <u>13357</u>	\fp_exp_integer_units:	<u>12485</u> , <u>12638</u> , <u>12640</u>
\fp_compare_absolute_a<b: <u>13357</u>	\fp_exp_internal: <u>12485</u> , <u>12516</u> , <u>12533</u> , <u>13339</u>
\fp_compare_absolute_a>b: <u>13357</u>	\fp_exp_overflow_msg: 12545, 12558, <u>13621</u> , <u>13627</u>
\fp_compare_aux:N <u>13357</u> , <u>13372</u> , <u>13383</u> , <u>13385</u>	\fp_exp_Taylor:	<u>12485</u> , <u>12703</u> , <u>12737</u> , <u>12782</u>
\fp_compare_aux_i:w	. <u>13561</u> , <u>13567</u> , <u>13571</u>	\fp_extended_normalise: <u>11754</u> , 11754, 11867, 12536, 13201, 13236
\fp_compare_aux_ii:w	<u>13561</u> , <u>13574</u> , <u>13577</u>	\fp_extended_normalise_aux:NNNNNNNN <u>11754</u>
\fp_compare_aux_iii:w	<u>13561</u> , <u>13580</u> , <u>13583</u>	\fp_extended_normalise_aux_i: <u>11754</u> , <u>11756</u> , <u>11759</u> , <u>11766</u>
\fp_compare_aux_iv:w	<u>13561</u> , <u>13586</u> , <u>13589</u>	\fp_extended_normalise_aux_i:w <u>11754</u> , <u>11764</u> , <u>11769</u>
\fp_compare_aux_v:w	. <u>13561</u> , <u>13592</u> , <u>13595</u>		
\fp_compare_aux_vi:w	<u>13561</u> , <u>13598</u> , <u>13601</u>		
\fp_compare_aux_vii:w	<u>13561</u> , <u>13604</u> , <u>13607</u>		
\fp_const:cn	<u>10569</u>		
\fp_const:Nn	165, <u>10569</u> , <u>10569</u> , <u>10574</u>		
\fp_cos:cn	<u>12091</u>		
\fp_cos:Nn 170, 6828, 7724, <u>12091</u> , <u>12091</u> , <u>12093</u>		
\fp_cos_aux:Nn	<u>12091</u> , <u>12091</u> , <u>12092</u> , <u>12095</u>		
\fp_cos_aux_i: <u>12091</u> , <u>12121</u> , <u>12131</u>		
\fp_cos_aux_ii:	<u>12091</u> , <u>12134</u> , <u>12164</u> , <u>12369</u>		

- \fp_extended_normalise_aux_ii: [11754](#), [11757](#), [11787](#), [11794](#)
- \fp_extended_normalise_aux_ii:w [11754](#), [11776](#), [11779](#)
- \fp_extended_normalise_ii_aux:NNNNNNNN [11792](#), [11797](#)
- \fp_extended_normalise_output: [11820](#), [11820](#), [11827](#), [12636](#), [12656](#), [13328](#)
- \fp_extended_normalise_output_aux:N [11820](#), [11848](#), [11850](#)
- \fp_extended_normalise_output_aux_i:NNNNNNNN [11820](#), [11825](#), [11830](#)
- \fp_extended_normalise_output_aux_ii:NNNNNNNN [11820](#), [11841](#), [11844](#)
- \fp_gabs:c [11157](#)
- \fp_gabs:N [169](#), [11157](#), [11158](#), [11160](#)
- \fp_gadd:cn [11208](#)
- \fp_gadd:Nn [169](#), [11208](#), [11209](#), [11211](#)
- \fp_gcos:cn [12091](#)
- \fp_gcos:Nn [170](#), [12091](#), [12092](#), [12094](#)
- \fp_gdiv:cn [11442](#)
- \fp_gdiv:Nn [169](#), [11442](#), [11443](#), [11445](#)
- \fp_gexp:cn [12485](#)
- \fp_gexp:Nn [170](#), [12485](#), [12486](#), [12488](#)
- \fp_gln:cn [12803](#)
- \fp_gln:Nn [170](#), [12803](#), [12804](#), [12806](#)
- \fp_gmul:cn [11329](#)
- \fp_gmul:Nn [169](#), [11329](#), [11330](#), [11332](#)
- \fp_gneg:c [11182](#)
- \fp_gneg:N [169](#), [11182](#), [11183](#), [11185](#)
- \fp_gpow:cn [13137](#)
- \fp_gpow:Nn [169](#), [13137](#), [13138](#), [13140](#)
- \fp_ground_figures:cn [11038](#)
- \fp_ground_figures:Nn [167](#), [11038](#), [11041](#), [11043](#)
- \fp_ground_places:cn [11073](#)
- \fp_ground_places:Nn [167](#), [11073](#), [11076](#), [11078](#)
- \fp_gset:cn [10587](#)
- \fp_gset:Nn [166](#), [10572](#), [10587](#), [10588](#), [10620](#)
- \fp_gset_eq:cc [10671](#), [10678](#)
- \fp_gset_eq:cN [10671](#), [10676](#)
- \fp_gset_eq:Nc [10671](#), [10677](#)
- \fp_gset_eq:NN [165](#), [10671](#), [10675](#)
- \fp_gset_from_dim:cn [10621](#)
- \fp_gset_from_dim:Nn [166](#), [10621](#), [10623](#), [10668](#)
- \fp_gsin:cn [11994](#)
- \fp_gsin:Nn [170](#), [11994](#), [11995](#), [11997](#)
- \fp_gsub:cn [11311](#)
- \fp_gsub:Nn [169](#), [11311](#), [11312](#), [11314](#)
- \fp_gtan:cn [12292](#)
- \fp_gtan:Nn [170](#), [12292](#), [12293](#), [12295](#)
- \fp_gzero:c [10575](#)
- \fp_gzero:N [166](#), [10575](#), [10577](#), [10580](#), [10584](#)
- \fp_gzero_new:c [10581](#)
- \fp_gzero_new:N . [166](#), [10581](#), [10583](#), [10586](#)
- \fp_if_undefined:N [13341](#), [13341](#)
- \fp_if_undefined:NTF [168](#)
- \fp_if_zero:N [13349](#), [13349](#)
- \fp_if_zero:NTF [168](#)
- \fp_level_input_exponents: [10504](#), [10504](#), [11227](#)
- \fp_level_input_exponents_a: [10504](#), [10507](#), [10512](#), [10519](#)
- \fp_level_input_exponents_a:NNNNNNNN [10504](#), [10517](#), [10522](#)
- \fp_level_input_exponents_b: [10504](#), [10509](#), [10537](#), [10544](#)
- \fp_level_input_exponents_b:NNNNNNNN [10504](#), [10542](#), [10547](#)
- \fp_ln:cn [12803](#)
- \fp_ln:Nn [170](#), [12803](#), [12803](#), [12805](#)
- \fp_ln_aux: [12803](#), [12821](#), [12840](#)
- \fp_ln_aux:NNn [12803](#), [12803](#), [12804](#), [12807](#)
- \fp_ln_const:nn [12920](#), [12930](#), [12981](#), [13020](#)
- \fp_ln_error_msg: [12828](#), [12836](#), [13629](#), [13632](#)
- \fp_ln_exponent: [12803](#), [12855](#), [12864](#)
- \fp_ln_exponent_tens: [12803](#)
- \fp_ln_exponent_tens:NN [12907](#), [12917](#)
- \fp_ln_exponent_units: [12803](#), [12915](#), [12927](#)
- \fp_ln_fixed: [12803](#), [13035](#), [13080](#), [13087](#)
- \fp_ln_fixed_aux:NNNNNNNN [12803](#), [13085](#), [13090](#)
- \fp_ln_integer_const:nn [12803](#)
- \fp_ln_internal: [12803](#), [12866](#), [12898](#), [13300](#)
- \fp_ln_mantissa: [12803](#), [12939](#), [12975](#)
- \fp_ln_mantissa_aux: [12803](#), [12979](#), [13000](#), [13005](#)
- \fp_ln_mantissa_divide_two: [12803](#), [13004](#), [13008](#)
- \fp_ln_normalise: [12803](#), [12931](#), [12941](#), [12948](#), [12982](#), [13068](#)
- \fp_ln_normalise_aux:NNNNNNNN [12946](#), [12953](#)
- \fp_ln_nornalise_aux:NNNNNNNN [12803](#)
- \fp_ln_Taylor: [12803](#), [12997](#), [13026](#)
- \fp_ln_Taylor_aux: [12803](#), [13048](#), [13105](#), [13134](#)

- \fp_mul:cn [11329](#)
- \fp_mul:Nn [169](#),
6826, 6836, 6837, 6847, 6848, 7513,
7518, 7547, 7722, 7795, 7796, 7800,
7801, 7898, 7901, [11329](#), [11329](#), [11331](#)
- \fp_mul:NNNNNN .. [11633](#), [11633](#), [12194](#),
[12241](#), [12245](#), [12741](#), [13043](#), [13108](#)
- \fp_mul:NNNNNNNN [11677](#),
[11677](#), [12629](#), [12649](#), [12704](#), [13317](#)
- \fp_mul_aux:NNn [11329](#), [11329](#), [11330](#), [11333](#)
- \fp_mul_end_level:
.... [11329](#), [11400](#), [11404](#), [11407](#),
[11411](#), [11431](#), [11657](#), [11662](#), [11666](#),
[11671](#), [11673](#), [11674](#), [11703](#), [11710](#),
[11716](#), [11723](#), [11727](#), [11730](#), [11734](#)
- \fp_mul_end_level:NNNNNNNN
..... [11329](#), [11435](#), [11437](#)
- \fp_mul_internal: .. [11329](#), [11343](#), [11384](#)
- \fp_mul_product:NN [11392](#)–
[11394](#), [11396](#)–[11399](#), [11401](#)–[11403](#),
[11405](#), [11406](#), [11410](#), [11426](#), [11645](#)–
[11650](#), [11652](#)–[11656](#), [11658](#)–[11661](#),
[11663](#)–[11665](#), [11669](#), [11670](#), [11672](#),
[11689](#)–[11694](#), [11696](#)–[11702](#), [11704](#)–
[11709](#), [11711](#)–[11715](#), [11719](#)–[11722](#),
[11724](#)–[11726](#), [11728](#), [11729](#), [11733](#)
- \fp_mul_split:NNNN [11329](#), [11386](#),
[11388](#), [11414](#), [11635](#), [11637](#), [11639](#),
[11641](#), [11679](#), [11681](#), [11683](#), [11685](#)
- \fp_mul_split:w [11329](#)
- \fp_mul_split_aux:w [11417](#), [11423](#)
- \fp_neg:c [11182](#)
- \fp_neg:N [169](#), [11182](#), [11182](#), [11184](#)
- \fp_neg:NN [11182](#)
- \fp_neg_aux:NN [11182](#), [11183](#), [11186](#)
- \fp_new:c [10563](#)
- \fp_new:N . [165](#), [6754](#)–[6756](#), [6766](#)–[6770](#),
[6896](#), [6897](#), [7114](#), [7133](#)–[7137](#), [7143](#),
[7144](#), [7149](#)–[7152](#), [7847](#), [7848](#), [10563](#),
[10563](#), [10568](#), [10571](#), [10582](#), [10584](#)
- \fp_overflow_msg:
..... [10432](#), [10499](#), [13613](#), [13619](#)
- \fp_pow:cn [13137](#)
- \fp_pow:Nn [169](#), [13137](#), [13137](#), [13139](#)
- \fp_pow_aux:NNn [13137](#), [13137](#), [13138](#), [13141](#)
- \fp_pow_aux_i: [13137](#), [13187](#), [13192](#)
- \fp_pow_aux_ii: [13137](#), [13196](#), [13210](#), [13228](#)
- \fp_pow_aux_iii: ... [13137](#), [13253](#), [13282](#)
- \fp_pow_aux_iv: [13137](#), [13231](#),
[13245](#), [13259](#), [13263](#), [13285](#), [13294](#)
- \fp_pow_negative: [13137](#)
- \fp_pow_positive: [13137](#)
- \fp_read:N [10350](#), [10350](#), [11047](#),
[11082](#), [11164](#), [11189](#), [11215](#), [11318](#),
[11336](#), [11449](#), [13144](#), [13377](#), [13382](#)
- \fp_read_aux:w [10350](#), [10351](#), [10352](#)
- \fp_round: ... [11050](#), [11086](#), [11109](#), [11109](#)
- \fp_round_aux:NNNNNNNN
..... [11109](#), [11116](#), [11118](#)
- \fp_round_figures:cn [11038](#)
- \fp_round_figures:Nn
..... [167](#), [11038](#), [11038](#), [11040](#)
- \fp_round_figures_aux:NNn
..... [11038](#), [11039](#), [11042](#), [11044](#)
- \fp_round_loop:N [11109](#), [11120](#), [11131](#), [11154](#)
- \fp_round_places:cn [11073](#)
- \fp_round_places:Nn
..... [167](#), [11073](#), [11073](#), [11075](#)
- \fp_round_places_aux:NNn
..... [11073](#), [11074](#), [11077](#), [11079](#)
- \fp_set:cn [10587](#)
- \fp_set:Nn [166](#), [6776](#), [6991](#), [6992](#),
[7720](#), [7883](#), [7884](#), [10587](#), [10587](#), [10619](#)
- \fp_set_aux:NNn [10587](#), [10587](#)–[10589](#)
- \fp_set_eq:cc [10671](#), [10674](#)
- \fp_set_eq:cN [10671](#), [10672](#)
- \fp_set_eq:Nc [10671](#), [10673](#)
- \fp_set_eq:NN [165](#), [6824](#),
[6834](#), [6835](#), [6845](#), [6846](#), [6958](#), [6978](#),
[7793](#), [7794](#), [7798](#), [7799](#), [10671](#), [10671](#)
- \fp_set_from_dim:cn [10621](#)
- \fp_set_from_dim:Nn [166](#), [6832](#),
[6833](#), [6843](#), [6844](#), [6907](#), [6908](#), [6910](#),
[6911](#), [6954](#), [6955](#), [6975](#), [6976](#), [7498](#)–
[7501](#), [7508](#), [7509](#), [7512](#), [7517](#), [7540](#)–
[7544](#), [7791](#), [7792](#), [7855](#), [7856](#), [7858](#),
[7859](#), [7897](#), [7900](#), [10621](#), [10621](#), [10667](#)
- \fp_set_from_dim_aux:NNn
..... [10621](#), [10622](#), [10624](#), [10625](#)
- \fp_set_from_dim_aux:w
.. [10621](#), [10632](#), [10661](#), [10663](#), [10666](#)
- \fp_show:c [10679](#), [10680](#)
- \fp_show:N [166](#), [10679](#), [10679](#)
- \fp_sin:cn [11994](#)
- \fp_sin:Nn
.. [170](#), [6827](#), [7723](#), [11994](#), [11994](#), [11996](#)
- \fp_sin_aux:NNn [11994](#), [11994](#), [11995](#), [11998](#)
- \fp_sin_aux_i: [11994](#), [12034](#), [12045](#)
- \fp_sin_aux_ii: [11994](#), [12048](#), [12078](#), [12392](#)

`\fp_split:Nn` [10363](#),
[10363](#), [10592](#), [10630](#), [11216](#), [11319](#),
[11337](#), [11450](#), [12001](#), [12098](#), [12299](#),
[12492](#), [12810](#), [13149](#), [13360](#), [13366](#)
`\fp_split_aux_i:w` .. [10363](#), [10402](#), [10406](#)
`\fp_split_aux_ii:w` .. [10363](#), [10407](#), [10408](#)
`\fp_split_aux_iii:w` . [10363](#), [10409](#), [10410](#)
`\fp_split_decimal:w` . [10363](#), [10413](#), [10416](#)
`\fp_split_decimal_aux:w`
..... [10363](#), [10417](#), [10418](#)
`\fp_split_exponent:` [10363](#)
`\fp_split_exponent:w` [10371](#), [10398](#)
`\fp_split_sign:`
.. [10363](#), [10369](#), [10373](#), [10384](#), [10394](#)
`\fp_standardise:NNNN` [10435](#),
[10435](#), [10593](#), [10635](#), [11217](#), [11237](#),
[11320](#), [11338](#), [11348](#), [11451](#), [11491](#),
[12002](#), [12056](#), [12099](#), [12142](#), [12300](#),
[12387](#), [12396](#), [12423](#), [12493](#), [12720](#),
[12811](#), [12876](#), [13150](#), [13361](#), [13367](#)
`\fp_standardise_aux:`
..... [10435](#), [10449](#), [10455](#),
[10465](#), [10466](#), [10472](#), [10489](#), [10502](#)
`\fp_standardise_aux:NNNN`
..... [10435](#), [10443](#), [10447](#)
`\fp_standardise_aux:w` [10435](#),
[10453](#), [10459](#), [10471](#), [10476](#), [10503](#)
`\fp_sub:cn` [11311](#)
`\fp_sub:Nn` [169](#), [6838](#), [7510](#), [7520](#),
[7522](#), [7545](#), [7797](#), [11311](#), [11311](#), [11313](#)
`\fp_sub:NNNNNNNN` [11610](#), [11610](#),
[11949](#), [11970](#), [11981](#), [12986](#), [13072](#)
`\fp_sub_aux:NNn` [11311](#), [11311](#), [11312](#), [11315](#)
`\fp_tan:cn` [12292](#)
`\fp_tan:Nn` [170](#), [12292](#), [12292](#), [12294](#)
`\fp_tan_aux:NNn` [12292](#), [12292](#), [12293](#), [12296](#)
`\fp_tan_aux_i:` [12292](#), [12332](#), [12343](#)
`\fp_tan_aux_ii:` [12292](#), [12346](#), [12353](#)
`\fp_tan_aux_iii:` [12292](#), [12376](#), [12379](#), [12382](#)
`\fp_tan_aux_iv:` [12292](#), [12406](#), [12409](#), [12412](#)
`\fp_tmp:w` [10562](#),
[10562](#), [10599](#), [10617](#), [10641](#), [10659](#),
[11053](#), [11071](#), [11089](#), [11107](#), [11166](#),
[11180](#), [11223](#), [11242](#), [11327](#), [11353](#),
[11382](#), [11460](#), [11470](#), [11479](#), [11496](#),
[12023](#), [12036](#), [12043](#), [12123](#), [12129](#),
[12321](#), [12334](#), [12341](#), [12518](#), [12531](#),
[12823](#), [12831](#), [12838](#), [12857](#), [13049](#),
[13060](#), [13163](#), [13169](#), [13180](#), [13190](#),
[13213](#), [13220](#), [13266](#), [13301](#), [13315](#)
`\fp_to_dim:c` [10744](#)
`\fp_to_dim:N`
[167](#), [6839](#), [6850](#), [7527](#), [7549](#), [7803](#),
[7804](#), [7899](#), [7902](#), [10744](#), [10744](#), [10745](#)
`\fp_to_int:c` [10746](#)
`\fp_to_int:N` [167](#), [10746](#), [10746](#), [10748](#)
`\fp_to_int_aux:w` [10746](#), [10747](#), [10749](#)
`\fp_to_int_large:w` .. [10746](#), [10757](#), [10772](#)
`\fp_to_int_large_aux:nnn`
[10746](#), [10784](#), [10786](#), [10788](#), [10790](#),
[10792](#), [10794](#), [10796](#), [10798](#), [10800](#)
`\fp_to_int_large_aux_1:w` [10746](#)
`\fp_to_int_large_aux_2:w` [10746](#)
`\fp_to_int_large_aux_3:w` [10746](#)
`\fp_to_int_large_aux_4:w` [10746](#)
`\fp_to_int_large_aux_5:w` [10746](#)
`\fp_to_int_large_aux_6:w` [10746](#)
`\fp_to_int_large_aux_7:w` [10746](#)
`\fp_to_int_large_aux_8:w` [10746](#)
`\fp_to_int_large_aux_i:w`
..... [10746](#), [10775](#), [10781](#)
`\fp_to_int_large_aux_ii:w`
..... [10746](#), [10777](#), [10808](#)
`\fp_to_int_none:w` [10746](#)
`\fp_to_int_small:w` .. [10746](#), [10755](#), [10761](#)
`\fp_to_tl:c` [10813](#)
`\fp_to_tl:N` [167](#), [10813](#), [10813](#), [10815](#)
`\fp_to_tl_aux:w` [10813](#), [10814](#), [10816](#)
`\fp_to_tl_large:w` .. [10813](#), [10824](#), [10828](#)
`\fp_to_tl_large_0:w` [10813](#)
`\fp_to_tl_large_1:w` [10813](#)
`\fp_to_tl_large_2:w` [10813](#)
`\fp_to_tl_large_3:w` [10813](#)
`\fp_to_tl_large_4:w` [10813](#)
`\fp_to_tl_large_5:w` [10813](#)
`\fp_to_tl_large_6:w` [10813](#)
`\fp_to_tl_large_7:w` [10813](#)
`\fp_to_tl_large_8:w` [10813](#)
`\fp_to_tl_large_8_aux:w` [10813](#)
`\fp_to_tl_large_9:w` [10813](#)
`\fp_to_tl_large_aux_i:w`
..... [10813](#), [10831](#), [10837](#)
`\fp_to_tl_large_aux_ii:w`
..... [10813](#), [10833](#), [10839](#)
`\fp_to_tl_large_zeros:NNNNNNNN`
..... [10813](#), [10842](#), [10848](#),
[10853](#), [10858](#), [10863](#), [10868](#), [10873](#),
[10878](#), [10883](#), [10893](#), [10951](#), [10954](#)
`\fp_to_tl_small:w` .. [10813](#), [10822](#), [10896](#)
`\fp_to_tl_small_aux:w` [10813](#), [10904](#), [10948](#)

- \fp_to_tl_small_one:w [10813](#), [10899](#), [10909](#)
 - \fp_to_tl_small_two:w [10813](#), [10902](#), [10928](#)
 - \fp_to_tl_small_zeros:NNNNNNNNN
 [10813](#),
 [10916](#), [10925](#), [10935](#), [10945](#), [10993](#)
 - \fp_trig_calc_cos: [12084](#),
 [12086](#), [12168](#), [12174](#), [12187](#), [12187](#)
 - \fp_trig_calc_sin: [12082](#),
 [12088](#), [12170](#), [12172](#), [12187](#), [12223](#)
 - \fp_trig_calc_Taylor:
 [12187](#), [12220](#), [12235](#), [12238](#), [12289](#)
 - \fp_trig_normalise:
 [11863](#), [11863](#), [12047](#), [12133](#), [12355](#)
 - \fp_trig_normalise_aux:
 [11863](#), [11868](#), [11882](#), [11887](#), [11895](#)
 - \fp_trig_octant: [11873](#), [11939](#), [11939](#)
 - \fp_trig_octant_aux_i:
 [11939](#), [11942](#), [11957](#), [11977](#), [11990](#)
 - \fp_trig_octant_aux_ii:
 [11939](#), [11964](#), [11967](#)
 - \fp_trig_overflow_msg:
 [11879](#), [12350](#), [13635](#), [13641](#)
 - \fp_trig_sub:NNN [11863](#), [11885](#), [11891](#), [11900](#)
 - \fp_use:c [10681](#)
 - \fp_use:N [166](#), [6933](#), [6935](#),
 [6939](#), [6941](#), [10681](#), [10681](#), [10683](#), [10744](#)
 - \fp_use_aux:w [10681](#), [10682](#), [10684](#)
 - \fp_use_i_to_iix:NNNNNNNNN
 [10813](#), [10913](#), [10918](#), [11036](#)
 - \fp_use_i_to_vii:NNNNNNNNN
 [10813](#), [10932](#), [10937](#), [11034](#)
 - \fp_use_iix_ix:NNNNNNNNN
 [10813](#), [10930](#), [11032](#)
 - \fp_use_ix:NNNNNNNNN [10813](#), [10911](#), [11033](#)
 - \fp_use_large:w [10681](#), [10690](#), [10708](#)
 - \fp_use_large_aux_1:w [10681](#)
 - \fp_use_large_aux_2:w [10681](#)
 - \fp_use_large_aux_3:w [10681](#)
 - \fp_use_large_aux_4:w [10681](#)
 - \fp_use_large_aux_5:w [10681](#)
 - \fp_use_large_aux_6:w [10681](#)
 - \fp_use_large_aux_7:w [10681](#)
 - \fp_use_large_aux_8:w [10681](#)
 - \fp_use_large_aux_i:w [10681](#), [10711](#), [10717](#)
 - \fp_use_large_aux_ii:w [10681](#), [10713](#), [10738](#)
 - \fp_use_none:w [10681](#), [10696](#), [10701](#)
 - \fp_use_small:w [10681](#), [10694](#), [10702](#)
 - \fp_zero:c [10575](#)
 - \fp_zero:N [166](#), [10575](#), [10575](#), [10579](#), [10582](#)
 - \fp_zero_new:c [10581](#)
 - \fp_zero_new:N [166](#), [10581](#), [10581](#), [10585](#)
 - \frozen@everydisplay [767](#)
 - \frozen@everymath [768](#)
 - \futurelet [361](#)
- G**
- \g [2276](#)
 - \g_cctab_allocate_int [13679](#),
 [13679](#), [13680](#), [13686](#), [13688](#), [13690](#)
 - \g_cctab_stack_int [13679](#), [13681](#),
 [13718](#), [13719](#), [13721](#), [13722](#), [13726](#)
 - \g_cctab_stack_seq
 [13679](#), [13682](#), [13716](#), [13727](#), [13729](#)
 - \g_file_current_name_tl
 [162](#), [10103](#), [10103](#), [10108](#),
 [10112](#), [10120](#), [10199](#), [10200](#), [10202](#)
 - \g_file_internal_ior
 [10131](#), [10131](#), [10146](#),
 [10147](#), [10150](#), [10168](#), [10169](#), [10179](#)
 - \g_file_record_seq [164](#), [10115](#), [10115](#),
 [10120](#), [10194](#), [10261](#), [10263](#), [10270](#)
 - \g_file_stack_seq
 [163](#), [10114](#), [10114](#), [10199](#), [10202](#)
 - \g_ior_internal_ior
 [8348](#), [8349](#), [8395](#), [8396](#), [8398](#)
 - \g_ior_streams_prop [8250](#),
 [8251](#), [8256](#), [8315](#), [8415](#), [8430](#), [8451](#)
 - \g_iow_internal_iow
 [8348](#), [8348](#), [8359](#), [8360](#), [8362](#)
 - \g_iow_streams_prop
 [8250](#), [8250](#), [8253](#)–[8255](#),
 [8328](#), [8334](#), [8342](#), [8379](#), [8443](#), [8453](#)
 - \g_keyval_level_int [9454](#), [9454](#),
 [9501](#), [9523](#), [9547](#), [9549](#), [9551](#), [9553](#)
 - \g_peek_token [56](#), [2960](#), [2961](#), [2971](#)
 - \g_prg_map_int [2242](#), [2248](#),
 [2258](#), [2260](#), [2308](#), [2308](#), [4693](#), [4694](#),
 [4697](#), [4699](#), [5486](#), [5488](#), [5492](#), [5494](#),
 [6021](#), [6022](#), [6024](#), [6026](#), [6432](#), [6433](#),
 [6436](#), [6439](#), [8703](#), [8704](#), [8711](#), [8715](#)
 - \g_scan_marks_tl [2494](#), [2494](#), [2497](#), [2503](#)
 - \g_tmpa_bool [36](#), [1928](#), [1929](#)
 - \g_tmpa_clist [113](#), [6077](#), [6079](#)
 - \g_tmpa_dim [77](#), [4222](#), [4225](#)
 - \g_tmpa_int [70](#), [3972](#), [3975](#)
 - \g_tmpa_skip [79](#), [4304](#), [4307](#)
 - \g_tmpa_tl [95](#), [4956](#), [4956](#)
 - \g_tmpb_clist [113](#), [6077](#), [6080](#)
 - \g_tmpb_dim [77](#), [4222](#), [4226](#)
 - \g_tmpb_int [70](#), [3972](#), [3976](#)

- \g_tmpb_skip 79, [4304](#), 4308
 - \g_tmpb_tl 95, [4956](#), 4957
 - \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: 1940,
[2270](#), [2270](#), [2992](#), [3010](#), [4548](#), [4983](#)
 - \group_align_safe_end 41
 - \group_align_safe_end:
..... [2037](#), [2038](#), [2270](#), [2272](#),
[2974](#), [2984](#), [2989](#), [3007](#), [4557](#), [5009](#)
 - \group_begin 9
 - \group_begin: [810](#), [811](#), [832](#), [1052](#),
[1827](#), [2275](#), [2290](#), [2613](#), [2626](#), [2633](#),
[2670](#), [2723](#), [2760](#), [2902](#), [3068](#), [3165](#),
[3172](#), [4482](#), [4500](#), [4649](#), [4930](#), [5352](#),
[6660](#), [6775](#), [6902](#), [6949](#), [6970](#), [6990](#),
[8206](#), [8282](#), [8295](#), [8505](#), [8510](#), [8539](#),
[8833](#), [8874](#), [9305](#), [9459](#), [9469](#), [10134](#),
[10207](#), [10591](#), [10627](#), [11046](#), [11081](#),
[11163](#), [11188](#), [11214](#), [11317](#), [11335](#),
[11448](#), [12000](#), [12097](#), [12298](#), [12491](#),
[12809](#), [13028](#), [13143](#), [13200](#), [13234](#),
[13296](#), [13359](#), [13376](#), [13563](#), [13755](#)
 - \group_end 9
 - \group_end: [810](#),
[812](#), [832](#), [1057](#), [1839](#), [2280](#), [2295](#),
[2625](#), [2629](#), [2642](#), [2677](#), [2731](#), [2770](#),
[2908](#), [3133](#), [3174](#), [3183](#), [4489](#), [4507](#),
[4653](#), [4656](#), [4935](#), [5363](#), [6665](#), [6785](#),
[6921](#), [6962](#), [6982](#), [7004](#), [8210](#), [8289](#),
[8302](#), [8509](#), [8529](#), [8573](#), [8838](#), [8880](#),
[9327](#), [9466](#), [9477](#), [10141](#), [10214](#),
[10601](#), [10643](#), [11055](#), [11091](#), [11168](#),
[11205](#), [11244](#), [11355](#), [11462](#), [11472](#),
[11498](#), [12025](#), [12038](#), [12125](#), [12323](#),
[12336](#), [12520](#), [12825](#), [12833](#), [12859](#),
[13051](#), [13165](#), [13171](#), [13182](#), [13206](#),
[13212](#), [13215](#), [13222](#), [13239](#), [13247](#),
[13256](#), [13268](#), [13303](#), [13390](#), [13401](#),
[13404](#), [13408](#), [13412](#), [13416](#), [13428](#),
[13432](#), [13435](#), [13444](#), [13447](#), [13458](#),
[13462](#), [13476](#), [13480](#), [13484](#), [13489](#),
[13494](#), [13497](#), [13508](#), [13512](#), [13516](#),
[13521](#), [13526](#), [13529](#), [13566](#), [13758](#)
 - \group_execute_after:N 1498
 - \group_insert_after:N . . . 9, [815](#), [815](#), 1498
- ## H
- \halign 378
 - \hangafter 556
 - \hangindent 557
 - \hbadness 618
 - \hbox 613
 - \hbox:n . . . [126](#), [6671](#), [6671](#), [6809](#), [7991](#), [8046](#)
 - \hbox_gset:cn [6672](#)
 - \hbox_gset:cw [6682](#), [6694](#)
 - \hbox_gset:Nn [127](#), [6672](#), [6673](#), [6675](#)
 - \hbox_gset:Nw . . . [127](#), [6682](#), [6684](#), [6687](#), [6693](#)
 - \hbox_gset_end [127](#)
 - \hbox_gset_end: [6682](#), [6689](#), [6695](#)
 - \hbox_gset_inline_begin:c ... [6690](#), [6694](#)
 - \hbox_gset_inline_begin:N ... [6690](#), [6693](#)
 - \hbox_gset_inline_end: [6690](#), [6695](#)
 - \hbox_gset_to_wd:cnn [6676](#)
 - \hbox_gset_to_wd:Nnn [127](#), [6676](#), [6678](#), [6681](#)
 - \hbox_overlap_left:n ... [127](#), [6699](#), [6699](#)
 - \hbox_overlap_right:n [127](#), [6699](#), [6701](#), [7029](#)
 - \hbox_set:cn [6672](#)
 - \hbox_set:cw [6682](#), [6691](#)
 - \hbox_set:Nn [127](#),
[6672](#), [6672](#)–[6674](#), [6773](#), [6805](#), [6806](#),
[6900](#), [6947](#), [6968](#), [6988](#), [7026](#), [7049](#),
[7054](#), [7062](#), [7072](#), [7084](#), [7091](#), [7098](#),
[7201](#), [7298](#), [7555](#), [7626](#), [7735](#), [8136](#)
 - \hbox_set:Nw
[127](#), [6682](#), [6682](#), [6685](#), [6686](#), [6690](#), [7245](#)
 - \hbox_set_end [127](#)
 - \hbox_set_end: ... [6682](#), [6688](#), [6692](#), [7249](#)
 - \hbox_set_inline_begin:c ... [6690](#), [6691](#)
 - \hbox_set_inline_begin:N ... [6690](#), [6690](#)
 - \hbox_set_inline_end: [6690](#), [6692](#)
 - \hbox_set_to_wd:cnn [6676](#)
 - \hbox_set_to_wd:Nnn
..... [127](#), [6676](#), [6676](#), [6679](#), [6680](#)
 - \hbox_to_wd:nn ... [127](#), [6696](#), [6696](#), [7036](#)

- \hbox_to_zero:n [127](#), [6696](#), [6698](#), [6700](#), [6702](#)
 - \hbox_unpack:c [6703](#)
 - \hbox_unpack:N
 - ... [127](#), [6703](#), [6703](#), [6705](#), [7559](#), [7708](#)
 - \hbox_unpack_clear:c [6703](#)
 - \hbox_unpack_clear:N [128](#), [6703](#), [6704](#), [6706](#)
 - \hcoffin_set:cn [7197](#)
 - \hcoffin_set:cw [7241](#)
 - \hcoffin_set:Nn [131](#), [7197](#),
 - [7197](#), [7213](#), [7988](#), [8000](#), [8043](#), [8083](#)
 - \hcoffin_set:Nw ... [131](#), [7241](#), [7241](#), [7257](#)
 - \hcoffin_set_end [131](#)
 - \hcoffin_set_end: [7241](#), [7246](#), [7256](#)
 - \Height [7338](#), [7340](#), [7344](#), [7348](#), [7355](#)
 - \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**
- \if [184](#), [387](#)
 - \if:w [23](#), [785](#),
 - [791](#), [961](#), [1029](#), [1046](#), [1786](#), [2945](#),
[3050](#), [3466](#), [10354](#), [10686](#), [10751](#), [10818](#)
 - \if_bool:N [42](#), [1886](#), [1886](#)
 - \if_box_empty:N ... [130](#), [6608](#), [6610](#), [6624](#)
 - \if_case:w [71](#),
 - [1297](#), [3301](#), [3306](#), [3726](#), [12080](#), [12166](#)
 - \if_catcode:w [23](#), [785](#), [793](#),
 - [2652](#), [2657](#), [2662](#), [2667](#), [2674](#), [2680](#),
[2685](#), [2690](#), [2695](#), [2700](#), [2705](#), [2715](#),
[2748](#), [3019](#), [4847](#), [4885](#), [4903](#), [8665](#)
 - \if_charcode:w
 - . [23](#), [785](#), [792](#), [2720](#), [4825](#), [4831](#), [4878](#)
 - \if_cs_exist:N [23](#), [799](#), [799](#), [1086](#), [1114](#), [2954](#)
 - \if_cs_exist:w [799](#),
 - [800](#), [823](#), [1095](#), [1123](#), [1270](#), [12029](#),
[12119](#), [12327](#), [12514](#), [12523](#), [12853](#)
 - \if_dim:w
 - [82](#), [4028](#), [4028](#), [4088](#), [4100](#), [4125](#)–[4131](#)
 - \if_eof:w [141](#), [8226](#), [8226](#), [8679](#)
 - \if_false [23](#)
 - \if_false: [785](#), [786](#), [2271](#),
 - [4918](#), [4926](#), [5329](#), [5332](#), [5435](#), [5439](#)
 - \if_hbox:N [130](#), [6608](#), [6608](#), [6612](#)
 - \if_int_compare:w
 - . . [71](#), [813](#), [813](#), [1457](#), [1463](#), [2271](#),
[2273](#), [2421](#), [2428](#), [2445](#), [2472](#), [2481](#),
[2739](#), [3301](#), [3316](#), [3324](#), [3335](#), [3346](#),
[3350](#), [3351](#), [3357](#), [3476](#), [3484](#), [3492](#),
[3500](#), [3508](#), [3516](#), [3524](#), [3532](#), [4268](#),
[4962](#), [8676](#), [10375](#), [10386](#), [10421](#),
[10429](#), [10437](#), [10451](#), [10468](#), [10490](#),
[10491](#), [10506](#), [10514](#), [10539](#), [10604](#),
[10646](#), [10689](#), [10692](#), [10710](#), [10754](#),
[10763](#), [10765](#), [10774](#), [10802](#), [10821](#),
[10830](#), [10898](#), [10901](#), [10911](#), [10912](#),
[10930](#), [10931](#), [10956](#)–[10964](#), [10995](#)–
[11003](#), [11049](#), [11058](#), [11085](#), [11094](#),
[11124](#), [11133](#), [11137](#), [11146](#), [11147](#),
[11153](#), [11193](#), [11228](#), [11247](#), [11273](#),
[11289](#), [11293](#), [11295](#), [11358](#), [11362](#),
[11456](#), [11466](#), [11501](#), [11505](#), [11536](#),
[11539](#), [11547](#), [11550](#), [11552](#), [11567](#),
[11599](#), [11604](#), [11615](#), [11619](#), [11623](#),
[11624](#), [11749](#), [11761](#), [11789](#), [11800](#),
[11822](#), [11865](#), [11869](#), [11884](#), [11889](#),
[11890](#), [11908](#), [11912](#), [11916](#), [11918](#),
[11943](#), [11959](#), [11969](#), [11979](#), [12009](#),
[12022](#), [12049](#), [12064](#), [12106](#), [12135](#),
[12150](#), [12176](#), [12177](#), [12181](#), [12189](#),
[12203](#), [12204](#), [12226](#), [12259](#), [12260](#),
[12264](#), [12270](#), [12280](#), [12284](#), [12307](#),
[12320](#), [12345](#), [12356](#), [12357](#), [12363](#),
[12370](#), [12371](#), [12401](#), [12402](#), [12431](#),
[12501](#), [12535](#), [12537](#), [12538](#), [12548](#),
[12563](#), [12575](#), [12591](#), [12592](#), [12614](#),
[12624](#), [12642](#), [12643](#), [12675](#), [12676](#),
[12682](#), [12711](#), [12714](#), [12749](#), [12753](#),
[12757](#), [12763](#), [12773](#), [12777](#), [12816](#),
[12817](#), [12867](#), [12870](#), [12883](#), [12900](#),
[12906](#), [12929](#), [12943](#), [12955](#), [12980](#),
[12983](#), [12994](#), [13002](#), [13062](#), [13069](#),
[13082](#), [13092](#), [13112](#), [13122](#), [13128](#),
[13155](#), [13159](#), [13176](#), [13194](#), [13199](#),
[13202](#), [13230](#), [13233](#), [13237](#), [13238](#),
[13396](#)–[13399](#), [13422](#), [13425](#), [13431](#),
[13440](#), [13443](#), [13457](#), [13461](#), [13465](#),
[13475](#), [13479](#), [13483](#), [13487](#), [13492](#),
[13507](#), [13511](#), [13515](#), [13519](#), [13524](#)

<code>\if_int_odd:w</code>	71, 3301, 3305, 3540, 3548, 11947, 13010, 13013
<code>\if_meaning:w</code>	23, 785, 794, 1066, 1083, 1101, 1111, 1129, 1281, 1394, 1557, 1558, 1866, 1918, 1948, 1958, 1967, 1970, 1980, 1983, 1996, 2005, 2407, 2413, 2439, 2452, 2460, 2710, 2755, 2792, 2795, 2814, 2817, 2834, 2837, 2852, 2855, 2928, 3028, 3473, 4120, 4602, 4614, 4626, 4637, 4652, 4870, 5001, 5357, 5456, 5551, 6356, 6382, 6422, 6484, 13343, 13351
<code>\if_mode_horizontal</code>	23
<code>\if_mode_horizontal:</code>	795, 796, 2265
<code>\if_mode_inner</code>	23
<code>\if_mode_inner:</code>	795, 798, 2267
<code>\if_mode_math</code>	23
<code>\if_mode_math:</code>	795, 795, 2269
<code>\if_mode_vertical</code>	23
<code>\if_mode_vertical:</code>	795, 797, 2263
<code>\if_num:w</code>	71, 2936, 3301, 3304
<code>\if_predicate:w</code>	42, 1886, 1887, 1932
<code>\if_true</code>	23
<code>\if_true:</code>	785, 785
<code>\if_vbox:N</code>	130, 6608, 6609, 6614
<code>\ifcase</code>	388
<code>\ifcat</code>	389
<code>\ifcsname</code>	672
<code>\ifdefined</code>	671
<code>\ifdim</code>	392
<code>\ifeof</code>	393
<code>\iffalse</code>	398
<code>\iffontchar</code>	701
<code>\ifhbox</code>	394
<code>\ifhmode</code>	400
<code>\ifinner</code>	403
<code>\ifmmode</code>	401
<code>\ifnum</code>	390
<code>\ifodd</code>	169, 205, 391
<code>\iftrue</code>	399
<code>\ifvbox</code>	395
<code>\ifvmode</code>	402
<code>\ifvoid</code>	396
<code>\ifx</code>	13, 62, 108, 135, 229, 233, 397
<code>\ignorespaces</code>	445
<code>\immediate</code>	407
<code>\indent</code>	541
<code>\initcatcodetable</code>	760
<code>\input</code>	415
<code>\input@path</code>	10158, 10161, 10176
<code>\inputlineno</code>	417
<code>\insert</code>	597
<code>\insertpenalties</code>	600
<code>\int_abs:n</code>	61, 3313, 3313
<code>\int_add:cn</code>	3429
<code>\int_add:Nn</code>	63, 3429, 3429, 3434, 3437, 8587, 8600, 8638
<code>\int_compare:n</code>	3460, 3460
<code>\int_compare:nF</code>	3564, 3579
<code>\int_compare:nNn</code>	3530, 3530
<code>\int_compare:nNnF</code>	2231, 3592, 3607, 8426, 8428, 8439, 8441
<code>\int_compare:nNnT</code> 3584, 3601, 4203, 5146, 5633, 8357, 8375, 8393, 8411, 13719
<code>\int_compare:nNnTF</code>	64, 2065, 2117, 2217, 2220, 3385, 3387, 3613, 3699, 3705, 3852, 3876, 3880, 3930, 5158, 5646, 6116, 6118, 6123, 6131, 6151, 8311, 8324, 8588, 13687
<code>\int_compare:nT</code>	3556, 3573
<code>\int_compare:nTF</code>	64
<code>\int_compare:<:w</code>	3460
<code>\int_compare:=:w</code>	3460
<code>\int_compare:>:w</code>	3460
<code>\int_compare_aux:Nw</code>	3460, 3464, 3472
<code>\int_compare_aux:nw</code>	3460, 3461, 3462
<code>\int_compare_p:nNn</code>	4280, 4281
<code>\int_const:cn</code>	3383, 3889–3902
<code>\int_const:Nn</code>	62, 3383, 3383, 3403, 3953–3971, 10280–10284
<code>\int_constdef:Nw</code> .	3383, 3394, 3406, 3410
<code>\int_convert_from_base_ten:nn</code>	3977, 3978
<code>\int_convert_to_base_ten:nn</code>	3977, 3980
<code>\int_convert_to_symbols:nnn</code>	3977, 3979
<code>\int_decr:c</code>	3441
<code>\int_decr:N</code>	63, 3441, 3443, 3448, 3450
<code>\int_div_round:nn</code>	61, 3343, 3368
<code>\int_div_truncate:nn</code> 62, 3343, 3343, 3372, 3628, 3718
<code>\int_do_until:nn</code> ...	65, 3554, 3576, 3580
<code>\int_do_until:nNnn</code> ..	65, 3582, 3604, 3608
<code>\int_do_while:nn</code> ...	65, 3554, 3570, 3574
<code>\int_do_while:nNnn</code> ..	64, 3582, 3598, 3602
<code>\int_eval:n</code>	61, 1325, 2111, 2227, 2235, 3307, 3308, 3311, 3368, 3610, 3696, 3765, 3775, 3834, 3848, 3852, 3855, 3870, 3879, 4733, 4738, 5079, 5144, 5160, 5619, 5631, 5648, 6083, 6092, 6120, 6133, 6155

`\int_eval:w`
 . 71, 1297, 2173, 2517, 2519, 2521,
 2587, 2589, 2591, 2593, 2595, 2597,
 2599, 2601, 2603, 2605, 2607, 2609,
 3301, 3302, 3308, 3311, 3316, 3319,
 3323, 3325, 3334, 3336, 3345, 3346,
 3350, 3351, 3357, 3371, 3395, 3430,
 3432, 3454, 3461, 3476, 3484, 3492,
 3500, 3508, 3516, 3524, 3532, 3540,
 3548, 3726, 3753, 3908, 3952, 8658,
 10401, 10404, 10422, 10438, 10805,
 10913, 10917, 10932, 10936, 11135,
 11136, 11229, 11255, 11266, 11270,
 11282, 11286, 11299, 11303, 11345,
 11359, 11363, 11376, 11429, 11457,
 11467, 11488, 11502, 11506, 11519,
 11537, 11582, 11591, 11596–11598,
 11612–11614, 11624, 11628, 11629,
 11741, 11748, 11773, 11783, 11903,
 11905, 11907, 11919, 11925, 11929,
 11933, 12017, 12072, 12114, 12158,
 12315, 12420, 12439, 12509, 12588,
 12621, 12684, 12690, 12694, 12731,
 12750, 12818, 12848, 12891, 12995,
 13113, 13156, 13160, 13177, 13203,
 13275, 13325, 13422, 13425, 13440
`\int_eval_end` 71
`\int_eval_end:` 1297, 2173, 2517, 2519,
 2521, 2587, 2589, 2591, 2593, 2595,
 2597, 2599, 2601, 2603, 2605, 2607,
 2609, 3301, 3303, 3308, 3311, 3319,
 3325, 3330, 3336, 3341, 3366, 3373,
 3395, 3430, 3432, 3454, 3476, 3484,
 3492, 3500, 3508, 3516, 3524, 3532,
 3540, 3548, 3726, 3753, 3952, 8661,
 10805, 10919, 10938, 11521, 11585,
 11591, 11596–11598, 11612–11614,
 11628, 11629, 11741, 11748, 11903,
 11905, 11907, 11927, 11931, 11935,
 12422, 12590, 12623, 12686, 12696
`\int_from_alpha:n` 68, 3832, 3832
`\int_from_alpha_aux:N` ... 3832, 3848, 3851
`\int_from_alpha_aux:n` ... 3832, 3837, 3840
`\int_from_alpha_aux:nN`
 3832, 3841, 3842, 3847
`\int_from_base:nn`
 68, 3853, 3853, 3884, 3886, 3888, 3980
`\int_from_base_aux:N` ... 3853, 3870, 3874
`\int_from_base_aux:nn` .. 3853, 3858, 3862
`\int_from_base_aux:nnN`
 3853, 3863, 3864, 3869
`\int_from_binary:n` 68, 3883, 3883
`\int_from_hexadecimal:n` .. 68, 3883, 3885
`\int_from_octal:n` 68, 3883, 3887
`\int_from_roman:n` 68, 3903, 3903
`\int_from_roman_aux:NN`
 3903, 3909, 3912, 3937, 3941
`\int_from_roman_clean_up:w`
 3903, 3920, 3927, 3929, 3948
`\int_from_roman_end:w` .. 3903, 3907, 3946
`\int_gadd:cn` 3429
`\int_gadd:Nn`
 .. 63, 3429, 3433, 3438, 13686, 13718
`\int_gdecr:c` 3441
`\int_gdecr:N` 63, 2260, 3441, 3447, 3452,
 4699, 5492, 6026, 6439, 8715, 9553
`\int_get_digits:n` 70, 3798, 3803, 3837, 3859
`\int_get_sign:n` 70, 3798, 3798, 3836, 3857
`\int_get_sign_and_digits_aux:nNNN` ..
 3798, 3800, 3805, 3808, 3831
`\int_get_sign_and_digits_aux:oNNN` ..
 3798, 3814, 3818, 3824
`\int_gincr:c` 3441
`\int_gincr:N` 63, 2258, 3441, 3445, 3451,
 4693, 5488, 6021, 6432, 8711, 9547
`\int_gset:cn` 3453
`\int_gset:Nn` 63, 3390, 3400, 3453, 3455, 3457
`\int_gset_eq:cc` 3423
`\int_gset_eq:cN` 3423
`\int_gset_eq:Nc` 3423
`\int_gset_eq:NN` 62, 3423, 3426–3428
`\int_gsub:cn` 3429
`\int_gsub:Nn` .. 63, 3429, 3435, 3440, 13726
`\int_gzero:c` 3413
`\int_gzero:N` ... 62, 3413, 3414, 3416, 3420
`\int_gzero_new:c` 3417
`\int_gzero_new:N` ... 62, 3417, 3419, 3422
`\int_if_even:n` 3538, 3546
`\int_if_even:nTF` 64
`\int_if_odd:n` 3538, 3538
`\int_if_odd:nTF` 64
`\int_incr:c` 3441
`\int_incr:N` 63, 3441, 3441, 3446, 3449,
 8374, 8410, 8606, 9690, 9717, 9784
`\int_max:nn` 62, 3313, 3321
`\int_min:nn` 62, 3313, 3332
`\int_mod:nn` 62, 3343, 3369, 3618, 3709
`\int_new:c` 3375

- \int_new:N 62, 2308, 3375,
3376, 3382, 3389, 3399, 3418, 3420,
3972–3976, 8260, 8493, 8495–8498,
9454, 9566, 10285, 10287, 10289,
10291, 10293, 10295, 10303–10331,
10333–10337, 10340, 10341, 10343,
10344, 10346–10349, 13679, 13681
- \int_set:cn 3453
- \int_set:Nn 63,
3453, 3453, 3455, 3456, 6661, 6662,
8309, 8322, 8336, 8344, 8360, 8396,
8494, 8540, 8553, 8585, 8613, 8910,
9686, 9712, 9780, 10286, 10288,
10290, 10292, 10294, 10296, 11048,
11083, 13541, 13543, 13545, 13547,
13549, 13551, 13553, 13555, 13680
- \int_set_eq:cc 3423
- \int_set_eq:cN 3423
- \int_set_eq:Nc 3423
- \int_set_eq:NN
62, 3423, 3423–3425, 6663, 8511, 8547
- \int_show:c 3949, 3950
- \int_show:N 69, 3949, 3949
- \int_show:n 69, 3951, 3951
- \int_sub:cn 3429
- \int_sub:Nn 63, 3429, 3431, 3436, 3439, 8644
- \int_to_Alph:n 66, 3631, 3663
- \int_to_alph:n 66, 3631, 3631
- \int_to_arabic:n 66, 3610, 3610
- \int_to_base:nn
67, 3695, 3695, 3757, 3759, 3761, 3978
- \int_to_base_aux_i:nn .. 3695, 3696, 3697
- \int_to_base_aux_ii:nnN
..... 3695, 3700, 3701, 3703, 3717
- \int_to_base_aux_iii:nnnN 3695, 3708, 3715
- \int_to_binary:n 67, 3756, 3756
- \int_to_hexadecimal:n ... 67, 3756, 3758
- \int_to_letter:n 70, 3695, 3706, 3709, 3723
- \int_to_octal:n 67, 3756, 3760
- \int_to_Roman:n 68, 3762, 3772
- \int_to_roman:n 68, 3762, 3762
- \int_to_roman:w
70, 813, 814, 914, 916, 1045, 2020,
2025, 2171, 3301, 3465, 3765, 3775
- \int_to_Roman_aux:N 3774, 3777, 3780
- \int_to_roman_aux:N 3762, 3764, 3767, 3770
- \int_to_Roman_c:w 3762, 3794
- \int_to_roman_c:w 3762, 3786
- \int_to_Roman_d:w 3762, 3795
- \int_to_roman_d:w 3762, 3787
- \int_to_Roman_i:w 3762, 3790
- \int_to_roman_i:w 3762, 3782
- \int_to_Roman_l:w 3762, 3793
- \int_to_roman_l:w 3762, 3785
- \int_to_Roman_m:w 3762, 3796
- \int_to_roman_m:w 3762, 3788
- \int_to_Roman_Q:w 3762, 3797
- \int_to_roman_Q:w 3762, 3789
- \int_to_Roman_v:w 3762, 3791
- \int_to_roman_v:w 3762, 3783
- \int_to_Roman_x:w 3762, 3792
- \int_to_roman_x:w 3762, 3784
- \int_to_symbol:n 3982, 3983
- \int_to_symbol_math:n .. 3982, 3987, 3990
- \int_to_symbol_text:n .. 3982, 3988, 4005
- \int_to_symbols:nnn .. 67, 3611, 3611,
3627, 3633, 3665, 3979, 3992, 4007
- \int_to_symbols_aux:nnnn 3615, 3625
- \int_until_do:nn ... 65, 3554, 3562, 3567
- \int_until_do:nNnn .. 65, 3582, 3590, 3595
- \int_use:c 3458, 3459
- \int_use:N
. 63, 2242, 2248, 3458, 3458, 3459,
4694, 4697, 5486, 5494, 6022, 6024,
6433, 6436, 8352, 8353, 8362, 8366,
8368, 8377, 8388, 8389, 8398, 8402,
8404, 8413, 8703, 8704, 8817, 9501,
9523, 9549, 9551, 9687, 9713, 9781,
10414, 10454, 10471, 10484, 10518,
10532, 10543, 10557, 10609, 10612,
10614, 10651, 10654, 10656, 11063,
11066, 11068, 11099, 11102, 11104,
11116, 11143, 11172, 11175, 11177,
11198, 11201, 11203, 11252, 11258,
11373, 11379, 11423, 11435, 11516,
11523, 11535, 11765, 11777, 11793,
11805, 11815, 11826, 11839, 11858,
12014, 12020, 12069, 12075, 12111,
12117, 12155, 12161, 12312, 12318,
12436, 12442, 12506, 12512, 12585,
12618, 12644, 12647, 12728, 12734,
12845, 12851, 12888, 12895, 12908,
12930, 12947, 12960, 12970, 12981,
13054, 13056, 13058, 13086, 13097,
13272, 13278, 13305, 13307, 13309,
13311, 13313, 13542, 13544, 13546,
13548, 13550, 13552, 13554, 13556
- \int_value:w 71, 1051, 1991, 1992,
2012, 2014–2016, 2173, 3301, 3301,
3308, 3311, 3315, 3323, 3334, 3345,

- 3371, 3461, 3753, 3908, 4097, 7165,
7189–7191, 7193, 7310, 7319, 7321,
7326–7329, 7333–7336, 7387, 7393,
7395, 7397, 7399, 7404, 7409, 7414,
7421, 7428, 7567, 7597, 7598, 7637,
7656, 7662, 7725, 7727, 7747, 7749,
7774, 7812, 7832, 7840, 7866, 7868,
7872, 7874, 7907, 7921, 7928, 8054,
8153, 8158, 8658, 10805, 10917,
10936, 11255, 11376, 11519, 12017,
12072, 12114, 12158, 12315, 12439,
12509, 12731, 12848, 12891, 13275
- \int_while_do:nn ... 66, 3554, 3554, 3559
- \int_while_do:nNnn .. 65, 3582, 3582, 3587
- \int_zero:c 3413
- \int_zero:N 62, 3413, 3413, 3415, 3418,
8541, 8543, 8632, 9680, 9699, 9774
- \int_zero_new:c 3417
- \int_zero_new:N 62, 3417, 3417, 3421
- \interactionmode 699
- \interlinepenalties 720
- \interlinepenalty 579
- \io_new:c 136
- \ior_alloc_read:n 8310, 8332, 8340
- \ior_close:N 137,
8308, 8422, 8448, 8716, 10150, 10179
- \ior_gto:NN 140, 8688, 8690
- \ior_if_eof:N 8672
- \ior_if_eof:Nf 8712, 8722, 10169
- \ior_if_eof:Ntf 140, 10147
- \ior_if_eof_p:N 8672
- \ior_list_streams 137
- \ior_list_streams: 8450, 8450, 8741
- \ior_list_streams_aux:Nn 8451, 8453, 8454
- \ior_map_inline:nn 141, 8696, 8696
- \ior_map_inline_aux:Nnn 8697, 8699, 8700
- \ior_map_inline_aux:NNNnn ... 8702, 8706
- \ior_map_inline_loop:NNN 8712, 8719, 8725
- \ior_new:c 8276
- \ior_new:N 136,
8276, 8276, 8277, 8349, 8708, 10131
- \ior_open:cn 8280
- \ior_open:Nn .. 136, 8280, 8280, 8292, 8710
- \ior_open_streams: 8740, 8741
- \ior_open_unsafe:Nn
.. 141, 8280, 8290, 8306, 10146, 10168
- \ior_raw_new:c 8262, 8402
- \ior_raw_new:N
... 141, 8262, 8264, 8272, 8274, 8395
- \ior_str_gto:NN 140, 8692, 8694
- \ior_str_map_inline:nn . 141, 8696, 8698
- \ior_str_map_inline_aux:Nnn 8696
- \ior_str_map_inline_aux:NNNnn ... 8696
- \ior_str_map_inline_loop:NNN 8696
- \ior_str_to:NN 140, 8692, 8692, 8699
- \ior_stream_alloc:N 8314, 8348, 8386
- \ior_stream_alloc_aux:
..... 8348, 8392, 8408, 8416, 8418
- \ior_to:NN 140, 8688, 8688, 8697
- \iow_alloc_write:n 8323, 8332, 8332
- \iow_char:N 138, 8492, 8492
- \iow_close:c 8422
- \iow_close:N .. 137, 8321, 8422, 8435, 8449
- \iow_indent:n 139, 8530, 8530, 8556
- \iow_indent_expandable:n 8530, 8531, 8556
- \iow_list_streams 137
- \iow_list_streams: 8452, 8742
- \iow_log:n . 137, 8483, 8484, 10262–10264
- \iow_log:x 1153, 1153, 1192,
1819, 8483, 8483, 8898, 8900, 8901
- \iow_new:c 8276
- \iow_new:N ... 136, 8276, 8278, 8279, 8348
- \iow_newline 138
- \iow_newline: 8491, 8491,
8564, 8885, 8887, 9397, 9401, 9406
- \iow_now:Nn 137, 8481,
8481, 8484, 8486, 8488, 8736, 8738
- \iow_now:Nx
.. 8480, 8480, 8482, 8483, 8485, 8490
- \iow_now_buffer_safe:Nn 8734, 8735
- \iow_now_buffer_safe:Nx 8734, 8737
- \iow_now_when_avail:Nn . 138, 8487, 8487
- \iow_now_when_avail:Nx 8487, 8489
- \iow_open:cn 8280
- \iow_open:Nn 137, 8280, 8293, 8305
- \iow_open_streams: 8740, 8742
- \iow_open_unsafe:Nn 141, 8280, 8303, 8319
- \iow_raw_new:c 8262, 8366
- \iow_raw_new:N
... 142, 8262, 8267, 8271, 8275, 8359
- \iow_shipout:Nn ... 138, 8477, 8477, 8479
- \iow_shipout:Nx 8477
- \iow_shipout_x:Nn
... 138, 8475, 8475, 8476, 8478, 8480
- \iow_shipout_x:Nx 8475
- \iow_stream_alloc:N 8327, 8348, 8350
- \iow_stream_alloc_aux:
..... 8348, 8356, 8372, 8380, 8382
- \iow_term:n 137, 8483, 8486

- \iow_term:x [1153](#), [1155](#), [8483](#),
8485, 8883, 8905, 8907, 8908, 9366
- \iow_wrap:xnnnN
[139](#), [8537](#), 8537, 8736, 8738, 8842,
8845, 8850, 8853, 8899, 8906, 9361
- \iow_wrap_end: [8648](#)
- \iow_wrap_end:w [8621](#)
- \iow_wrap_indent: [8636](#)
- \iow_wrap_indent:w [8621](#)
- \iow_wrap_loop:w
..... [8567](#), [8576](#), 8576, 8591, [8626](#)
- \iow_wrap_new_marker:n
..... [8510](#), 8514, 8525–8528
- \iow_wrap_newline: [8628](#)
- \iow_wrap_newline:w [8621](#)
- \iow_wrap_special:w [8580](#), [8621](#), 8621, [8625](#)
- \iow_wrap_unindent: [8642](#)
- \iow_wrap_unindent:w [8621](#)
- \iow_wrap_word: [8581](#), [8583](#), 8583
- \iow_wrap_word_fits: ... [8583](#), 8589, 8593
- \iow_wrap_word_newline: [8583](#), 8590, 8609
- J**
- \jobname [654](#)
- K**
- \kern [532](#)
- \kernel_register_show:c [1409](#), 1418, 3950
- \kernel_register_show:N [1409](#),
1409, 1419, 3949, 4209, 4298, 4353
- \keys_bool_set:NN [9637](#), 9637, 9804, 9806
- \keys_bool_set_inverse:NN
..... [9652](#), 9652, 9808, 9810
- \keys_choice_code_store:x
..... [9719](#), 9719, 9820, [9822](#)
- \keys_choice_find:n
..... [9670](#), 9760, [10009](#), 10009
- \keys_choice_make: [9640](#),
9655, [9667](#), 9667, 9679, 9698, 9812
- \keys_choices_generate:n [9693](#), 9693, 9852
- \keys_choices_generate_aux:n
..... [9693](#), 9700, 9707
- \keys_choices_make:nn .. [9677](#), 9677, 9814
- \keys_cmd_set:nn 9645, 9660, 9669, 9671,
[9730](#), 9730, 9751, 9763, 9765, 9816
- \keys_cmd_set:nx
9641, 9643, 9656, 9658, 9683, 9709,
[9730](#), 9735, 9756, 9777, 9796, 9818
- \keys_cmd_set_aux:n [9730](#), 9732, 9737, 9740
- \keys_default_set:n
.. 9650, 9665, [9746](#), 9746, 9748, 9832
- \keys_define_set:V [9746](#), 9834
- \keys_define:nn .. [152](#), [9575](#), 9575, 10048
- \keys_define_aux:nnn ... [9575](#), 9577, 9583
- \keys_define_aux:onn [9575](#), 9576
- \keys_define_elt:n 9580, [9584](#), 9584
- \keys_define_elt:nn 9580, [9584](#), 9589
- \keys_define_elt_aux:nn
..... [9584](#), 9587, 9592, 9594
- \keys_define_key:n 9597, [9620](#), 9620
- \keys_define_key_aux:w . [9620](#), 9624, 9635
- \keys_execute: 9955, [9980](#), 9980
- \keys_execute:nn
[9980](#), 9981, 9984, 10000, 10011, 10012
- \keys_execute_unknown:
.. 9913, 9915, 9980, 9981, 9982, 9990
- \keys_execute_unknown_alt:
..... 9913, 9980, 9991
- \keys_execute_unknown_std:
..... 9915, [9980](#), 9990
- \keys_if_choice_exist:nnn . [10020](#), 10020
- \keys_if_choice_exist:nnTF [160](#)
- \keys_if_exist:nn [10014](#), 10014
- \keys_if_exist:nnTF [160](#)
- \keys_if_value:n 9973
- \keys_if_value_p:n 9938, 9948, [9973](#)
- \keys_meta_make:n [9749](#), 9749, 9862
- \keys_meta_make:x [9749](#), 9754, 9864
- \keys_multichoice_find:n [9759](#), 9759, 9764
- \keys_multichoice_make:
..... [9759](#), 9761, 9773, 9866
- \keys_multichoices_make:nn
..... [9759](#), 9771, 9868
- \keys_property_find:n .. 9595, [9603](#), 9603
- \keys_property_find_aux:w
..... [9603](#), 9607, 9610, 9616
- \keys_set:nn
.. [159](#), 9752, 9757, [9897](#), 9897, 9905
- \keys_set:no [9897](#)
- \keys_set:nV [9897](#)
- \keys_set:nv [9897](#)
- \keys_set_aux:nnn [9897](#), 9899, 9906
- \keys_set_aux:onn [9897](#), 9898
- \keys_set_elt:n .. 9902, 9914, [9921](#), 9921
- \keys_set_elt:nn . 9902, 9914, [9921](#), 9926
- \keys_set_elt_aux:nn [9921](#), 9924, 9929, 9931
- \keys_set_known:nnN [160](#), [9907](#), 9907, 9919
- \keys_set_known:noN [9907](#)
- \keys_set_known:nVN [9907](#)

- \keys_set_known:nvN 9907
 - \keys_set_known_aux:nnnN 9907, 9909, 9920
 - \keys_set_known_aux:onnN 9907, 9908
 - \keys_show:nn 160, 10026, 10026
 - \keys_value_or_default:n 9935, 9958, 9958
 - \keys_value_requirement:n
 - 9787, 9787, 9894, 9896
 - \keys_variable_set:cnN .. 9793, 9826,
 - 9838, 9846, 9856, 9872, 9880, 9884
 - \keys_variable_set:cnNN . 9793, 9830,
 - 9842, 9850, 9860, 9876, 9888, 9892
 - \keys_variable_set:NnN
 - 9793, 9799, 9802, 9824,
 - 9836, 9844, 9854, 9870, 9878, 9882
 - \keys_variable_set:NnNN
 - ... 9793, 9793, 9800, 9801, 9828,
 - 9840, 9848, 9858, 9874, 9886, 9890
 - \keyval_parse:n 9459, 9467, 9552
 - \keyval_parse:NnN 161, 9545,
 - 9545, 9580, 9902, 9914, 10092–10094
 - \keyval_parse_elt:w
 - 9475, 9481, 9481, 9484, 9489
 - \keyval_split_key:w 9495, 9513, 9513
 - \keyval_split_key_value:w 9488, 9493, 9493
 - \keyval_split_key_value_aux:wTF
 - 9493, 9506, 9511
 - \keyval_split_value:w .. 9507, 9518, 9518
 - \keyval_split_value_aux:w ... 9536, 9539
 - \KV_process_no_space_removal_no_sanitiz:NNn
 - 10091, 10094
 - \KV_process_space_removal_no_sanitiz:NNn
 - 10091, 10093
 - \KV_process_space_removal_sanitiz:NNn
 - 10091, 10092
- L
- \l@expl@log@functions@bool 1184
 - \l_box_angle_fp .. 6754, 6754, 6776, 6824
 - \l_box_bottom_dim .. 6757, 6758, 6791,
 - 6856, 6860, 6865, 6871, 6876, 6880,
 - 6889, 6891, 6904, 6912, 6935, 6941,
 - 6951, 6956, 6972, 6994, 7013, 7016
 - \l_box_bottom_new_dim
 - 6761, 6762, 6817, 6857, 6868, 6879,
 - 6890, 6934, 6940, 7013, 7017, 7033
 - \l_box_cos_fp 6755, 6755,
 - 6780, 6796, 6801, 6828, 6836, 6847
 - \l_box_internal_box 6765, 6765,
 - 6805, 6806, 6812, 6816–6818, 6820,
 - 7026, 7032, 7033, 7039, 7044, 7045
 - \l_box_internal_fp
 - 6765, 6766, 6824–6828, 6835, 6837,
 - 6838, 6846, 6848, 6849, 6908, 6909,
 - 6911, 6913, 6955, 6957, 6976, 6977
 - \l_box_left_dim 6757, 6759, 6793,
 - 6856, 6858, 6867, 6871, 6876, 6882,
 - 6887, 6891, 6906, 6953, 6974, 6996
 - \l_box_left_new_dim 6761, 6763,
 - 6808, 6819, 6859, 6870, 6881, 6892
 - \l_box_right_dim 6757,
 - 6760, 6792, 6854, 6860, 6865, 6869,
 - 6878, 6880, 6889, 6893, 6905, 6908,
 - 6952, 6973, 6976, 6995, 7020, 7021
 - \l_box_right_new_dim 6761, 6764,
 - 6819, 6861, 6872, 6883, 6894, 6928,
 - 6929, 7020, 7021, 7036, 7038, 7044
 - \l_box_scale_x_fp 6896,
 - 6896, 6907, 6909, 6914, 6958, 6975,
 - 6977–6979, 6991, 6997, 7019, 7034
 - \l_box_scale_y_fp
 - 6896, 6897, 6910, 6913,
 - 6916, 6933, 6935, 6939, 6941, 6954,
 - 6957–6959, 6978, 6992, 6999, 7010
 - \l_box_sin_fp 6755,
 - 6756, 6778, 6794, 6827, 6837, 6848
 - \l_box_top_dim 6757, 6757, 6790,
 - 6854, 6858, 6867, 6869, 6878, 6882,
 - 6887, 6893, 6903, 6912, 6933, 6939,
 - 6950, 6956, 6971, 6993, 7012, 7017
 - \l_box_top_new_dim
 - 6761, 6761, 6816, 6855, 6866, 6877,
 - 6888, 6932, 6938, 7012, 7016, 7032
 - \l_box_x_fp
 - .. 6767, 6767, 6832, 6834, 6843, 6846
 - \l_box_x_new_fp
 - .. 6767, 6769, 6834, 6836, 6838, 6839
 - \l_box_y_fp
 - .. 6767, 6768, 6833, 6835, 6844, 6845
 - \l_box_y_new_fp
 - .. 6767, 6770, 6845, 6847, 6849, 6850
 - \l_cctab_internal_tl
 - 13714, 13728–13730, 13752
 - \l_char_active_seq
 - 51, 2630, 2630, 2643, 4931
 - \l_char_special_seq . 51, 2630, 2647, 2648
 - \l_clist_internal_clist
 - 5759, 5759, 5847, 5848,
 - 5860, 5861, 5968, 5969, 6031, 6032,
 - 6048, 6049, 6070, 6072, 6074, 9292

\l_clist_internal_remove_clist	\l_coffin_display_pole_coffin
.. 5908, 5908, 5915, 5918, 5919, 5921	.. 7934, 7936, 7988, 7999, 8043, 8081
\l_coffin_aligned_coffin	\l_coffin_display_poles_prop
..... 7297, 7299, 7554, 7978, 7978,
7555, 7559, 7565, 7567, 7568, 7584,	8053, 8058, 8061, 8063, 8065, 8072
7585, 7591–7595, 7597, 7599, 7603,	\l_coffin_display_x_dim
7604, 7609–7613, 7647, 7662, 7707, 7976, 7976, 8078, 8133
7709, 8136, 8143, 8145, 8147, 8149	\l_coffin_display_y_dim
\l_coffin_aligned_internal_coffin 7976, 7977, 8079, 8135
..... 7297, 7300, 7626, 7633	\l_coffin_error_bool 7138, 7138, 7440,
\l_coffin_bottom_corner_dim	7444, 7458, 7473, 7506, 8074, 8076
..... 7714, 7716,	\l_coffin_Height_dim . . . 7153, 7154, 7344
7739, 7743, 7810, 7819, 7835, 7843	\l_coffin_internal_box . . 7112, 7112,
\l_coffin_bounding_prop	7229, 7233, 7237, 7272, 7277, 7282
..... 7712, 7712, 7730,	\l_coffin_internal_dim . . 7112, 7113,
7755, 7757, 7760, 7762, 7768, 7825	7560, 7562, 7563, 7759, 7761, 7763
\l_coffin_bounding_shift_dim	\l_coffin_internal_fp
..... 7713, 7713, 7737, 7824, 7829 7112, 7114, 7720–7724,
\l_coffin_calc_a_fp	7794, 7796, 7797, 7799, 7801, 7802,
7133, 7133, 7498, 7502, 7509, 7511–	7856, 7857, 7859, 7860, 7897–7902
7513, 7516–7518, 7521, 7541, 7545	\l_coffin_internal_tl 7112,
\l_coffin_calc_b_fp	7115, 7122–7132, 7645, 7646, 7648,
..... 7133, 7134, 7499, 7502,	8012, 8013, 8016, 8017, 8025, 8030,
7505, 7514, 7522, 7525, 7542, 7548	8095, 8096, 8099, 8100, 8109, 8114
\l_coffin_calc_c_fp	\l_coffin_left_corner_dim 7714, 7714,
.. 7133, 7135, 7500, 7503, 7543, 7547	7738, 7746, 7811, 7817, 7834, 7842
\l_coffin_calc_d_fp . 7133, 7136, 7501,	\l_coffin_offset_x_dim
7503, 7505, 7519, 7523, 7544, 7546 7139, 7139, 7557,
\l_coffin_calc_result_fp	7558, 7561, 7569, 7571, 7573, 7579,
... 7133, 7137, 7508, 7510, 7515,	7582, 7602, 7622, 7630, 8132, 8140
7520, 7524, 7527, 7540, 7545–7549	\l_coffin_offset_y_dim
\l_coffin_cos_fp 7139, 7140, 7572, 7574, 7579,
..... 7143, 7144, 7724, 7795, 7800	7582, 7602, 7624, 7631, 8134, 8141
\l_coffin_Depth_dim 7153, 7153, 7345	\l_coffin_pole_a_tl
\l_coffin_display_coffin 7141, 7141, 7438, 7443, 7671,
..... 7934, 7934, 8062, 8068,	7674, 7675, 7678, 8055, 8057, 8060
8138, 8139, 8144, 8146, 8148, 8149	\l_coffin_pole_b_tl
\l_coffin_display_coord_coffin	7141, 7142, 7439, 7443, 7672, 7674,
..... 7934, 7935,	7676, 7678, 8056, 8057, 8059, 8060
8000, 8020, 8036, 8083, 8103, 8122	\l_coffin_right_corner_dim
\l_coffin_display_font_tl 7714, 7715, 7746, 7809, 7818
.. 7979, 7979, 7981, 7984, 8008, 8091	\l_coffin_scale_x_fp 7847, 7847,
\l_coffin_display_handles_prop	7855, 7857, 7870, 7883, 7888, 7898
7937, 7937, 7938, 7940, 7942, 7944,	\l_coffin_scale_y_fp 7847,
7946, 7948, 7950, 7952, 7954, 7956,	7848, 7858, 7860, 7884, 7885, 7901
7958, 7960, 7962, 7964, 7966, 7968,	\l_coffin_scaled_total_height_dim . .
7970, 7972, 8011, 8015, 8094, 8098 7849, 7849, 7886, 7887, 7892
\l_coffin_display_offset_dim . 7974,	\l_coffin_scaled_width_dim
7974, 7975, 8037, 8038, 8123, 8124 7849, 7850, 7889, 7890, 7892

\l_coffin_sin_fp 12372, 12403, 12428, 12429, 12499,
 7143, 7143, 7723, 7796, 7801
 \l_coffin_top_corner_dim 12514, 12523, 12525, 12553, 12593,
 7714, 7717, 7743, 7808, 7820 12725, 12842, 12853, 12861, 12881
 \l_coffin_TotalHeight_dim 7153, 7155, 7346
 \l_coffin_Width_dim 7153, 7156, 7347
 \l_coffin_x_dim 7145, 7145, 7447,
 7456, 7476, 7479, 7486, 7493, 7495,
 7526, 7529, 7619, 7623, 7642, 7650,
 7767, 7769, 7773, 7775, 7779, 7784,
 7906, 7908, 7912, 7915, 8078, 8130
 \l_coffin_x_fp 7149, 7149, 7791, 7793, 7799
 \l_coffin_x_prime_dim ... 7145, 7147,
 7619, 7623, 7781, 7785, 8130, 8133
 \l_coffin_x_prime_fp
 7149, 7151, 7793, 7795, 7797, 7803
 \l_coffin_y_dim
 7145, 7146, 7448, 7461, 7464, 7471,
 7488, 7530, 7620, 7625, 7643, 7650,
 7767, 7769, 7773, 7775, 7779, 7784,
 7906, 7908, 7912, 7915, 8079, 8131
 \l_coffin_y_fp 7149, 7150, 7792, 7794, 7798
 \l_coffin_y_prime_dim ... 7145, 7148,
 7620, 7625, 7781, 7786, 8131, 8135
 \l_coffin_y_prime_fp
 7149, 7152, 7798, 7800, 7802, 7804
 \l_doc_pTF_name_tl 21,
 22, 36, 37, 40, 41, 44, 52, 53, 54,
 55, 64, 74, 78, 87, 88, 93, 94, 100,
 109, 114, 117, 125, 126, 140, 160, 168
 \l_exp_internal_tl 32, 881,
 882, 1521, 1521, 1540, 1541, 1718, 1719
 \l_expl_status_bool
 96, 294, 309, 323, 327, 328
 \l_expl_status_stack_tl 197
 \l_file_internal_name_tl 164,
 10123, 10123, 10135, 10136, 10139,
 10142, 10183, 10184, 10190, 10191,
 10201, 10208, 10209, 10212, 10215
 \l_file_internal_saved_path_seq
 164, 10125, 10126, 10160, 10177
 \l_file_internal_seq 164, 10128,
 10129, 10161, 10163, 10269, 10270
 \l_file_search_path_seq 164,
 10124, 10124, 10160, 10162, 10163,
 10166, 10177, 10254, 10255, 10258
 \l_fp_arg_tl 10302, 10302, 12007,
 12026, 12030, 12040, 12061, 12062,
 12104, 12119, 12127, 12147, 12148,
 12305, 12324, 12328, 12338, 12348,
 12372, 12403, 12428, 12429, 12499,
 12514, 12523, 12525, 12553, 12593,
 12725, 12842, 12853, 12861, 12881
 \l_fp_count_int 10303, 10303,
 11529, 11564, 11576, 11584, 12202,
 12234, 12251, 12253, 12256, 12258,
 12702, 12739, 12747, 12977, 12980,
 12981, 13003, 13047, 13107, 13118
 \l_fp_div_offset_int .. 10304, 10304,
 11485, 11539, 11584, 11586, 12417
 \l_fp_exp_decimal_int . 10305, 10306,
 12577, 12611, 12630, 12650, 12669,
 12678, 12683, 12689, 12705, 12754,
 12759, 12763, 12766, 12770, 12774,
 12777, 12779, 12923, 12933, 12944,
 12947, 12956, 12964, 12990, 13053,
 13061, 13065, 13076, 13119, 13120
 \l_fp_exp_exponent_int
 10305, 10308, 12579,
 12613, 12635, 12655, 12671, 12921,
 12925, 12943, 12950, 12973, 13057
 \l_fp_exp_extended_int 10305,
 10307, 12578, 12612, 12630, 12650,
 12670, 12679, 12682, 12687, 12693,
 12705, 12755, 12757, 12760, 12771,
 12773, 12775, 12924, 12933, 12966,
 12970, 12972, 12990, 13055, 13062,
 13064, 13067, 13076, 13119, 13121
 \l_fp_exp_integer_int
 10305, 10305, 12576, 12610,
 12630, 12650, 12668, 12677, 12681,
 12705, 12765, 12778, 12922, 12933,
 12955, 12960, 12963, 12990, 13052
 \l_fp_input_a_decimal_int
 10309, 10311, 10360, 10551, 10552,
 10557, 10559, 10596, 10598, 10612,
 10638, 10640, 10654, 11052, 11066,
 11088, 11102, 11114, 11116, 11123,
 11127, 11165, 11175, 11190, 11201,
 11271, 11287, 11386, 11468, 11533,
 11535, 11537, 11553, 11566, 11567,
 11569, 11592, 11763, 11765, 11774,
 11782, 11783, 11790, 11793, 11801,
 11809, 11890, 11904, 11905, 11909,
 11912, 11914, 11920, 11928, 11930,
 11943, 11944, 11951, 11953, 11961,
 11971, 11974, 11980, 11982, 11986,
 12005, 12018, 12102, 12115, 12189,
 12195, 12196, 12226, 12229, 12232,
 12243, 12247, 12303, 12316, 12370,

- 12394, 12399, 12401, 12496, 12510,
 12675, 12678, 12685, 12691, 12700,
 12742, 12814, 12819, 12849, 12996,
 13010, 13014, 13017, 13032, 13037,
 13041, 13044, 13045, 13109, 13111,
 13114, 13117, 13147, 13153, 13161,
 13178, 13204, 13237, 13287, 13298,
 13318, 13331, 13364, 13380, 13398,
 13423, 13493, 13525, 13545, 13554
- \l_fp_input_a_exponent_int
 10309, 10312, 10361,
 10506, 10514, 10539, 10560, 10597,
 10614, 10639, 10656, 11068, 11084,
 11104, 11128, 11177, 11203, 11236,
 11346, 11489, 11761, 11785, 11789,
 11818, 11865, 12006, 12020, 12022,
 12103, 12117, 12304, 12318, 12320,
 12345, 12395, 12400, 12421, 12497,
 12512, 12535, 12815, 12851, 12900,
 12901, 12906, 12908, 12919, 12929,
 12930, 13030, 13039, 13148, 13154,
 13199, 13233, 13288, 13299, 13326,
 13333, 13365, 13381, 13400, 13475,
 13479, 13507, 13511, 13547, 13556
- \l_fp_input_a_extended_int
 10317, 10317, 11775,
 11777, 11784, 11811, 11815, 11817,
 11866, 11906–11908, 11910, 11920,
 11932, 11934, 11945, 11952, 11954,
 11962, 11972, 11975, 11983, 11987,
 12195, 12196, 12230, 12233, 12243,
 12247, 12279, 12498, 12679, 12695,
 12701, 12742, 12772, 12978, 13011,
 13018, 13038, 13042, 13044, 13045,
 13109, 13111, 13114, 13117, 13198,
 13204, 13235, 13316, 13319, 13332
- \l_fp_input_a_integer_int
 10309, 10310, 10359, 10540, 10543,
 10550, 10595, 10609, 10637, 10651,
 11063, 11099, 11122, 11124, 11126,
 11172, 11198, 11267, 11283, 11396,
 11401, 11405, 11410, 11468, 11532,
 11537, 11547, 11550, 11565, 11568,
 11590, 11591, 11762, 11772, 11773,
 11800, 11805, 11808, 11869, 11871,
 11884, 11889, 11902, 11903, 11913,
 11916, 11922, 11924, 11926, 11951,
 11953, 11969, 11971, 11974, 11982,
 11986, 12004, 12014, 12101, 12111,
 12302, 12312, 12371, 12393, 12398,
- 12402, 12495, 12506, 12538, 12548,
 12563, 12575, 12585, 12587, 12589,
 12614, 12618, 12620, 12622, 12642,
 12644, 12647, 12813, 12819, 12845,
 12996, 13002, 13013, 13016, 13029,
 13036, 13146, 13152, 13161, 13178,
 13238, 13286, 13297, 13318, 13330,
 13363, 13379, 13397, 13423, 13483,
 13488, 13515, 13520, 13543, 13552
- \l_fp_input_a_sign_int
 10309, 10309, 10355, 10357,
 10594, 10604, 10636, 10646, 11058,
 11094, 11193, 11230, 11264, 11308,
 11360, 11503, 11870, 11875, 11917,
 12003, 12009, 12057, 12064, 12100,
 12106, 12143, 12150, 12176, 12178,
 12183, 12301, 12307, 12356, 12397,
 12494, 12501, 12537, 12591, 12624,
 12643, 12676, 12699, 12740, 12812,
 12816, 13145, 13151, 13230, 13284,
 13329, 13362, 13378, 13396, 13443,
 13457, 13461, 13465, 13541, 13550
- \l_fp_input_b_decimal_int
 10309, 10315, 10526, 10527,
 10532, 10534, 11220, 11271, 11287,
 11323, 11341, 11388, 11454, 11458,
 11553, 11566, 12385, 12390, 12700,
 12743, 12744, 12746, 12748, 12751,
 12754, 12770, 13032, 13046, 13110,
 13147, 13157, 13290, 13298, 13308,
 13320, 13370, 13380, 13398, 13426,
 13441, 13493, 13525, 13546, 13553
- \l_fp_input_b_exponent_int
 10309, 10316, 10506, 10514, 10535,
 10539, 11221, 11324, 11342, 11346,
 11455, 11489, 12386, 12391, 12421,
 13033, 13148, 13291, 13299, 13312,
 13326, 13371, 13381, 13400, 13475,
 13479, 13507, 13511, 13548, 13555
- \l_fp_input_b_extended_int
 10317, 10318, 12701,
 12743, 12744, 12746, 12748, 12751,
 12756, 13046, 13110, 13310, 13321
- \l_fp_input_b_integer_int
 10309, 10314, 10515, 10518,
 10525, 11219, 11267, 11283, 11322,
 11340, 11399, 11403, 11406, 11410,
 11453, 11458, 11547, 11550, 11565,
 12384, 12389, 13031, 13146, 13157,
 13289, 13297, 13306, 13320, 13369,

- 13379, 13397, 13426, 13441, 13483,
13488, 13515, 13520, 13544, 13551
- \l_fp_input_b_sign_int 10309, 10313,
11218, 11230, 11294, 11321, 11325,
11339, 11360, 11452, 11503, 12388,
12699, 12740, 12753, 13145, 13194,
13304, 13329, 13368, 13378, 13396,
13431, 13457, 13461, 13542, 13549
- \l_fp_internal_dim
..... 10621, 10629, 10633, 10669
- \l_fp_internal_int 10344,
10344, 10412, 10414, 11135–11138,
11143, 11739–11741, 11746–11748
- \l_fp_internal_skip
..... 10621, 10628, 10629, 10670
- \l_fp_internal_t1
10345, 10345, 10365–10367, 10371,
10376, 10378, 10381, 10387, 10389,
10392, 10481, 10486, 10528, 10534,
10553, 10559, 11191, 11206, 11803,
11809, 11812, 11817, 11835, 11842,
11854, 11860, 12582, 12589, 12596,
12602, 12615, 12622, 12625, 12627,
12958, 12964, 12967, 12972, 13095,
13101, 13539, 13558, 13564, 13569
- \l_fp_mul_a_i_int 10319, 10319,
11387, 11392, 11397, 11402, 11406,
11636, 11645, 11652, 11658, 11663,
11669, 11672, 11680, 11689, 11697,
11705, 11712, 11720, 11725, 11729
- \l_fp_mul_a_ii_int
.... 10319, 10320, 11387, 11393,
11398, 11403, 11636, 11646, 11653,
11659, 11664, 11670, 11680, 11690,
11698, 11706, 11713, 11721, 11726
- \l_fp_mul_a_iii_int 10319,
10321, 11387, 11394, 11399, 11636,
11647, 11654, 11660, 11665, 11680,
11691, 11699, 11707, 11714, 11722
- \l_fp_mul_a_iv_int 10319,
10322, 11638, 11648, 11655, 11661,
11682, 11692, 11700, 11708, 11715
- \l_fp_mul_a_v_int
.... 10319, 10323, 11638, 11649,
11656, 11682, 11693, 11701, 11709
- \l_fp_mul_a_vi_int 10319, 10324,
11638, 11650, 11682, 11694, 11702
- \l_fp_mul_b_i_int 10319, 10325,
11389, 11394, 11398, 11402, 11405,
11640, 11650, 11656, 11661, 11665,
11670, 11672, 11684, 11694, 11701,
11708, 11714, 11721, 11725, 11728
- \l_fp_mul_b_ii_int
.... 10319, 10326, 11389, 11393,
11397, 11401, 11640, 11649, 11655,
11660, 11664, 11669, 11684, 11693,
11700, 11707, 11713, 11720, 11724
- \l_fp_mul_b_iii_int 10319,
10327, 11389, 11392, 11396, 11640,
11648, 11654, 11659, 11663, 11684,
11692, 11699, 11706, 11712, 11719
- \l_fp_mul_b_iv_int 10319,
10328, 11642, 11647, 11653, 11658,
11686, 11691, 11698, 11705, 11711
- \l_fp_mul_b_v_int
.... 10319, 10329, 11642, 11646,
11652, 11686, 11690, 11697, 11704
- \l_fp_mul_b_vi_int 10319, 10330,
11642, 11645, 11686, 11689, 11696
- \l_fp_mul_output_int
..... 10331, 10331, 11390,
11395, 11428, 11429, 11433, 11435,
11440, 11643, 11651, 11687, 11695
- \l_fp_mul_output_t1 ... 10331, 10332,
11391, 11408, 11409, 11412, 11439,
11644, 11667, 11668, 11675, 11688,
11717, 11718, 11731, 11732, 11735
- \l_fp_output_decimal_int
.... 10333, 10335, 11240, 11256,
11269, 11273, 11276, 11285, 11289,
11291, 11295, 11298, 11300, 11351,
11364, 11377, 11408, 11483, 11494,
11507, 11520, 11581, 11583, 11834,
11839, 11847, 11877, 12052, 12053,
12059, 12073, 12138, 12139, 12145,
12159, 12191, 12206, 12210, 12215,
12219, 12227, 12229, 12261, 12266,
12270, 12273, 12277, 12281, 12284,
12286, 12384, 12393, 12415, 12426,
12440, 12567, 12611, 12631, 12633,
12651, 12653, 12706, 12708, 12713,
12714, 12716, 12723, 12732, 12869,
12870, 12872, 12879, 12892, 12911,
12923, 12934, 12936, 12988, 12991,
13037, 13040, 13041, 13054, 13074,
13077, 13083, 13086, 13093, 13101,
13120, 13124, 13128, 13131, 13276,
13290, 13309, 13322, 13331, 13335
- \l_fp_output_exponent_int ... 10333,
10336, 11236, 11241, 11258, 11344,

- 11352, 11379, 11487, 11495, 11523,
11861, 11878, 12050, 12054, 12060,
12075, 12136, 12140, 12146, 12161,
12419, 12427, 12442, 12569, 12613,
12635, 12655, 12724, 12734, 12880,
12895, 12913, 12925, 12943, 12950,
13039, 13058, 13082, 13103, 13278,
13291, 13313, 13324, 13333, 13337
- \l_fp_output_extended_int
..... 10337, 10337, 11852,
11853, 11858, 11860, 12053, 12139,
12207, 12211, 12216, 12218, 12230,
12262, 12264, 12267, 12278, 12280,
12282, 12385, 12394, 12568, 12612,
12632, 12634, 12652, 12654, 12707,
12709, 12711, 12867, 12912, 12924,
12935, 12937, 12989, 12992, 13042,
13056, 13075, 13078, 13121, 13122,
13125, 13311, 13323, 13332, 13336
- \l_fp_output_integer_int
..... 10333, 10334, 11239,
11252, 11265, 11275, 11281, 11290,
11293, 11296, 11302, 11304, 11350,
11364, 11373, 11412, 11482, 11493,
11507, 11516, 11576, 11822, 11823,
11826, 11833, 11876, 12049, 12052,
12058, 12069, 12135, 12138, 12144,
12155, 12190, 12205, 12209, 12214,
12225, 12272, 12285, 12414, 12425,
12436, 12566, 12592, 12610, 12631,
12633, 12651, 12653, 12706, 12708,
12717, 12722, 12728, 12873, 12878,
12888, 12910, 12922, 12934, 12936,
12988, 12991, 13036, 13074, 13077,
13092, 13097, 13100, 13130, 13272,
13289, 13307, 13322, 13330, 13334
- \l_fp_output_sign_int
..... 10333, 10333, 11238,
11247, 11264, 11294, 11308, 11349,
11492, 12358, 12360, 12364, 12366,
12424, 12431, 12721, 12877, 12883,
12902, 12904, 12983, 13069, 13305
- \l_fp_round_carry_bool 10338, 10338,
11111, 11121, 11134, 11140, 11148
- \l_fp_round_decimal_tl 10339, 10339,
11113, 11123, 11142, 11143, 11145
- \l_fp_round_position_int 10340, 10340,
11112, 11133, 11146, 11152, 11153
- \l_fp_round_target_int
..... 10340, 10341, 11048,
11049, 11083, 11085, 11133, 11146
- \l_fp_sign_tl
.. 10342, 10342, 13195, 13207, 13271
- \l_fp_split_sign_int
.. 10343, 10343, 10368, 10370, 10383
- \l_fp_trig_decimal_int
.... 10347, 10348, 12197, 12199,
12201, 12204, 12219, 12232, 12242,
12244, 12246, 12248, 12250, 12252,
12255, 12257, 12259, 12261, 12277
- \l_fp_trig_extended_int 10347, 10349,
12197, 12199, 12201, 12203, 12218,
12233, 12242, 12244, 12246, 12248,
12250, 12252, 12255, 12257, 12263
- \l_fp_trig_octant_int
.... 10346, 10346, 11941, 11947,
11959, 11960, 11976, 11988, 12080,
12166, 12177, 12181, 12357, 12363
- \l_fp_trig_sign_int 10347,
10347, 12193, 12231, 12240, 12260
- \l_ior_internal_tl .. 8258, 8258, 8283,
8284, 8287, 8290, 8696, 8721, 8724
- \l_ior_stream_int
..... 8260, 8261, 8309, 8311,
8315, 8344, 8388, 8389, 8393, 8396,
8402, 8404, 8410, 8411, 8413, 8415
- \l_iow_current_indentation_int
..... 8496, 8498,
8541, 8601, 8616, 8638, 8644, 8646
- \l_iow_current_indentation_tl 8499,
8501, 8542, 8599, 8619, 8639, 8645
- \l_iow_current_line_int
..... 8496, 8496, 8543,
8587, 8588, 8600, 8606, 8613, 8632
- \l_iow_current_line_tl
..... 8499, 8499, 8544, 8598,
8604, 8612, 8618, 8631, 8633, 8651
- \l_iow_current_word_int
..... 8496, 8497, 8585, 8587, 8615
- \l_iow_current_word_tl .. 8499, 8500,
8578, 8579, 8586, 8599, 8605, 8619
- \l_iow_internal_tl
.. 8258, 8259, 8296, 8297, 8300, 8303
- \l_iow_line_length_int
..... 139, 8493, 8493, 8494, 8540
- \l_iow_line_start_bool
.. 8504, 8504, 8546, 8595, 8597, 8634
- \l_iow_stream_int 8260,
8260, 8261, 8322, 8324, 8328, 8336,

- 8352, 8353, 8357, 8360, 8362, 8366,
8368, 8374, 8375, 8377, 8379, 8398
- \l_iow_target_length_int
..... 8495, 8495, 8540, 8588
- \l_iow_wrap_tl
.. 8502, 8502, 8545, 8559, 8562, 8568
- \l_iow_wrapped_tl
.. 8503, 8503, 8574, 8611, 8630, 8650
- \l_keys_choice_int .. 158, 9566, 9566,
9680, 9686, 9687, 9690, 9699, 9712,
9713, 9717, 9774, 9780, 9781, 9784
- \l_keys_choice_tl . 158, 9685, 9711, 9779
- \l_keys_choices_tl 9566, 9567
- \l_keys_key_tl 159, 9568,
9568, 9648, 9663, 9933, 9934, 9995
- \l_keys_module_tl
... 9569, 9569, 9576, 9579, 9581,
9605, 9752, 9757, 9898, 9901, 9903,
9908, 9911, 9916, 9934, 9984, 9987
- \l_keys_no_value_bool
... 9570, 9570, 9586, 9591, 9622,
9923, 9928, 9939, 9949, 9961, 9996
- \l_keys_path_tl
.... 159, 9571, 9571, 9600, 9605,
9612, 9615, 9630, 9641, 9643, 9645,
9656, 9658, 9660, 9669, 9671, 9674,
9683, 9696, 9704, 9709, 9715, 9722,
9725, 9727, 9747, 9751, 9756, 9763,
9765, 9768, 9777, 9790, 9796, 9816,
9818, 9934, 9943, 9953, 9963, 9965,
9968, 9976, 9981, 9987, 10011, 10012
- \l_keys_property_tl . 9572, 9572, 9596,
9600, 9618, 9625, 9626, 9629, 9633
- \l_keys_unknown_clist
..... 9573, 9573, 9912, 9917, 9993
- \l_keys_value_tl 159, 9574,
9574, 9953, 9960, 9967, 9997, 10005
- \l_keyval_key_tl
..... 9455, 9455, 9502, 9515, 9524
- \l_keyval_parse_tl .. 9457, 9458, 9474,
9478, 9498, 9520, 9529, 9533, 9542
- \l_keyval_sanitise_tl
..... 9457, 9457, 9470-9473, 9476
- \l_keyval_value_tl 9455,
9456, 9526, 9528, 9531, 9541, 9543
- \l_last_box 7104, 7104
- \l_msg_class_tl 9011, 9012, 9062, 9063, 9066
- \l_msg_current_class_tl
..... 9011, 9013, 9044, 9063
- \l_msg_current_module_tl 9011, 9014, 9045
- \l_msg_internal_tl 8751, 8751,
8861, 8864, 8872, 9382-9384, 9389
- \l_msg_redirect_classes_prop 8918, 8918
- \l_msg_redirect_classes_seq
..... 9011, 9011, 9019, 9024, 9027
- \l_msg_redirect_kernel_info_prop . 9163
- \l_msg_redirect_kernel_warning_prop
..... 9141
- \l_msg_redirect_names_prop
..... 8918, 8919, 9046, 9077
- \l_msg_text_tl 8830, 8830, 8857, 8890, 8892
- \l_peek_search_tl
..... 2963, 2963, 2981, 3002, 3045
- \l_peek_search_token
.. 2962, 2962, 2980, 3001, 3020, 3028
- \l_peek_token 56, 2960, 2960, 2969, 3020,
3028, 3038-3040, 3059, 3188-3190
- \l_seq_internal_a_tl
..... 5227, 5227, 5256, 5262,
5267, 5268, 5334, 5339, 5353, 5357
- \l_seq_internal_b_tl
.. 5227, 5228, 5330, 5334, 5356, 5357
- \l_seq_internal_remove_seq
.. 5302, 5302, 5309, 5312, 5313, 5315
- \l_tl_internal_a_tl 4647, 4650, 4652, 4660
- \l_tl_internal_b_tl 4647, 4651, 4652, 4661
- \l_tmpa_bool 36, 1928, 1928
- \l_tmpa_box 126, 6641, 6642, 6645
- \l_tmpa_clist 113, 6077, 6077
- \l_tmpa_dim 77, 4222, 4222
- \l_tmpa_int 70, 3972, 3972
- \l_tmpa_skip 79, 4304, 4304
- \l_tmpa_tl 5, 95, 4958, 4958
- \l_tmpb_box 126, 6641, 6647
- \l_tmpb_clist 113, 6077, 6078
- \l_tmpb_dim 77, 4222, 4223
- \l_tmpb_int 70, 3972, 3973
- \l_tmpb_skip 79, 4304, 4305
- \l_tmpb_tl 95, 4958, 4959
- \l_tmpe_dim 77, 4222, 4224
- \l_tmpe_int 70, 3972, 3974
- \l_tmpe_skip 79, 4304, 4306
- \language 449
- \lastbox 606
- \lastkern 539
- \lastlinefit 719
- \lastnodetype 700
- \lastpenalty 645
- \lastskip 540
- \latelua 761

<code>\lccode</code>	668	<code>\mag</code>	448
<code>\leaders</code>	536	<code>\mark</code>	450
<code>\left</code>	504	<code>\marks</code>	676
<code>\lefthyphenmin</code>	560	<code>\mathaccent</code>	461
<code>\leftskip</code>	562	<code>\mathbin</code>	491
<code>\leqno</code>	479	<code>\mathchar</code>	462, 2787
<code>\let</code>	59, 230, 336, 337, 349	<code>\mathchardef</code>	359
<code>\limits</code>	496	<code>\mathchoice</code>	459
<code>\linepenalty</code>	552	<code>\mathclose</code>	492
<code>\lineskip</code>	546	<code>\mathcode</code>	670
<code>\lineskiplimit</code>	547	<code>\mathinner</code>	493
<code>\long</code>	33, 339, 368	<code>\mathop</code>	494
<code>\looseness</code>	564	<code>\mathopen</code>	498
<code>\lower</code>	601	<code>\mathord</code>	499
<code>\lowercase</code>	640	<code>\mathparagraph</code>	3998
<code>\lua_now:n</code>	172, 13656, 13673	<code>\mathpunct</code>	500
<code>\lua_now:x</code>	4949,	<code>\mathrel</code>	501
13656, 13658, 13662, 13665, 13674		<code>\mathsection</code>	3997
<code>\lua_shipout:n</code> ..	172, 13656, 13676, 13678	<code>\mathsurround</code>	512
<code>\lua_shipout:x</code>	13656	<code>\maxdeadcycles</code>	582
<code>\lua_shipout_x:n</code>	173, 13656,	<code>\maxdepth</code>	583
13659, 13667, 13670, 13675, 13677		<code>\maxdimen</code>	4220
<code>\lua_shipout_x:x</code>	13656	<code>\meaning</code>	642
<code>\luaescapestring</code>	39, 40	<code>\medmuskip</code>	513
<code>\luatex_catcodetable:D</code>		<code>\message</code>	419
758, 773, 13716, 13717, 13722, 13730		<code>\MessageBreak</code>	222, 238–244
<code>\luatex_directlua:D</code>	759, 1443, 13658	<code>\middle</code>	724
<code>\luatex_if_engine:</code>	1420	<code>\mkern</code>	466
<code>\luatex_if_engine:F</code>		<code>\mode_if_horizontal:</code>	2264, 2264
1421, 1446, 13695, 13732, 13760		<code>\mode_if_horizontalTF</code>	40
<code>\luatex_if_engine:T</code> .	1420, 1445, 4947,	<code>\mode_if_inner:</code>	2266, 2266
13704, 13746, 13768, 13774, 13783		<code>\mode_if_innerTF</code>	41
<code>\luatex_if_engine:TF</code> ..	1422, 1447, 13656	<code>\mode_if_math:</code>	2268, 2268
<code>\luatex_if_engine_p:</code> ...	1429, 1451, 1501	<code>\mode_if_math:TF</code>	3986
<code>\luatex_if_engineTF</code>	22	<code>\mode_if_mathTF</code>	41
<code>\luatex_initcatcodetable:D</code>		<code>\mode_if_vertical:</code>	2262, 2262
760, 774, 13691, 13710		<code>\mode_if_verticalTF</code>	41
<code>\luatex_latelua:D</code>	761, 775, 13659	<code>\month</code>	652
<code>\luatex luatexversion:D</code>	762, 845	<code>\moveleft</code>	602
<code>\luatex_savecatcodetable:D</code>		<code>\moveright</code>	603
763, 776, 13721, 13757		<code>\msg_aux_show:n</code> 5546, 6066, 6074, 9395, 9395	
<code>\luatexcatcodetable</code>	773	<code>\msg_aux_show:nn</code>	6449, 9395, 9399
<code>\luatexinitcatcodetable</code>	774	<code>\msg_aux_show:Nnx</code>	150, 5543,
<code>\luatexlatalua</code>	775	6063, 6071, 6446, 8155, 9368, 9368	
<code>\luatexsavecatcodetable</code>	776	<code>\msg_aux_show:w</code>	9368, 9388, 9394
<code>\luatexversion</code>	762	<code>\msg_aux_show:x</code>	
		150, 8164, 8458, 9368, 9373, 9380	
		<code>\msg_aux_show_unbraced:nn</code>	
		8159, 8459, 9395, 9404	
		<code>\msg_aux_use:nn</code> 150, 8163, 8456, 9357, 9357	
M			
<code>\M</code>	2724		
<code>\m@ne</code>	834		

\msg_aux_use:nnxxxx	9357 , 9358 , 9359 , 9372	\msg_if_more_text:N	8934 , 8934
\msg_class_new:nn	9409 , 9410	\msg_if_more_text:NF	8943
\msg_class_set:nn	145 , 8920 , 8920 , 8945 , 8956 , 8967 , 8989 , 8997 , 9005 , 9010 , 9410	\msg_if_more_text:NT	8942
\msg_critical:nn	8956	\msg_if_more_text:NTF	8944
\msg_critical:nnx	8956	\msg_if_more_text_p:N	8941
\msg_critical:nnxx	8956	\msg_info:nn	8997
\msg_critical:nnxxx	8956	\msg_info:nnx	8997
\msg_critical:nnxxxx	145 , 8956	\msg_info:nnxx	8997
\msg_critical_text:n	144 , 8911 , 8912 , 8959	\msg_info:nnxxx	8997
\msg_direct_interrupt:xxxxx	9419 , 9424	\msg_info:nnxxxx	146 , 8997
\msg_direct_log:xx	9419 , 9425	\msg_info_text:n	145 , 8911 , 8915 , 9001 , 9170
\msg_direct_term:xx	9419 , 9426	\msg_interrupt:xxx	148 , 8831 , 8831 , 8947 , 8958 , 8971 , 8980 , 9088 , 9110 , 9123 , 9431
\msg_error:nn	8967	\msg_interrupt_aux:	8831 , 8837 , 8881
\msg_error:nnx	8967	\msg_interrupt_details:xxx	8831 , 8836 , 8848
\msg_error:nnxx	8967	\msg_interrupt_more_text:n	8831 , 8844 , 8852 , 8858
\msg_error:nnxxx	8967	\msg_interrupt_no_details:xx	8831 , 8835 , 8840
\msg_error:nnxxxx	146 , 8967	\msg_interrupt_text:n	8831 , 8846 , 8854 , 8856
\msg_error_text:n	144 , 8911 , 8913 , 8972 , 8981 , 9111 , 9124	\msg_kernel_bug:x	9428 , 9429
\msg_expandable_error:n	150 , 9305 , 9313 , 9330	\msg_kernel_error:nn	1157 , 1171 , 7446 , 9025 , 9106 , 9139 , 9508 , 13620 , 13628 , 13633 , 13642
\msg_expandable_error_aux:w	9319 , 9325	\msg_kernel_error:nnx	1157 , 1169 , 1414 , 2499 , 4544 , 5457 , 6653 , 7168 , 7173 , 8286 , 8299 , 9034 , 9106 , 9137 , 9376 , 9608 , 9647 , 9662 , 9703 , 9942 , 10138 , 10211 , 13699 , 13741
\msg_expandable_kernel_error:nn	2199 , 5224 , 9328 , 9352	\msg_kernel_error:nnxx	1157 , 1157 , 1170 , 1172 , 1179 , 1189 , 1202 , 1324 , 7312 , 8758 , 9040 , 9106 , 9135 , 9599 , 9628 , 9673 , 9767 , 9952 , 9986 , 13736 , 13764
\msg_expandable_kernel_error:nnn	1572 , 2222 , 4728 , 9328 , 9347 , 13664 , 13669	\msg_kernel_error:nnxxx	9106 , 9133
\msg_expandable_kernel_error:nnnn	9328 , 9342	\msg_kernel_error:nnxxxx	149 , 9106 , 9106 , 9134 , 9136 , 9138 , 9140
\msg_expandable_kernel_error:nnnnn	9328 , 9337	\msg_kernel_fatal:nn	8312 , 8325 , 9086 , 9104 , 13720
\msg_expandable_kernel_error:nnnnnn	150 , 9328 , 9328 , 9339 , 9344 , 9349 , 9354	\msg_kernel_fatal:nnx	9086 , 9102 , 13693
\msg_fatal:nn	8945	\msg_kernel_fatal:nnxx	9086 , 9100
\msg_fatal:nnx	8945	\msg_kernel_fatal:nnxxx	9086 , 9098
\msg_fatal:nnxx	8945	\msg_kernel_fatal:nnxxxx	149 , 9086 , 9086 , 9099 , 9101 , 9103 , 9105
\msg_fatal:nnxxx	8945	\msg_kernel_info:nn	9141 , 9183
\msg_fatal:nnxxxx	145 , 8945	\msg_kernel_info:nnx	9141 , 9181
\msg_fatal_text:n	144 , 8911 , 8911 , 8948 , 9089	\msg_kernel_info:nnxx	9141 , 9179
\msg_generic_new:nn	9419 , 9421		
\msg_generic_new:nnn	9419 , 9420		
\msg_generic_set:nn	9419 , 9423		
\msg_generic_set:nnn	9419 , 9422		
\msg_gset:nnn	8754 , 8781		
\msg_gset:nnnn	143 , 8754 , 8761 , 8774 , 8782		
\msg_if_more_text:c	8934		
\msg_if_more_text:cTF	8969 , 9108		

\msg_kernel_info:nnxxx	9141, 9177	\msg_term:x	148, 8896, 8903, 8991, 9146
\msg_kernel_info:nnxxxx	149, 9141, 9164, 9178, 9180, 9182, 9184	\msg_trace:nn	9412, 9417
\msg_kernel_new:nnn	8186, 8194, 9078, 9080, 9267, 9269, 9271, 9273, 9275, 9282, 9289, 9297, 9299, 9301, 9303	\msg_trace:nnx	9412, 9416
\msg_kernel_new:nnnn	149, 8168, 8176, 8179, 8461, 8468, 8728, 9078, 9078, 9185, 9193, 9201, 9208, 9215, 9223, 9232, 9239, 9246, 9253, 9260, 9555, 10028, 10031, 10037, 10044, 10053, 10059, 10065, 10072, 10079, 10085, 13613, 13621, 13629, 13635, 13650	\msg_trace:nnxx	9412, 9415
\msg_kernel_set:nnn	9078, 9084	\msg_trace:nnxxx	9412, 9414
\msg_kernel_set:nnnn	149, 9078, 9082	\msg_trace:nnxxxx	9412, 9413
\msg_kernel_warning:nn	9141, 9161	\msg_two_newlines	147
\msg_kernel_warning:nnx	9141, 9159	\msg_two_newlines:	8815, 8816
\msg_kernel_warning:nnxx	9141, 9157	\msg_use:nnnnxxxx	8924, 9015, 9015, 9144, 9166
\msg_kernel_warning:nnxxx	9141, 9155	\msg_use_aux:nn	9015, 9048, 9050
\msg_kernel_warning:nnxxxx	149, 9141, 9142, 9156, 9158, 9160, 9162	\msg_use_aux:nnn	9015, 9039, 9042
\msg_line_context	144	\msg_use_code:	9015, 9017, 9057, 9065, 9069
\msg_line_context:	1173, 1173, 1192, 8817, 8818	\msg_use_loop:n	9015, 9022, 9070, 9071
\msg_line_number	144	\msg_use_loop:o	9015, 9066
\msg_line_number:	8817, 8817, 8822, 9556	\msg_use_loop_check:nn	9015, 9047, 9053, 9056, 9060
\msg_log:nn	9005, 9417	\msg_warning:nn	8989
\msg_log:nnx	9005, 9416	\msg_warning:nnx	8989
\msg_log:nnxx	9005, 9415	\msg_warning:nnxx	8989
\msg_log:nnxxx	9005, 9414	\msg_warning:nnxxx	8989
\msg_log:nnxxxx	146, 9005, 9413	\msg_warning:nnxxxx	146, 8989
\msg_log:x	148, 8896, 8896, 8999, 9007, 9168	\msg_warning_text:n	145, 8911, 8914, 8993, 9148
\msg_new:nnn	8754, 8763, 9081	\mskip	463
\msg_new:nnnn	143, 8754, 8754, 8764, 9079	\muexpr	712
\msg_newline	147	\multiply	364
\msg_newline:	8815, 8815, 8869	\muskip	660
\msg_no_more_text:xxxx	8934, 8936, 8940	\muskip_add:cn	4339
\msg_none:nn	9010	\muskip_add:Nn	80, 4339, 4339, 4341, 4342
\msg_none:nnx	9010	\muskip_eval:n	81, 4349, 4349
\msg_none:nnxx	9010	\muskip_gadd:cn	4339
\msg_none:nnxxx	9010	\muskip_gadd:Nn	80, 4339, 4341, 4343
\msg_none:nnxxxx	146, 9010	\muskip_gset:cn	4328
\msg_redirect_class:nn	147, 9072, 9072	\muskip_gset:Nn	80, 4328, 4330, 4332
\msg_redirect_module:nnn	147, 9074, 9074	\muskip_gset_eq:cc	4333
\msg_redirect_name:nnn	147, 9076, 9076	\muskip_gset_eq:cN	4333
\msg_see_documentation_text:n	8916, 8916, 8951, 8962, 8975, 8984, 9093, 9115, 9128, 9434	\muskip_gset_eq:Nc	4333
\msg_set:nnn	8754, 8772, 9085	\muskip_gset_eq:NN	80, 4333, 4336-4338
\msg_set:nnnn	143, 8754, 8765, 8773, 9083	\muskip_gsub:cn	4339
		\muskip_gsub:Nn	80, 4339, 4346, 4348
		\muskip_gzero:c	4317
		\muskip_gzero:N	80, 4317, 4319, 4321, 4325
		\muskip_gzero_new:c	4322
		\muskip_gzero_new:N	80, 4322, 4324, 4327
		\muskip_new:c	4309
		\muskip_new:N	80, 4309, 4310, 4316, 4323, 4325
		\muskip_set:cn	4328
		\muskip_set:Nn	80, 4328, 4328, 4330, 4331

\muskip_set_eq:cc	4333	\or:	785 , 787 , 1299–1307 , 1508 ,
\muskip_set_eq:cN	4333			3728–3752 , 12081 , 12083 , 12085 ,
\muskip_set_eq:Nc	4333			12087 , 12167 , 12169 , 12171 , 12173
\muskip_set_eq>NN	..	80 , 4333 , 4333–4335	\outer	369
\muskip_show:c	4353	\output	584
\muskip_show:N	81 , 4353 , 4353 , 4354	\outputpenalty	594
\muskip_show:n	81 , 4355 , 4355	\over	471
\muskip_sub:cn	4339	\overfullrule	622
\muskip_sub:Nn	..	80 , 4339 , 4344 , 4346 , 4347	\overline	502
\muskip_use:c	4351	\overwithdelims	472
\muskip_use:N	..	81 , 4350 , 4351 , 4351 , 4352			
\muskip_zero:c	4317			
\muskip_zero:N		P		
....	80 , 4317 , 4317 , 4319 , 4320 , 4323		\P	1830
\muskip_zero_new:c	4322	\package_check_loaded_expl:	783 , 1519 ,	
\muskip_zero_new:N	..	80 , 4322 , 4322 , 4326		1884 , 2396 , 2514 , 3299 , 4026 , 4375 ,	
\muskipdef	358		5220 , 5757 , 6234 , 6539 , 7110 , 8204 ,	
\mutoglu	718		8224 , 8749 , 9452 , 10101 , 10278 , 13648	
			\PackageError	219 , 235
N			\pagedepth	586
\name_primitive:NN	339 , 339 , 346–763	\pagediscards	727
\newbox	6545	\pagefilllstretch	590
\newcatcodetable	13709	\pagefillstretch	589
\newcount	3379	\pagefilstretch	588
\newdimen	4035	\pagegoal	592
\newlinechar	249 , 414	\pageshrink	591
\newmuskip	4313	\pagestretch	587
\newread	8272	\pagetotal	593
\newskip	4231	\par	542 , 6707 , 6708 ,
\newwrite	8271		6710 , 6712 , 6714 , 6719 , 6725 , 6738	
\noalign	382	\parfillskip	573
\noboundary	517	\parindent	566
\noexpand	35 , 39 , 40 , 166 , 169 , 172 , 174 ,		\parshape	558
	175 , 184 , 187–189 , 191 , 193 , 194 ,		\parshapedimen	708
	203 , 205–209 , 268 , 270 , 275 , 277 , 375		\parshapeindent	706
\noindent	543	\parshapelength	707
\nolimits	497	\parskip	565
\nonscript	477	\patterns	648
\nonstopmode	440	\pausing	435
\nulldelimiterspace	510	\pdf@strcmp	59
\nullfont	628	\pdfcolorstack	738
\number	637	\pdfcompresslevel	739
\numexpr	709	\pdfcreationdate	737
O			\pdfdecimaldigits	740
\O	1832	\pdfhorigin	741
\omit	383	\pdfinfo	742
\openin	409	\pdflastxform	743
\openout	410	\pdfliteral	744
\or	23 , 71 , 406	\pdfminorversion	745
			\pdfobjcompresslevel	746
			\pdfoutput	747

<code>\pdfpkresolution</code>	752	<code>\peek_charcode:N</code>	3101
<code>\pdfrefxform</code>	748	<code>\peek_charcode:NTF</code>	57
<code>\pdfrestore</code>	749	<code>\peek_charcode_ignore_spaces:N</code> ..	3101
<code>\pdfsave</code>	750	<code>\peek_charcode_ignore_spaces:NTF</code> ...	57
<code>\pdfsetmatrix</code>	751	<code>\peek_charcode_remove:N</code>	3101
<code>\pdfstrcmp</code> . 33, 59, 230, 235, 238, 252, 756		<code>\peek_charcode_remove:NTF</code>	57
<code>\pdftex_if_engine:</code>	1420	<code>\peek_charcode_remove_ignore_spaces:N</code>	3101
<code>\pdftex_if_engine:F</code>	1424, 1435, 1449	<code>\peek_charcode_remove_ignore_spaces:NTF</code>	57
<code>\pdftex_if_engine:T</code>	1423, 1434, 1448	<code>\peek_def:nnnn</code>	3068, 3069, 3085, 3089, 3093, 3097, 3101, 3105, 3109, 3113, 3117, 3121, 3125, 3129
<code>\pdftex_if_engine:TF</code> 1425, 1436, 1450, 3404		<code>\peek_def_aux:nnnnn</code> 3068, 3071–3073, 3075	
<code>\pdftex_if_engine_p:</code> 1430, 1440, 1452, 1502		<code>\peek_execute_branches:</code>	3064, 3080
<code>\pdftex_if_engineTF</code>	22	<code>\peek_execute_branches_catcode:</code>	3017, 3017, 3088, 3090, 3096, 3098
<code>\pdftex_pdfcolorstack:D</code>	738	<code>\peek_execute_branches_charcode:</code> ...	3034, 3034, 3104, 3106, 3112, 3114
<code>\pdftex_pdfcompresslevel:D</code>	739	<code>\peek_execute_branches_charcode:NN</code> 3034	
<code>\pdftex_pdfcreationdate:D</code>	737	<code>\peek_execute_branches_charcode_aux:NN</code>	3044, 3048
<code>\pdftex_pdfdecimaldigits:D</code>	740	<code>\peek_execute_branches_meaning:</code>	3017, 3026, 3120, 3122, 3128, 3130
<code>\pdftex_pdfhorigin:D</code>	741	<code>\peek_execute_branches_N_type:</code>	3184, 3184, 3196, 3198, 3200
<code>\pdftex_pdfinfo:D</code>	742	<code>\peek_false:w</code>	2964, 2966, 2987, 3005, 3023, 3031, 3042, 3053, 3192
<code>\pdftex_pdflastxform:D</code>	743	<code>\peek_gafter:NN</code>	3225, 3227
<code>\pdftex_pdfliteral:D</code>	744	<code>\peek_gafter:Nw</code>	56, 2970, 3227
<code>\pdftex_pdfminorversion:D</code>	745	<code>\peek_ignore_spaces_execute_branches:</code>	3057, 3057, 3067, 3092, 3100, 3108, 3116, 3124, 3132
<code>\pdftex_pdfobjcompresslevel:D</code>	746	<code>\peek_ignore_spaces_execute_branches_aux:</code>	3057, 3061, 3066
<code>\pdftex_pdfoutput:D</code>	747	<code>\peek_meaning:N</code>	3117
<code>\pdftex_pdfpkresolution:D</code>	752	<code>\peek_meaning:NTF</code>	57
<code>\pdftex_pdfrefxform:D</code>	748	<code>\peek_meaning_ignore_spaces:N</code> ...	3117
<code>\pdftex_pdfrestore:D</code>	749	<code>\peek_meaning_ignore_spaces:NTF</code>	58
<code>\pdftex_pdfsave:D</code>	750	<code>\peek_meaning_remove:N</code>	3117
<code>\pdftex_pdfsetmatrix:D</code>	751	<code>\peek_meaning_remove:NTF</code>	58
<code>\pdftex_pdftextrevision:D</code>	753	<code>\peek_meaning_remove_ignore_spaces:N</code>	3117
<code>\pdftex_pdfvorigin:D</code>	754	<code>\peek_meaning_remove_ignore_spaces:NTF</code>	58
<code>\pdftex_pdfxform:D</code>	755	<code>\peek_N_type:</code>	3184
<code>\pdftex_strcmp:D</code>	756, 1457, 1463, 2421, 2428, 2445, 2472, 2481, 2739, 4269, 4962, 10375, 10386	<code>\peek_N_type:F</code>	3199
<code>\pdftexrevision</code>	753	<code>\peek_N_type:T</code>	3197
<code>\pdfvorigin</code>	754	<code>\peek_N_type:TF</code>	3195
<code>\pdfxform</code>	755	<code>\peek_N_typeTF</code>	60
<code>\peek_after:NN</code>	3225, 3226		
<code>\peek_after:Nw</code>	56, 2968, 2968, 2993, 3011, 3067, 3226		
<code>\peek_catcode:N</code>	3085		
<code>\peek_catcode:NTF</code>	56		
<code>\peek_catcode_ignore_spaces:N</code> ...	3085		
<code>\peek_catcode_ignore_spaces:NTF</code>	56		
<code>\peek_catcode_remove:N</code>	3085		
<code>\peek_catcode_remove:NTF</code>	56		
<code>\peek_catcode_remove_ignore_spaces:N</code>	3085		
<code>\peek_catcode_remove_ignore_spaces:NTF</code>	57		

- \peek_tmp:w [2964](#), [2967](#), [2976](#), [3062](#)
- \peek_token_generic:NN [2978](#)
- \peek_token_generic:N NF [2997](#), [3200](#)
- \peek_token_generic:N NT [2995](#), [3198](#)
- \peek_token_generic:N NTF
..... [2978](#), [2996](#), [2998](#), [3196](#)
- \peek_token_remove_generic:NN ... [2999](#)
- \peek_token_remove_generic:N NF .. [3015](#)
- \peek_token_remove_generic:N NT .. [3013](#)
- \peek_token_remove_generic:N NTF
..... [2999](#), [3014](#), [3016](#)
- \peek_true:w [2964](#), [2964](#),
[2982](#), [3003](#), [3021](#), [3029](#), [3051](#), [3193](#)
- \peek_true_aux:w . [2964](#), [2965](#), [2975](#), [3004](#)
- \peek_true_remove:w [2972](#), [2972](#), [3003](#)
- \penalty [643](#)
- \postdisplaypenalty [490](#)
- \pdisplaydirection [734](#)
- \pdisplaypenalty [489](#)
- \pdisplaysize [488](#)
- \pretolerance [569](#)
- \prevdepth [616](#)
- \prevgraf [575](#)
- \prg_break_point:n [42](#), [1466](#),
[1466–1468](#), [2213](#), [2260](#), [2309](#), [4681](#),
[4699](#), [4708](#), [5153](#), [5363](#), [5376](#), [5389](#),
[5402](#), [5430](#), [5465](#), [5500](#), [5511](#), [5562](#),
[5569](#), [5582](#), [5589](#), [5596](#), [5603](#), [5641](#),
[5660](#), [5731](#), [5993](#), [6007](#), [6026](#), [6043](#),
[6372](#), [6418](#), [6439](#), [6480](#), [6496](#), [8713](#)
- \prg_case_dim:nnn [39](#), [2121](#), [2121](#)
- \prg_case_dim_aux:nnn .. [2121](#), [2124](#), [2126](#)
- \prg_case_dim_aux:nw [2121](#), [2127](#), [2128](#), [2132](#)
- \prg_case_end:nw [2107](#),
[2107](#), [2118](#), [2131](#), [2142](#), [2154](#), [2165](#)
- \prg_case_int:nnn [38](#), [2108](#), [2108](#), [3617](#), [3623](#)
- \prg_case_int_aux:nnn .. [2108](#), [2111](#), [2113](#)
- \prg_case_int_aux:nw [2108](#), [2114](#), [2115](#), [2119](#)
- \prg_case_str:nnn [39](#), [2134](#), [2134](#), [2145](#), [5131](#)
- \prg_case_str:onn [2134](#)
- \prg_case_str:xxn [2134](#), [2146](#)
- \prg_case_str_aux:nw [2134](#), [2137](#), [2139](#), [2143](#)
- \prg_case_str_x_aux:nw
..... [2134](#), [2149](#), [2151](#), [2155](#)
- \prg_case_tl:cnn [2157](#)
- \prg_case_tl:Nnn ... [39](#), [2157](#), [2157](#), [2168](#)
- \prg_case_tl_aux:Nw [2157](#), [2160](#), [2162](#), [2166](#)
- \prg_conditional_form_F:nnn [1040](#)
- \prg_conditional_form_p:nnn [1037](#)
- \prg_conditional_form_T:nnn [1039](#)
- \prg_conditional_form_TF:nnn [1038](#)
- \prg_define_quicksort:nnn [2309](#), [2309](#), [2384](#)
- \prg_do_nothing [9](#)
- \prg_do_nothing: [1454](#), [1454](#),
[4497](#), [4511](#), [4569](#), [4574](#), [5258](#), [5265](#),
[6145](#), [6149](#), [6156](#), [9392](#), [10381](#), [10392](#)
- \prg_generate_conditional_aux:nnNNnnnn
..... [927](#), [942](#), [951](#), [951](#)
- \prg_generate_conditional_aux:nnw ..
..... [951](#), [953](#), [959](#), [965](#)
- \prg_generate_conditional_count_aux:NNnn
..... [930](#), [931](#), [933](#), [935](#), [937](#), [938](#)
- \prg_generate_conditional_parm_aux:NNpnn
..... [917](#), [918](#), [920](#), [922](#), [924](#), [925](#)
- \prg_generate_F_form_count:Nnnnn
..... [996](#), [1012](#)
- \prg_generate_F_form_parm:Nnnnn [967](#), [983](#)
- \prg_generate_p_form_count:Nnnnn
..... [996](#), [996](#)
- \prg_generate_p_form_parm:Nnnnn [967](#), [967](#)
- \prg_generate_T_form_count:Nnnnn
..... [996](#), [1004](#)
- \prg_generate_T_form_parm:Nnnnn [967](#), [975](#)
- \prg_generate_TF_form_count:Nnnnn ..
..... [996](#), [1020](#)
- \prg_generate_TF_form_parm:Nnnnn
..... [967](#), [991](#)
- \prg_map_break:
... [1466](#), [1467](#), [2223](#), [2309](#), [2440](#),
[2447](#), [4720](#), [5450](#), [5452](#), [6059](#), [6442](#)
- \prg_map_break:n [42](#), [1466](#), [1468](#), [2309](#),
[4721](#), [5451](#), [5453](#), [6060](#), [6443](#), [6501](#)
- \prg_new_conditional:Nnn [930](#),
[932](#), [1888](#), [2450](#), [2458](#), [2470](#), [2479](#), [8672](#)
- \prg_new_conditional:Npnn .. [33](#), [917](#),
[919](#), [1392](#), [1455](#), [1461](#), [1888](#), [1916](#),
[1930](#), [2262](#), [2264](#), [2266](#), [2268](#), [2650](#),
[2655](#), [2660](#), [2665](#), [2672](#), [2678](#), [2683](#),
[2688](#), [2693](#), [2698](#), [2703](#), [2708](#), [2713](#),
[2718](#), [2732](#), [2746](#), [2751](#), [2771](#), [2780](#),
[2790](#), [2808](#), [2832](#), [2850](#), [2868](#), [2880](#),
[2889](#), [2909](#), [3460](#), [3530](#), [3538](#), [3546](#),
[4098](#), [4103](#), [4266](#), [4276](#), [4590](#), [4600](#),
[4612](#), [4633](#), [4635](#), [4829](#), [4845](#), [4861](#),
[4895](#), [4901](#), [4916](#), [4972](#), [4974](#), [5164](#),
[6197](#), [6354](#), [6366](#), [6611](#), [6613](#), [6623](#),
[8934](#), [9973](#), [10014](#), [10020](#), [13341](#), [13349](#)
- \prg_new_eq_conditional:Nnn
..... [35](#), [947](#), [949](#), [1888](#),
[5345](#), [5347](#), [5956–5961](#), [6529–6532](#)

- \prg_new_map_functions:Nn ... [2387](#), [2388](#)
- \prg_new_protected_conditional:Nnn .
 - [930](#), [936](#), [1888](#), [10181](#)
- \prg_new_protected_conditional:Npnn
 - [33](#), [917](#), [923](#),
[1888](#), [4647](#), [4668](#), [5349](#), [5556](#), [5564](#),
[5577](#), [5584](#), [5591](#), [5598](#), [5962](#), [5966](#),
[6397](#), [6452](#), [6458](#), [13357](#), [13374](#), [13561](#)
- \prg_quicksort:n [2384](#)
- \prg_quicksort_compare:nnTF . [2385](#), [2386](#)
- \prg_quicksort_function:n ... [2385](#), [2385](#)
- \prg_replicate:nn [40](#), [2169](#),
[2169](#), [8646](#), [9274](#), [10705](#), [10741](#), [10811](#)
- \prg_replicate_ [2169](#), [2180](#)
- \prg_replicate_0:n [2169](#)
- \prg_replicate_1:n [2169](#)
- \prg_replicate_2:n [2169](#)
- \prg_replicate_3:n [2169](#)
- \prg_replicate_4:n [2169](#)
- \prg_replicate_5:n [2169](#)
- \prg_replicate_6:n [2169](#)
- \prg_replicate_7:n [2169](#)
- \prg_replicate_8:n [2169](#)
- \prg_replicate_9:n [2169](#)
- \prg_replicate_aux:N [2169](#), [2176](#), [2177](#), [2179](#)
- \prg_replicate_first_ -:n [2169](#)
- \prg_replicate_first_0:n [2169](#)
- \prg_replicate_first_1:n [2169](#)
- \prg_replicate_first_2:n [2169](#)
- \prg_replicate_first_3:n [2169](#)
- \prg_replicate_first_4:n [2169](#)
- \prg_replicate_first_5:n [2169](#)
- \prg_replicate_first_6:n [2169](#)
- \prg_replicate_first_7:n [2169](#)
- \prg_replicate_first_8:n [2169](#)
- \prg_replicate_first_9:n [2169](#)
- \prg_replicate_first_aux:N
..... [2169](#), [2172](#), [2178](#)
- \prg_return_false [35](#)
- \prg_return_false: ... [913](#), [915](#), [1084](#),
[1089](#), [1102](#), [1107](#), [1115](#), [1132](#), [1395](#),
[1459](#), [1464](#), [1888](#), [1921](#), [1935](#), [2263](#),
[2265](#), [2267](#), [2269](#), [2455](#), [2463](#), [2476](#),
[2485](#), [2653](#), [2658](#), [2663](#), [2668](#), [2675](#),
[2681](#), [2686](#), [2691](#), [2696](#), [2701](#), [2706](#),
[2711](#), [2716](#), [2721](#), [2742](#), [2749](#), [2756](#),
[2758](#), [2793](#), [2796](#), [2815](#), [2818](#), [2835](#),
[2838](#), [2853](#), [2856](#), [2912](#), [2931](#), [2948](#),
[2957](#), [3479](#), [3487](#), [3493](#), [3503](#), [3511](#),
[3517](#), [3525](#), [3535](#), [3543](#), [3549](#), [4101](#),
[4109](#), [4273](#), [4284](#), [4605](#), [4617](#), [4630](#),
[4640](#), [4657](#), [4672](#), [4838](#), [4858](#), [4873](#),
[4881](#), [4891](#), [4911](#), [4925](#), [4965](#), [5362](#),
[5552](#), [5976](#), [6200](#), [6359](#), [6385](#), [6401](#),
[6456](#), [6462](#), [6612](#), [6614](#), [6624](#), [8682](#),
[8937](#), [9978](#), [10018](#), [10024](#), [10185](#),
[13346](#), [13354](#), [13391](#), [13405](#), [13409](#),
[13413](#), [13417](#), [13429](#), [13433](#), [13448](#),
[13463](#), [13481](#), [13490](#), [13498](#), [13513](#),
[13522](#), [13530](#), [13575](#), [13581](#), [13587](#),
[13593](#), [13599](#), [13605](#), [13610](#), [13611](#)
- \prg_return_true [35](#)
- \prg_return_true: [913](#), [913](#), [1087](#), [1104](#),
[1112](#), [1117](#), [1130](#), [1135](#), [1395](#), [1459](#),
[1464](#), [1888](#), [1919](#), [1933](#), [2263](#), [2265](#),
[2267](#), [2269](#), [2453](#), [2461](#), [2474](#), [2483](#),
[2653](#), [2658](#), [2663](#), [2668](#), [2675](#), [2681](#),
[2686](#), [2691](#), [2696](#), [2701](#), [2706](#), [2711](#),
[2716](#), [2721](#), [2740](#), [2749](#), [2756](#), [2810](#),
[2812](#), [2929](#), [2955](#), [3477](#), [3485](#), [3495](#),
[3501](#), [3509](#), [3519](#), [3527](#), [3533](#), [3541](#),
[3551](#), [4101](#), [4107](#), [4271](#), [4283](#), [4603](#),
[4615](#), [4628](#), [4638](#), [4654](#), [4672](#), [4836](#),
[4856](#), [4871](#), [4889](#), [4913](#), [4924](#), [4963](#),
[5365](#), [5560](#), [5568](#), [5581](#), [5588](#), [5595](#),
[5602](#), [5976](#), [6201](#), [6357](#), [6383](#), [6406](#),
[6468](#), [6612](#), [6614](#), [6624](#), [8677](#), [8680](#),
[8686](#), [8938](#), [9977](#), [10017](#), [10023](#),
[10186](#), [13344](#), [13352](#), [13402](#), [13436](#),
[13445](#), [13459](#), [13477](#), [13485](#), [13495](#),
[13509](#), [13517](#), [13527](#), [13575](#), [13581](#),
[13587](#), [13593](#), [13599](#), [13605](#), [13611](#)
- \prg_set_conditional:Nnn . [930](#), [930](#), [1888](#)
- \prg_set_conditional:Npnn [33](#),
[917](#), [917](#), [1081](#), [1093](#), [1109](#), [1121](#), [1888](#)
- \prg_set_eq_conditional:NNn
..... [35](#), [947](#), [947](#), [1888](#)
- \prg_set_eq_conditional_aux:NNNn ...
..... [948](#), [950](#), [1025](#), [1025](#)
- \prg_set_eq_conditional_aux:NNNw ...
..... [1025](#), [1026](#), [1027](#), [1035](#)
- \prg_set_map_functions:Nn ... [2387](#), [2389](#)
- \prg_set_protected_conditional:Nnn .
..... [930](#), [934](#), [1888](#)
- \prg_set_protected_conditional:Npnn
..... [33](#), [917](#), [921](#), [1888](#)
- \prg_stepwise_aux:nnnn
..... [2210](#), [2212](#), [2215](#), [2259](#)
- \prg_stepwise_aux:NnnnN
..... [2210](#), [2218](#), [2225](#), [2229](#), [2234](#)

<code>\prg_stepwise_aux:NNnnnn</code>	<code>\prop_get_aux_true:Nnnn</code> 6397 , 6400 , 6403
..... 2238 , 2240 , 2246 , 2255	<code>\prop_get_gdel:NnN</code>
<code>\prg_stepwise_function:nnnN</code>	6517 , 6518
..... 40 , 2210 , 2210	<code>\prop_get_Nn_aux:nwn</code> 6491 , 6493 , 6498 , 6502
<code>\prg_stepwise_inline:nnnn</code>	<code>\prop_gget:cnN</code>
..... 40 , 2238 , 2238 , 13798 , 13803	6509
<code>\prg_stepwise_variable:nnnNn</code>	<code>\prop_gget:cVN</code>
..... 40 , 2238 , 2244	6509
<code>\prg_variable_get_scope:N</code> 41 , 2275 , 2281	<code>\prop_gget:NnN</code> ... 6509 , 6510 , 6514 , 6515
<code>\prg_variable_get_scope_aux:w</code>	<code>\prop_gget:NVN</code>
..... 2275 , 2284 , 2287	6509
<code>\prg_variable_get_type:N</code> . 42 , 2275 , 2296	<code>\prop_gget_aux:Nnnn</code> ... 6509 , 6511 , 6512
<code>\prg_variable_get_type:w</code>	<code>\prop_gpop:cnN</code>
2275	6296 , 6452
<code>\prg_variable_get_type_aux:w</code>	<code>\prop_gpop:coN</code>
..... 2298 , 2301 , 2305	6296
<code>\prop_clear:c</code>	<code>\prop_gpop:NnN</code>
6240 , 6241 , 7566	116 , 6296 , 6302 , 6315 , 6316 , 6458 , 6518
<code>\prop_clear:N</code> 115 , 6240 , 6240 , 6245	<code>\prop_gpop:NnNF</code>
<code>\prop_clear_new:c</code> 6244 , 7189 , 7190 , 8922	6474
<code>\prop_clear_new:N</code> . 115 , 6244 , 6244 , 6246	<code>\prop_gpop:NnNT</code>
<code>\prop_del:cn</code>	6473
6276	<code>\prop_gpop:NnNTF</code>
<code>\prop_del:cV</code>	119 , 6475
6276	<code>\prop_gpop:NoN</code>
<code>\prop_del:Nn</code>	6296
117 , 6276 , 6276 , 6282 , 6283 , 8058 , 8061 , 8065	<code>\prop_gput:ccx</code>
<code>\prop_del:NV</code>	6525
6276	<code>\prop_gput:cnN</code>
<code>\prop_del_aux:NNnnn</code> 6276 , 6277 , 6279 , 6280	6317
<code>\prop_display:c</code>	<code>\prop_gput:cno</code>
6505 , 6507	6317
<code>\prop_display:N</code>	<code>\prop_gput:cnV</code>
6505 , 6506	6317
<code>\prop_gclear:c</code>	<code>\prop_gput:cnx</code>
6240 , 6243	6317
<code>\prop_gclear:N</code> 115 , 6240 , 6242 , 6248	<code>\prop_gput:con</code>
<code>\prop_gclear_new:c</code>	6317
6244	<code>\prop_gput:coo</code>
<code>\prop_gclear_new:N</code> . 115 , 6244 , 6247 , 6249	6317
<code>\prop_gdel:cn</code>	<code>\prop_gput:cVn</code>
6276	6317
<code>\prop_gdel:cV</code>	<code>\prop_gput:cVV</code>
6276	6317
<code>\prop_gdel:Nn</code> . 117 , 6276 , 6278 , 6284 , 6285	<code>\prop_gput:Nnn</code>
<code>\prop_gdel:NV</code> 116 , 6317 , 6318 , 6335 , 6337 , 6526
6276 , 8430 , 8443	<code>\prop_gput:Nno</code>
<code>\prop_get:cn</code>	6317
6491	<code>\prop_gput:NnV</code>
<code>\prop_get:cnN</code>	6317
6286 , 6397 , 9062	<code>\prop_gput:Nnx</code>
<code>\prop_get:cnNF</code>	6317
7309	<code>\prop_gput:Non</code>
<code>\prop_get:coN</code>	6317
6397	<code>\prop_gput:Noo</code>
<code>\prop_get:cVN</code>	6317
6286 , 6397	<code>\prop_gput:NVn</code>
<code>\prop_get:Nn</code> 120 , 6491 , 6491 , 6504	6317 , 8315 , 8328
<code>\prop_get:NnN</code>	<code>\prop_gput:NVV</code>
... 116 , 6286 , 6286 , 6294 , 6295 , 6397 , 6397 , 8011 , 8015 , 8094 , 8098	6317
<code>\prop_get:NnNF</code>	<code>\prop_gput_if_new:cnN</code>
6409 , 6412	6339
<code>\prop_get:NnNT</code>	<code>\prop_gput_if_new:Nnn</code> 116 , 6339 , 6341 , 6353
6408 , 6411	<code>\prop_gset_eq:cc</code> . 6250 , 6257 , 7333 , 7335
<code>\prop_get:NnNTF</code>	<code>\prop_gset_eq:cN</code> . 6250 , 6256 , 7191 , 7193
117 , 6410 , 6413	<code>\prop_gset_eq:Nc</code>
<code>\prop_get:NoN</code>	6250 , 6255
6286 , 6397	<code>\prop_gset_eq:NN</code>
<code>\prop_get:NVN</code>	115 , 6250 , 6254
6286 , 6397	<code>\prop_if_empty:c</code>
<code>\prop_get_aux:Nnnn</code> 6286 , 6289 , 6292	6354
	<code>\prop_if_empty:N</code>
	6354 , 6354
	<code>\prop_if_empty:NF</code>
	6365
	<code>\prop_if_empty:NT</code>
	6364
	<code>\prop_if_empty:NTF</code> . 117 , 6363 , 8457 , 9285
	<code>\prop_if_empty_p:N</code>
	6362
	<code>\prop_if_eq:cc</code>
	6528 , 6532
	<code>\prop_if_eq:cN</code>
	6528 , 6530
	<code>\prop_if_eq:Nc</code>
	6528 , 6531
	<code>\prop_if_eq:NN</code>
	6528 , 6529

<code>\prop_if_in:cc</code>	6520	<code>\prop_put:cnn</code>	6317 , 7387 , 9073 , 9075
<code>\prop_if_in:cn</code>	6366	<code>\prop_put:cno</code>	6317
<code>\prop_if_in:cnTF</code>	9052 , 9055	<code>\prop_put:cnV</code>	6317
<code>\prop_if_in:co</code>	6366	<code>\prop_put:cnx</code>	
<code>\prop_if_in:cV</code>	6366	...	6317 , 7393 , 7395 , 7397 , 7399 ,
<code>\prop_if_in:Nn</code>	6366 , 6366		7404 , 7409 , 7414 , 7421 , 7428 , 7661 ,
<code>\prop_if_in:NnF</code> 6393 , 6394 , 6522 , 8334 , 8342			7774 , 7832 , 7840 , 7907 , 7921 , 7928
<code>\prop_if_in:NnT</code>	6391 , 6392 , 6521	<code>\prop_put:con</code>	6317
<code>\prop_if_in:NnTF</code> 117 , 6395 , 6396 , 6523 , 9046		<code>\prop_put:coo</code>	6317
<code>\prop_if_in:No</code>	6366	<code>\prop_put:cVn</code>	6317
<code>\prop_if_in:NV</code>	6366	<code>\prop_put:cVV</code>	6317
<code>\prop_if_in:NVT</code>	8379 , 8415	<code>\prop_put:Nnn</code> ..	116 , 6317 , 6317 , 6331 ,
<code>\prop_if_in_aux:N</code>	6366 , 6377 , 6380		6333 , 7117 – 7120 , 7938 , 7940 , 7942 ,
<code>\prop_if_in_aux:nwn</code> 6366 , 6368 , 6374 , 6378			7944 , 7946 , 7948 , 7950 , 7952 , 7954 ,
<code>\prop_if_in_p:Nn</code>	6389 , 6390		7956 , 7958 , 7960 , 7962 , 7964 , 7966 ,
<code>\prop_map_break</code>	118		7968 , 7970 , 7972 , 8253 – 8256 , 9077
<code>\prop_map_break:</code>		<code>\prop_put:Nno</code> 6317 , 7123 – 7125 , 7127 – 7132	
.....	6387 , 6423 , 6442 , 6442 , 6485	<code>\prop_put:NnV</code>	6317
<code>\prop_map_break:n</code>	119 , 6442 , 6443	<code>\prop_put:Nnx</code>	
<code>\prop_map_function:cc</code>	6414	..	6317 , 7755 , 7757 , 7760 , 7762 , 7768
<code>\prop_map_function:cN</code>	6414 , 8157	<code>\prop_put:Non</code>	6317
<code>\prop_map_function:Nc</code>	6414	<code>\prop_put:Noo</code>	6317
<code>\prop_map_function:NN</code>		<code>\prop_put:NVn</code>	6317
118 , 6414 , 6414 , 6428 , 6429 , 6449 , 8459		<code>\prop_put:NVV</code>	6317
<code>\prop_map_function_aux:Nwn</code>		<code>\prop_put_aux:NNnn</code>	6317 – 6319
.....	6414 , 6416 , 6420 , 6426 , 6435	<code>\prop_put_aux:NNnnnnn</code> ..	6317 , 6321 , 6323
<code>\prop_map_inline:cn</code>		<code>\prop_put_if_new:cnn</code>	6339
6430 , 7637 , 7656 , 7725 , 7727 , 7747 ,		<code>\prop_put_if_new:Nnn</code> 116 , 6339 , 6339 , 6352	
7749 , 7812 , 7866 , 7868 , 7872 , 7874		<code>\prop_put_if_new_aux:NNnn</code> 6340 , 6342 , 6343	
<code>\prop_map_inline:Nn</code>	118 , 6430 ,	<code>\prop_set_eq:cc</code> 6250 , 6253 , 7326 , 7328 , 7596	
6430 , 6441 , 7730 , 7825 , 8063 , 8072		<code>\prop_set_eq:cN</code> ..	6250 , 6252 , 7319 , 7321
<code>\prop_map_tokens:cn</code>	6476	<code>\prop_set_eq:Nc</code>	6250 , 6251 , 8053
<code>\prop_map_tokens:Nn</code> 119 , 6476 , 6476 , 6490		<code>\prop_set_eq:NN</code>	115 , 6250 , 6250
<code>\prop_map_tokens_aux:nwn</code>		<code>\prop_show:c</code>	6444 , 6507
.....	6476 , 6478 , 6482 , 6488	<code>\prop_show:N</code> ..	119 , 6444 , 6444 , 6451 , 6506
<code>\prop_new:c</code>	6238 , 6239	<code>\prop_split:Nnn</code> 120 , 6270 , 6270 , 6321 , 6511	
<code>\prop_new:N</code>	115 , 6238 , 6238 , 6245 ,	<code>\prop_split:NnTF</code>	120 ,
6248 , 7116 , 7121 , 7712 , 7937 , 7978 ,			6258 , 6258 , 6272 , 6277 , 6279 , 6288 ,
8250 , 8251 , 8918 , 8919 , 9141 , 9163			6298 , 6304 , 6345 , 6399 , 6454 , 6460
<code>\prop_pop:cnN</code>	6296 , 6452	<code>\prop_split_aux:nnnn</code> ..	6258 , 6264 , 6268
<code>\prop_pop:coN</code>	6296	<code>\prop_split_aux:NnTF</code> ..	6258 , 6259 , 6260
<code>\prop_pop:NnN</code>		<code>\prop_split_aux:w</code> 6258 , 6262 , 6265 , 6269	
116 , 6296 , 6296 , 6313 , 6314 , 6452 , 6452		<code>\protect</code>	235
<code>\prop_pop:NnNF</code>	6471	<code>\protected</code>	68 , 82 , 98 ,
<code>\prop_pop:NnNT</code>	6470		104 , 126 , 133 , 141 , 143 , 147 , 152 ,
<code>\prop_pop:NnNTF</code>	118 , 6472		157 , 213 , 266 , 273 , 292 , 325 , 736 , 2897
<code>\prop_pop:NoN</code>	6296	<code>\protected@edef</code>	8562 , 8864
<code>\prop_pop_aux:NNNnnn</code> 6296 , 6299 , 6305 , 6308		<code>\ProvidesClass</code>	154
<code>\prop_pop_aux_true:NNNnnn</code>		<code>\ProvidesExplClass</code>	6 , 146 , 152
.....	6452 , 6455 , 6461 , 6464	<code>\ProvidesExplFile</code>	6 , 146 , 157

<code>\ProvidesExplPackage</code>	3909, 3946, 4195, 4200, 4561, 4576, 4749, 4751, 4756, 4758, 4774, 4795, 4805, 4806, 4808, 4810, 4811, 4818, 4826, 4828, 4834, 4850, 4870, 5211, 5213, 5375, 5378, 5388, 5391, 5559, 5580, 5587, 5668, 5670, 5675, 5806, 5811, 5868, 5869, 5878, 5880, 5940, 6127, 6160, 6202, 6210, 6263, 6266, 8571, 8660, 9488, 9493, 9495, 9506, 9511, 9513, 9536, 9539, 9607, 9610, 9616, 9625, 9635, 10221, 10223, 10226, 10229, 10232, 10249, 10351, 10352, 10371, 10376, 10381, 10387, 10392, 10398, 10404, 10406, 10407, 10410, 10414, 10418, 10454, 10459, 10682, 10684, 10699, 10701, 10702, 10708, 10715, 10717, 10720, 10722, 10723, 10725, 10727, 10729, 10731, 10733, 10735, 10737, 10738, 10747, 10749, 10759, 10761, 10772, 10779, 10781–10783, 10785, 10787, 10789, 10791, 10793, 10795, 10797, 10799, 10808, 10814, 10816, 10826, 10828, 10835, 10837–10839, 10845, 10850, 10855, 10860, 10865, 10870, 10875, 10880, 10890, 10895, 10896, 10907, 10909, 10928, 10948, 11418, 11423, 11535, 11588, 11765, 11770, 11777, 11780, 13567, 13571, 13574, 13577, 13580, 13583, 13586, 13589, 13592, 13595, 13598, 13601, 13604, 13607
<code>\ProvidesFile</code>	159
<code>\ProvidesPackage</code>	47, 149
Q	
<code>\q</code>	2020, 2025
<code>\q_mark</code> <i>43</i> , 1844, 1846, 1850, <u>2399</u> , 2400, 3470, 3472, 4552, 4561, 4565, 4576, 4747, 4748, 4751, 4754, 4755, 4766, 4769, 4770, 4776, 4780, 4782, 4784, 4810, 4811, 5801, 5802, 5818, 5827, 5832, 5934, 5940, 5953, 6006, 6014, 6200, 6201, 6210, 6263, 6265, 6266	
<code>\q_nil</code> 898, 901, 2312, 2316, <u>2399</u> , 2399, 2452, 2473, 3841, 3863, 4614, 4626, 4627, 4768, 4772, 4789, 4792, 4795, 4834, 4850, 4870, 5800, 5804, 5811, 5878, 5887, 9476, 9507, 9527, 9534, 9539, 10221, 10226, 13567, 13574, 13580, 13586, 13592, 13598, 13604	
<code>\q_no_value</code>	<i>43</i> , 2049, <u>2399</u> , 2401, 2460, 2482, 6274, 6290, 6300, 6306, 9476, 9484, 9489, 9506, 9512, 9743
<code>\q_prop</code>	<i>120</i> , <u>6236</u> , 6236, 6237, 6263, 6264, 6266, 6328, 6349, 6370, 6374, 6382, 6417, 6420, 6438, 6479, 6482, 6495, 6498
<code>\q_recursion_stop</code> <i>44</i> , 900, 903, 957, 1026, 1783, 2107, 2114, 2127, 2137, 2149, 2160, <u>2403</u> , 2404, 5819, 6042, 6096
<code>\q_recursion_tail</code>	<u>2403</u> , 2403, 2407, 2413, 2422, 2429, 2439, 2446, 4680, 4698, 4707, 5152, 5819, 5992, 6006, 6025, 6042, 6096, 6371, 6417, 6422, 6438, 6479, 6484
<code>\q_stop</code> . <i>43</i> , 899, 902, 1062, 1064, 1072, 1074, 1288, 1292, 1805, 1847, 1850, 2049, 2052, 2285, 2287, 2299, 2301, 2305, 2312, 2316, 2378, <u>2399</u> , 2402, 2735, 2737, 2776, 2785, 2789, 2801, 2807, 2823, 2831, 2843, 2849, 2861, 2867, 2874, 2879, 2885, 2895, 2899, 2915, 2918, 2921, 2943, 3136, 3143, 3152, 3161, 3461, 3462, 3470, 3474, 3482, 3490, 3498, 3506, 3514, 3522,	
<code>\q_tl_act_mark</code> <i>97</i> , <u>2492</u> , 2492, <u>4980</u> , 4984, 5001
<code>\q_tl_act_stop</code>	<i>97</i> , <u>2492</u> , 2493, <u>4980</u> , 4984, 4988, 4997, 4999, 5005, 5010, 5013, 5017, 5020
<code>\quark_if_nil:N</code>	<u>2450</u> , 2450
<code>\quark_if_nil:n</code>	<u>2470</u> , 2470
<code>\quark_if_nil:nF</code>	2491
<code>\quark_if_nil:nT</code>	2319, 2323, 2490
<code>\quark_if_nil:NTF</code> ..	<i>44</i> , 3844, 3866, 9531
<code>\quark_if_nil:nTF</code> <i>44</i> , 2327, 2336, 2345, 2354, 2489, 5883, 10225, 10234, 13573, 13579, 13585, 13591, 13597, 13603, 13609
<code>\quark_if_nil:o</code>	<u>2470</u>
<code>\quark_if_nil:oF</code>	9486
<code>\quark_if_nil:V</code>	<u>2470</u>
<code>\quark_if_nil_p:n</code>	2488
<code>\quark_if_no_value:c</code>	<u>2450</u>

\quark_if_no_value:cF	9963	\savingvdiscards	726
\quark_if_no_value:N	2458	\scan_align_safe_stop	41
\quark_if_no_value:n	2470, 2479	\scan_align_safe_stop: ..	2274, 2274, 3985
\quark_if_no_value:N.	2450	\scan_new:N	45, 2495, 2495, 2507
\quark_if_no_value:NF	2468	\scan_stop	9
\quark_if_no_value:NT	2467	\scan_stop:	308,
\quark_if_no_value:NTF			322, 810, 810, 1029, 1053, 1083,
44, 2054, 2469, 8013, 8017, 8096, 8100			1101, 1111, 1129, 1558, 1828–1836,
\quark_if_no_value:nTF	44		2276, 2291, 2504, 2671, 2748, 3145,
\quark_if_no_value_p:N	2466		3154, 3163, 3196, 3198, 3200, 4246,
\quark_if_recursion_tail_break:N ...			4257, 4262, 4287, 4292, 4295, 4301,
.....	45, 2437, 2437, 4715		4329, 4340, 4345, 4350, 4356, 4366,
\quark_if_recursion_tail_break:n ...			4367, 4483–4486, 4826, 6671, 7991,
..	2437, 2443, 4687, 5157, 5998, 6011		8046, 8316, 8329, 10359–10361,
\quark_if_recursion_tail_stop:N			10401, 10412, 10420, 10425, 10461,
.....	44, 2405, 2405, 6054		10462, 10478, 10486, 10525, 10534,
\quark_if_recursion_tail_stop:n			10550, 10559, 10628, 11123, 11135,
....	45, 2419, 2419, 2435, 5823, 6101		11136, 11268, 11272, 11284, 11288,
\quark_if_recursion_tail_stop:o ..	2419		11301, 11305, 11347, 11408, 11412,
\quark_if_recursion_tail_stop_do:Nn			11419–11421, 11429, 11440, 11490,
.....	45, 2405, 2411		11592, 11667, 11675, 11717, 11731,
\quark_if_recursion_tail_stop_do:nn			11735, 11773, 11774, 11783, 11784,
.....	45, 2419, 2426, 2436		11801, 11809, 11817, 11833, 11847,
\quark_if_recursion_tail_stop_do:on			11860, 12215, 12538, 12548, 12592,
.....	2419		12668–12671, 12692, 12893, 12919,
\quark_new:N			12956, 12964, 12972, 13054, 13056,
43, 2398, 2398–2404, 2492, 2493, 6236			13058, 13093, 13101, 13305, 13307,
			13309, 13311, 13313, 13327, 13730
R			
\R	1831	\scantokens	684
\radical	464	\scriptfont	630
\raise	604	\scriptscriptfont	631
\read	411	\scriptscriptstyle	476
\readline	686	\scriptspace	516
\relax	3–6, 9, 13,	\scriptstyle	475
62, 70–80, 84–93, 96, 101, 131, 133,		\scrollmode	441
141, 143, 229, 233, 249, 281–289, 446		\seq_break	106
\relpenalty	507	\seq_break:	5413, 5445, 5450,
\RequirePackage	57, 58		5450, 5458, 5553, 5561, 5568, 5581,
\reverse_if:N 23, 785, 790, 4129–4131, 4825			5588, 5595, 5602, 5639, 5659, 5667
\right	505	\seq_break:n	
\righthyphenmin	561	...	106, 5362, 5365, 5450, 5451, 5647
\rightskip	563	\seq_clear:c	5231, 5232
\romannumeral	638	\seq_clear:N ...	98, 5231, 5231, 5309, 9019
\rule	7995, 8050	\seq_clear_new:c	5235, 5236
		\seq_clear_new:N	98, 5235, 5235
S			
\s_stop	45, 2507, 2507, 2508	\seq_concat:ccc	5279
\savecatcodetable	763	\seq_concat:NNN 99, 5279, 5279, 5283, 10162	
\savinghyphcodes	725	\seq_display:c	5748, 5750
		\seq_display:N	5748, 5749
		\seq_gclear:c	5231, 5234

<code>\seq_gclear:N</code>	98 , 5231 , 5233	<code>\seq_gput_left:co</code>	5293 , 5533
<code>\seq_gclear_new:c</code>	5235 , 5238	<code>\seq_gput_left:cV</code>	5293 , 5531
<code>\seq_gclear_new:N</code>	98 , 5235 , 5237	<code>\seq_gput_left:cv</code>	5293 , 5532
<code>\seq_gconcat:ccc</code>	5279	<code>\seq_gput_left:cx</code>	5293 , 5534
<code>\seq_gconcat:NNN</code> 99 , 5279 , 5281 , 5284 , 10270		<code>\seq_gput_left:Nn</code>	
<code>\seq_get:cN</code>	5535 , 5536	99 , 5293 , 5293 , 5297 , 5298 , 5525
<code>\seq_get:NN</code>	102 , 5535 , 5535	<code>\seq_gput_left:No</code>	5293 , 5528
<code>\seq_get_left:cN</code>	5372 , 5536 , 5556 , 5746	<code>\seq_gput_left:NV</code>	5293 , 5526
<code>\seq_get_left:NN</code>	99 , 5372 , 5372 , 5380 , 5535 , 5556 , 5556 , 5745	<code>\seq_gput_left:Nv</code>	5293 , 5527
<code>\seq_get_left:NNF</code>	5572	<code>\seq_gput_left:Nx</code>	5293 , 5529
<code>\seq_get_left:NNT</code>	5571	<code>\seq_gput_right:cn</code>	5293
<code>\seq_get_left:NNTF</code>	103 , 5573	<code>\seq_gput_right:co</code>	5293
<code>\seq_get_left_aux:NnwN</code>	5372 , 5375 , 5378	<code>\seq_gput_right:cV</code>	5293
<code>\seq_get_left_aux:Nw</code>	5559	<code>\seq_gput_right:cv</code>	5293
<code>\seq_get_right:cN</code>	5398 , 5556	<code>\seq_gput_right:cx</code>	5293
<code>\seq_get_right:NN</code>		<code>\seq_gput_right:Nn</code>	
.	99 , 5398 , 5398 , 5421 , 5556 , 5564	99 , 5293 , 5295 , 5299 , 5300
<code>\seq_get_right:NNF</code>	5575	<code>\seq_gput_right:No</code>	5293
<code>\seq_get_right:NNT</code>	5574	<code>\seq_gput_right:NV</code>	5293 , 10120
<code>\seq_get_right:NNTF</code>	103 , 5576	<code>\seq_gput_right:Nv</code>	5293
<code>\seq_get_right_aux:NN</code>		<code>\seq_gput_right:Nx</code>	5293 , 10194
.	5398 , 5401 , 5404 , 5567	<code>\seq_gremove_all:cn</code>	5319
<code>\seq_get_right_loop:nn</code>		<code>\seq_gremove_all:Nn</code> 100 , 5319 , 5321 , 5344	
.	5398 , 5407 , 5416 , 5419 , 5436	<code>\seq_gremove_duplicates:c</code>	5303
<code>\seq_gpop:cN</code>	5535 , 5540	<code>\seq_gremove_duplicates:N</code>	
<code>\seq_gpop:NN</code> 102 , 5535 , 5539 , 10202 , 13729		100 , 5303 , 5305 , 5318
<code>\seq_gpop_left:cN</code>	5381 , 5540 , 5577	<code>\seq_greverse:c</code>	5705
<code>\seq_gpop_left:NN</code>		<code>\seq_greverse:N</code>	105 , 5705 , 5708 , 5723
.	100 , 5381 , 5383 , 5397 , 5539 , 5577 , 5584	<code>\seq_gset_eq:cc</code>	5239 , 5246
<code>\seq_gpop_left:NNF</code>	5609	<code>\seq_gset_eq:cN</code>	5239 , 5245
<code>\seq_gpop_left:NNT</code>	5608	<code>\seq_gset_eq:Nc</code>	5239 , 5244
<code>\seq_gpop_left:NNTF</code>	103 , 5610	<code>\seq_gset_eq:NN</code>	98 , 5239 , 5243 , 5306
<code>\seq_gpop_right:cN</code>	5422 , 5577	<code>\seq_gset_filter:NNn</code>	105 , 5724 , 5726
<code>\seq_gpop_right:NN</code>		<code>\seq_gset_from_clist:cc</code>	5679
.	100 , 5422 , 5424 , 5449 , 5577 , 5598	<code>\seq_gset_from_clist:cN</code>	5679
<code>\seq_gpop_right:NNF</code>	5615	<code>\seq_gset_from_clist:cn</code>	5679
<code>\seq_gpop_right:NNT</code>	5614	<code>\seq_gset_from_clist:Nc</code>	5679
<code>\seq_gpop_right:NNTF</code>	104 , 5616	<code>\seq_gset_from_clist:NN</code>	
<code>\seq_gpush:cn</code>	5515 , 5530	105 , 5679 , 5689 , 5702 , 5703
<code>\seq_gpush:co</code>	5515 , 5533	<code>\seq_gset_from_clist:Nn</code> 5679 , 5694 , 5704	
<code>\seq_gpush:cV</code>	5515 , 5531	<code>\seq_gset_map:NNn</code>	105 , 5734 , 5736
<code>\seq_gpush:cv</code>	5515 , 5532	<code>\seq_gset_split:Nnn</code>	98 , 5247 , 5249
<code>\seq_gpush:cx</code>	5515 , 5534	<code>\seq_if_empty:c</code>	5345 , 5347
<code>\seq_gpush:Nn</code>	103 , 5515 , 5525	<code>\seq_if_empty:N</code>	5345 , 5345
<code>\seq_gpush:No</code>	5515 , 5528	<code>\seq_if_empty:NNTF</code> 100 , 6172 , 9278 , 13727	
<code>\seq_gpush:Nv</code>	5515 , 5526 , 10199	<code>\seq_if_empty_break_return_false:N</code> .	
<code>\seq_gpush:Nv</code>	5515 , 5527	5549 , 5549 , 5558 , 5566 , 5579 , 5586 , 5593 , 5600
<code>\seq_gpush:Nx</code>	5515 , 5529 , 13716	<code>\seq_if_empty_err_break:N</code>	
<code>\seq_gput_left:cn</code>	5293 , 5530	105 , 5374 , 5387 , 5400 , 5428 , 5454 , 5454

<code>\seq_if_in:cn</code>	5349	<code>\seq_new:N</code>	4 , 98 , 2630 ,
<code>\seq_if_in:co</code>	5349		2647 , 5229 , 5229 , 5302 , 9011 , 10114 ,
<code>\seq_if_in:cV</code>	5349		10115 , 10124 , 10126 , 10129 , 13682
<code>\seq_if_in:cv</code>	5349	<code>\seq_pop:cN</code>	5535 , 5538
<code>\seq_if_in:cx</code>	5349	<code>\seq_pop:NN</code>	102 , 5535 , 5537
<code>\seq_if_in:Nn</code>	5349 , 5349	<code>\seq_pop_item_def</code>	106
<code>\seq_if_in:NnF</code> ...	5312 , 5368 , 5369 , 10254	<code>\seq_pop_item_def:</code> ..	5341 , 5412 , 5444 ,
<code>\seq_if_in:NnT</code>	5366 , 5367		5474 , 5490 , 5500 , 5511 , 5732 , 5742
<code>\seq_if_in:NnTF</code> ...	101 , 5370 , 5371 , 9024	<code>\seq_pop_left:cN</code>	5381 , 5538 , 5577
<code>\seq_if_in:No</code>	5349	<code>\seq_pop_left:NN</code>	
<code>\seq_if_in:Nv</code>	5349		99 , 5381 , 5381 , 5396 , 5537 , 5577 , 5577
<code>\seq_if_in:Nv</code>	5349	<code>\seq_pop_left:NNF</code>	5606
<code>\seq_if_in:Nx</code>	5349	<code>\seq_pop_left:NNT</code>	5605
<code>\seq_if_in_aux:</code>	5349 , 5358 , 5365	<code>\seq_pop_left:NNTF</code>	103 , 5607
<code>\seq_item:cn</code>	5627	<code>\seq_pop_left_aux:NNN</code>	
<code>\seq_item:n</code>	105 , 5222 ,		5381 , 5382 , 5384 , 5385
	5222 , 5286 , 5288 , 5294 , 5296 , 5301 ,	<code>\seq_pop_left_aux:NnwNNN</code>	
	5354 , 5378 , 5391 , 5477 , 5482 , 5487 ,		5381 , 5388 , 5391 , 5580 , 5587
	5493 , 5712 , 5713 , 5715 , 5720 , 5740	<code>\seq_pop_right:cN</code>	5422 , 5577
<code>\seq_item:Nn</code>	104 , 5627 , 5627 , 5650	<code>\seq_pop_right:NN</code>	
<code>\seq_item_aux:nnn</code>	5627 , 5629 , 5643 , 5648		... 100 , 5422 , 5422 , 5448 , 5577 , 5591
<code>\seq_length:c</code>	5617	<code>\seq_pop_right:NNF</code>	5612
<code>\seq_length:N</code> .	104 , 5617 , 5617 , 5626 , 5634	<code>\seq_pop_right:NNT</code>	5611
<code>\seq_length_aux:n</code>	5617 , 5622 , 5625	<code>\seq_pop_right:NNTF</code>	104 , 5613
<code>\seq_map_break</code>	101	<code>\seq_pop_right_aux:NNN</code>	
<code>\seq_map_break:</code> ..	5450 , 5452 , 5464 , 10172		5422 , 5423 , 5425 , 5426
<code>\seq_map_break:n</code>	102 , 5450 , 5453	<code>\seq_pop_right_aux_ii:NNN</code>	
<code>\seq_map_function:cN</code>	5461		5422 , 5429 , 5432 , 5594 , 5601
<code>\seq_map_function:NN</code> ...	4 , 101 , 5461 ,	<code>\seq_push:cn</code>	5515 , 5520
	5461 , 5473 , 5546 , 5622 , 5651 , 6178	<code>\seq_push:co</code>	5515 , 5523
<code>\seq_map_function_aux:NNn</code>		<code>\seq_push:cV</code>	5515 , 5521
	5461 , 5463 , 5467 , 5471	<code>\seq_push:cv</code>	5522
<code>\seq_map_inline:cn</code>	5496	<code>\seq_push:cx</code>	5515 , 5524
<code>\seq_map_inline:Nn</code>	101 , 4931 ,	<code>\seq_push:Nn</code>	103 , 5515 , 5515
	5310 , 5496 , 5496 , 5502 , 10166 , 10263	<code>\seq_push:No</code>	5515 , 5518
<code>\seq_map_variable:ccn</code>	5503	<code>\seq_push:Nv</code>	5515 , 5516
<code>\seq_map_variable:cNn</code>	5503	<code>\seq_push:Nv</code>	5515 , 5517
<code>\seq_map_variable:Ncn</code>	5503	<code>\seq_push:Nx</code>	5515 , 5519
<code>\seq_map_variable:NNn</code>		<code>\seq_push_item_def:n</code>	106 , 5325 , 5406 ,
	101 , 5503 , 5503 , 5513 , 5514		5434 , 5474 , 5474 , 5498 , 5730 , 5740
<code>\seq_mapthread_function:ccN</code>	5653	<code>\seq_push_item_def:x</code> ...	5474 , 5479 , 5505
<code>\seq_mapthread_function:cNN</code>	5653	<code>\seq_push_item_def_aux:</code>	
<code>\seq_mapthread_function:NcN</code>	5653		5474 , 5476 , 5481 , 5484
<code>\seq_mapthread_function:NNN</code>		<code>\seq_put_left:cn</code>	5285 , 5520
	104 , 5653 , 5653 , 5677 , 5678	<code>\seq_put_left:co</code>	5285 , 5523
<code>\seq_mapthread_function_aux:NN</code> ...		<code>\seq_put_left:cV</code>	5285 , 5521
	5653 , 5655 , 5662	<code>\seq_put_left:cv</code>	5285 , 5522
<code>\seq_mapthread_function_aux:Nnnwnn</code> .		<code>\seq_put_left:cx</code>	5285 , 5524
	5653 , 5664 , 5670 , 5675	<code>\seq_put_left:Nn</code>	
<code>\seq_new:c</code>	5229 , 5230		... 99 , 5285 , 5285 , 5289 , 5290 , 5515

<code>\seq_put_left:No</code>	5285 , 5518	<code>\seq_set_split_aux:NNnn</code>	
<code>\seq_put_left:Nv</code>	5285 , 5516	5247 , 5248 , 5250 , 5251
<code>\seq_put_left:Nx</code>	5285 , 5517	<code>\seq_set_split_aux_end:</code>	
<code>\seq_put_right:cn</code>	5285 , 5519	..	5247 , 5260 , 5264 , 5271 , 5275 , 5277
<code>\seq_put_right:co</code>	5285	<code>\seq_set_split_aux_i:w</code>	
<code>\seq_put_right:cV</code>	5285	5247 , 5258 , 5265 , 5271
<code>\seq_put_right:cv</code>	5285	<code>\seq_set_split_aux_ii:w</code> 5247 , 5273 , 5277	
<code>\seq_put_right:cx</code>	5285	<code>\seq_show:c</code>	5541 , 5750
<code>\seq_put_right:Nn</code>	99 , 5285 , 5287 , 5291 , 5292 , 5313 , 9027 , 10255	<code>\seq_show:N</code> ...	103 , 5541 , 5541 , 5548 , 5749
<code>\seq_put_right:No</code>	5285	<code>\seq_tmp:w</code>	5705 , 5712 , 5715
<code>\seq_put_right:Nv</code>	5285	<code>\seq_top:cN</code>	5744 , 5746
<code>\seq_put_right:Nx</code>	5285	<code>\seq_top:NN</code>	5744 , 5745
<code>\seq_remove_all:cn</code>	5319	<code>\seq_use:c</code>	5651
<code>\seq_remove_all:Nn</code>		<code>\seq_use:N</code>	104 , 5651 , 5651 , 5652
.....	100 , 5319 , 5319 , 5343 , 10258	<code>\seq_wrap_item:n</code>	
<code>\seq_remove_all_aux:NNn</code>	5254 , 5278 , 5301 , 5301 , 5337 , 5434 , 5682 , 5687 , 5692 , 5697 , 5730
.....	5319 , 5320 , 5322 , 5323	<code>\set@color</code>	8217
<code>\seq_remove_duplicates:c</code>	5303	<code>\setbox</code>	612
<code>\seq_remove_duplicates:N</code>		<code>\setlanguage</code>	370
.....	100 , 5303 , 5303 , 5317 , 10261	<code>\sfcode</code>	667
<code>\seq_remove_duplicates_aux:NN</code>		<code>\sffamily</code>	7984
.....	5303 , 5304 , 5306 , 5307	<code>\shipout</code>	577
<code>\seq_reverse:c</code>	5705	<code>\show</code>	420
<code>\seq_reverse:N</code> ...	105 , 5705 , 5706 , 5722	<code>\showbox</code>	422
<code>\seq_reverse_aux:NN</code> 5705 , 5707 , 5709 , 5710		<code>\showboxbreadth</code>	436
<code>\seq_reverse_aux_item:nwn</code> 5705 , 5713 , 5717		<code>\showboxdepth</code>	437
<code>\seq_set_eq:cc</code>	5239 , 5242	<code>\showgroups</code>	697
<code>\seq_set_eq:cN</code>	5239 , 5241	<code>\showifs</code>	698
<code>\seq_set_eq:Nc</code>	5239 , 5240	<code>\showlists</code>	423
<code>\seq_set_eq:NN</code>		<code>\showthe</code>	421
..	98 , 5239 , 5239 , 5304 , 10160 , 10177	<code>\showtokens</code>	685
<code>\seq_set_filter:NNn</code> ...	105 , 5724 , 5724	<code>\skewchar</code>	634
<code>\seq_set_filter_aux:NNNn</code>		<code>\skip</code>	658
.....	5724 , 5725 , 5727 , 5728	<code>\skip_add:cn</code>	4256
<code>\seq_set_from_clist:cc</code>	5679	<code>\skip_add:Nn</code> ...	77 , 4256 , 4256 , 4258 , 4259
<code>\seq_set_from_clist:cN</code>	5679	<code>\skip_eval:n</code>	78 , 4269 , 4286 , 4286
<code>\seq_set_from_clist:cn</code>	5679	<code>\skip_gadd:cn</code>	4256
<code>\seq_set_from_clist:Nc</code>	5679	<code>\skip_gadd:Nn</code> ...	77 , 4256 , 4258 , 4260
<code>\seq_set_from_clist:NN</code>	105 , 5679 , 5679 , 5699 , 5700 , 10161 , 10269	<code>\skip_gset:cn</code>	4245
<code>\seq_set_from_clist:Nn</code>		<code>\skip_gset:Nn</code>	77 , 4245 , 4247 , 4249
.....	2643 , 2648 , 5679 , 5684 , 5701	<code>\skip_gset_eq:cc</code>	4250
<code>\seq_set_map:NNn</code>	105 , 5734 , 5734	<code>\skip_gset_eq:cN</code>	4250
<code>\seq_set_map_aux:NNNn</code>		<code>\skip_gset_eq:Nc</code>	4250
.....	5734 , 5735 , 5737 , 5738	<code>\skip_gset_eq:NN</code> ...	78 , 4250 , 4253 – 4255
<code>\seq_set_split:Nnn</code>	98 , 5247 , 5247	<code>\skip_gsub:cn</code>	4256
		<code>\skip_gsub:Nn</code>	78 , 4256 , 4263 , 4265
		<code>\skip_gzero:c</code>	4235
		<code>\skip_gzero:N</code> ..	77 , 4235 , 4236 , 4238 , 4242
		<code>\skip_gzero_new:c</code>	4239

- \skip_gzero_new:N . . . 77, 4239, 4241, 4244
 - \skip_horizontal:c 4290
 - \skip_horizontal:N 81, 4290, 4290, 4292, 4296
 - \skip_horizontal:n 4290, 4291, 7056, 7086
 - \skip_if_eq:nn 4266, 4266
 - \skip_if_eq:nnTF 78
 - \skip_if_infinite_glue:n 4276, 4276
 - \skip_if_infinite_glue:nTF 78, 4359
 - \skip_new:c 4227
 - \skip_new:N 77, 4227, 4228, 4234, 4240, 4242, 4304–4308, 10670
 - \skip_set:cn 4245
 - \skip_set:Nn 77, 4245, 4245, 4247, 4248
 - \skip_set_eq:cc 4250
 - \skip_set_eq:cN 4250
 - \skip_set_eq:Nc 4250
 - \skip_set_eq:NN 78, 4250, 4250–4252
 - \skip_show:c 4298
 - \skip_show:N 79, 4298, 4298, 4299
 - \skip_show:n 79, 4300, 4300
 - \skip_split_finite_else_action:nnNN 82, 4357, 4357
 - \skip_sub:cn 4256
 - \skip_sub:Nn 78, 4256, 4261, 4263, 4264
 - \skip_use:c 4288
 - \skip_use:N 79, 4287, 4288, 4288, 4289
 - \skip_vertical:c 4290
 - \skip_vertical:N 81, 4290, 4293, 4295, 4297
 - \skip_vertical:n 4290, 4294
 - \skip_zero:c 4235
 - \skip_zero:N 77, 4235, 4235–4237, 4240
 - \skip_zero_new:c 4239
 - \skip_zero_new:N 77, 4239, 4239, 4243
 - \skipdef 357
 - \space 49, 205
 - \spacefactor 576
 - \spaceskip 571
 - \span 384
 - \special 646
 - \splitbotmark 455
 - \splitbotmarks 681
 - \splitdiscards 728
 - \splitfirstmark 454
 - \splitfirstmarks 680
 - \splitmaxdepth 624
 - \splittopskip 625
 - \str_head:n 93, 4814, 4814, 4835, 4878
 - \str_head_aux:w 4814, 4816, 4820
 - \str_if_eq:nn 1455, 1455
 - \str_if_eq:nnF 1875, 1876, 9292
 - \str_if_eq:nnT 1873, 1874, 5327
 - \str_if_eq:nnTF 22, 1877, 1878, 2141, 3914, 3917, 9496
 - \str_if_eq:no 1871
 - \str_if_eq:nV 1871
 - \str_if_eq:on 1871
 - \str_if_eq:Vn 1871
 - \str_if_eq:VV 1871
 - \str_if_eq:xx 1455, 1461
 - \str_if_eq:xxTF 2153, 6376, 6500, 8624
 - \str_if_eq_p:nn 1871, 1872
 - \str_if_eq_return:xx 2773, 2782, 2798, 2820, 2840, 2858, 2871, 2882, 2892, 4897, 4960, 4960, 4973, 4977, 4978, 5165
 - \str_length_loop:NNNNNNNN 8654, 8659, 8663, 8669
 - \str_length_skip_spaces:N 8586, 8654, 8654
 - \str_length_skip_spaces:n 8654, 8655, 8656
 - \str_tail:n 93, 4814, 4822
 - \str_tail_aux:w 4814, 4824, 4828
 - \strcmp 230
 - \string 223, 238, 252, 639
- T**
- \T 1833, 2727, 2905
 - \tabskip 385
 - \tempa 106, 108, 109, 118, 124
 - \tempb 107, 108
 - \tex_above:D 467
 - \tex_abovedisplayshortskip:D 480
 - \tex_abovedisplayskip:D 481
 - \tex_abovewithdelims:D 468
 - \tex_accent:D 518
 - \tex_adjdemerits:D 555
 - \tex_advance:D 362, 3430, 3432, 3442, 3444, 4075, 4080, 4257, 4262, 4340, 4345, 10452, 10463, 10469, 10479, 10487, 10515, 10526, 10535, 10540, 10551, 10560, 10598, 10640, 11052, 11088, 11114, 11122, 11128, 11152, 11165, 11190, 11275, 11276, 11290, 11291, 11416, 11424, 11433, 11533, 11564–11566, 11568, 11569, 11601, 11602, 11606, 11607, 11616, 11617, 11620, 11621, 11627, 11750, 11751, 11763, 11775, 11785, 11790, 11818, 11823, 11834, 11852, 11861, 11909, 11910, 11913, 11914, 11976,

11258, 12218, 12219, 12253, 12258,	\tex_delimiter:D	460
12261, 12262, 12266, 12267, 12272,	\tex_delimiterfactor:D	509
12273, 12277, 12278, 12281, 12282,	\tex_delimitershortfall:D	508
12285, 12286, 12635, 12655, 12713,	\tex_dimen:D	657, 2792
12717, 12739, 12754, 12755, 12759,	\tex_dimendef:D	356, 2795
12760, 12765, 12766, 12770, 12771,	\tex_discretionary:D	520
12774, 12775, 12778, 12779, 12869,	\tex_displayindent:D	485
12873, 12921, 12944, 12973, 13003,	\tex_displaylimits:D	495
13011, 13014, 13061, 13064, 13065,	\tex_displaystyle:D	473
13067, 13083, 13103, 13107, 13120,	\tex_displaywidowpenalty:D	484
13121, 13124, 13125, 13130, 13131	\tex_displaywidth:D	486
\tex_afterassignment:D	\tex_divide:D	363, 10480, 10527,
..... 372, 2975, 3061, 10402	10552, 11127, 11395, 11586, 11651,	
\tex_aftergroup:D	11695, 11740, 11743, 11745, 11747,	
..... 373, 815	11811, 11853, 12966, 13016–13018	
\tex_atop:D	\tex_doublehyphendemerits:D	553
..... 469	\tex_dp:D	664, 6582
\tex_atopwithdelims:D	\tex_dump:D	647
..... 470	\tex_edef:D	351, 851
\tex_badness:D	\tex_else:D	404, 788, 847
..... 617	\tex_emergencystretch:D	568
\tex_baselineskip:D	\tex_end:D	442, 766, 1167, 8954, 9096
..... 545	\tex_endcsname:D	444, 807
\tex_batchmode:D	\tex_endgroup:D	377, 764, 812
..... 438	\tex_endinput:D	416, 8965
\tex_begingroup:D	\tex_endlinechar:D	307, 308, 322, 458, 4502
..... 376, 811	\tex_eqno:D	478
\tex_belowdisplayshortskip:D	\tex_errhelp:D	424, 8872
..... 482	\tex_errmessage:D	418, 1159, 8892
\tex_belowdisplayskip:D	\tex_errorcontextlines:D	425, 8910
..... 483	\tex_errorstopmode:D	439
\tex_binoppenalty:D	\tex_escapechar:D	457, 8511, 8547, 8553
..... 506	\tex_everycr:D	386
\tex_botmark:D	\tex_everydisplay:D	487, 767
..... 453	\tex_everyhbox:D	626
\tex_box:D	\tex_everyjob:D	655,
..... 661, 6576, 6596	4944, 4946, 10105, 10107, 10117, 10119	
\tex_boxmaxdepth:D	\tex_everymath:D	511, 768
..... 623	\tex_everypar:D	574
\tex_brokenpenalty:D	\tex_everyvbox:D	627
..... 580	\tex_exhyphenpenalty:D	550
\tex_catcode:D	\tex_expandafter:D	374, 801
..... 665, 1054, 1830–1836, 2277, 2292,	\tex_fam:D	366
2517, 2519, 2521, 3202, 4485, 4486	\tex_fi:D	405, 789, 849
\tex_char:D	\tex_finalhyphendemerits:D	554
..... 519	\tex_firstmark:D	452
\tex_chardef:D	\tex_floatingpenalty:D	599
..... 354, 840–	\tex_font:D	365
844, 846, 1041, 1042, 1911, 1913,	\tex_fontdimen:D	632
2901, 3410, 3411, 8265, 8268, 13690	\tex_fontname:D	456
\tex_cleaders:D		
..... 537		
\tex_closein:D		
..... 413, 8429, 8442		
\tex_closeout:D		
..... 408		
\tex_clubpenalty:D		
..... 548		
\tex_copy:D		
..... 605, 6570, 6597		
\tex_count:D		
..... 656, 2814		
\tex_countdef:D		
..... 355, 837, 2817		
\tex_cr:D		
..... 380		
\tex_crcr:D		
..... 381		
\tex_csname:D		
..... 443, 806		
\tex_day:D		
..... 651		
\tex_deadcycles:D		
..... 585		
\tex_def:D		
..... 350, 819–821, 831, 850		
\tex_defaultthyphenchar:D		
..... 635		
\tex_defaultskewchar:D		
..... 636		
\tex_delcode:D		
..... 666		

<code>\tex_futurelet:D</code>	361, 2969, 2971	<code>\tex_immediate:D</code> 407, 1154, 1156, 8329, 8480	
<code>\tex_gdef:D</code>	352, 864	<code>\tex_indent:D</code>	541
<code>\tex_global:D</code>	336, 341, 343, 367, 816, 816, 1257, 1262, 1913, 2971, 3394, 3414, 3426, 3434, 3436, 3446, 3448, 3455, 4040, 4051, 4057, 4076, 4081, 4236, 4247, 4253, 4258, 4263, 4319, 4330, 4336, 4341, 4346, 5177, 6572, 6578, 6632, 6673, 6679, 6685, 6715, 6721, 6727, 6733, 8691, 8695, 13690	<code>\tex_input:D</code>	415, 770, 10201
<code>\tex_globaldefs:D</code>	371	<code>\tex_inputlineno:D</code>	417, 1174, 1823, 8817
<code>\tex_halign:D</code>	378	<code>\tex_insert:D</code>	597
<code>\tex_hangafter:D</code>	556	<code>\tex_insertpenalties:D</code>	600
<code>\tex_hangindent:D</code>	557	<code>\tex_interlinepenalty:D</code>	579
<code>\tex_hbadness:D</code>	618	<code>\tex_italic_correction:D</code>	771
<code>\tex_hbox:D</code>	613, 6671, 6672, 6677, 6683, 6697, 6698	<code>\tex_italiccor:D</code>	347
<code>\tex_hfil:D</code>	521	<code>\tex_jobname:D</code>	654, 4954, 10108
<code>\tex_hfill:D</code>	523	<code>\tex_kern:D</code> 532, 6808, 7038, 7558, 7563, 7629, 7630, 7737, 7738, 8139, 8140	
<code>\tex_hfilneg:D</code>	522	<code>\tex_language:D</code>	449
<code>\tex_hfuzz:D</code>	620	<code>\tex_lastbox:D</code>	606, 6630, 7104
<code>\tex_hoffset:D</code>	595	<code>\tex_lastkern:D</code>	539
<code>\tex_holdinginserts:D</code>	598	<code>\tex_lastpenalty:D</code>	645
<code>\tex_hruler:D</code>	534	<code>\tex_lastskip:D</code>	540
<code>\tex_hsize:D</code>	559, 7220, 7263	<code>\tex_lccode:D</code>	668, 1053, 1828, 1829, 2276, 2291, 2593, 2595, 2597, 3204, 4483, 4484
<code>\tex_hskip:D</code>	524, 4290	<code>\tex_leaders:D</code>	536
<code>\tex_hss:D</code>	525, 6700, 6702, 7040	<code>\tex_left:D</code>	504
<code>\tex_ht:D</code>	663, 6581	<code>\tex_lefthyphenmin:D</code>	560
<code>\tex_hyphen:D</code>	348, 769	<code>\tex_leftskip:D</code>	562
<code>\tex_hyphenation:D</code>	649	<code>\tex_leqno:D</code>	479
<code>\tex_hyphenchar:D</code>	633	<code>\tex_let:D</code>	337, 341, 343, 349, 766–776, 785–818, 834, 850, 851, 864, 865, 1250, 1512, 1886, 1887
<code>\tex_hyphenpenalty:D</code>	551	<code>\tex_limits:D</code>	496
<code>\tex_if:D</code>	387, 791, 792	<code>\tex_linepenalty:D</code>	552
<code>\tex_ifcase:D</code>	388, 3306	<code>\tex_lineskip:D</code>	546
<code>\tex_ifcat:D</code>	389, 793	<code>\tex_lineskiplimit:D</code>	547
<code>\tex_ifdim:D</code>	392, 4028	<code>\tex_long:D</code>	368, 816, 817, 819, 821, 853, 855, 861, 863, 867, 869, 875, 877
<code>\tex_ifeof:D</code>	393, 8226	<code>\tex_looseness:D</code>	564
<code>\tex_iffalse:D</code>	398, 786	<code>\tex_lower:D</code>	601, 6607
<code>\tex_ifhbox:D</code>	394, 6608	<code>\tex_lowercase:D</code> 640, 1055, 1837, 4487, 4526	
<code>\tex_ifhmode:D</code>	400, 796	<code>\tex_mag:D</code>	448
<code>\tex_ifinner:D</code>	403, 798	<code>\tex_mark:D</code>	450
<code>\tex_ifmmode:D</code>	401, 795	<code>\tex_mathaccent:D</code>	461
<code>\tex_ifnum:D</code>	390, 813, 3304	<code>\tex_mathbin:D</code>	491
<code>\tex_ifodd:D</code>	391, 1184, 1886, 1887, 3305	<code>\tex_mathchar:D</code>	462
<code>\tex_iftrue:D</code>	399, 785	<code>\tex_mathchardef:D</code>	359, 848, 3406, 3407
<code>\tex_ifvbox:D</code>	395, 6609	<code>\tex_mathchoice:D</code>	459
<code>\tex_ifvmode:D</code>	402, 797	<code>\tex_mathclose:D</code>	492
<code>\tex_ifvoid:D</code>	396, 6610	<code>\tex_mathcode:D</code> 670, 2587, 2589, 2591, 3203	
<code>\tex_ifx:D</code>	397, 794	<code>\tex_mathinner:D</code>	493
<code>\tex_ignorespaces:D</code>	445	<code>\tex_mathop:D</code>	494
		<code>\tex_mathopen:D</code>	498
		<code>\tex_mathord:D</code>	499

<code>\tex_mathpunct:D</code>	500	<code>\tex_parshape:D</code>	558
<code>\tex_mathrel:D</code>	501	<code>\tex_parskip:D</code>	565
<code>\tex_mathsurround:D</code>	512	<code>\tex_patterns:D</code>	648
<code>\tex_maxdeadcycles:D</code>	582	<code>\tex_pausing:D</code>	435
<code>\tex_maxdepth:D</code>	583	<code>\tex_penalty:D</code>	643
<code>\tex_meaning:D</code>	642, 804, 808	<code>\tex_postdisplaypenalty:D</code>	490
<code>\tex_medmuskip:D</code>	513	<code>\tex_predisdisplaypenalty:D</code>	489
<code>\tex_message:D</code>	419	<code>\tex_predisplaysize:D</code>	488
<code>\tex_mkern:D</code>	466	<code>\tex_pretolerance:D</code>	569
<code>\tex_month:D</code>	652	<code>\tex_prevdepth:D</code>	616
<code>\tex_moveleft:D</code>	602, 6601	<code>\tex_prevgraf:D</code>	575
<code>\tex_moveright:D</code>	603, 6603	<code>\tex_protected:D</code> 816, 818, 831, 852, 854, 856–859, 861, 863, 871, 873, 875, 877	
<code>\tex_mskip:D</code>	463	<code>\tex_radical:D</code>	464
<code>\tex_multiply:D</code>	364, 11325, 11532, 11746, 11762, 12740, 13329	<code>\tex_raise:D</code>	604, 6605
<code>\tex_muskip:D</code>	660	<code>\tex_read:D</code>	411, 8689, 8691
<code>\tex_muskipdef:D</code>	358	<code>\tex_relax:D</code>	446, 810, 3303, 4030
<code>\tex_newlinechar:D</code>	414, 4503	<code>\tex_relpenalty:D</code>	507
<code>\tex_noalign:D</code>	382	<code>\tex_right:D</code>	505
<code>\tex_noboundary:D</code>	517	<code>\tex_righthyphenmin:D</code>	561
<code>\tex_noexpand:D</code>	375, 802	<code>\tex_rightskip:D</code>	563
<code>\tex_noindent:D</code>	543	<code>\tex_romannumeral:D</code>	
<code>\tex_nolimits:D</code>	497 638, 814, 1534, 1546, 1552, 1594, 1598, 1603, 1609, 1615, 1621, 1633, 1638, 1640, 1647, 1702, 1709, 1714, 1722, 1724, 1727, 1734, 1740, 1749, 1764, 1768, 1773, 2110, 2123, 2136, 2148, 2159, 4918, 4980, 5036, 5059, 5107, 5115, 5138, 9317	
<code>\tex_nonscript:D</code>	477	<code>\tex_scriptfont:D</code>	630
<code>\tex_nonstopmode:D</code>	440	<code>\tex_scriptscriptfont:D</code>	631
<code>\tex_nulldelimiterspace:D</code>	510	<code>\tex_scriptscriptstyle:D</code>	476
<code>\tex_nullfont:D</code>	628, 2928	<code>\tex_scriptspace:D</code>	516
<code>\tex_number:D</code>	637, 3301	<code>\tex_scriptstyle:D</code>	475
<code>\tex_omit:D</code>	383	<code>\tex_scrollmode:D</code>	441
<code>\tex_openin:D</code>	409, 8316	<code>\tex_setbox:D</code>	
<code>\tex_openout:D</code>	410, 8329 612, 6570, 6576, 6630, 6672, 6677, 6683, 6714, 6719, 6725, 6731, 6753	
<code>\tex_or:D</code>	406, 787	<code>\tex_setlanguage:D</code>	370
<code>\tex_outer:D</code>	369	<code>\tex_sfcode:D</code> . 667, 2605, 2607, 2609, 3206	
<code>\tex_output:D</code>	584	<code>\tex_shipout:D</code>	577
<code>\tex_outputpenalty:D</code>	594	<code>\tex_show:D</code>	420, 809
<code>\tex_over:D</code>	471	<code>\tex_showbox:D</code>	422, 6651
<code>\tex_overfullrule:D</code>	622	<code>\tex_showboxbreadth:D</code>	436, 6661
<code>\tex_overline:D</code>	502	<code>\tex_showboxdepth:D</code>	437, 6662
<code>\tex_overwithdelims:D</code>	472	<code>\tex_showlists:D</code>	423
<code>\tex_pagedepth:D</code>	586	<code>\tex_showthe:D</code> . 421, 1412, 2521, 2591, 2597, 2603, 2609, 3211, 3214, 3217, 3220, 3223, 3952, 4212, 4301, 4356	
<code>\tex_pagefillllstretch:D</code>	590	<code>\tex_skewchar:D</code>	634
<code>\tex_pagefillstretch:D</code>	589		
<code>\tex_pagefilstretch:D</code>	588		
<code>\tex_pagegoal:D</code>	592		
<code>\tex_pageshrink:D</code>	591		
<code>\tex_pagestretch:D</code>	587		
<code>\tex_pagetotal:D</code>	593		
<code>\tex_par:D</code>	542, 8209		
<code>\tex_parfillskip:D</code>	573		
<code>\tex_parindent:D</code>	566		

<code>\tex_skip:D</code>	658, 2834	<code>\tex_valign:D</code>	379
<code>\tex_skipdef:D</code>	357, 2837	<code>\tex_vbadness:D</code>	619
<code>\tex_space:D</code>	346	<code>\tex_vbox:D</code>	
<code>\tex_spacefactor:D</code>	576		614, 6707, 6710, 6712, 6714, 6725, 6731
<code>\tex_spaceskip:D</code>	571	<code>\tex_vcenter:D</code>	465
<code>\tex_span:D</code>	384	<code>\tex_vfil:D</code>	526
<code>\tex_special:D</code>	646	<code>\tex_vfill:D</code>	528
<code>\tex_splitbotmark:D</code>	455	<code>\tex_vfilneg:D</code>	527
<code>\tex_splitfirstmark:D</code>	454	<code>\tex_vfuzz:D</code>	621
<code>\tex_splitmaxdepth:D</code>	624	<code>\tex_voffset:D</code>	596
<code>\tex_splittopskip:D</code>	625	<code>\tex_vrule:D</code>	535, 7991, 8046
<code>\tex_string:D</code>	639, 805	<code>\tex_vsize:D</code>	578
<code>\tex_tabskip:D</code>	385	<code>\tex_vskip:D</code>	529, 4293
<code>\tex_textfont:D</code>	629	<code>\tex_vsplit:D</code>	607, 6753
<code>\tex_textstyle:D</code>	474	<code>\tex_vss:D</code>	530
<code>\tex_the:D</code>		<code>\tex_vtop:D</code>	615, 6708, 6719
	308, 447, 1174, 1564, 1568, 1823,	<code>\tex_wd:D</code>	662, 6583
	2519, 2589, 2595, 2601, 2607, 3209,	<code>\tex_widowpenalty:D</code>	549
	3212, 3215, 3218, 3221, 3458, 4207,	<code>\tex_write:D</code>	412, 1154, 1156, 8475
	4288, 4351, 4946, 10107, 10119, 13716	<code>\tex_xdef:D</code>	353, 865
<code>\tex_thickmuskip:D</code>	515	<code>\tex_xleaders:D</code>	538
<code>\tex_thinmuskip:D</code>	514	<code>\tex_xspaceskip:D</code>	572
<code>\tex_time:D</code>	650	<code>\tex_year:D</code>	653
<code>\tex_toks:D</code>	659, 2852	<code>\textasteriskcentered</code>	4009, 4015
<code>\tex_toksdef:D</code>	360, 2855	<code>\textbardbl</code>	4014
<code>\tex_tolerance:D</code>	570	<code>\textdagger</code>	4010, 4016
<code>\tex_topmark:D</code>	451	<code>\textdaggerdbl</code>	4011, 4017
<code>\tex_topskip:D</code>	581	<code>\textfont</code>	629
<code>\tex_tracingcommands:D</code>	426	<code>\textparagraph</code>	4013
<code>\tex_tracinglostchars:D</code>	427	<code>\textsection</code>	4012
<code>\tex_tracingmacros:D</code>	428	<code>\textstyle</code>	474
<code>\tex_tracingonline:D</code>	429, 6663	<code>\TeXETstate</code>	729
<code>\tex_tracingoutput:D</code>	430	<code>\the</code>	70–79, 447
<code>\tex_tracingpages:D</code>	431	<code>\thickmuskip</code>	515
<code>\tex_tracingparagraphs:D</code>	432	<code>\thinmuskip</code>	514
<code>\tex_tracingrestores:D</code>	433	<code>\time</code>	650
<code>\tex_tracingstats:D</code>	434	<code>\tiny</code>	7984
<code>\tex_uccode:D</code>	669, 2599, 2601, 2603, 3205	<code>\tl_act:NNNnn</code>	4980, 4980
<code>\tex_uchyph:D</code>	567	<code>\tl_act_aux:NNNnn</code>	4980,
<code>\tex_undefined:D</code>	336, 343		4980, 4981, 5037, 5060, 5081, 5121
<code>\tex_underline:D</code>	503, 772	<code>\tl_act_case_aux:nn</code>	5108, 5116, 5119, 5138
<code>\tex_unhbox:D</code>	608, 6704	<code>\tl_act_case_group:nn</code>	5103, 5123, 5135
<code>\tex_unhcopy:D</code>	609, 6703	<code>\tl_act_case_normal:nN</code>	5103, 5122, 5127
<code>\tex_unkern:D</code>	533	<code>\tl_act_case_space:n</code>	5103, 5124, 5126
<code>\tex_unpenalty:D</code>	644	<code>\tl_act_end:w</code>	4980
<code>\tex_unskip:D</code>	531	<code>\tl_act_end:wn</code>	5002, 5008
<code>\tex_unvbox:D</code>	610, 6749	<code>\tl_act_group:nwnNNN</code>	4980, 4994, 5010
<code>\tex_unvcopy:D</code>	611, 6748	<code>\tl_act_group_recurse:Nnn</code>	4980, 5027, 5051
<code>\tex_uppercase:D</code>	641, 4527	<code>\tl_act_length_group:nn</code>	5077, 5083, 5091
<code>\tex_vadjust:D</code>	544	<code>\tl_act_length_normal:nN</code>	5077, 5082, 5089

<code>\tl_act_length_space:n</code>	5077 , 5084 , 5090	<code>\tl_gput_left:cn</code>	4434
<code>\tl_act_loop:w</code>	4980 , 4984 , 4988 , 5005 , 5013 , 5020	<code>\tl_gput_left:co</code>	4434
<code>\tl_act_normal:NwnNNN</code>	4980 , 4991 , 4999	<code>\tl_gput_left:cV</code>	4434
<code>\tl_act_output:n</code>	4980 , 5023 , 5126 , 5129 , 5137	<code>\tl_gput_left:cx</code>	4434
<code>\tl_act_result:n</code>	4986 , 5008 , 5023 – 5026	<code>\tl_gput_left:Nn</code>	85 , 4434 , 4442 , 4454 , 5294
<code>\tl_act_reverse_group:nn</code>	5032 , 5039 , 5049	<code>\tl_gput_left:No</code>	4434 , 4446 , 4456
<code>\tl_act_reverse_group_preserve:nn</code>	5062 , 5068	<code>\tl_gput_left:NV</code>	4434 , 4444 , 4455
<code>\tl_act_reverse_normal:nN</code>	5032 , 5038 , 5047 , 5061	<code>\tl_gput_left:Nx</code>	4434 , 4448 , 4457
<code>\tl_act_reverse_output:n</code>	4980 , 5025 , 5046 , 5048 , 5052 , 5069	<code>\tl_gput_right:cn</code>	4458
<code>\tl_act_reverse_space:n</code>	5032 , 5040 , 5045 , 5063	<code>\tl_gput_right:co</code>	4458
<code>\tl_act_space:wwnNNN</code>	4980 , 4995 , 5017	<code>\tl_gput_right:cV</code>	4458
<code>\tl_clear:c</code>	4396 , 5232 , 5764	<code>\tl_gput_right:cx</code>	4458
<code>\tl_clear:N</code>	84 , 4396 , 4396 , 4400 , 4403 , 5231 , 5763 , 8542 , 8544 , 8545 , 8633 , 9470 , 9474 , 10156 , 10219 , 10220 , 10239 , 11113 , 11391 , 11409 , 11644 , 11668 , 11688 , 11718 , 11732	<code>\tl_gput_right:Nn</code>	85 , 2503 , 4458 , 4466 , 4478 , 5296
<code>\tl_clear_new:c</code>	4402 , 5236 , 5768 , 9742 , 9744	<code>\tl_gput_right:No</code>	4458 , 4470 , 4480
<code>\tl_clear_new:N</code>	85 , 4402 , 4402 , 4406 , 5235 , 5767	<code>\tl_gput_right:NV</code>	4458 , 4468 , 4479
<code>\tl_const:cn</code>	4383 , 12445 – 12484 , 12791 – 12802	<code>\tl_gput_right:Nx</code>	4458 , 4472 , 4481 , 6342
<code>\tl_const:cx</code>	4383 , 8516 , 12881	<code>\tl_gremove_all:cn</code>	4584 , 5199
<code>\tl_const:Nn</code>	84 , 2398 , 2628 , 4383 , 4383 , 4393 , 4395 , 4490 , 4955 , 5093 , 5098 , 6237 , 8230 , 8508 , 8752 , 8753 , 8783 , 8788 , 8790 , 8792 , 8794 , 8796 , 8801 , 8802 , 8809 , 8827 , 9438 , 9440 , 9561 – 9565 , 10297 – 10301	<code>\tl_gremove_all:Nn</code>	86 , 4584 , 4586 , 4589 , 5198
<code>\tl_const:Nx</code>	4383 , 4388 , 4394 , 4954 , 6195 , 8512	<code>\tl_gremove_all_in:cn</code>	5191 , 5199
<code>\tl_elt_count:c</code>	5201 , 5206	<code>\tl_gremove_all_in:Nn</code>	5191 , 5198
<code>\tl_elt_count:N</code>	5201 , 5205	<code>\tl_gremove_in:cn</code>	5191 , 5195
<code>\tl_elt_count:n</code>	5201 , 5202	<code>\tl_gremove_in:Nn</code>	5191 , 5194
<code>\tl_elt_count:o</code>	5201 , 5204	<code>\tl_gremove_once:cn</code>	4578 , 5195
<code>\tl_elt_count:V</code>	5201 , 5203	<code>\tl_gremove_once:Nn</code>	86 , 4578 , 4580 , 4583 , 5194
<code>\tl_expandable_lowercase:n</code>	96 , 5103 , 5111	<code>\tl_greplace_all:cnn</code>	4528 , 5189
<code>\tl_expandable_uppercase:n</code>	96 , 5103 , 5103	<code>\tl_greplace_all:Nnn</code>	86 , 4528 , 4534 , 4539 , 4587 , 5188
<code>\tl_gclear:c</code>	4396 , 5234 , 5766	<code>\tl_greplace_all_in:cnn</code>	5181 , 5189
<code>\tl_gclear:N</code>	84 , 4396 , 4398 , 4401 , 4405 , 5233 , 5765	<code>\tl_greplace_all_in:Nnn</code>	5181 , 5188
<code>\tl_gclear_new:c</code>	4402 , 5238 , 5770	<code>\tl_greplace_in:cnn</code>	5181 , 5185
<code>\tl_gclear_new:N</code>	85 , 4402 , 4404 , 4407 , 5237 , 5769	<code>\tl_greplace_in:Nnn</code>	5181 , 5184
		<code>\tl_greplace_once:cnn</code>	4528 , 5185
		<code>\tl_greplace_once:Nnn</code>	85 , 4528 , 4530 , 4537 , 4581 , 5184
		<code>\tl_greverse:c</code>	5071
		<code>\tl_greverse:N</code>	91 , 5071 , 5073 , 5076
		<code>\tl_gset:cf</code>	4416
		<code>\tl_gset:cn</code>	4416
		<code>\tl_gset:co</code>	4416
		<code>\tl_gset:cx</code>	4416 , 12062 , 12148 , 12429
		<code>\tl_gset:Nc</code>	5175 , 5176
		<code>\tl_gset:Nf</code>	4416 , 5875
		<code>\tl_gset:Nn</code>	85 , 4416 , 4422 , 4431 , 4433 , 4495 , 5170 , 5384 , 5587 , 6279 , 6305 , 6461 , 6513 , 10200 , 10588 , 11042 , 11077 , 11158 , 11183 ,

- 11209, 11312, 11330, 11443, 11995,
 12092, 12293, 12486, 12804, 13138
 \tl_gset:No 4416, 4424
 \tl_gset:Nv 4416
 \tl_gset:Nx 4416,
 4426, 4432, 4531, 4535, 4802, 5074,
 5250, 5282, 5322, 5425, 5601, 5691,
 5696, 5709, 5727, 5737, 5782, 5838,
 5928, 6169, 6318, 10108, 10624, 12788
 \tl_gset_eq:cc
 . 4408, 4415, 5246, 5778, 6257, 10678
 \tl_gset_eq:cN
 . 4408, 4413, 5245, 5777, 6256, 10676
 \tl_gset_eq:Nc
 . 4408, 4414, 5244, 5776, 6255, 10677
 \tl_gset_eq:NN
 85, 4399, 4408, 4412, 5243,
 5775, 6254, 10112, 10566, 10578, 10675
 \tl_gset_rescan:cn 4492
 \tl_gset_rescan:cno 4492
 \tl_gset_rescan:cnx 4492
 \tl_gset_rescan:Nnn
 86, 4492, 4494, 4524, 4525
 \tl_gset_rescan:Nno 4492
 \tl_gset_rescan:Nnx 4492
 \tl_gtrim_spaces:c 4760
 \tl_gtrim_spaces:N .. 92, 4760, 4801, 4804
 \tl_head:f 4805
 \tl_head:n 92, 4805, 4807, 4812, 5209
 \tl_head:V 4805
 \tl_head:v 4805
 \tl_head:w 92, 4805, 4805, 4808, 4821,
 4834, 4850, 4870, 5210, 10376, 10387
 \tl_head_i:n 5208, 5209
 \tl_head_i:w 5208, 5210
 \tl_head_iii:f 5208
 \tl_head_iii:n 5208, 5211, 5212
 \tl_head_iii:w 5208, 5211, 5213
 \tl_if_blank:n 4590, 4590
 \tl_if_blank:nF .. 3905, 4594, 4598, 6102
 \tl_if_blank:nT 4593, 4597
 \tl_if_blank:nTF ... 87, 4595, 4599, 6148
 \tl_if_blank:o 4590
 \tl_if_blank:oTF 9483
 \tl_if_blank:V 4590
 \tl_if_blank_p:n 4592, 4596
 \tl_if_blank_p_aux:NNw 4590
 \tl_if_empty:c 4600, 5347, 5957
 \tl_if_empty:N ... 4600, 4600, 5345, 5956
 \tl_if_empty:n 4612, 4612
 \tl_if_empty:NF 4610, 10191, 10246
 \tl_if_empty:nF 3079, 4623, 6012
 \tl_if_empty:NT 4609, 9383
 \tl_if_empty:nT 4622
 \tl_if_empty:NTF
 87, 4611, 9528, 10184, 10236
 \tl_if_empty:nTF 87, 3947,
 4542, 4621, 5253, 5824, 8834, 9636
 \tl_if_empty:o 4624, 4633
 \tl_if_empty:oTF
 .. 2920, 4671, 4923, 5974, 6185, 6206
 \tl_if_empty:V 4612
 \tl_if_empty:x 5164, 5164
 \tl_if_empty_p:N 4608
 \tl_if_empty_p:n 4620
 \tl_if_empty_return:o
 4591, 4624, 4624, 4634
 \tl_if_eq:cc 4635, 5961, 6532
 \tl_if_eq:ccTF 9975
 \tl_if_eq:cN 4635, 5960, 6530
 \tl_if_eq:Nc 4635, 5959, 6531
 \tl_if_eq:NN 4635, 4635, 5958, 6529
 \tl_if_eq:nn 4647, 4647
 \tl_if_eq:NNF 4646
 \tl_if_eq:NNT ... 4645, 5334, 8057, 8060
 \tl_if_eq:NNTF . 88, 2164, 4644, 8579, 9063
 \tl_if_eq:nnTF 88
 \tl_if_eq_p:NN 4643
 \tl_if_head_eq_catcode:nN ... 4829, 4845
 \tl_if_head_eq_catcode:nNTF 93
 \tl_if_head_eq_charcode:fN 4829
 \tl_if_head_eq_charcode:nN .. 4829, 4829
 \tl_if_head_eq_charcode:nNF 4844
 \tl_if_head_eq_charcode:nNT 4843
 \tl_if_head_eq_charcode:nNTF
 94, 3810, 3823, 4842
 \tl_if_head_eq_charcode_p:nN ... 4841
 \tl_if_head_eq_meaning:nN ... 4829, 4861
 \tl_if_head_eq_meaning:nNTF ... 94, 9512
 \tl_if_head_eq_meaning_aux_normal:nN
 4864, 4868
 \tl_if_head_eq_meaning_aux_special:nN
 4865, 4876
 \tl_if_head_group:n 4901, 4901
 \tl_if_head_group:nTF 94, 4852, 4886, 4993
 \tl_if_head_N_type:n 4895, 4895
 \tl_if_head_N_type:nTF
 94, 4833, 4849, 4863, 4976, 4990
 \tl_if_head_space:n 4916, 4916

<code>\tl_if_head_space:nTF</code>	94	<code>\tl_map_inline:Nn</code> ..	89, 4691, 4701, 4703
<code>\tl_if_head_space_aux:w</code>	4916, 4919, 4921	<code>\tl_map_inline:nn</code> ..	89, 2766, 4691, 4691, 4702
<code>\tl_if_in:cn</code>	4662	<code>\tl_map_variable:cNn</code>	4704
<code>\tl_if_in:Nn</code>	4662	<code>\tl_map_variable:NNn</code> ..	89, 4704, 4710, 4719
<code>\tl_if_in:nn</code>	4668, 4668	<code>\tl_map_variable:nNn</code> ..	89, 4704, 4704, 4711
<code>\tl_if_in:NnF</code>	4663, 4666	<code>\tl_map_variable_aux:Nnn</code>	4704, 4706, 4712, 4717
<code>\tl_if_in:nnF</code>	4663, 4675	<code>\tl_new:c</code>	4377, 5230, 5762, 9724, 12061, 12147, 12428
<code>\tl_if_in:NnT</code>	4662, 4665, 8284, 8297, 10136, 10209	<code>\tl_new:cn</code>	5166
<code>\tl_if_in:nnT</code>	4662, 4674	<code>\tl_new:N</code>	84, 2494, 2963, 4377, 4377, 4382, 4403, 4405, 4660, 4661, 4956–4959, 5169, 5227–5229, 5759, 5761, 7115, 7141, 7142, 7979, 8258, 8259, 8499–8503, 8751, 8825, 8830, 9012–9014, 9455–9458, 9567–9569, 9571–9574, 10103, 10123, 10302, 10332, 10339, 10342, 10345, 10565, 12787, 13752
<code>\tl_if_in:NnTF</code>	88, 2497, 4664, 4667	<code>\tl_new:Nn</code>	5166, 5167, 5172, 5173
<code>\tl_if_in:nnTF</code>	88, 4664, 4676, 7644, 9606, 9613	<code>\tl_new:Nx</code>	5166
<code>\tl_if_in:no</code>	4668	<code>\tl_put_left:cn</code>	4434
<code>\tl_if_in:on</code>	4668	<code>\tl_put_left:co</code>	4434
<code>\tl_if_in:Vn</code>	4668	<code>\tl_put_left:cV</code>	4434
<code>\tl_if_single:N</code>	4968	<code>\tl_put_left:cx</code>	4434
<code>\tl_if_single:n</code>	4972, 4972	<code>\tl_put_left:Nn</code> ..	85, 4434, 4434, 4450, 5286
<code>\tl_if_single:NF</code>	4970	<code>\tl_put_left:No</code>	4434, 4438, 4452
<code>\tl_if_single:nF</code>	4970	<code>\tl_put_left:NV</code>	4434, 4436, 4451
<code>\tl_if_single:NT</code>	4969	<code>\tl_put_left:Nx</code>	4434, 4440, 4453
<code>\tl_if_single:nT</code>	4969	<code>\tl_put_right:cn</code>	4458
<code>\tl_if_single:NTF</code>	88, 4971	<code>\tl_put_right:co</code>	4458
<code>\tl_if_single:nTF</code>	88, 4971	<code>\tl_put_right:cV</code>	4458
<code>\tl_if_single_p:N</code>	4968	<code>\tl_put_right:cx</code>	4458
<code>\tl_if_single_p:n</code>	4968	<code>\tl_put_right:Nn</code>	85, 4458, 4458, 4474, 5288, 9529, 10228
<code>\tl_if_single_token:n</code>	4974, 4974	<code>\tl_put_right:No</code> ..	4458, 4462, 4476, 8890
<code>\tl_if_single_token:nTF</code>	88	<code>\tl_put_right:NV</code>	4458, 4460, 4475
<code>\tl_item:cn</code>	5140	<code>\tl_put_right:Nx</code> ..	4458, 4464, 4477, 6340, 8598, 8604, 8611, 8630, 8639, 8650, 9498, 9520, 9533, 9542, 10244
<code>\tl_item:Nn</code>	5140, 5162, 5163	<code>\tl_remove_all:cn</code>	4584, 5197
<code>\tl_item:nn</code>	97, 5140, 5140, 5162	<code>\tl_remove_all:Nn</code> ..	86, 4584, 4584, 4588, 5196
<code>\tl_item_aux:nn</code> ..	5140, 5142, 5155, 5160	<code>\tl_remove_all_in:cn</code>	5191, 5197
<code>\tl_length:c</code>	4731, 5206	<code>\tl_remove_all_in:Nn</code>	5191, 5196
<code>\tl_length:N</code> ..	91, 4731, 4736, 4743, 5205	<code>\tl_remove_in:cn</code>	5191, 5193
<code>\tl_length:n</code> ..	90, 4731, 4731, 4742, 5147, 5202	<code>\tl_remove_in:Nn</code>	5191, 5192
<code>\tl_length:o</code>	4731, 5204	<code>\tl_remove_once:cn</code>	4578, 5193
<code>\tl_length:V</code>	4731, 5203	<code>\tl_remove_once:Nn</code>	86, 4578, 4578, 4582, 5192
<code>\tl_length_aux:n</code> ..	4731, 4734, 4739, 4741	<code>\tl_replace_all:cnn</code>	4528, 5187
<code>\tl_length_tokens:n</code> ..	96, 5077, 5077, 5092		
<code>\tl_map_break</code>	90		
<code>\tl_map_break:</code> ...	4720, 4720, 8337, 8345		
<code>\tl_map_break:n</code>	4720, 4721, 5159		
<code>\tl_map_function:cN</code>	4677		
<code>\tl_map_function:NN</code>	89, 4677, 4683, 4690, 4739, 8310, 8323		
<code>\tl_map_function:nN</code>	89, 4677, 4677, 4684, 4734, 5254		
<code>\tl_map_function_aux:Nn</code>	4677, 4679, 4685, 4688, 4696		
<code>\tl_map_inline:cn</code>	4691		

`\tl_replace_all:Nnn` .. 86, [4528](#), 4532,
 4538, 4585, 5186, 5262, 9472, 9473
`\tl_replace_all_aux:`
 [4528](#), 4533, 4535, 4566, 4569
`\tl_replace_all_in:cnn` [5181](#), [5187](#)
`\tl_replace_all_in:Nnn` [5181](#), [5186](#)
`\tl_replace_aux:NNNnn`
 .. [4528](#), 4529, 4531, 4533, 4535, 4540
`\tl_replace_aux_ii:w` [4528](#), 4565, 4568, 4573
`\tl_replace_in:cnn` [5181](#), [5183](#)
`\tl_replace_in:Nnn` [5181](#), [5182](#)
`\tl_replace_once:cnn` [4528](#), [5183](#)
`\tl_replace_once:Nnn`
 85, [4528](#), 4528, 4536, 4579, 5182
`\tl_replace_once_aux:`
 [4528](#), 4529, 4531, 4571
`\tl_replace_once_aux_end:w`
 [4528](#), 4574, 4576
`\tl_rescan:nn` 86, [4492](#), 4496
`\tl_rescan_aux:w` [4492](#), 4510, 4518
`\tl_reverse:c` [5071](#)
`\tl_reverse:N` 91, [5071](#), [5071](#), [5075](#)
`\tl_reverse:n`
 91, [5055](#), [5055](#), 5070, 5072, 5074
`\tl_reverse:o` [5055](#)
`\tl_reverse:V` [5055](#)
`\tl_reverse_group_preserve:nn` ... [5055](#)
`\tl_reverse_items:n` 91, [4744](#), [4744](#)
`\tl_reverse_items_aux:nwNwn`
 [4744](#), 4746, 4747, 4751, 4754
`\tl_reverse_items_aux:wn`
 [4744](#), 4748, 4755, 4758
`\tl_reverse_tokens:n` 96, [5032](#), 5032, 5053
`\tl_set:cf` [4416](#)
`\tl_set:cn` [4416](#), 9743, 9747
`\tl_set:co` [4416](#)
`\tl_set:cx` [4416](#), 9727
`\tl_set:Nc` [5175](#), 5177, 5178
`\tl_set:Nf` [4416](#), 5873
`\tl_set:Nn` 85, 2251, 2981, 3002, [4416](#),
 4416, 4428, 4430, 4493, 4650, 4651,
 4714, 4936, 5256, 5330, 5339, 5353,
 5356, 5379, 5382, 5394, 5409, 5440,
 5507, 5580, 5870, 5882, 6053, 6277,
 6290, 6293, 6299, 6300, 6306, 6310,
 6405, 6455, 6466, 7122, 7126, 7314,
 7645, 7646, 7981, 7984, 8578, 8857,
 9044, 9045, 9471, 9581, 9618, 9685,
 9711, 9779, 9903, 9916, 9960, 10151,
 10238, 10241, 10587, 11039, 11074,
 11157, 11182, 11208, 11311, 11329,
 11442, 11994, 12091, 12292, 12485,
 12803, 13137, 13195, 13207, 13728
`\tl_set:No` [4416](#), 4418, 5179
`\tl_set:Nv` [4416](#)
`\tl_set:Nv` [4416](#)
`\tl_set:Nx` [4416](#), 4420, 4429,
 4529, 4533, 4800, 4938, 5072, 5248,
 5267, 5280, 5320, 5423, 5594, 5681,
 5686, 5707, 5725, 5735, 5780, 5836,
 5926, 6167, 6317, 8559, 8618, 8645,
 8861, 9382, 9384, 9515, 9526, 9541,
 9579, 9605, 9612, 9615, 9901, 9911,
 9933, 9934, 10171, 10365, 10378,
 10389, 10481, 10528, 10553, 10622,
 11142, 11145, 11191, 11439, 11803,
 11812, 11835, 11854, 12007, 12104,
 12305, 12499, 12582, 12615, 12842,
 12958, 12967, 13095, 13539, 13564
`\tl_set_eq:cc` [4408](#),
 4411, 5242, 5774, 6253, 9789, 10674
`\tl_set_eq:cN`
 . [4408](#), 4409, 5241, 5773, 6252, 10672
`\tl_set_eq:Nc` [4408](#),
 4410, 5240, 5772, 6251, 9967, 10673
`\tl_set_eq:NN` 85, 4397, [4408](#),
 4408, 5239, 5771, 6250, 10576, 10671
`\tl_set_rescan:cnn` [4492](#)
`\tl_set_rescan:cno` [4492](#)
`\tl_set_rescan:cnx` [4492](#)
`\tl_set_rescan:Nnn`
 86, [4492](#), 4492, 4522, 4523
`\tl_set_rescan:Nno` [4492](#), 10366
`\tl_set_rescan:Nnx` [4492](#)
`\tl_set_rescan_aux:NNnn`
 [4492](#), 4493, 4495, 4497, 4498
`\tl_show:c` [4940](#), 10680
`\tl_show:N` ... 95, [4940](#), 4940, 4941, 10679
`\tl_show:n` 95, [4942](#), 4942
`\tl_tail:f` [4805](#)
`\tl_tail:n` 93, [4805](#), 4809, 4813
`\tl_tail:V` [4805](#)
`\tl_tail:v` [4805](#)
`\tl_tail:w` .. 93, [4805](#), 4806, 10381, 10392
`\tl_tail_aux:w` 4810, 4811
`\tl_tmp:w` 4549,
 4569, 4574, 4670, 4671, 4760, 4798
`\tl_to_lowercase:n` 87,
 2278, 2293, 2729, 2768, 2907, 3174,
 [4526](#), 4526, 8508, 8878, 9311, 9464

\tl_to_str:c	4723	\token_if_alignment_tab:N	NTF	3280
\tl_to_str:N	90 , 4723 , 4723 , 4724 , 4938 , 8568 , 8569	\token_if_alignment_tab_p:N	3277
\tl_to_str:n	90 , 829 , 3136 , 4114 , 4200 , 4545 , 4614 , 4627 , 4722 , 4722 , 4817 , 4826 , 6259 , 6328 , 6349 , 6369 , 6370 , 6494 , 6495 , 8009 , 8092 , 8513 , 8659 , 9579 , 9612 , 9901 , 9911 , 9933 , 10011 , 10027 , 10664	\token_if_chardef:N	2760 , 2771
\tl_to_str_active_safe:Nx 97 , 4928 , 4928 , 8283 , 8296 , 10135 , 10208	\token_if_chardef:N	NTF	54 , 2810
\tl_to_uppercase:n 87 , 4526 , 4527	\token_if_chardef_aux:w	2760 , 2775 , 2784 , 2789
\tl_trim_spaces:c 4760	\token_if_cs:N	2746 , 2746
\tl_trim_spaces:N	.. 92 , 4760 , 4799 , 4803	\token_if_cs:N	NTF	54
\tl_trim_spaces:n	... 91 , 4760 , 4762 , 4800 , 4802 , 5274 , 6163 , 9516 , 9541	\token_if_dim_register:N	2760 , 2790
\tl_trim_spaces_aux_i:w 4760 , 4765 , 4776 , 4779 , 5798	\token_if_dim_register:N	NTF	55
\tl_trim_spaces_aux_ii:w	4770 , 4784 , 5802	\token_if_dim_register_aux:w	2760 , 2800 , 2807
\tl_trim_spaces_aux_ii:w\tl_trim_spaces_aux_iii:w 4760	\token_if_eq_catcode:NN	2713 , 2713
\tl_trim_spaces_aux_iii:w 4771 , 4786 , 4789 , 4793 , 5803	\token_if_eq_catcode:NNTF	53
\tl_trim_spaces_aux_iv:w	4760 , 4773 , 4795	\token_if_eq_catcode_p:NN	3038 , 3039 , 3188 , 3189
\tl_use:c 4725 , 5907	\token_if_eq_charcode:NN	2718 , 2718
\tl_use:N 90 , 4725 , 4725 , 4730 , 5906	\token_if_eq_charcode:NNTF	53
\token_get_arg_spec:N	... 58 , 3134 , 3147	\token_if_eq_meaning:NN	2708 , 2708
\token_get_prefix_arg_replacement_aux:wN 3134 , 3135 , 3142 , 3151 , 3160	\token_if_eq_meaning:NNTF	2288
\token_get_prefix_spec:N	59 , 3134 , 3138	\token_if_eq_meaning:NNTF	54 , 2303 , 3059	
\token_get_replacement_spec:N 59 , 3134 , 3156	\token_if_eq_meaning_p:NN	... 3040 , 3190	
\token_if_active:N 2703 , 2703	\token_if_expandable:N	2751 , 2751
\token_if_active:N	NF	\token_if_expandable:N	NTF	54
\token_if_active:N	NT	\token_if_group_begin:N	2650 , 2650
\token_if_active:N	NTF	\token_if_group_begin:N	NTF	52
\token_if_active_char:N 3276	\token_if_group_end:N	2655 , 2655
\token_if_active_char:N	NF	\token_if_group_end:N	NTF	52
\token_if_active_char:N	NT	\token_if_int_register:N	2760 , 2808
\token_if_active_char:N	NTF	\token_if_int_register:N	NTF	55
\token_if_active_char_p:N 3289	\token_if_int_register_aux:w	2760 , 2822 , 2831
\token_if_active_p:N 3289	\token_if_letter:N	2693 , 2693
\token_if_alignment:N 2665 , 2665	\token_if_letter:N	NTF	53
\token_if_alignment:N	NF	\token_if_long_macro:N	2760 , 2880
\token_if_alignment:N	NT	\token_if_long_macro:N	NTF	54
\token_if_alignment:N	NTF	\token_if_long_macro_aux:w	2760 , 2884 , 2894 , 2899
\token_if_alignment_p:N 3277	\token_if_macro:N	2723 , 2732
\token_if_alignment_tab:N 3276	\token_if_macro:N	NTF	54 , 2911 , 3140 , 3149 , 3158
\token_if_alignment_tab:N	NF	\token_if_macro_p_aux:w	2723 , 2734 , 2737	
\token_if_alignment_tab:N	NT	\token_if_math_shift:N	3276
\token_if_alignment_tab:N	NTF	\token_if_math_shift:N	NF	3283
\token_if_alignment_tab:N	NT	\token_if_math_shift:N	NT	3282
\token_if_alignment_tab:N	NTF	\token_if_math_shift:N	NTF	3284
\token_if_alignment_tab:N	NT	\token_if_math_shift_p:N	3281
\token_if_alignment_tab:N	NTF	\token_if_math_subscript:N	..	2683 , 2683
\token_if_alignment_tab:N	NT	\token_if_math_subscript:N	NTF	53
\token_if_alignment_tab:N	NT	\token_if_math_superscript:N	2678 , 2678	

<code>\token_if_math_superscript:NTF</code>	53	<code>\token_if_toks_register_aux:w</code>	
<code>\token_if_math_toggle:N</code>	2660, 2660		2760, 2860, 2867
<code>\token_if_math_toggle:NF</code>	3283	<code>\token_new:Nn</code>	51, 2610,
<code>\token_if_math_toggle:NT</code>	3282	2610, 2615, 2617–2619, 2621–2624	
<code>\token_if_math_toggle:NTF</code>	52, 3284	<code>\token_to_meaning:N</code>	52, 804, 804,
<code>\token_if_math_toggle_p:N</code>	3281	1180, 1190, 1203, 1843, 2735, 2776,	
<code>\token_if_mathchardef:N</code>	2760, 2780	2785, 2801, 2823, 2843, 2861, 2874,	
<code>\token_if_mathchardef:NTF</code>	55, 2812	2885, 2895, 2915, 3143, 3152, 3161	
<code>\token_if_other:N</code>	2698, 2698	<code>\token_to_str:c</code>	820, 820
<code>\token_if_other:NF</code>	3287	<code>\token_to_str:N</code>	5, 52, 804, 805,
<code>\token_if_other:NT</code>	3286	820, 1046, 1047, 1180, 1190, 1192,	
<code>\token_if_other:NTF</code>	53, 3288	1203, 1325, 1415, 1821, 2299, 2500,	
<code>\token_if_other_char:N</code>	3276	2778, 2787, 2803, 2825, 2845, 2863,	
<code>\token_if_other_char:NF</code>	3287	2876, 2887, 2897, 4907, 4932, 5457,	
<code>\token_if_other_char:NT</code>	3286	6654, 7169, 7174, 7313, 8188, 8192,	
<code>\token_if_other_char:NTF</code>	3288	8548–8552, 9277, 9284, 9292, 9377	
<code>\token_if_other_char_p:N</code>	3285	<code>\toks</code>	659
<code>\token_if_other_p:N</code>	3285	<code>\toksdef</code>	360
<code>\token_if_parameter:N</code>	2670, 2672	<code>\tolerance</code>	570
<code>\token_if_parameter:NTF</code>	53	<code>\topmark</code>	451
<code>\token_if_primitive:N</code>	2901, 2909	<code>\topmarks</code>	677
<code>\token_if_primitive:NTF</code>	55	<code>\topskip</code>	581
<code>\token_if_primitive_aux:NNw</code>		<code>\TotalHeight</code>	7338, 7342,
	2901, 2914, 2918	7346, 7350, 7357, 7859, 7886, 7887	
<code>\token_if_primitive_aux_loop:N</code>		<code>\tracingassigns</code>	687
	2901, 2921, 2934, 2940	<code>\tracingcommands</code>	426
<code>\token_if_primitive_aux_nullfont:N</code> . .		<code>\tracinggroups</code>	694
	2901, 2922, 2926	<code>\tracingifs</code>	690
<code>\token_if_primitive_aux_space:w</code>		<code>\tracinglostchars</code>	427
	2901, 2920, 2925	<code>\tracingmacros</code>	428
<code>\token_if_primitive_aux_undefined:N</code> . .		<code>\tracingnesting</code>	689
	2901, 2946, 2952	<code>\tracingonline</code>	429
<code>\token_if_primitive_auxii:Nw</code>		<code>\tracingoutput</code>	430
	2901, 2937, 2943	<code>\tracingpages</code>	431
<code>\token_if_protected_long_macro:N</code> . . .		<code>\tracingparagraphs</code>	432
	2760, 2889	<code>\tracingrestores</code>	433
<code>\token_if_protected_long_macro:NTF</code> . .	54	<code>\tracingscantokens</code>	688
<code>\token_if_protected_macro:N</code>	2760, 2868	<code>\tracingstats</code>	434
<code>\token_if_protected_macro:NTF</code>	54		
<code>\token_if_protected_macro_aux:w</code>			
	2760, 2873, 2878		
<code>\token_if_skip_register:N</code>	2760, 2832		
<code>\token_if_skip_register:NTF</code>	55		
<code>\token_if_skip_register_aux:w</code>			
	2760, 2842, 2849		
<code>\token_if_space:N</code>	2688, 2688		
<code>\token_if_space:NTF</code>	53		
<code>\token_if_toks_register:N</code>	2760, 2850		
<code>\token_if_toks_register:NTF</code>	55		

U

<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>\use_none:n</code>	18,
<code>\unvcopy</code>	611	904 , 904, 980, 1009, 1273, 1420,	
<code>\uppercase</code>	641	1424, 1426, 1434, 1438, 1446, 1448,	
<code>\use:c</code>	16, 878 , 878, 964,	1803, 1867, 2416, 2432, 2949, 3700,	
1146, 1148, 1150, 1152, 1945, 1955,		3815, 3819, 3824, 4591, 4796, 4882,	
2027, 2028, 3473, 3769, 3779, 3922,		4904, 4923, 4926, 4977, 5225, 5408,	
3931, 3933, 3935, 3936, 3940, 4117,		5437, 5469, 5645, 5672, 5673, 5812,	
8623, 8950, 8961, 8974, 8977, 8983,		5947, 6177, 8488, 8490, 9483, 9516,	
8994, 9002, 9008, 9030, 9091, 9113,		10484, 10531, 10556, 10611, 10653,	
9118, 9126, 9149, 9171, 9363, 9626,		11065, 11101, 11174, 11200, 11254,	
9633, 9795, 10004, 10370, 10400,		11375, 11518, 11838, 11857, 12016,	
10403, 10420, 10423, 10424, 10427,		12071, 12113, 12157, 12314, 12438,	
10430, 10720, 10782, 10838, 10888,		12508, 12730, 12847, 12890, 13274	
12040, 12127, 12338, 12525, 12861,		<code>\use_none:nn</code>	904 , 905,
13388, 13451, 13466, 13468, 13559		1802, 4759, 4973, 5335, 6206, 13338	
<code>\use:n</code>	17, 884 , 884,	<code>\use_none:nnn</code> ..	904 , 906, 1801, 6346, 9527
940, 980, 1009, 1271, 1412, 1421,		<code>\use_none:nnnn</code> ..	904 , 907, 1800, 10441
1423, 1427, 1435, 1437, 1445, 1449,		<code>\use_none:nnnnn</code>	904 , 908, 1799
1466, 2631, 4497, 4716, 4879, 4898,		<code>\use_none:nnnnnn</code>	904 , 909, 1798
5651, 6055, 6274, 6487, 6651, 9323		<code>\use_none:nnnnnnn</code>	904 , 910, 1797
<code>\use:nn</code> 884 , 885, 1537, 2640, 3134, 4112, 6039		<code>\use_none:nnnnnnnn</code>	904 , 911, 1796
<code>\use:nnn</code>	884 , 886	<code>\use_none:nnnnnnnnn</code>	
<code>\use:nnnn</code>	884 , 887	904 , 912, 1288, 1794, 1795
<code>\use:x</code>	19,	<code>\use_none_delimit_by_q_nil:w</code>	19, 898 , 898
879 , 879, 4197, 4505, 4516, 4933, 8565		<code>\use_none_delimit_by_q_recursion_stop:w</code>	
<code>\use_i:nn</code>	18, 824, 888 , 888, 914, 1067,	19, 46 ,
1096, 1124, 1282, 1425, 1439, 1447,		898 , 900, 962, 1030, 1787, 2408, 2423	
10470, 10516, 10541, 11115, 11434,		<code>\use_none_delimit_by_q_stop:w</code>	
11791, 11824, 12617, 12945, 13084		19, 898 , 899, 2319, 2323,
<code>\use_i:nnn</code>		4553, 5934, 6119, 6125, 8652, 8666	
18, 890 , 890, 1078, 1310, 3143, 12584		<code>\use_none_delimit_by_s_stop:w</code>	
<code>\use_i:nnnn</code>	18, 890 , 894	46 , 2508 , 2508
<code>\use_i_after_else:nw</code>	1505 , 1507	<code>\usepackage</code>	223
<code>\use_i_after_fi:nw</code>	1505 , 1506		
<code>\use_i_after_or:nw</code>	1505 , 1508		
<code>\use_i_after_orelse:nw</code>	1505 , 1509		
<code>\use_i_delimit_by_q_nil:nw</code> ..	19, 901 , 901		
<code>\use_i_delimit_by_q_recursion_stop:nw</code>			
.....	19, 46 , 901 , 903, 2414, 2430		
<code>\use_i_delimit_by_q_stop:nw</code>			
.....	19, 901 , 902, 1793, 6132		
<code>\use_i_ii:nnn</code>	18, 890 , 893, 1562		
<code>\use_ii:nn</code>	18, 826, 888 ,		
889, 916, 1069, 1098, 1126, 1284,			
1422, 1428, 1436, 1450, 6266, 9486			
<code>\use_ii:nnn</code> ..	18, 890 , 891, 1080, 3152, 9534		
<code>\use_ii:nnnn</code>	18, 890 , 895		
<code>\use_iii:nnn</code>	18, 890 , 892, 3161		
<code>\use_iii:nnnn</code>	18, 890 , 896		
<code>\use_iv:nnnn</code>	18, 890 , 897		

V

<code>\vadjust</code>	544
<code>\valign</code>	379
<code>\vbadness</code>	619
<code>\vbox</code>	614
<code>\vbox:n</code>	128, 6707 , 6707
<code>\vbox_gset:cn</code>	6713
<code>\vbox_gset:cw</code>	6730 , 6746
<code>\vbox_gset:Nn</code>	128, 6713 , 6715, 6717
<code>\vbox_gset:Nw</code> ..	129, 6730 , 6732, 6735, 6745
<code>\vbox_gset_end</code>	129
<code>\vbox_gset_end:</code>	6730 , 6741, 6747
<code>\vbox_gset_inline_begin:c</code> ...	6742 , 6746
<code>\vbox_gset_inline_begin:N</code> ...	6742 , 6745
<code>\vbox_gset_inline_end:</code>	6742 , 6747
<code>\vbox_gset_to_ht:cnn</code>	6724

[illegible]